An Introduction to NLP for LLMs

A large language model is a computer program that learns and generates human-like language. Large Language Models (LLMs) are **foundational machine learning models** that use deep learning algorithms to process and understand natural language.

These models are trained on massive amounts of text data to learn patterns and entity relationships in the language.
LLMs can perform many types of language tasks, such as

- translating languages,
- analyzing sentiments,
- chatbot conversations, and more.

They can understand

- complex textual data,
- identify entities and relationships between them,
- and generate new text that is coherent and grammatically accurate, making them ideal for sentiment analysis.

At the core of it all A language model is a machine learning model that aims to predict and generate plausible language.

Question: Is autocomplete a LLM?
Autocomplete is a language model,but **not an LLM**.

These models work by estimating the probability of a word or sequence of words occurring within a longer sequence of tokens. Consider the following sentence:

```
Once upon a ____
```

a **language model** determines the probabilities of different words or sequences of words to replace that underscore. For example, a language model might determine the following probabilities:

```
time 49.4%
dream 15.2%
reaction 3.6%
try 2.5%
```

```
location 2.2%
...
```

Estimating the probability of what comes next in a sequence is useful for all kinds of things: generating text, translating languages, and answering questions.

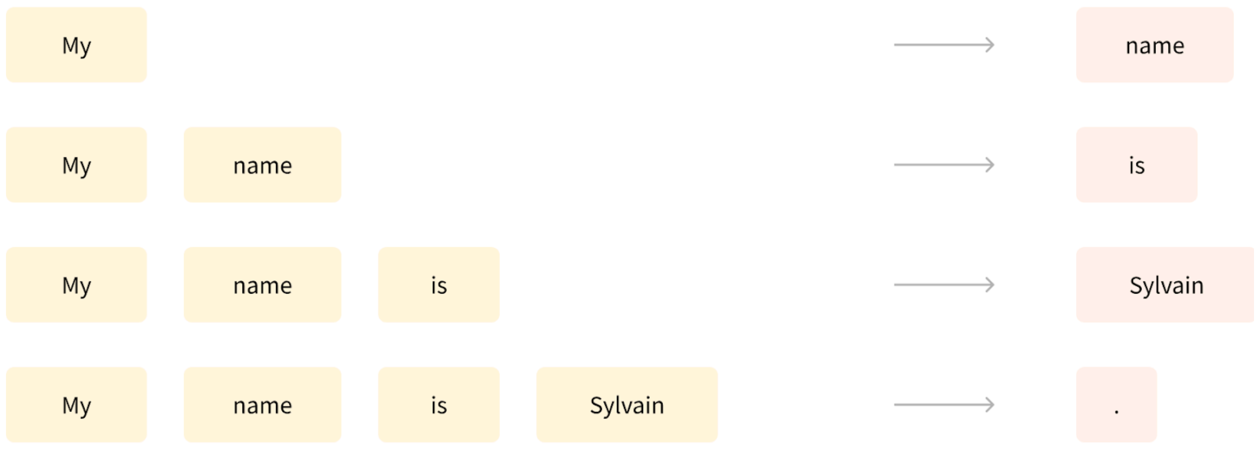
Modeling human language at scale is a highly complex and resource-intensive endeavor.
The path to reaching the current capabilities of language models and large language models has spanned several decades.

As models are built **bigger and bigger**, their complexity and efficacy increases.

Early language models could predict the probability of a single word;

Modern large language models can predict the probability of sentences, paragraphs, or even entire documents.

The size and capability of language models has exploded over the last few years as computer memory, dataset size, and processing power increases, and more effective techniques for modeling longer text sequences are developed.



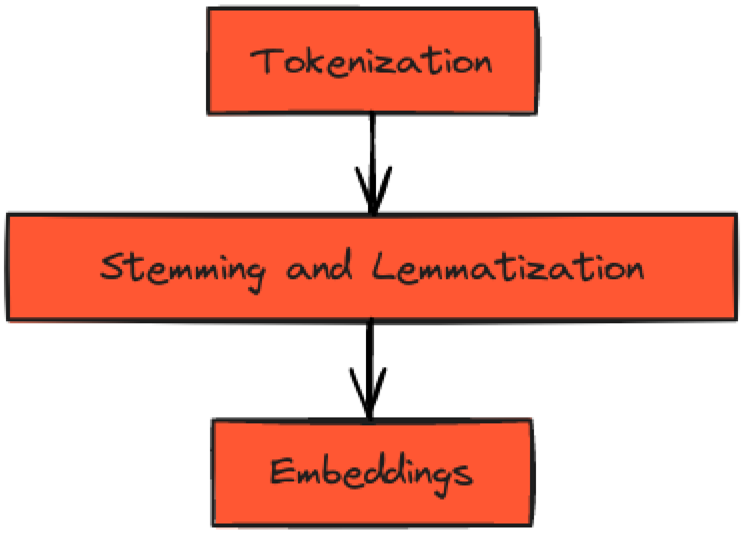
How does a computer handle language: NLP

NLP enables machines to not only gather text and speech but also identify the core meaning it should respond to.
Human language is complex, and constantly evolving, which means natural language processing has quite the challenge.

The general steps for handling language are:

- Breaking Down the Text: **Tokenization**
- Understanding the Words: **Stemming and Lemmatization**
- Converting to numbers: **Embeddings**

Along with this we also apply some **common preprocessing like removing stop words** that contribute much to the meaning of sentence (e.g., "the," "a," "is").



Language as seen by Models: Tokens

Tokenization is one of the many pieces of the puzzle in how NLP works.

Tokenization is a **way of separating a piece of text into smaller units called tokens.**

Imagine reading a sentence.
Your brain first recognizes individual words.

1. NLP mimics this process through tokenization.
2. It breaks down the text into smaller units called tokens,
3. which are usually words or sometimes subwords.
4. These tokens can be as small as characters or as long as words.
5. The primary reason this process matters is that it helps machines understand human language by breaking it down into bite-sized pieces, which are easier to analyze.

Here’s an example of a string of data:

```
"Hello, how are you"
```

In order for this sentence to be understood by a machine, tokenization is performed on the string to break it into individual parts. With tokenization, we'd get something like this:

```
'Hello' ',' 'how' 'are' 'you'
```

This is broken down into 5 tokens

Tokenizer

tokens can be individual words, punctuation marks, or even groups of characters depending on the situation.
Here is types of tokenization in nlp:

```
"Don't waste food."
```

- Word Tokenization: Common for languages with clear separation (e.g., English). ["Don't", "waste", "food"]

- Subwords Tokenization: Smaller than words, but bigger than characters (useful for complex languages or unknown words). ["Do", "n't", "waste", "food"]

- Character Tokenization: Useful for languages without clear separation or for very detailed analysis.Some languages, like **Chinese or Japanese**, don't have clear spaces between words.["D", "o", "n"," ' ", "t", "w", "a", "s", "t", "e", "f", "o", "o", "d"]

Now there are a **lot of different tokeniser** that we can use to build tokens according to need.

Popular Tokenizer Libraries:

- **NLTK (Python):** A widely used NLP library with basic tokenizers for words and sentences.
- **SpaCy (Python):** Another popular NLP library with advanced tokenizers that can handle various tasks and languages.
- **Hugging Face Transformers (Python):** Provides pre-trained models for many NLP tasks, each with specific tokenizers designed for the model's needs.

Try out this wonderful tokeniser playground:
https://huggingface.co/spaces/Xenova/the-tokenizer-playground (select different tokenizers in this for experiments)
https://gptforwork.com/tools/tokenizer

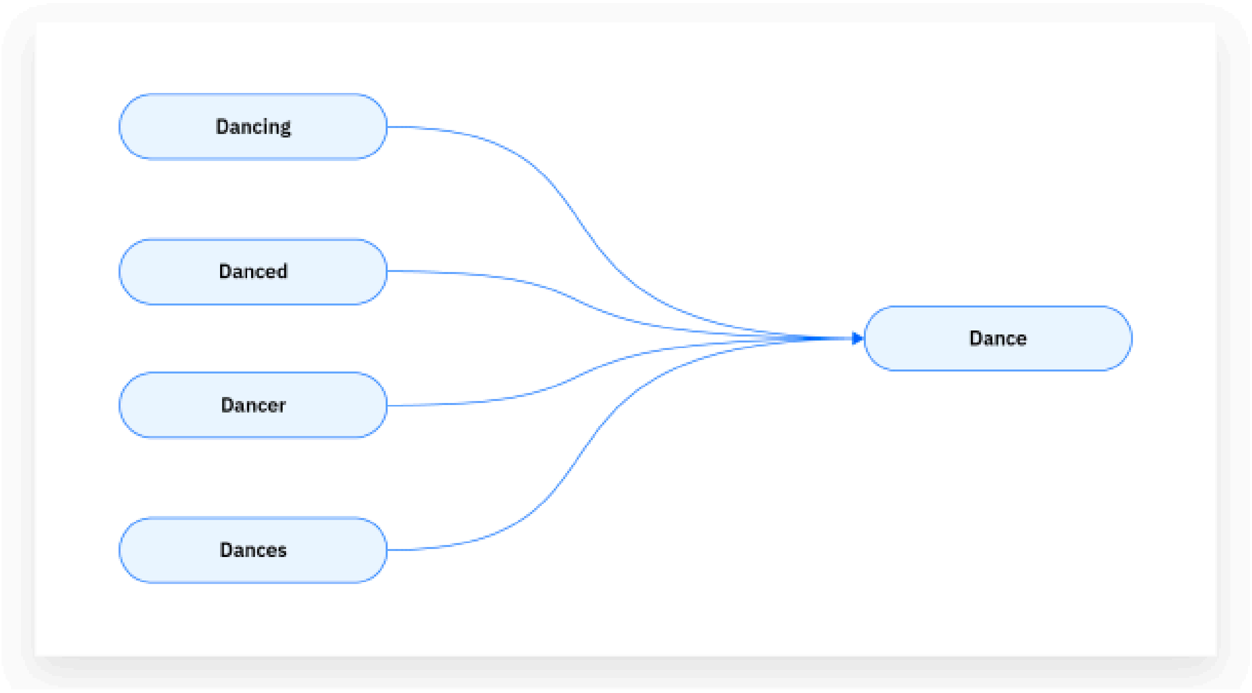
different word tokenizers from different sources (like ChatGPT, Hugging Face Transformers, or NLTK) can potentially produce different outputs for the same piece of text. Here's why:

Variations in Tokenization Rules:

- **Handling punctuation:** Some tokenizers might keep punctuation marks as separate tokens, while others might remove them or group them with adjacent words.
- **Contractions and special characters:** Tokenizers might handle contractions ("don't") differently, splitting them or keeping them as a single token. The treatment of special characters like emojis or symbols can also vary.
- **Text normalization:** Some tokenizers might convert all text to lowercase for consistency, while others might preserve the original case.

Understanding the Words: Stemming and Lemmatization

Once a computer tokenizes a text, it needs to understand different word forms. Consider the words “running”, “runner”, and “ran”. To us, they are related. But a computer sees them as separate words. Enter stemming and lemmatization.



stemming and lemmatization help computers understand the core meaning of words, regardless of their grammatical variations.
By reducing words to their base forms, computers can group words with similar meanings together.
Imagine searching for books about "happiness." This would group "happy," "happiest," and "happiness" under the same category, making searches more efficient.

1. Stemming: The Chop-It-Off Approach

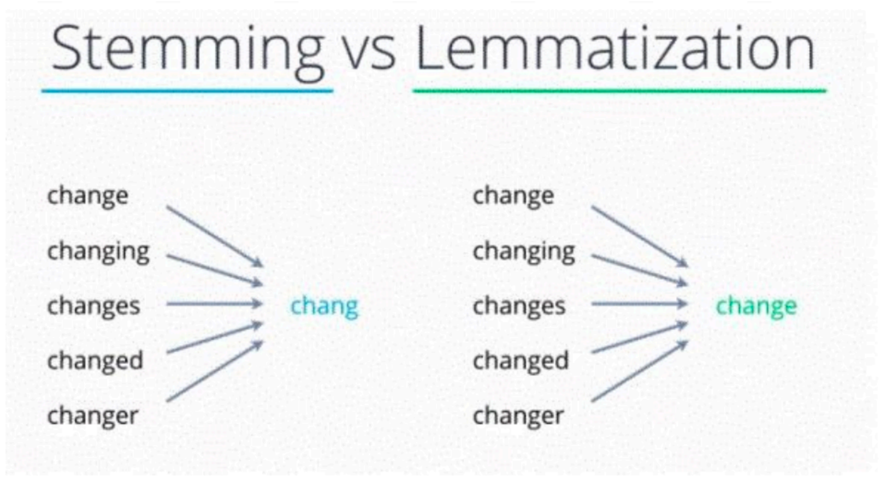
- Stemming is a simpler method. It works like this:
  - Imagine a book titled "Running." Stemming might chop off the ending "-ing" from "running" to get "run."
  - when we stem a word, **we chop off its inflections and keep what hopefully represents the main essence of the word.**This shortens the word to its "stem" or base form, like the root of a plant.
- But here's the catch: Stemming can be a bit blunt. Sometimes, chopping off the end can change the meaning of the word.
  - For example, "play" and "playing" would both be reduced to "pla" using stemming, which isn't quite right.

2. Lemmatization: The Dictionary Approach

- Lemmatization is a more sophisticated method. It uses a dictionary to find the base or "lemma" of a word, similar to looking up a word's definition.Lemmatization does not simply chop off inflections, but instead relies on a lexical knowledge base like **WordNet** to obtain the correct base forms of words.

- **WordNet** is like a giant digital dictionary on steroids, specifically designed for NLP. It goes beyond simply defining words; a manually constructed database of words, their synonyms, and relationships between them. It's like a giant digital thesaurus that captures **semantic connections**.

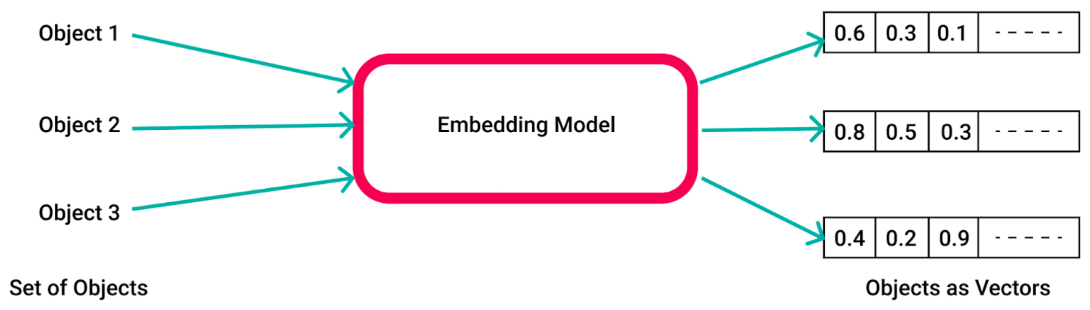
- Here's how it works:
    - The computer checks the dictionary for "running" and finds its lemma is "run."
    - This ensures the base word makes sense grammatically and has a real meaning.
    -
- Lemmatization is generally more accurate than stemming, but it can be computationally more expensive, like looking up every word in a giant dictionary.



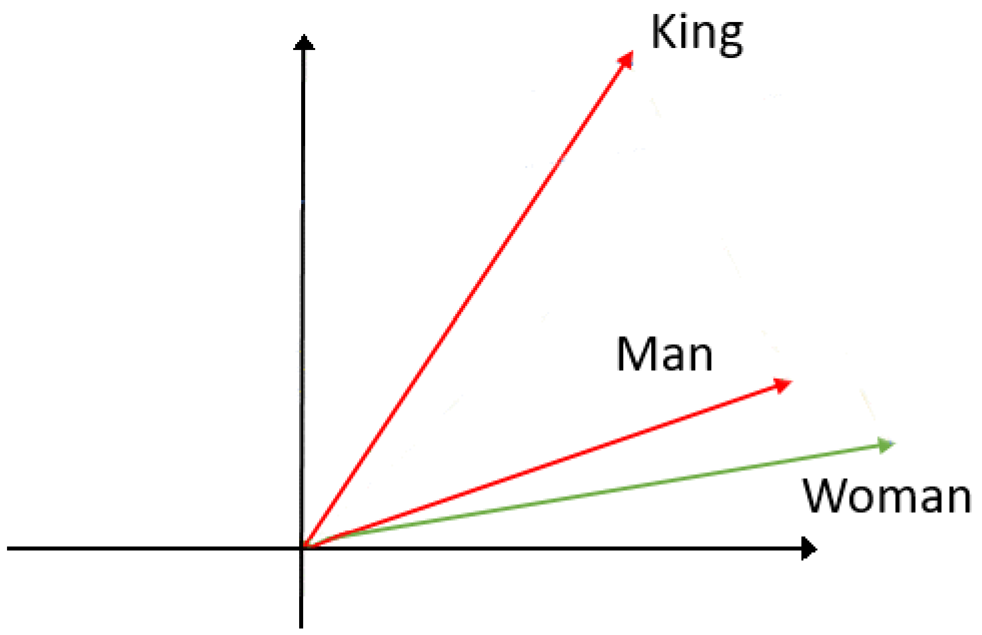
Converting to numbers: Embeddings

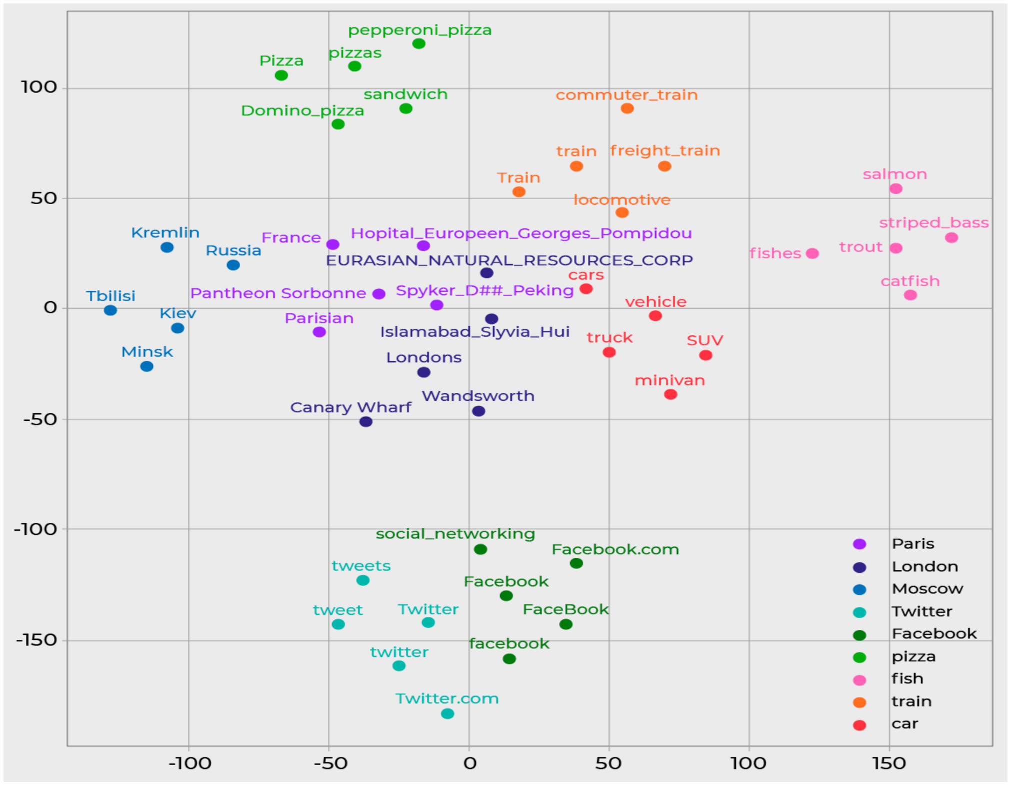
Imagine embeddings as a form of translating each token into a language that machines understand—numbers.
Embeddings convert words (or sometimes complete sentences) into numerical vectors, essentially creating a numerical code for each word. This allows LLMs to process language data mathematically.
Embeddings go beyond just assigning a number to a word. They capture the semantic relationships between words. Words with similar meanings (synonyms) will have similar vector representations in the embedding space. This lets the computers understand the

nuances of language, like the difference between "happy" and "sad" and the similarity between “happy” and “merry”.



Words with similar meanings tend to have similar vectors. This numerical representation allows computers to perform mathematical operations on words, leading to tasks like finding word similarities or even analogies.





- After the text is split into tokens Each token is mapped to a high-dimensional vector. These vectors are initially either:
    - Pre-trained (e.g., Word2Vec, GloVe) using large corpora.
        - These are simple neural network models specially trained to convert txt to numbers such that similar meaning words are converted to similar numbers, and are placed close to each other in a plot. Like the one shown above
    - Learned from scratch as part of the model's training process, as in the case of Transformers and newer models as we'll see after this..

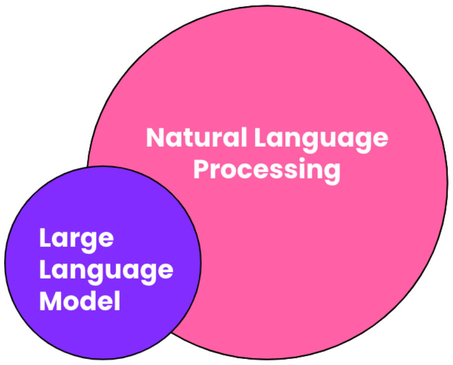
NLP and LLMs

Though both technologies are critical to the world of AI and language processing, NLP and LLMs are very different tools. NLP is a form of artificial intelligence and set of algorithms designed to understand and work with languages.

LLMs are a subset of NLP that are powered by some specific type of deep learning models, trained on massive datasets and are advanced architectures used in NLP

- NLP: Encompasses a broad range of techniques and tools designed to solve language-based tasks. This includes speech recognition, language translation, and sentiment analysis.

- LLM: Primarily focuses on generating and understanding text based on the training it has received from large datasets. It is a subset of the broader NLP field.



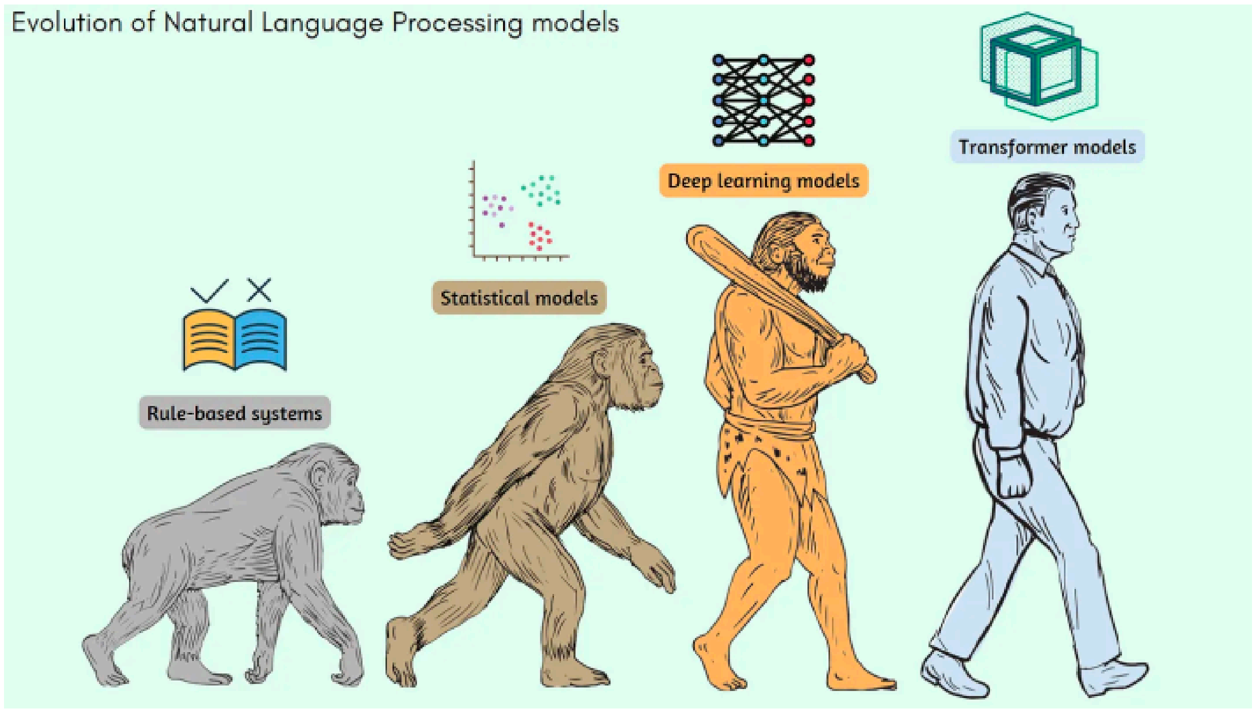
From NLP to LLMS: Transformers

Traditional NLP techniques involved

1. breaking down and analyzing text data using methods that could be quite complex and rigid.
2. While these techniques were useful, they often struggled with understanding context, meaning, and the nuances of human language. They treated words in isolation and had difficulty capturing relationships and meanings across larger text spans.

- **Sequential Processing:** Traditional NLP models often process text one word at a time, like reading a sentence from left to right. This can miss important connections between words that appear further apart in the sentence.

- **Limited Context:** These models might struggle to consider the full context of a sentence, making it difficult to understand the true meaning, especially in complex or nuanced language.



The Rise of Transformers

The introduction of Transformers marked a revolutionary shift in NLP, leading to the development of Large Language Models (LLMs) like BERT, GPT-3, and more.

Transformers brought in a **new way of processing text that overcame many limitations of previous methods**.

A transformer is a type of neural network architecture which is well suited for tasks that involve **processing sequences as inputs**.

The transformer consists of an **encoder-decoder transformer block** where the **encoder block takes input** and an **output translation sentence is generated by the decoder block**.

Encoder, Decoder Architecture

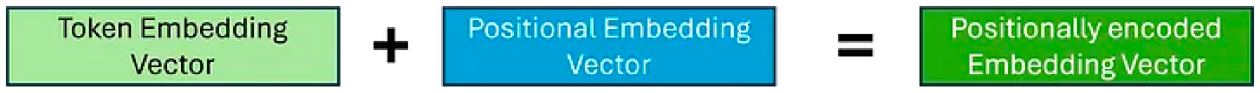
The model is primarily composed of two blocks:

- Encoders in Transformers are neural network layers that process the input sequence and produce a continuous representation, or embedding, of the input.

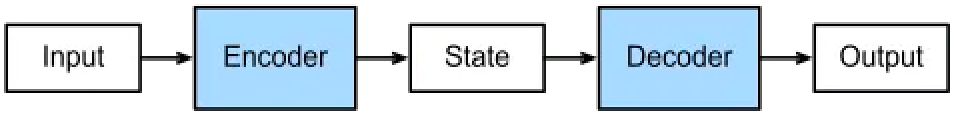
- The decoder then uses these embeddings to generate the output sequence.

So now we are taking in input and the model is internally converting it to embeddings in the form it requires.

To help the model understand the order, each token's embedding is modified by adding another vector called **positional encoding**. This encoding carries information about the position of a token within the sequence, allowing the model to understand sequences and the relative positioning of words.


Token Embedding Vector
+
Positional Embedding Vector
=
Positionally encoded Embedding Vector

the decoder's output is transformed into the actual output sequence, often one token at a time. Each output token is predicted based on the context provided by the input and what has been generated so far. The predicted tokens are then typically converted back into text, producing the final output that might be a translated sentence, a summary, or any other form of textual content.



Learning in a transformers

1. Encoder learning: The encoder's job is to compress input information into a compact representation. During training, it learns to:
    - Identify important features in the input
    - Discard irrelevant or redundant information
    - Create a condensed "summary" of the input

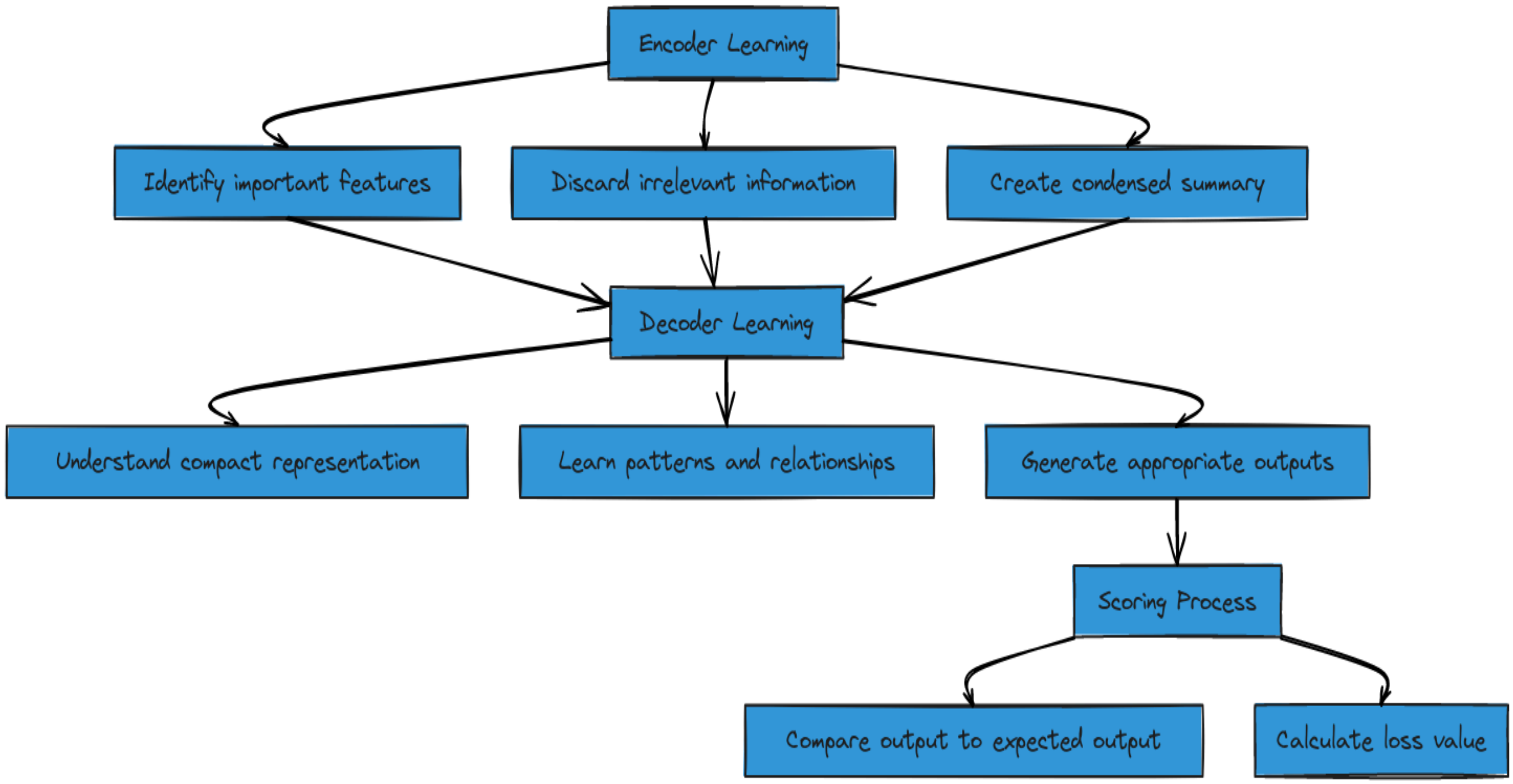
It's like learning to write concise notes from a long lecture. With practice, you get better at capturing the key points and leaving out unnecessary details.

2. Decoder learning: The decoder learns to **reconstruct** or generate output from the encoder's compressed representation. Its training process involves:
    - Understanding the compact representation
    - Learning patterns and relationships within the data
    - Generating appropriate outputs based on the encoded information

This is similar to learning how to expand bullet points into a full paragraph. You practice reconstructing the original meaning from the condensed form.

Scoring process:

- The encoder processes the input and creates a compressed representation.
- The decoder takes this representation and generates an output.
- This output is compared to the expected output using the loss function.
- The resulting score (loss value) indicates how well the model performed.



Encoder Learning
Identify important features
Discard irrelevant information
Create condensed summary
Decoder Learning
Understand compact representation
Learn patterns and relationships
Generate appropriate outputs
Scoring Process
Compare output to expected output
Calculate loss value

Attention

The key innovation adding to Transformers is the concept of "attention." **Attention allows models to focus on different parts of a text to understand the context and relationships between words better. Here's a simple way to understand this:**

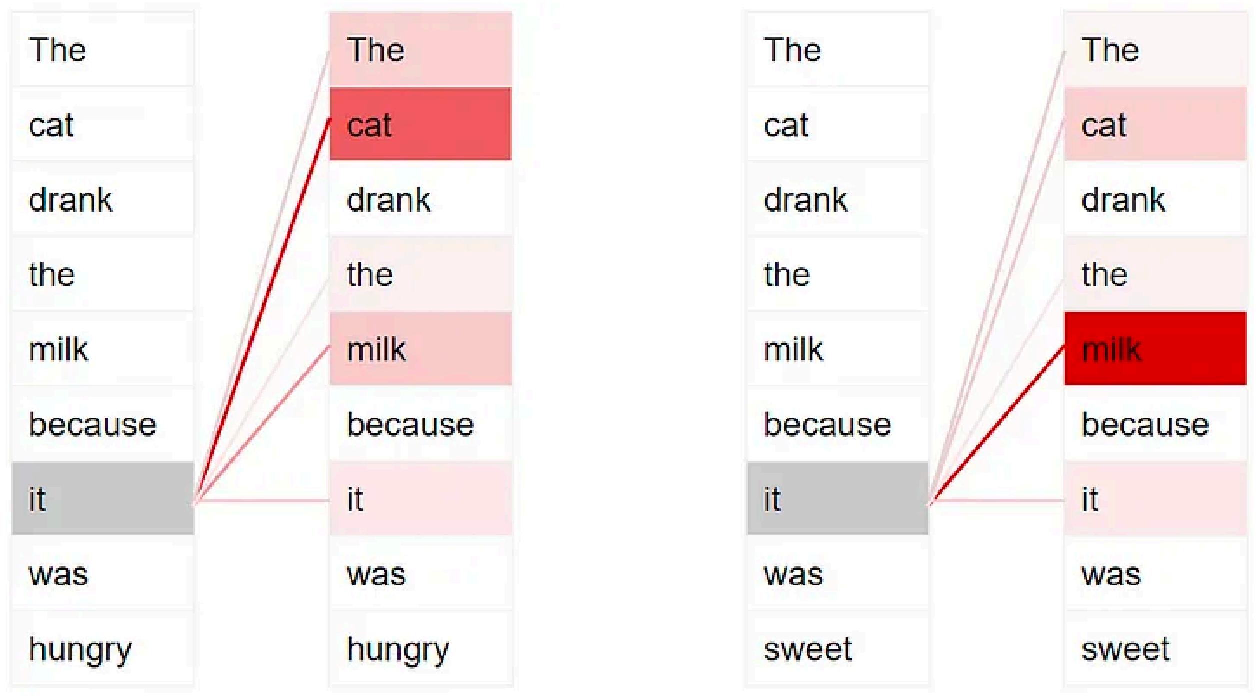
Take the word “bat” as an example.
In the sentence **“I played with the bat,”** “bat” refers to a sporting tool. However, in the sentence **“The bat flew in the night,”** “bat” indicates a flying mammal.

As you read, you n**aturally pay more attention to certain words based on the context provided by other words.**

The Transformer architecture uses self-attention by relating every word in the input sequence to every other word.
eg. Consider two sentences:

- The cat drank the milk because it was hungry.
- The cat drank the milk because it was sweet.

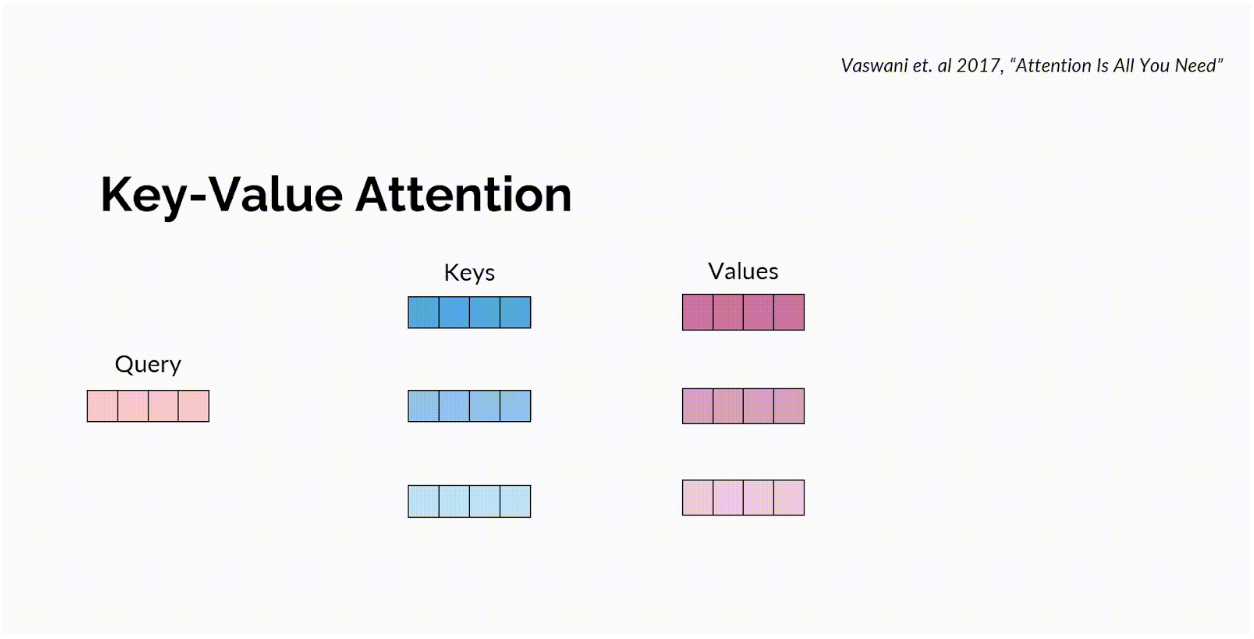
In the **first sentence,** the word **'it' refers to 'cat'**, while in the **second it refers to 'milk.**
When the model processes the word 'it', self-attention gives the model more information about its meaning so that it can associate 'it' with the correct word.



Basic Concept: Queries, Keys, and Values

To understand how attention works in Transformers, **imagine you're in a library looking for specific information in various books.** The process involves three main components: queries, keys, and values. Here's how each plays a role:

1. **Query (Q)**: This is like your **question** or the information you are seeking.
2. **Key (K)**: These are **potential matches** to your query, found in each book.
3. **Value (V)**: These are the **actual pieces of information** in the books that you get **if a key matches your query**.



Step-by-Step Intuitive Process

1. **Imagine a Sentence**: Let's take the sentence "The cat sat on the mat."
2. **Creating Queries, Keys, and Values**:
    - For each word in the sentence, the model creates a query, a key, and a value.
    - Think of the **query** as a **question each word has about the other words**.
    - The **key** is like a label or **identifier that helps match the query.**
    - The **value** is the actual content or **meaning of the word**.
3. **Matching Queries to Keys**:
    - **Each word's query is compared against every other word's key to see how well they match.**
    - For example, the word **"cat" has a query that might be looking for actions associated with it.**
    - **The keys from other words are checked to see if they match this query.**
4. **Scoring Matches**:
    - The model assigns a score to each match. This score reflects how relevant or related the words are.
    - For instance, the query from "cat" might find a high match with the key from "sat" because "cat" and "sat" are directly related in the context of this sentence.
5. **Creating a Weighted Summary**:
    - These scores are used to create a weighted summary of values.

    - Words with higher scores (stronger matches) contribute more to the final representation of each word.
    - **So, the value of "sat" will contribute significantly to the final understanding of "cat" because their match score is high.**

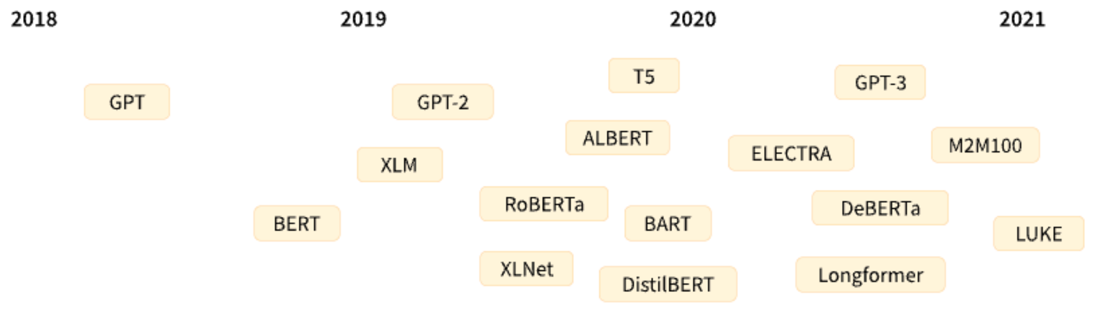
How Transformers Work

1. **Understanding Context**: Transformers read the entire text and understand the context by paying attention to the relationships between all words. This means that the meaning of a word can be influenced by words that are far apart in the text, which is something earlier models struggled with.
2. **Parallel Processing**: Unlike older models that processed text sequentially (word by word), Transformers can process all words in a sentence simultaneously. This makes them faster and more efficient.
3. **Building Blocks**: Transformers are built using layers that refine their understanding of the text at each stage. Each layer pays attention to different aspects of the text, gradually building a richer and more nuanced understanding.

We dont need to understand the details of the architecture of transformers but this is the basic architecture

Why Transformers Are Game-Changing

- **Contextual Understanding**: By focusing on the context, Transformers can understand the meaning of words based on their usage in the sentence. This is crucial for tasks like translation, where the same word can mean different things in different contexts.
- **Flexibility and Power**: Transformers are highly flexible and powerful. They can be fine-tuned for a variety of tasks, from answering questions to generating coherent and creative text, making them versatile tools in NLP.
- **Scalability**: Transformers can be scaled up to handle vast amounts of data, which is why models like GPT-3 can generate human-like text and perform complex tasks with impressive accuracy.



Conclusion:

- We learnt the basics on NLP: breaking down text (tokenization), understanding words (stemming and lemmatization), and converting text to numerical representations (embeddings) for machine comprehension.
- Transformers: Transformers revolutionized NLP by using attention mechanisms to process entire sequences of text simultaneously, capturing context and relationships between words more effectively than traditional sequential models.
    - introduced concepts like queries, keys, and values to enhance understanding of text. Their encoder-decoder architecture and ability to scale with large datasets enable models like BERT and GPT-3 to perform complex language tasks.
- Impact and Future: The advancements in NLP and LLMs, particularly through the development of Transformers, have significantly improved the accuracy and versatility of AI in understanding and generating human language, paving the way for future innovations.
- In the next lecture we will start on the practical implementation of really fun things using state of the art models to do all NLP tasks in a couple of lines of codes