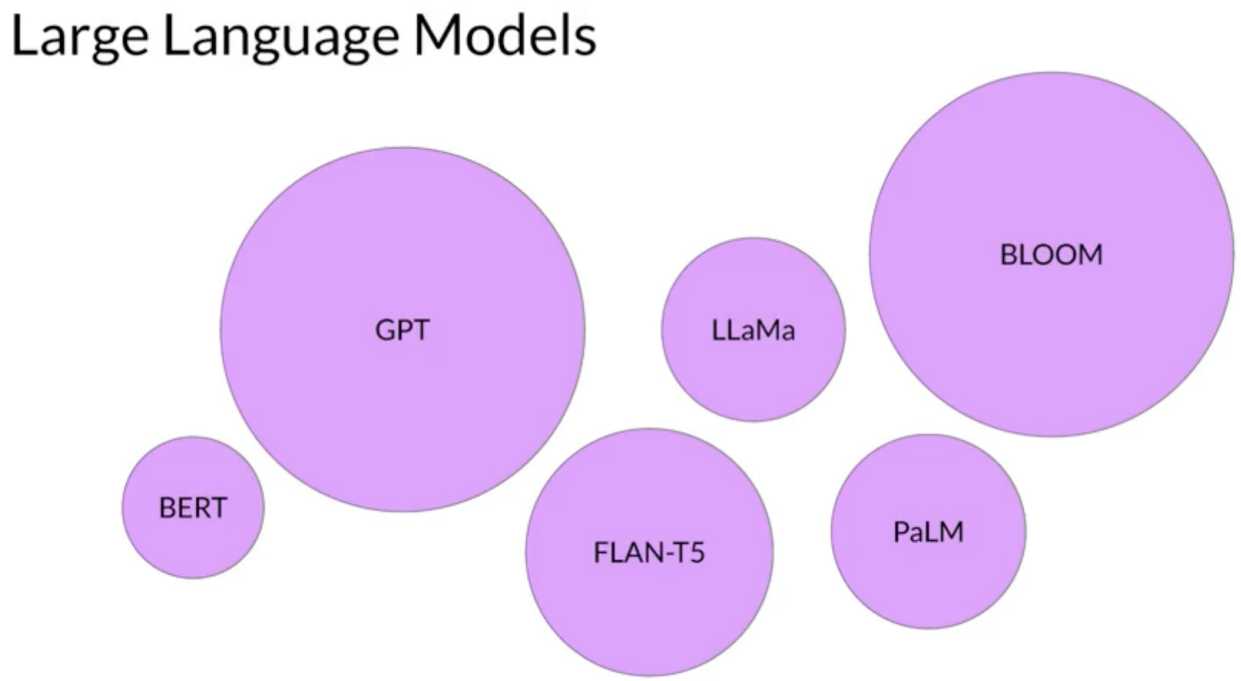
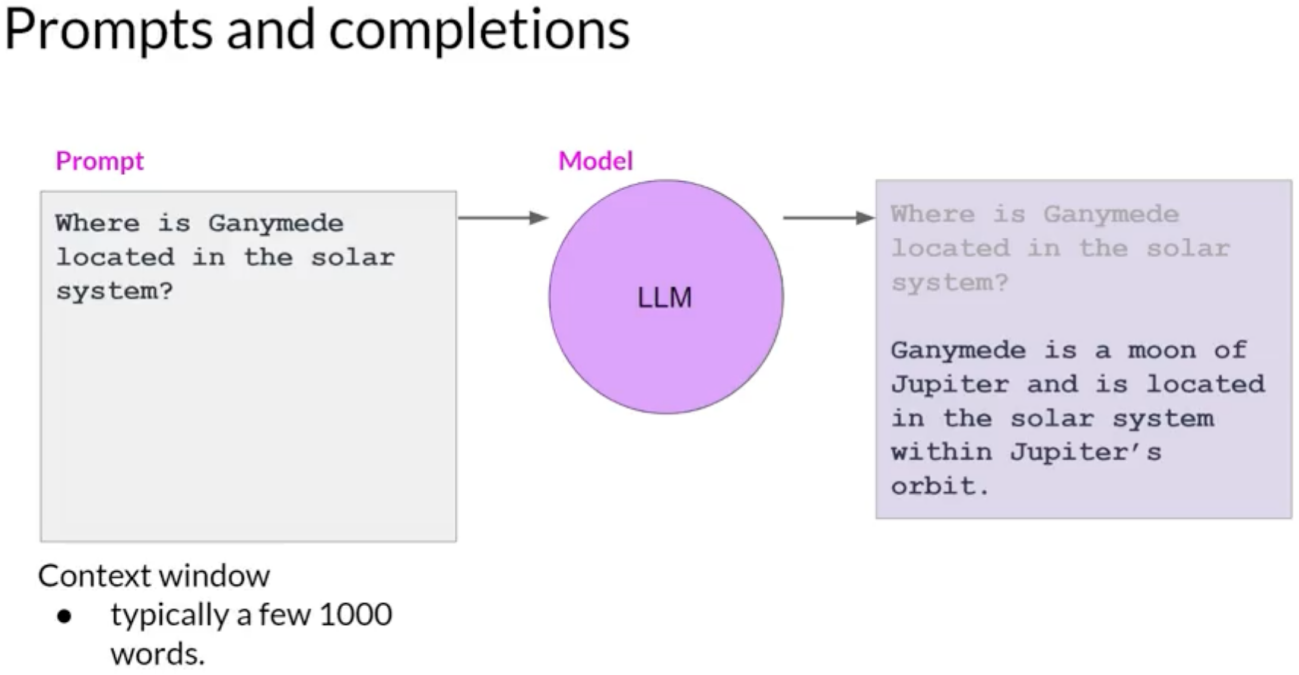
Generative AI & LLMs

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

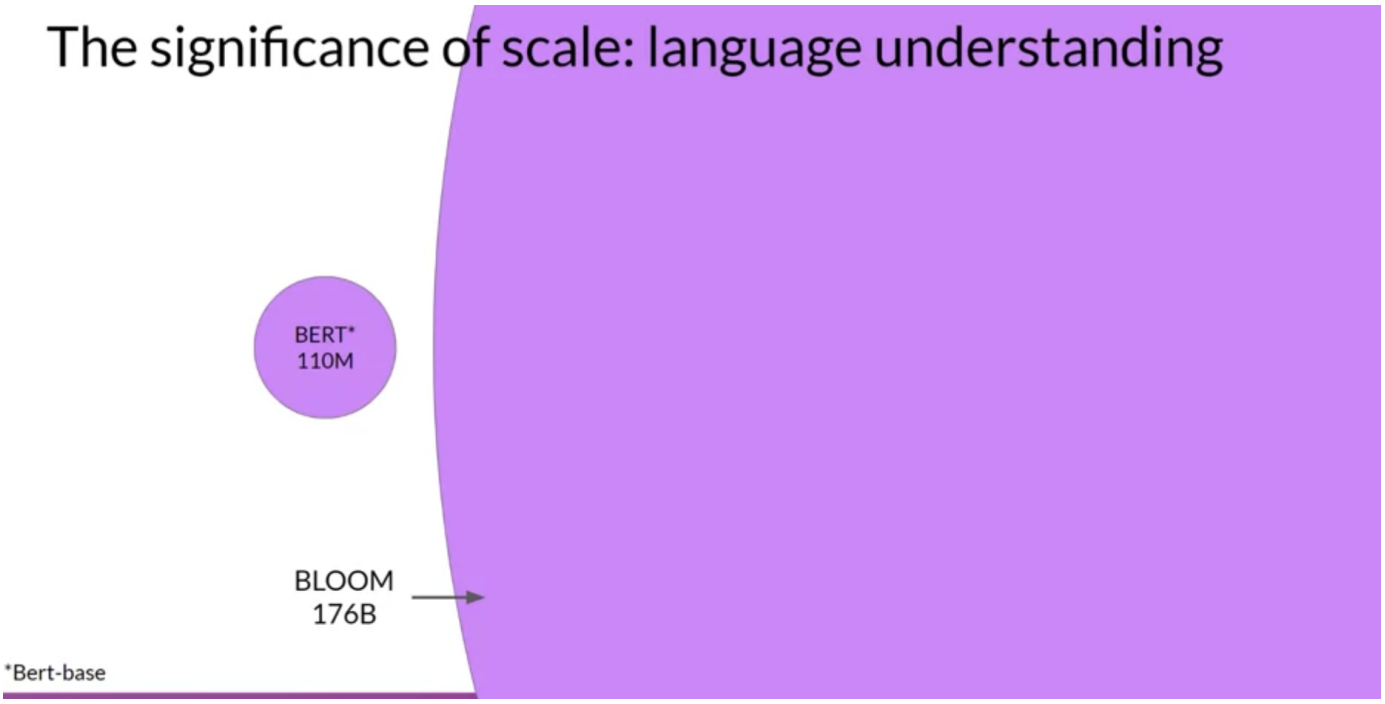
Fundation (Base) Models



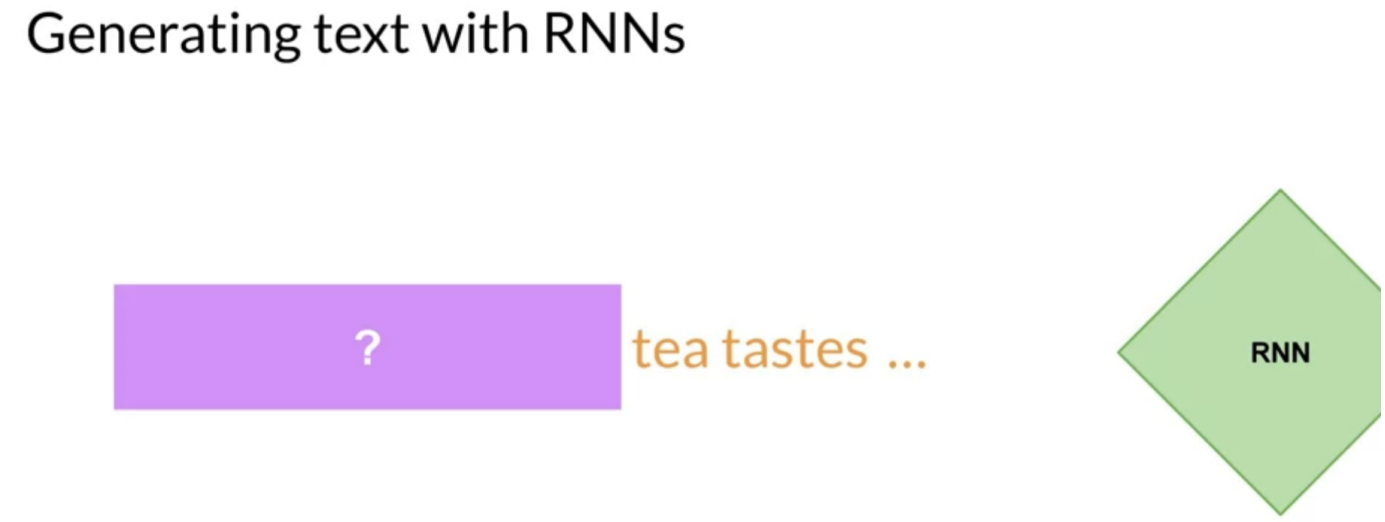
more parameters = more memory



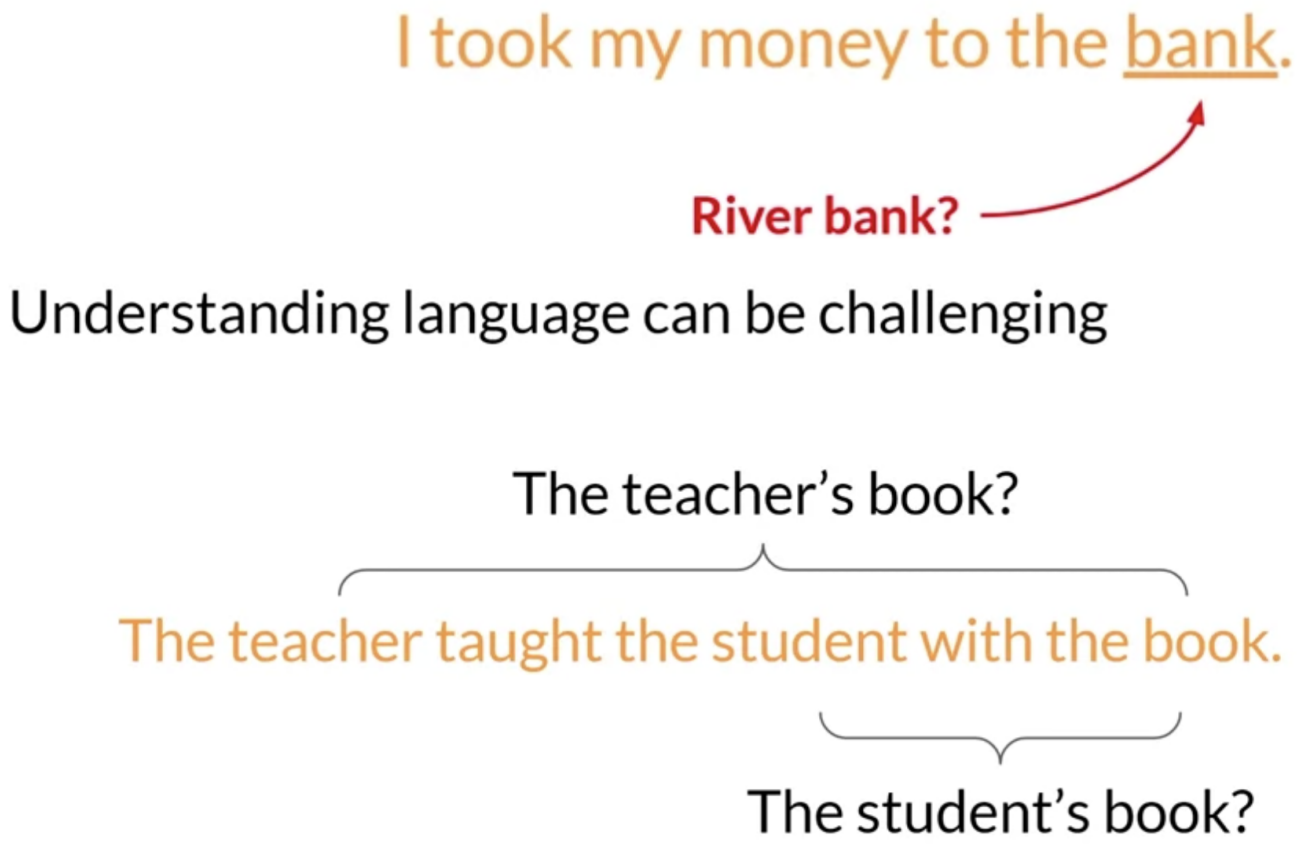
The output of the model is called a **completion**. The act of using the model to generate text is known as inference.



Generative algorithms are not new, Recurrent Neural Networks.



Understanding language can be challenging



Transformer Architecture

Transformers

Attention Is All You Need

Ashish Vaswani[*] Google Brain avaswani@google.com

Noam Shazeer[*] Google Brain noam@google.com

Niki Parmar[*] Google Research nikip@google.com

Jakob Uszkoreit[*] Google Research usz@google.com

Llion Jones[*] Google Research llion@google.com

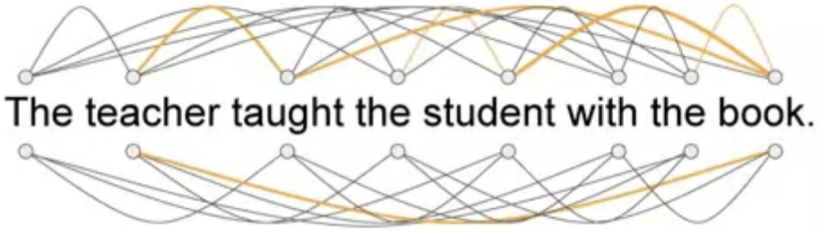
Aidan N. Gomez[*] [†] University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com

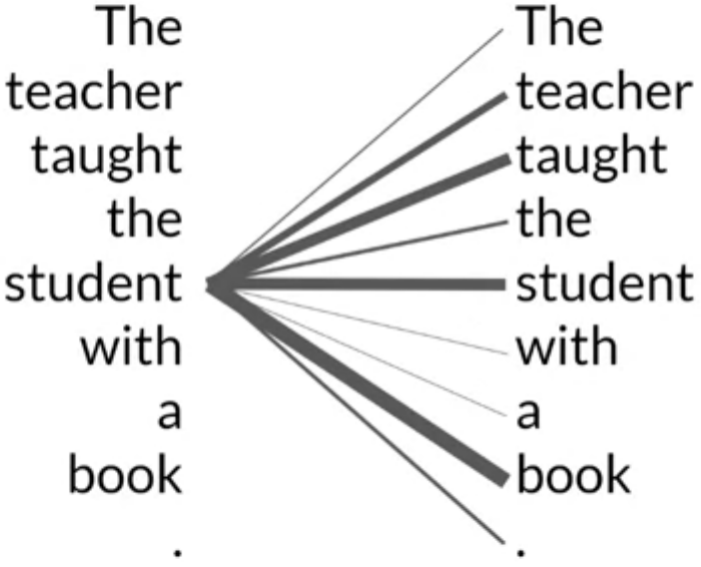
Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com

- Scale efficiently
- Parallel process
- Attention to input meaning

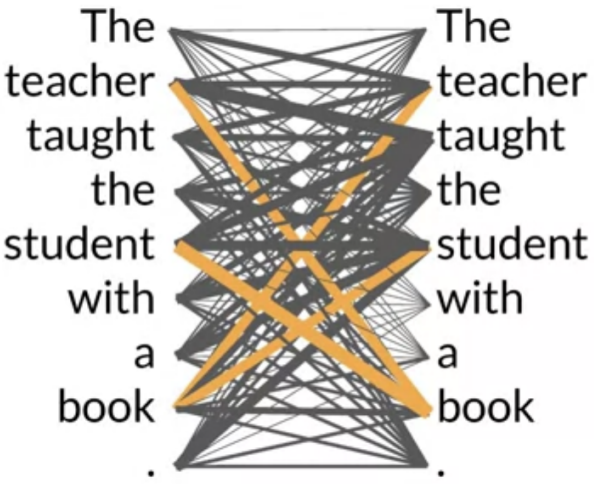
Transformers



Self-attention

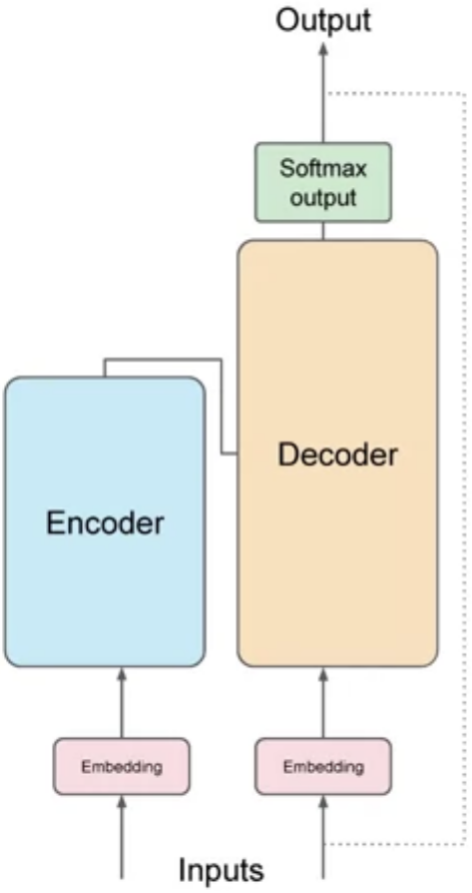


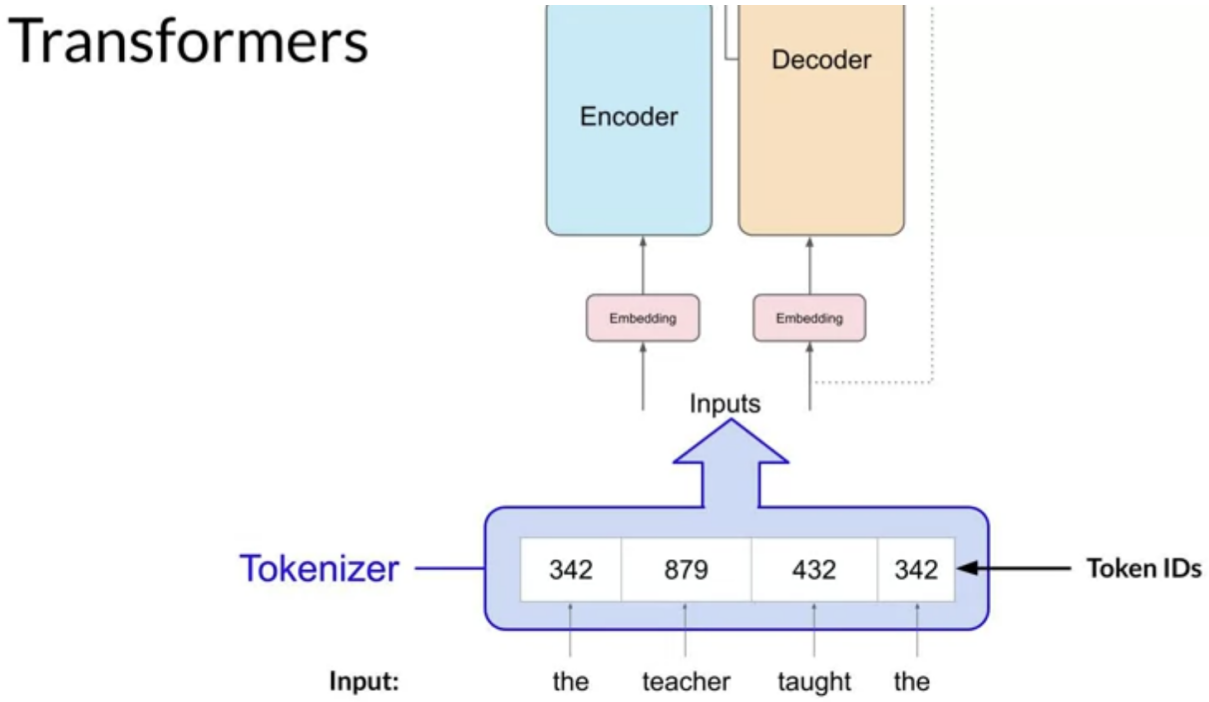
Self-attention



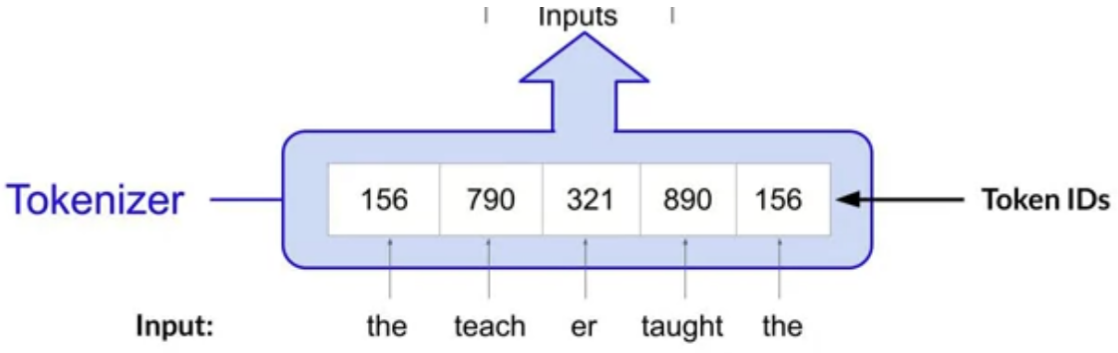
How the model works?

Transformers

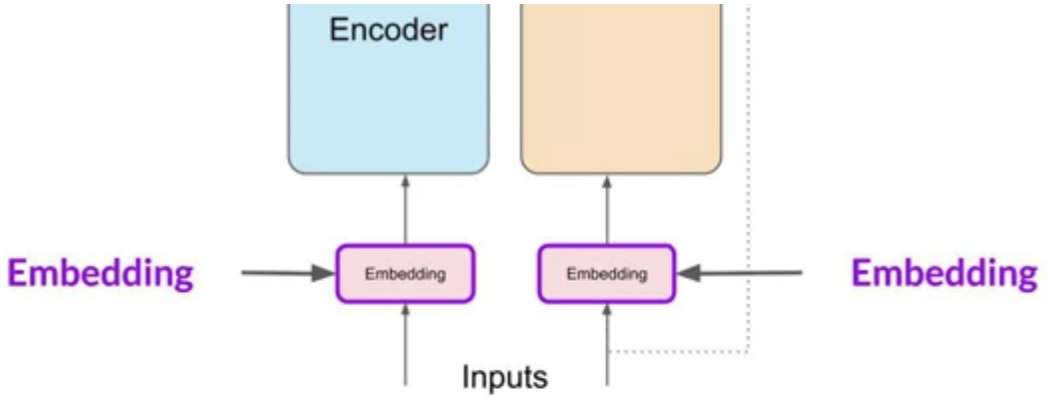




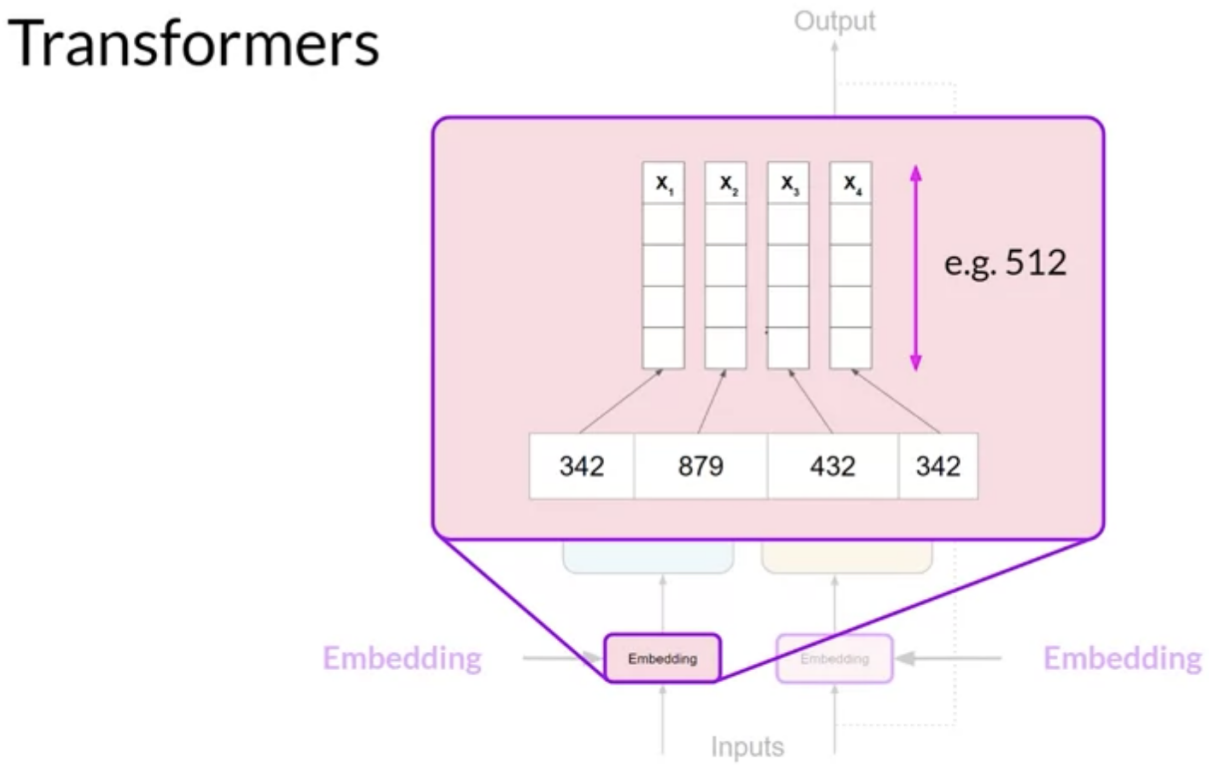
Now, machine-learning models are just big statistical calculators and they work with numbers, not words. So before passing texts into the model to process, you must first tokenize the words. Simply put, this converts the words into numbers, with each number representing a position in a dictionary of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



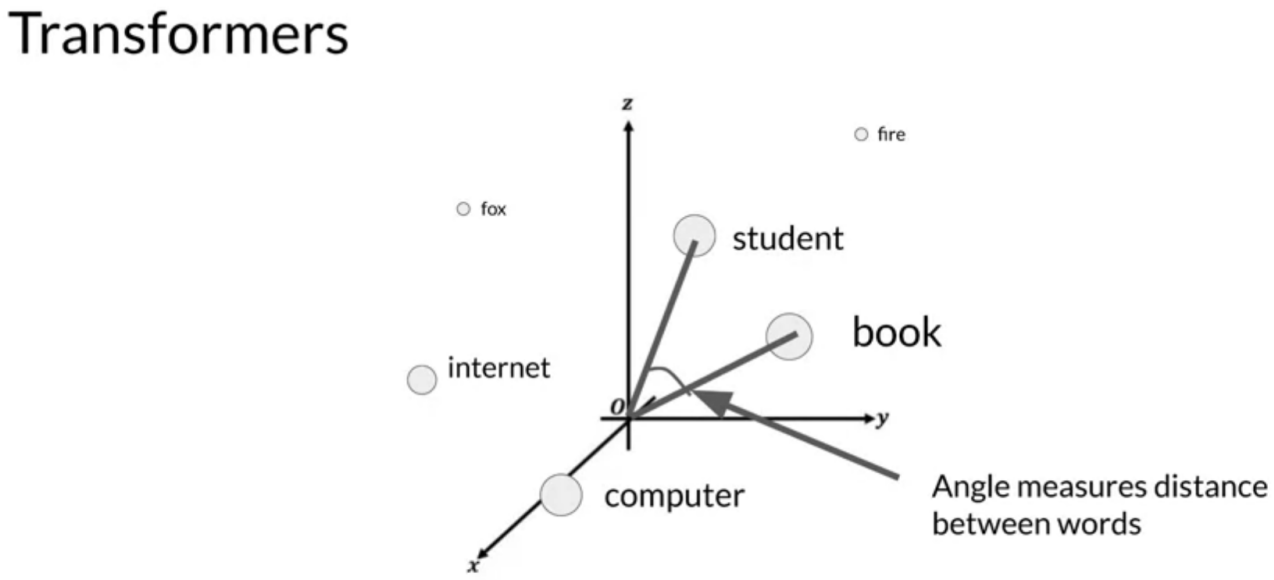
Now that your input is represented as numbers, you can pass it to the embedding layer. This layer is a trainable vector embedding space, a high-dimensional space where each token is represented as a vector and occupies a unique location within that space. Each token ID in the vocabulary is matched to 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.



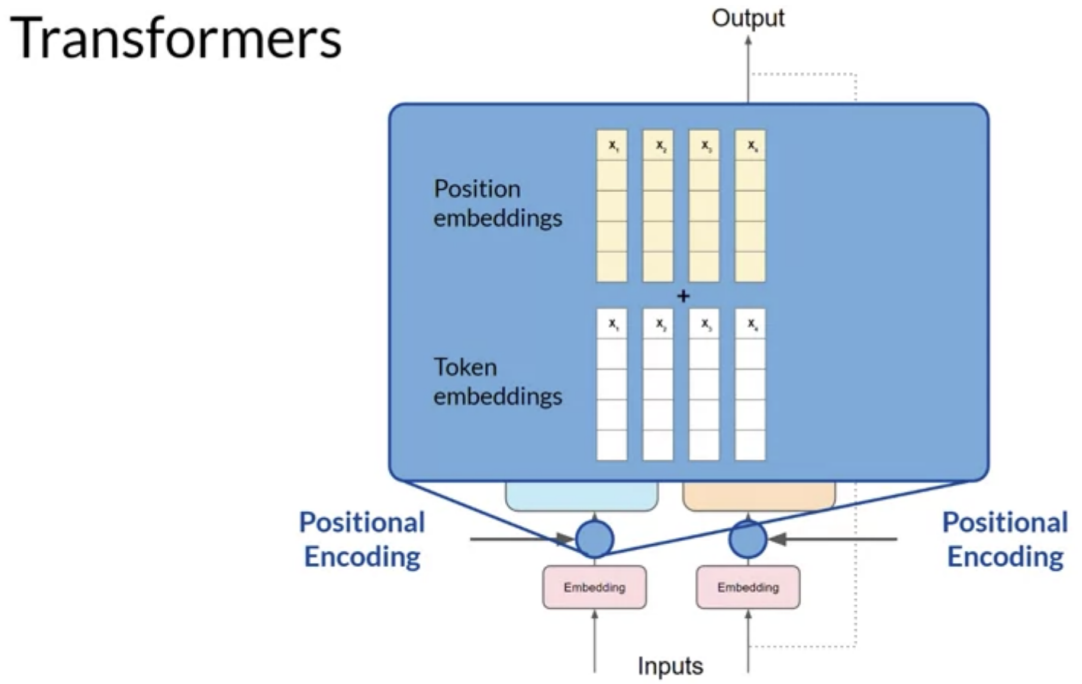
From the Transformer paper



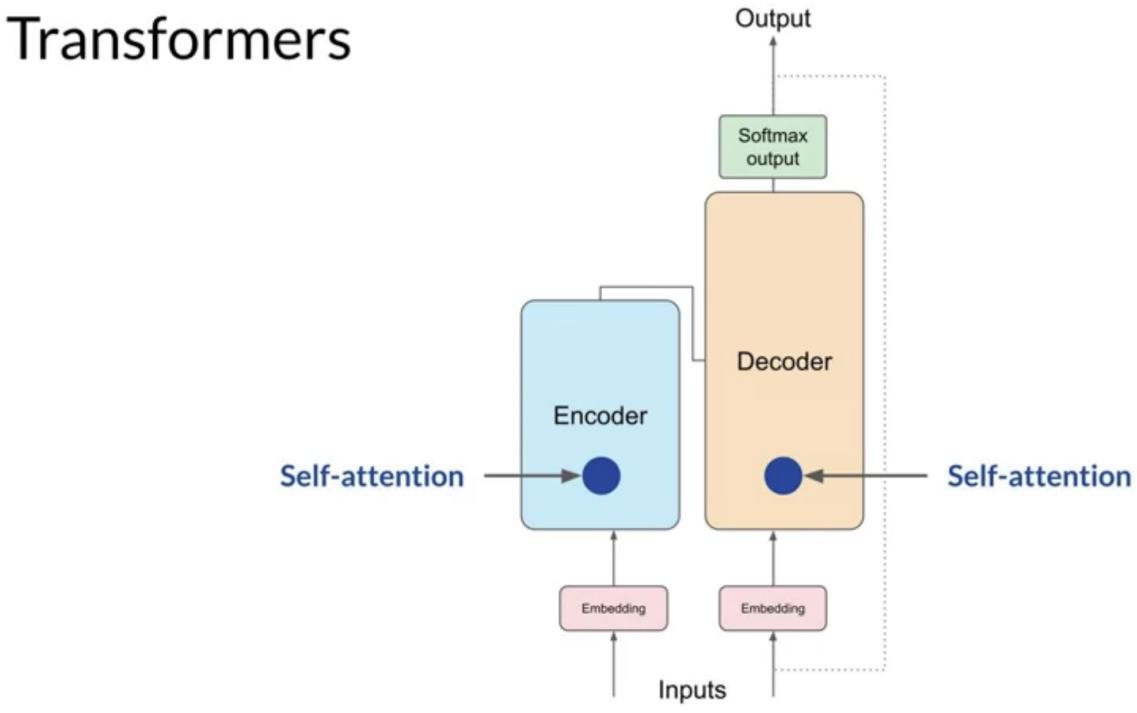
In case of 3 dimension per embedding:



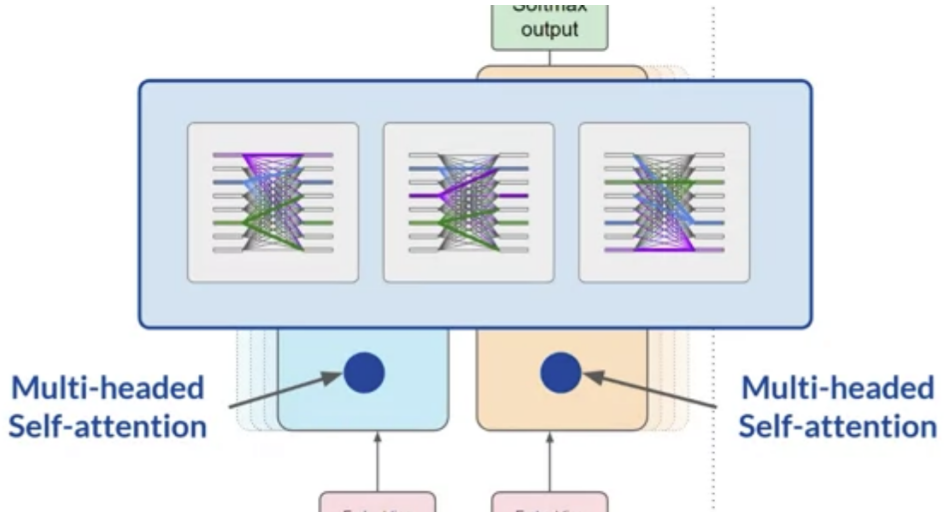
Positional Encoding to preserve the word order (because everything append in parallel). So by adding the positional encoding to the token embeddings you preserve the information and don't lose the relevance of the position of the words in the sentence.



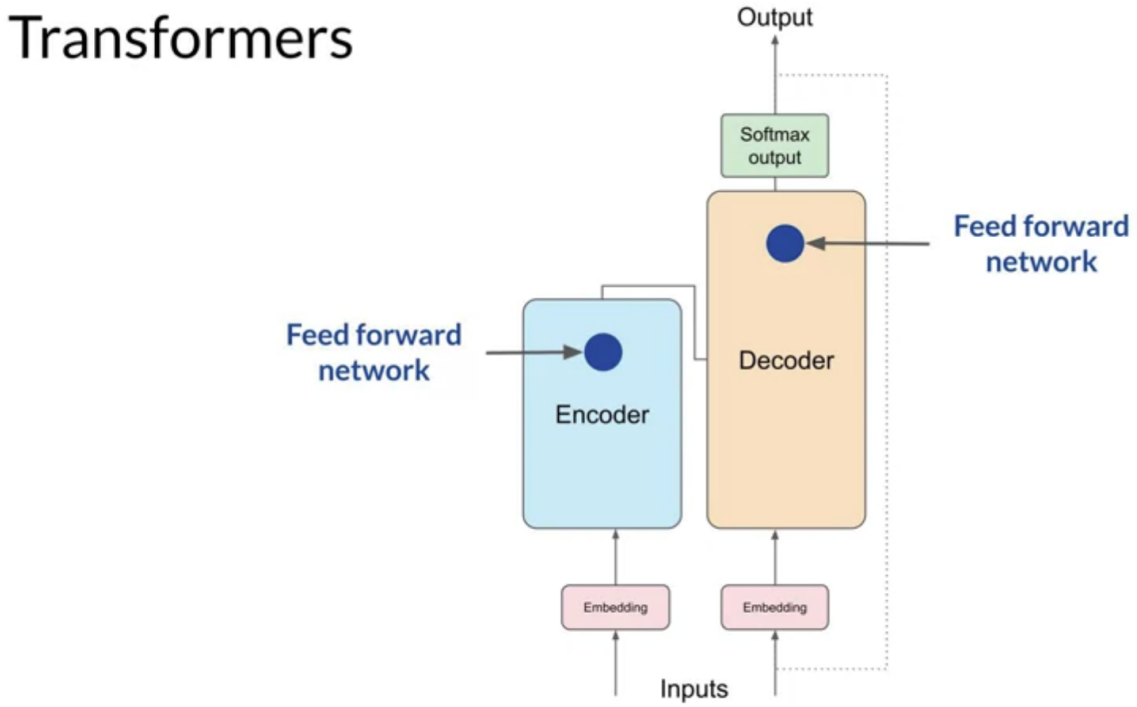
The model analyzes the relationships between the tokens in your input sequence. As you saw earlier, this allows the model to attend to different parts of the input sequence to better capture the contextual dependencies between the words. The self-attention weights that are learned during training and 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.



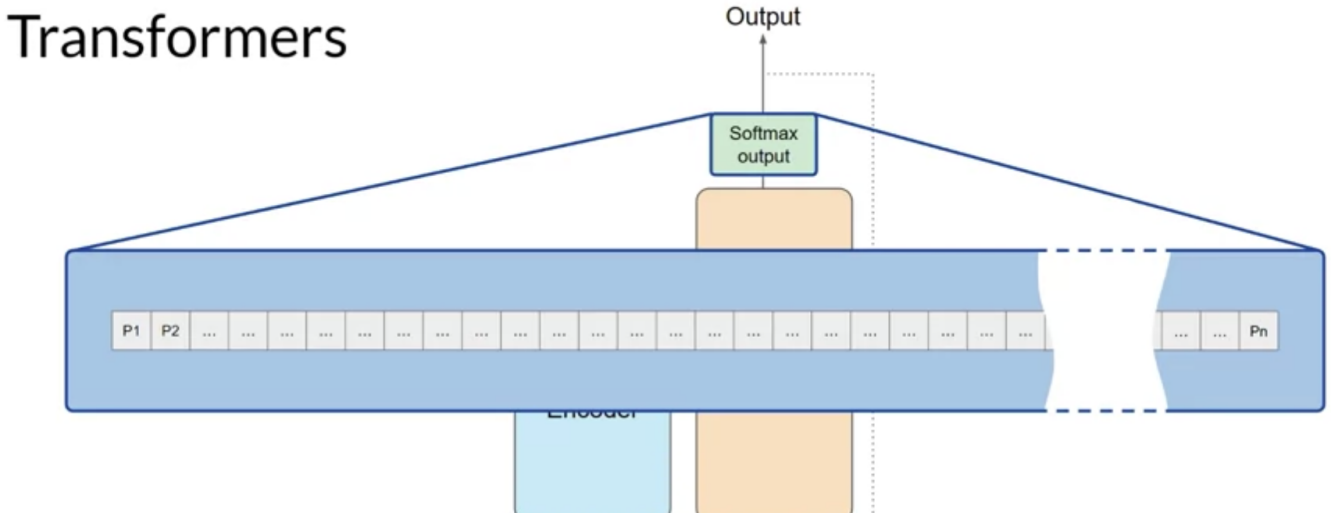
The number of attention heads included in the attention layer varies from model to model, but numbers in the range of 12-100 are common. The intuition here is that each self-attention head will learn a different aspect of language. For example, one head may see the relationship between the people entities 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-forwad 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.

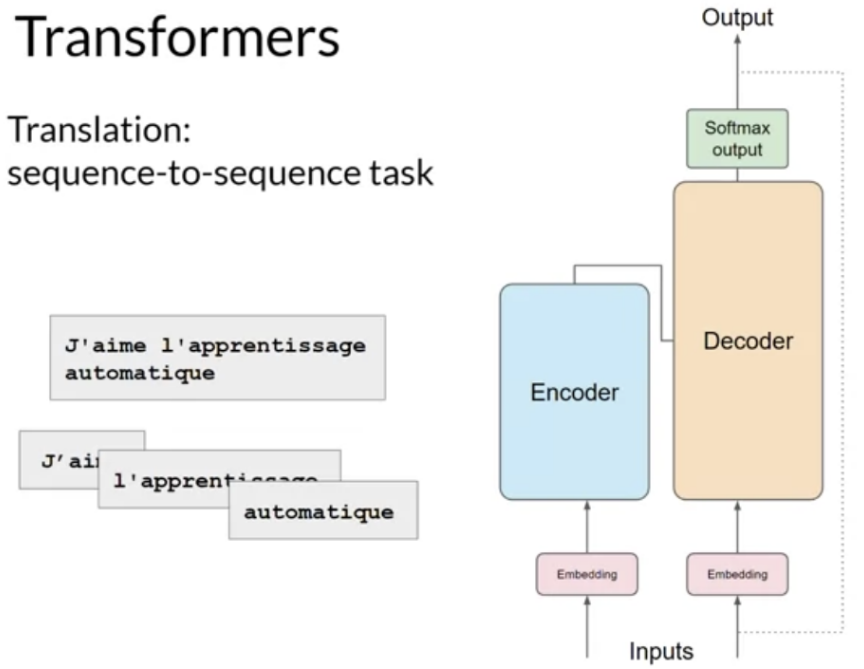


Those logits are pass to a softmax layer, where they are normalized into a probability score for each word.

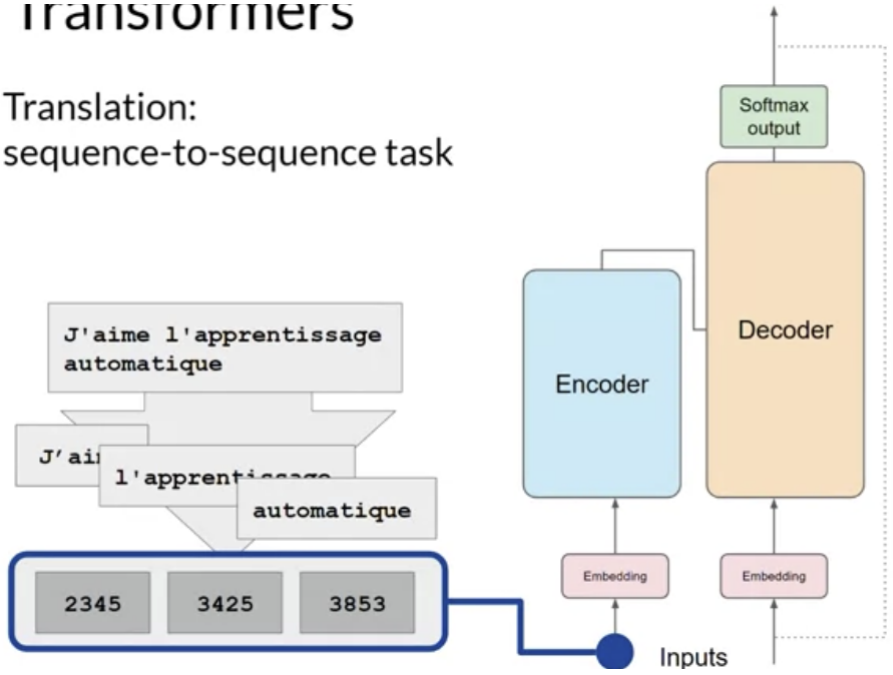


The output is a probability for every single word in the vocabolary. So there is likely to be thousands of scores. One single token will have the score higher than the rest. **This is the most likely predicted token.**

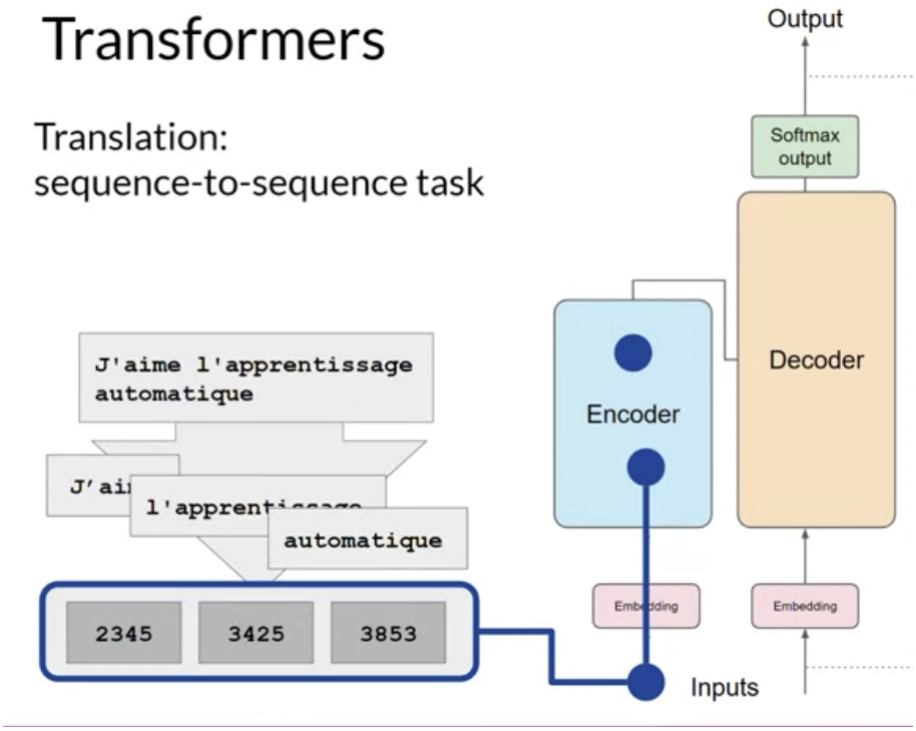
Example End-2-End Translation



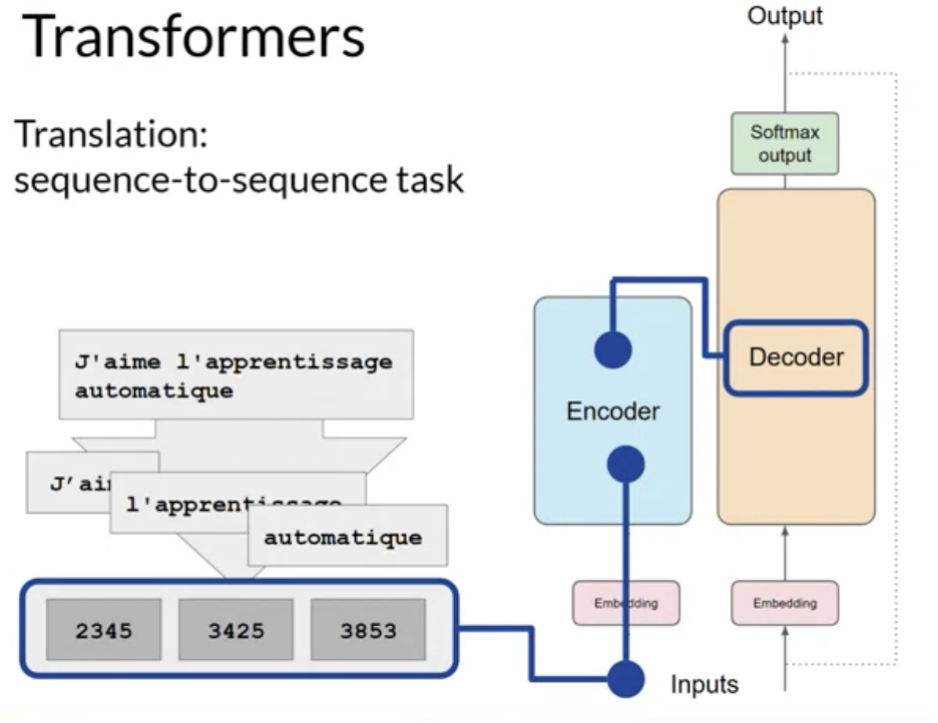
1. Tokenizer



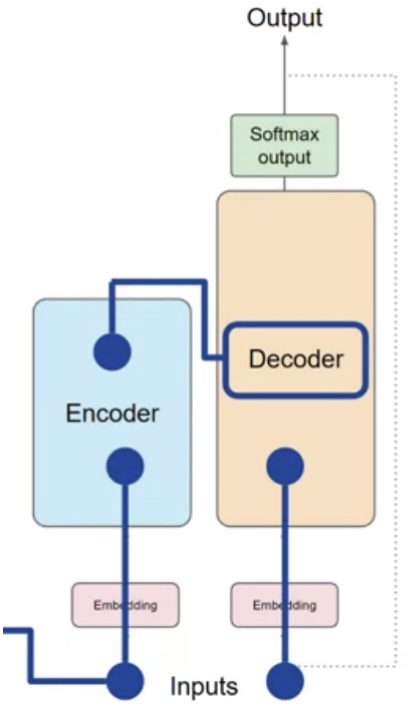
2. Embedding Layer + Encoder Layer (Multi-head attention layers)



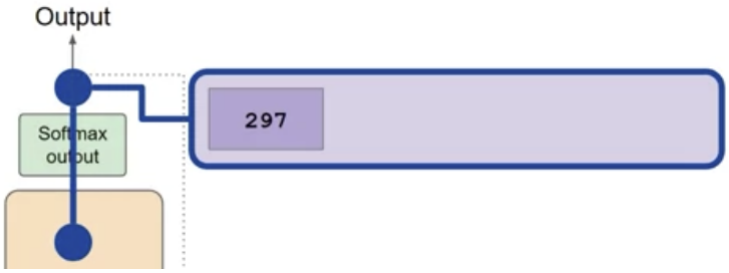
3. The output of the multi-head attention is injected into the feed-forward NN to the output of the encoder.
4. At this point, the data that leaves the encoder is a deep representation of the structure and the meaning of the input sequence. This is inserted into the middle of the Decoder (i.e., self-attention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



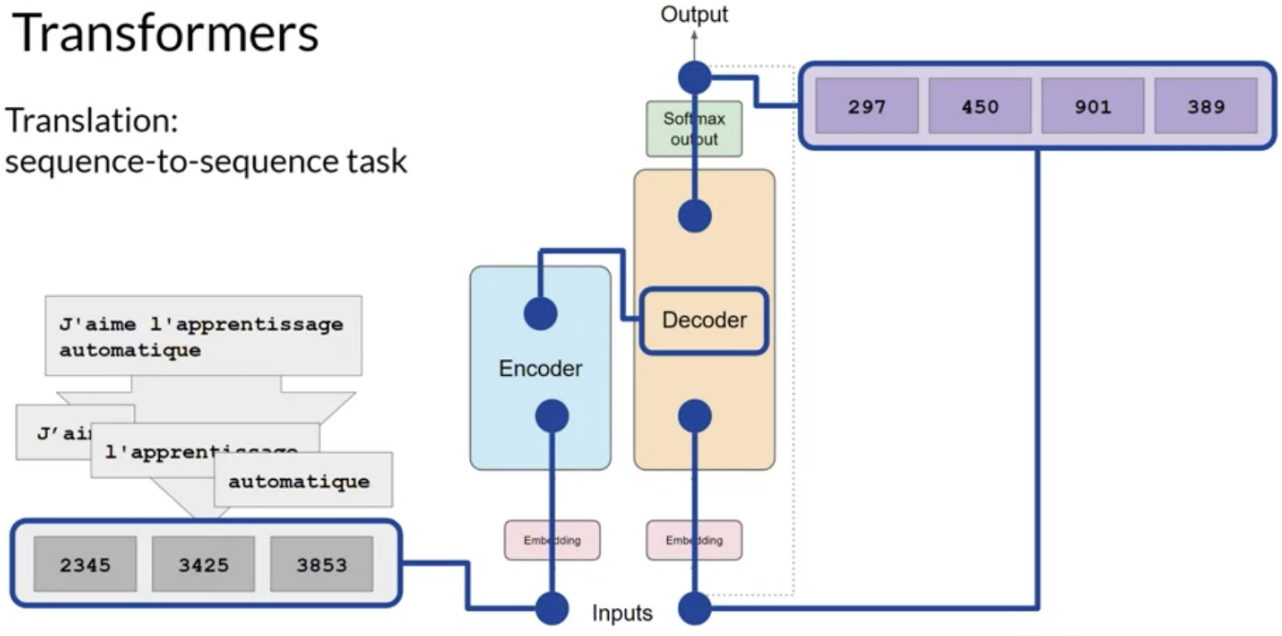
5. Decode, a start of sequence token is added to the input of the decoder trigging the decoder to predict the next token. It does this besed on the contextual understanding (output of the encoder to the decoder).



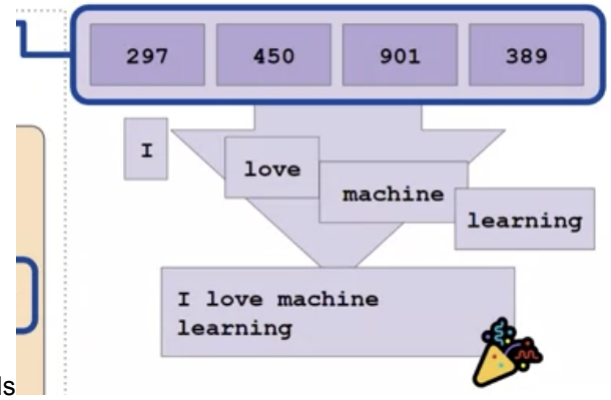
6. The output of the decoder passes throw the decoder feed-forward NN and a final softmax output layer.



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.

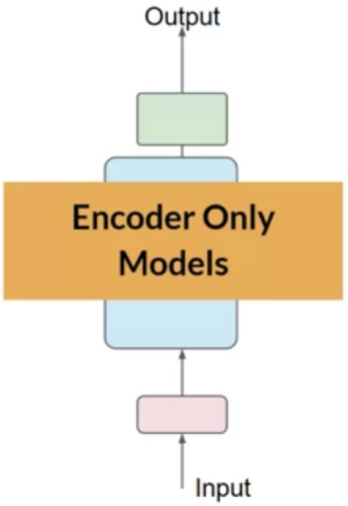


8. The final sequence of token can be de-tokenized into words



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.

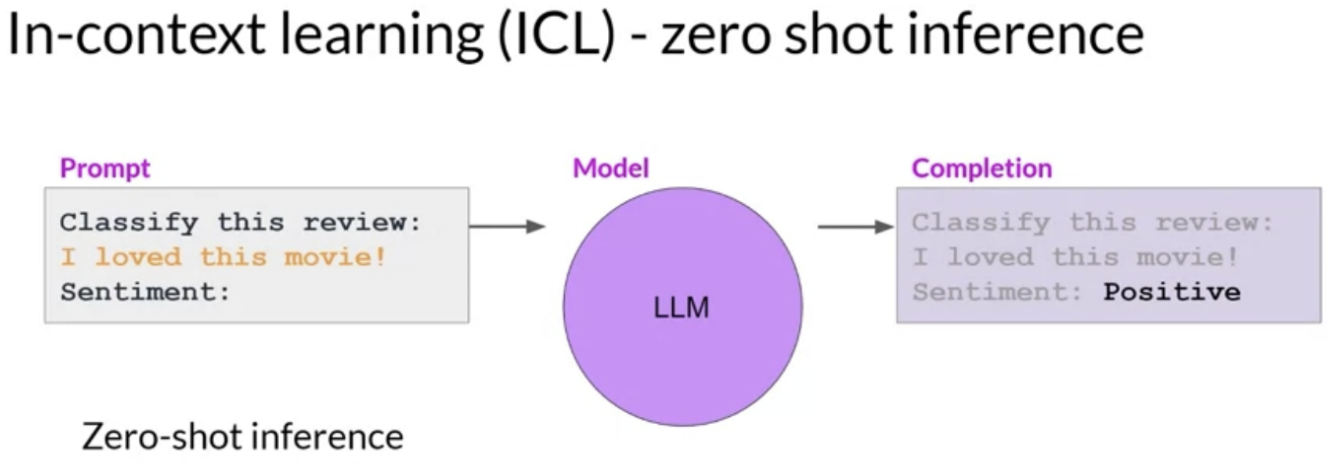
In addition, the authors introduce a positional encoding scheme that encodes the position of each token in the input sequence, enabling the model to capture the order of the sequence without the need for recurrent or convolutional operations.

Prompting and prompt engineering

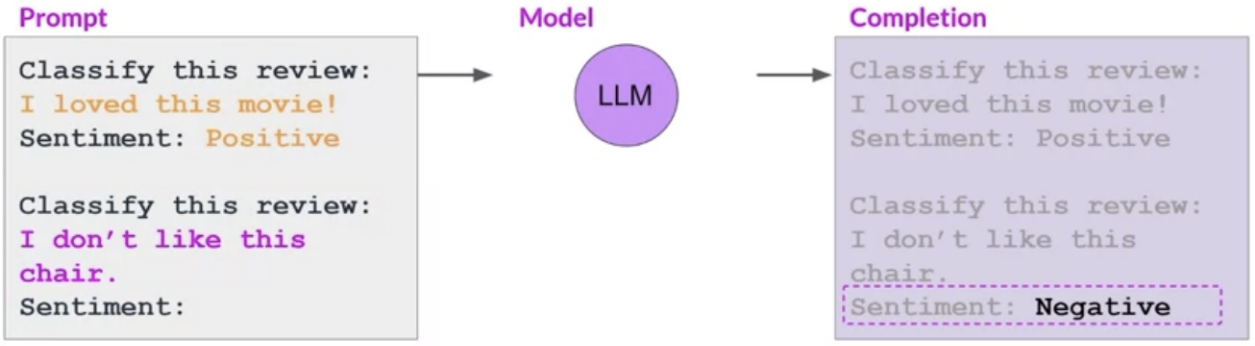
The input is called prompt, the act of generating text is know as inference, the output of the model is called completion. The context window is where you type the prompt and it appears as output at the begging.

in-context learning (ICL)

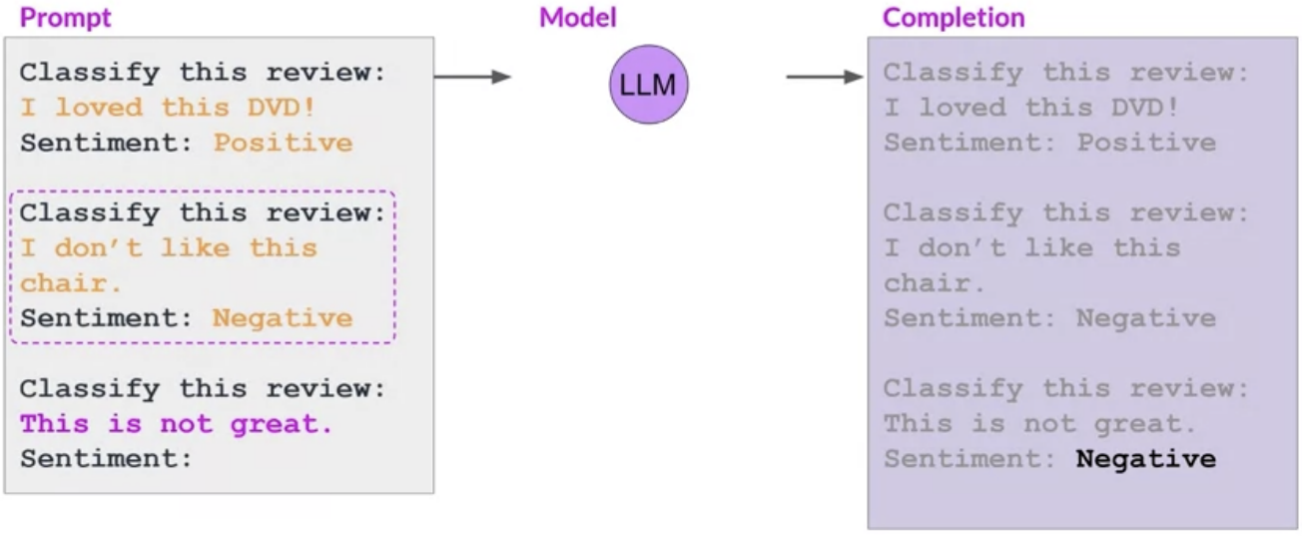
When the user provides the example inside the context window.



In-context learning (ICL) - one shot inference



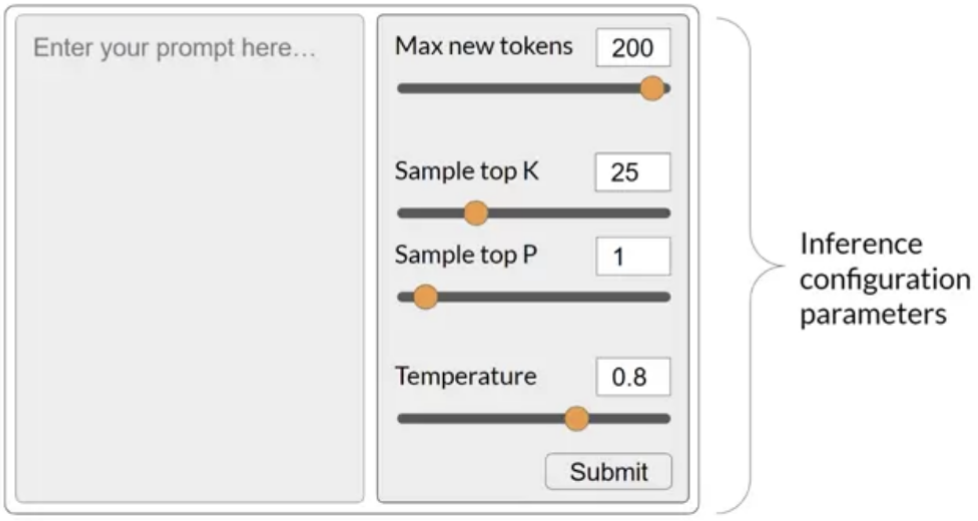
In-context learning (ICL) - few shot inference



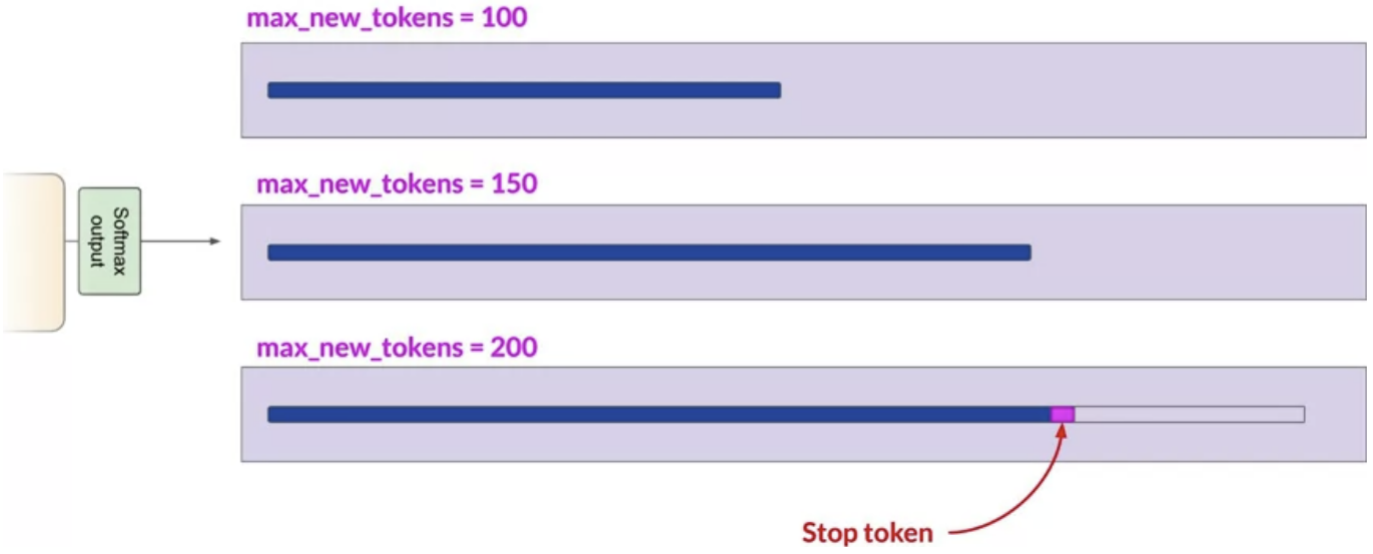
Summary of in-context learning (ICL)

Parameters

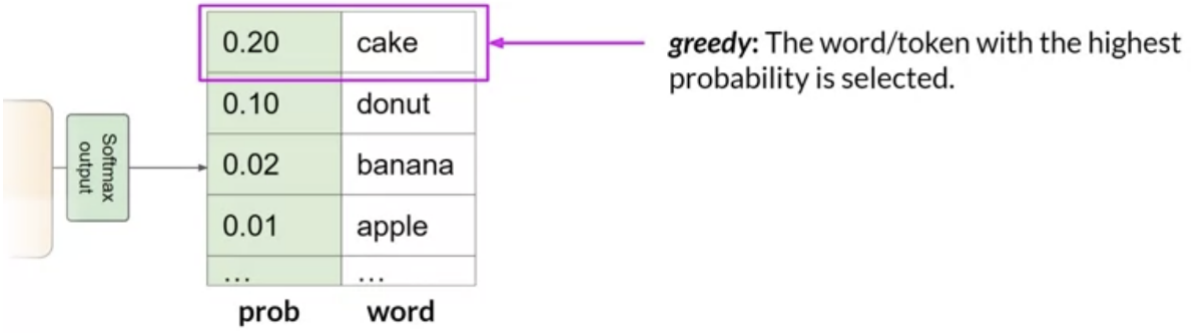
Generative configuration - inference parameters



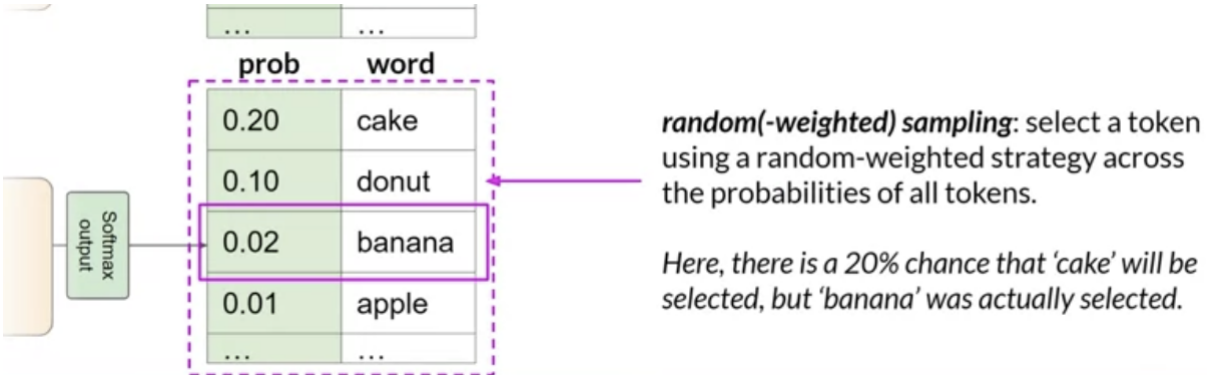
Generative config - max new tokens



Generative config - greedy vs. random sampling

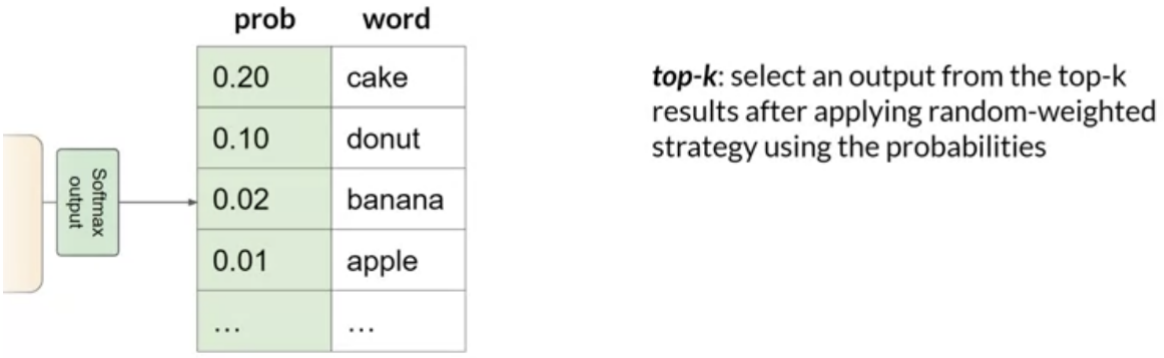


be careful with repetitions. Instead random sampling is more creative:



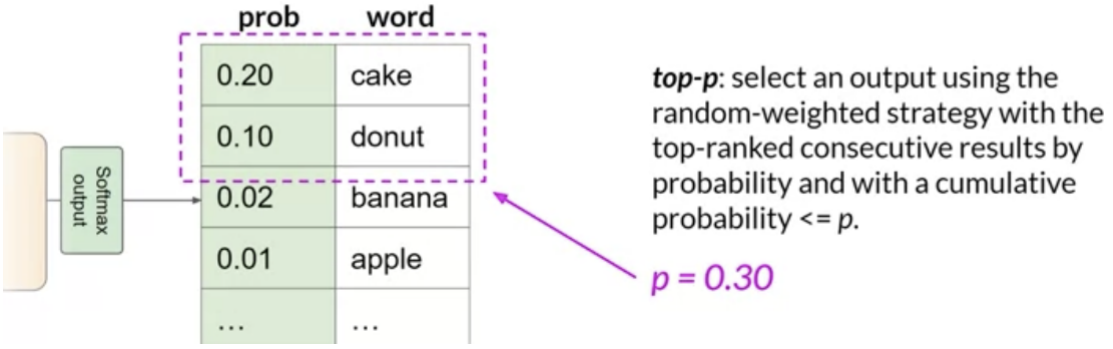
Top-K based on the probability, then random.

Generative config - top-k sampling



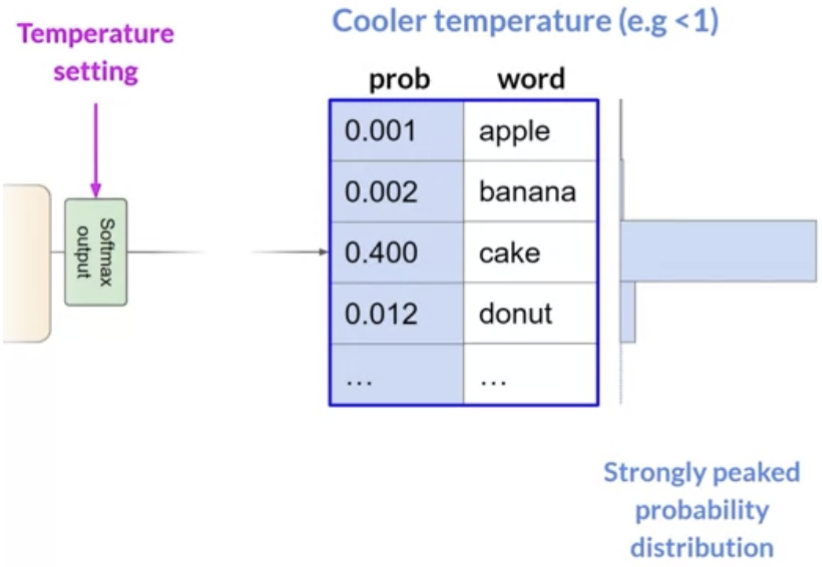
Top-p based on the probability threshold, then random.

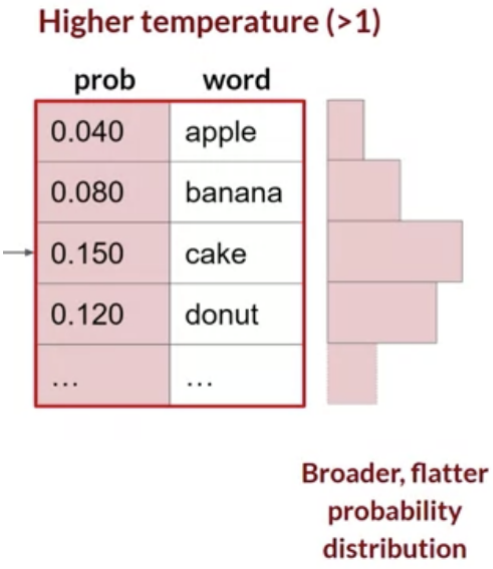
Generative config - top-p sampling



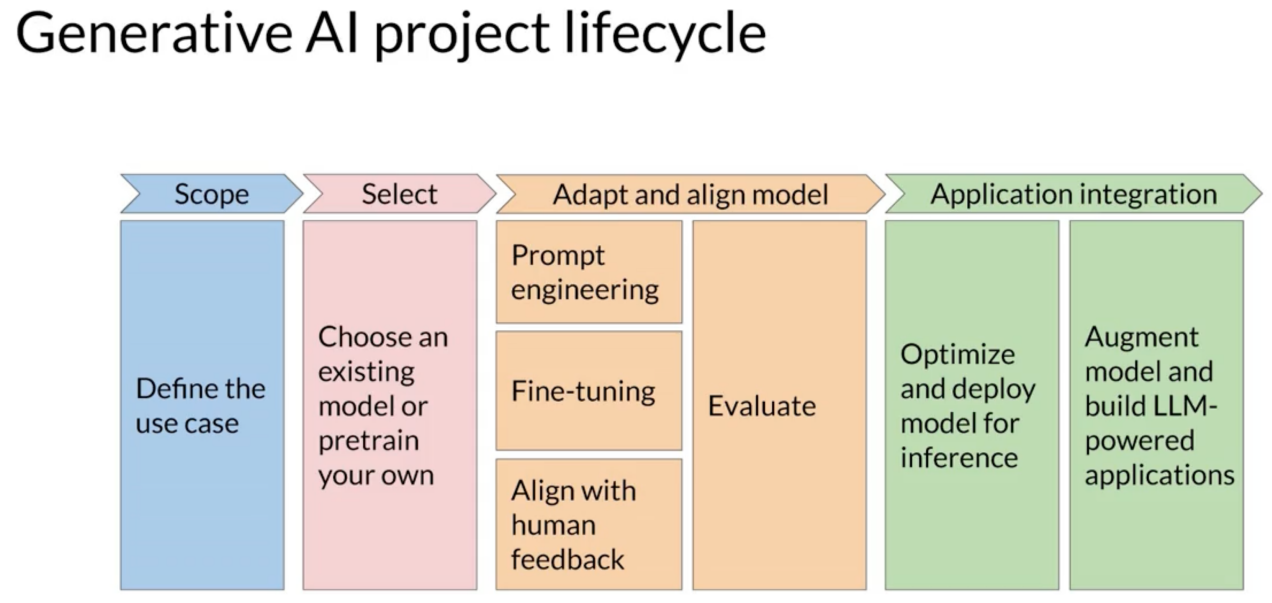
Temperature controls randomness

Generative config - temperature

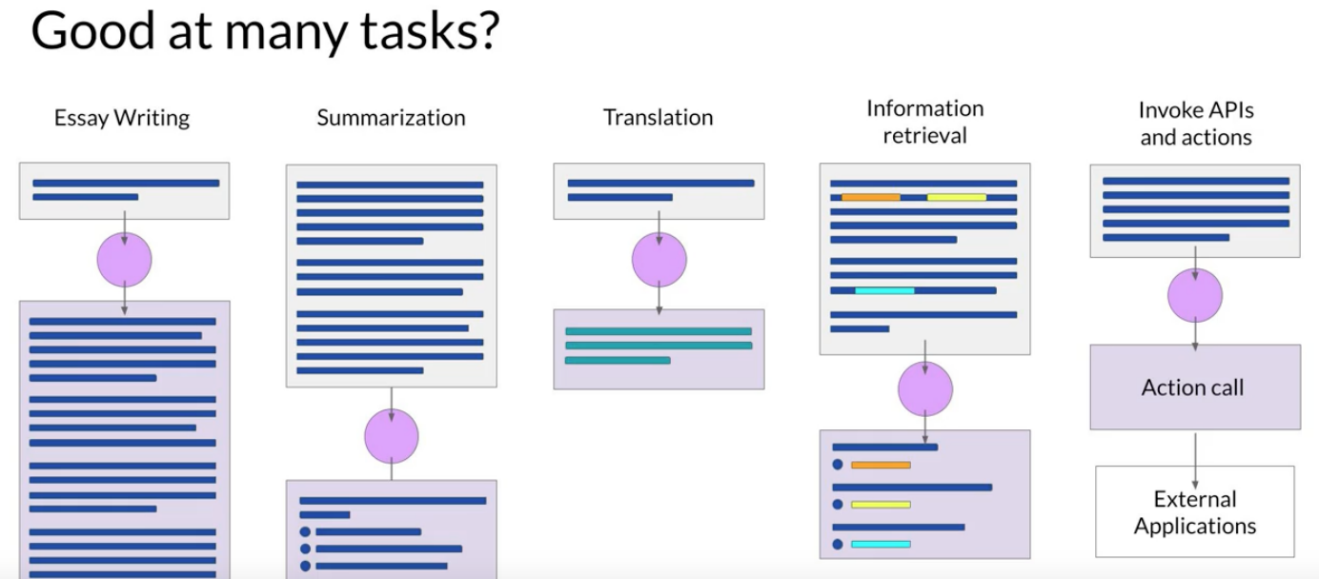




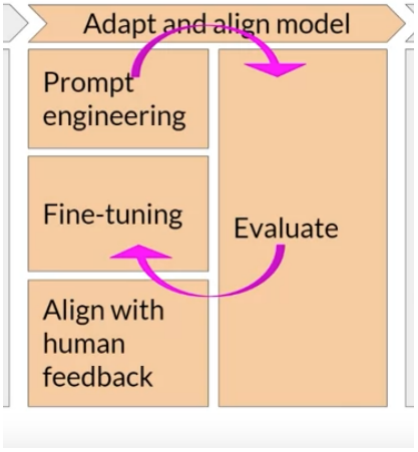
Generative AI Project Lifecycle



- Scope: as narrow as possible. For instance:

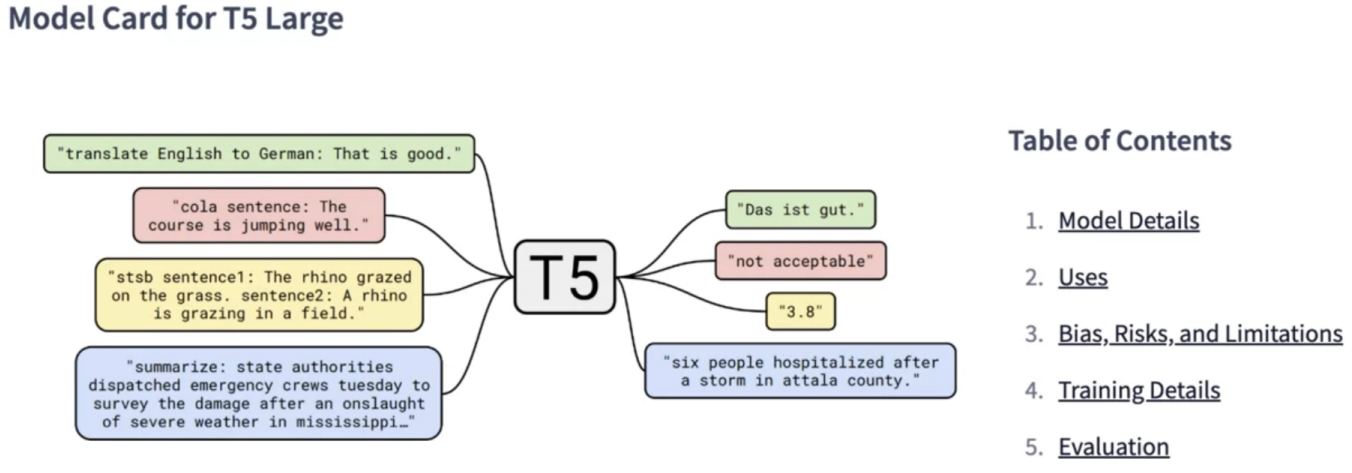


- Select: work with existing model or train from scratch
- Adapt and align mode: assess the performance of the chosen model on the assigned task(s)



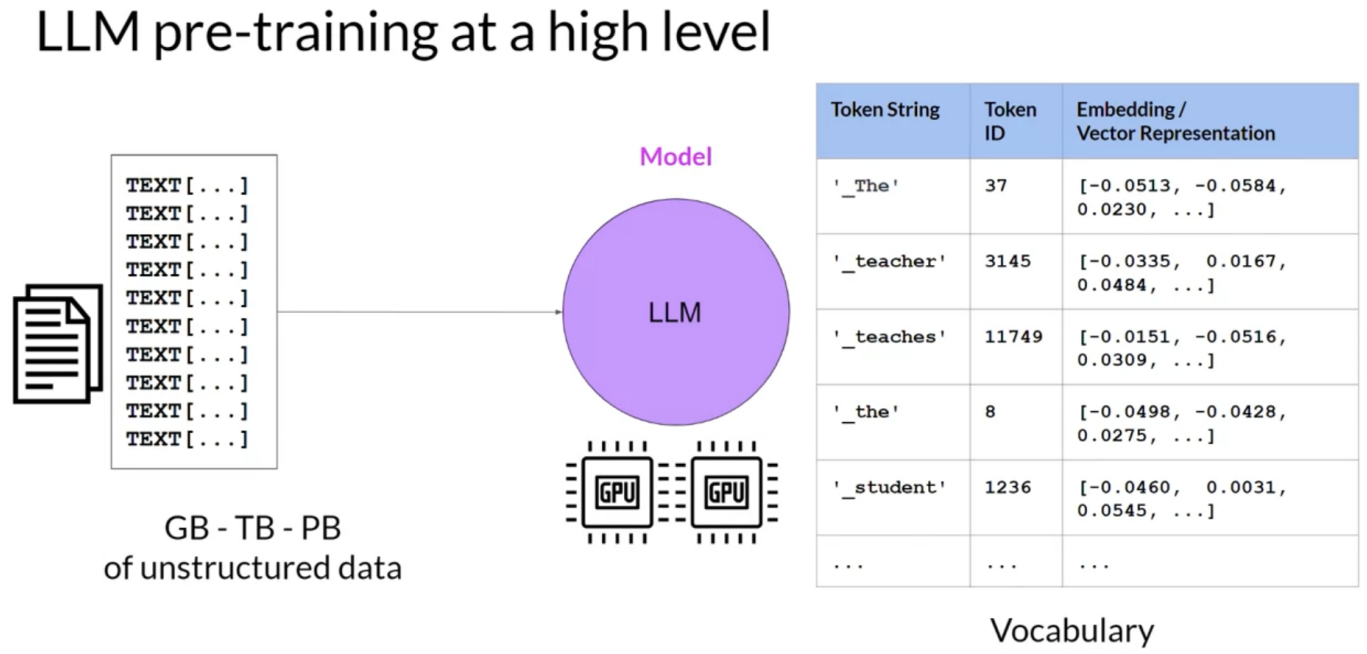
- Application Integration: Optimize and deploy, additional infrastructure.

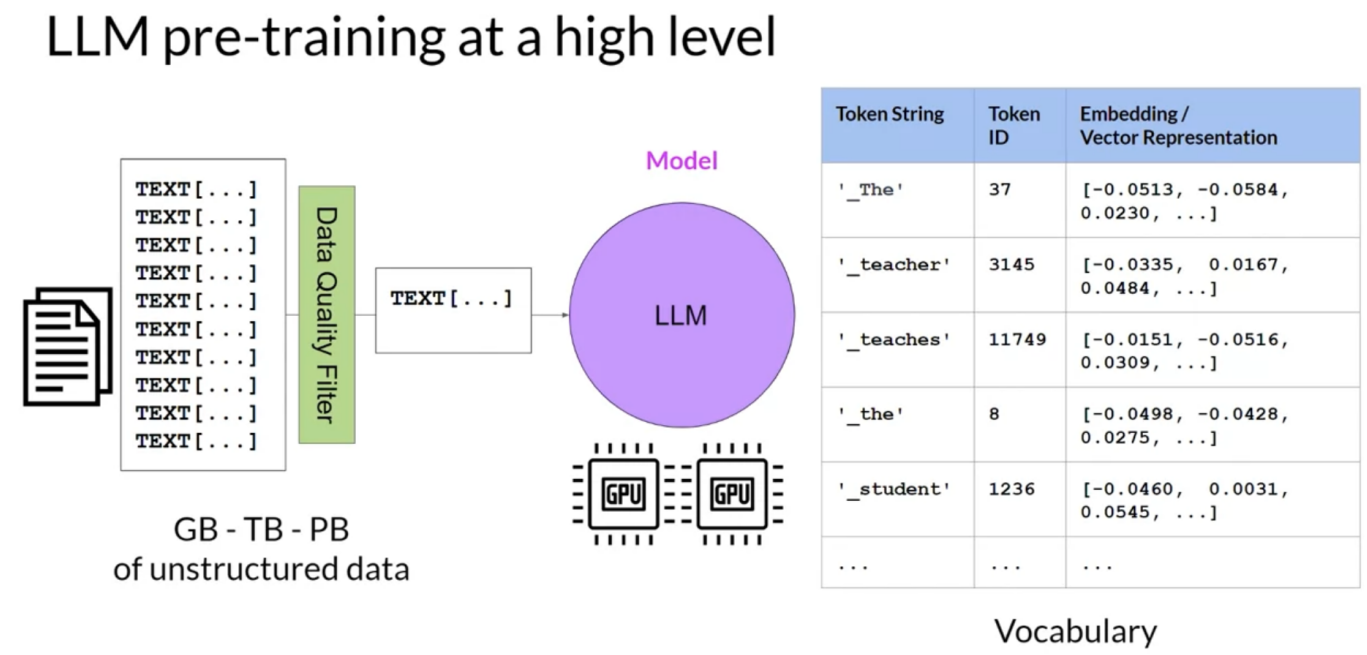
Model Hubs

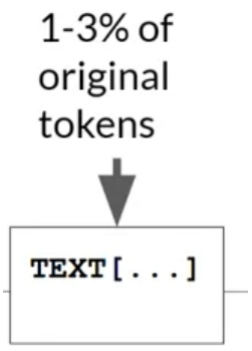


Pre-training

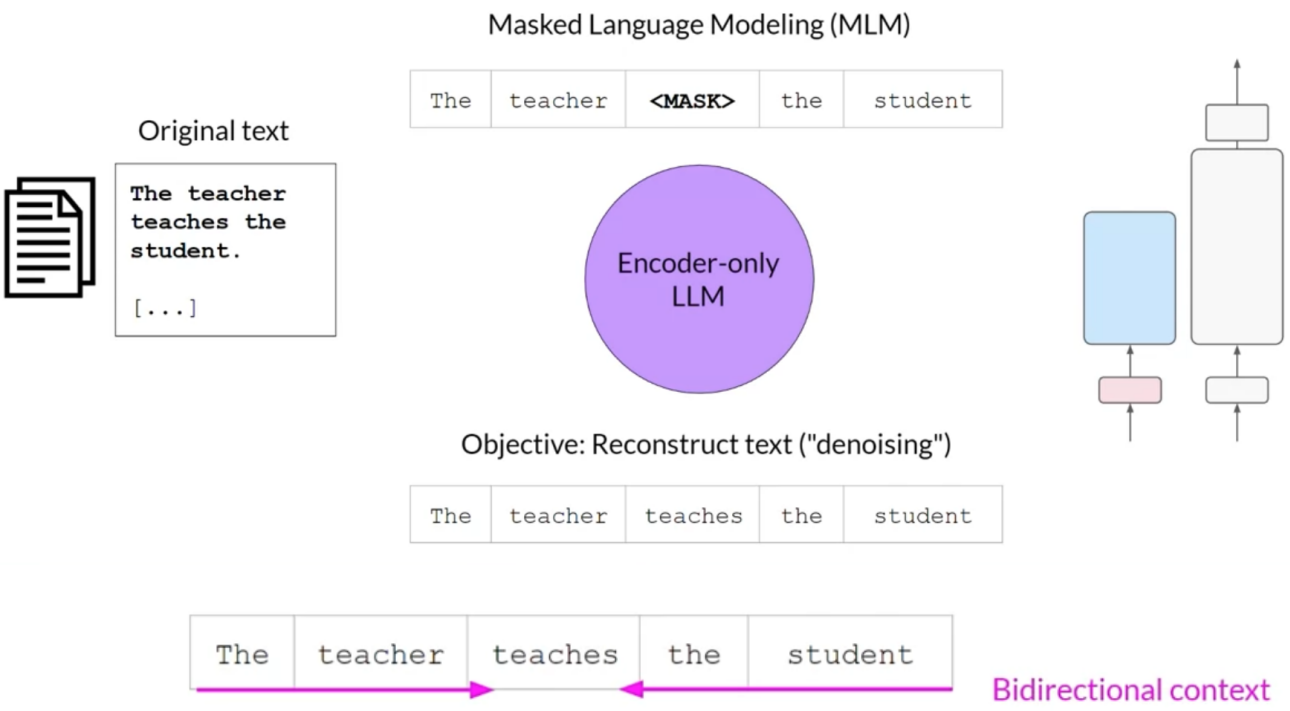
LLMs capture a deep statistical representation of language.







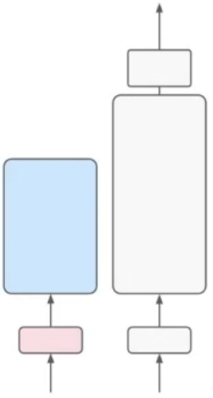
Autoencoding models



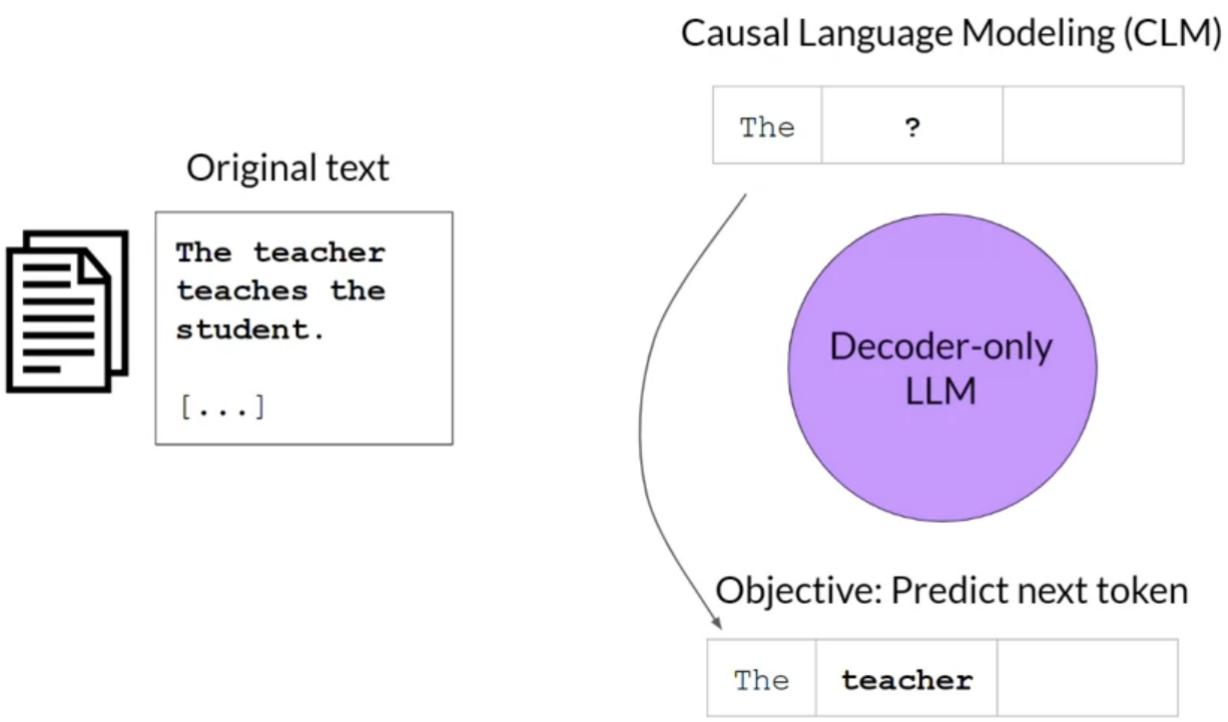
Autoencoding models

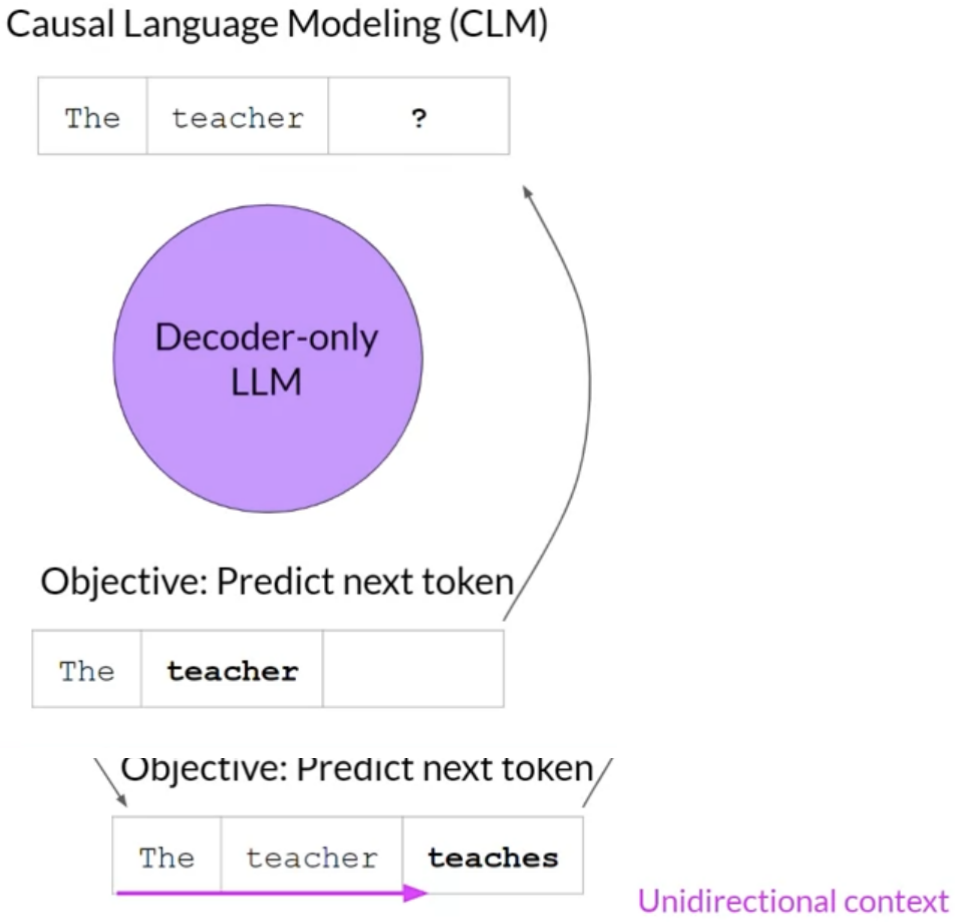
Good use cases:

- Sentiment analysis
- Named entity recognition
- Word classification



Autoregressive models





Autoregressive models

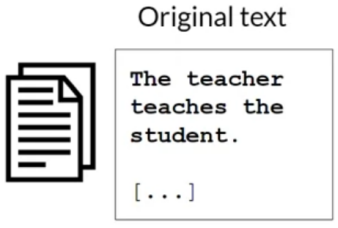
Good use cases:

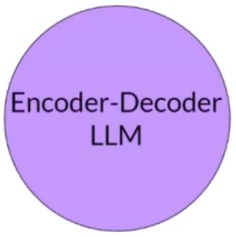
- Text generation
- Other emergent behavior
  - Depends on model size

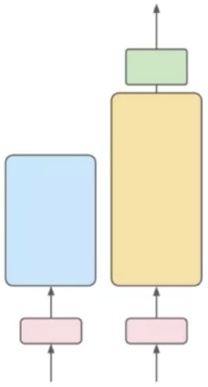
Example models:

- GPT
- BLOOM

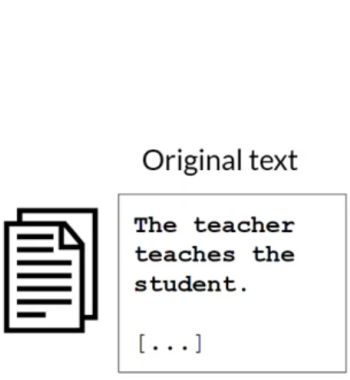
Sequence-to-sequence models

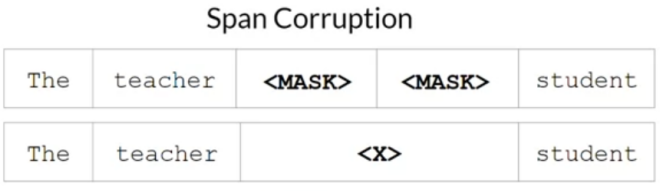


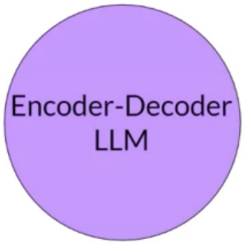


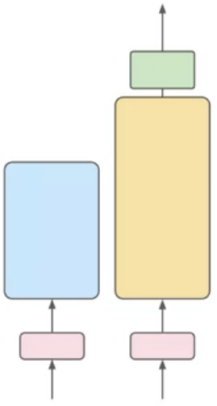


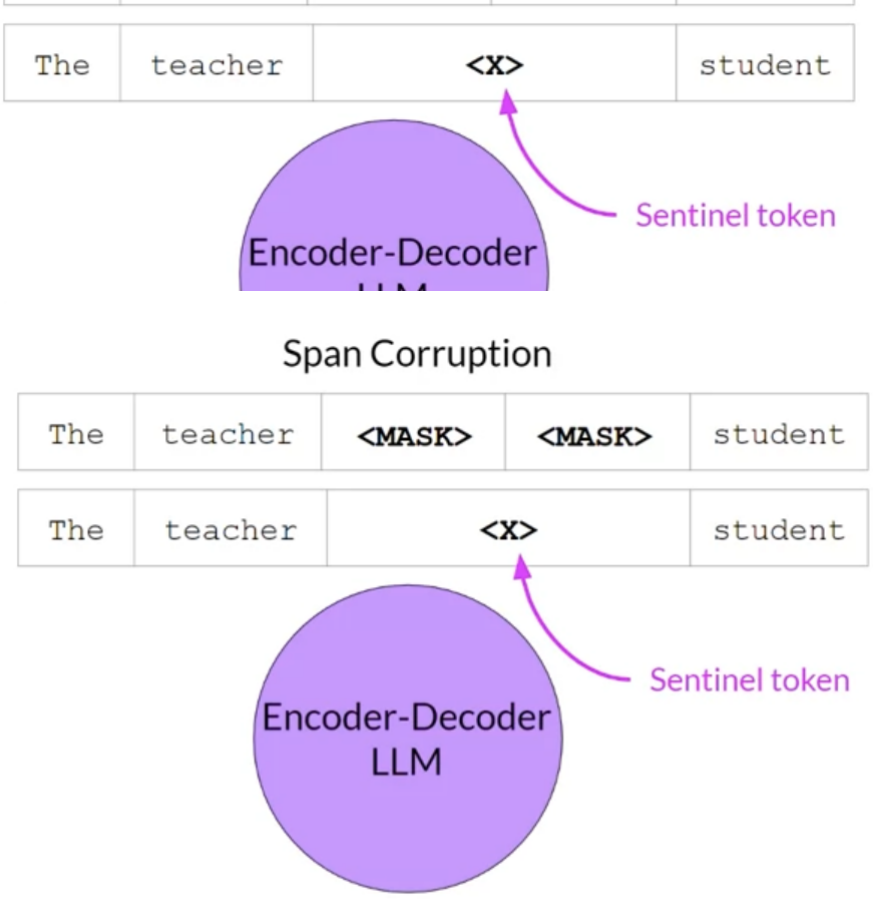
Sequence-to-sequence models









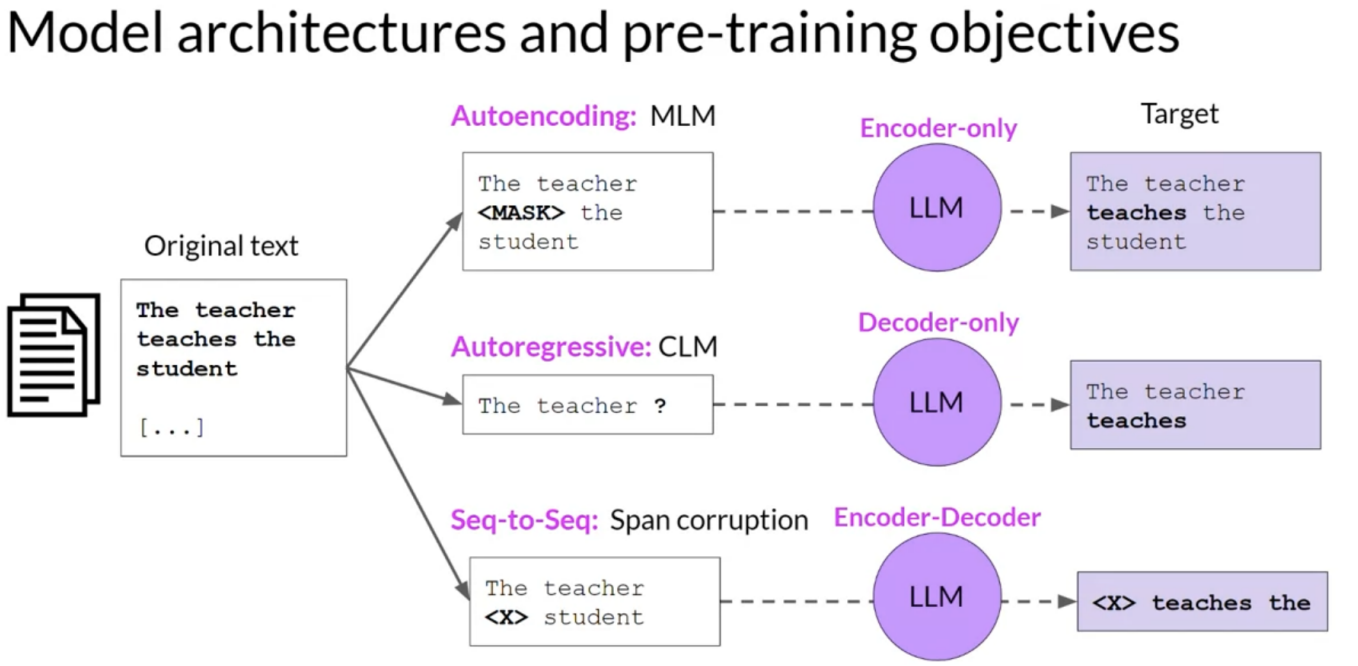


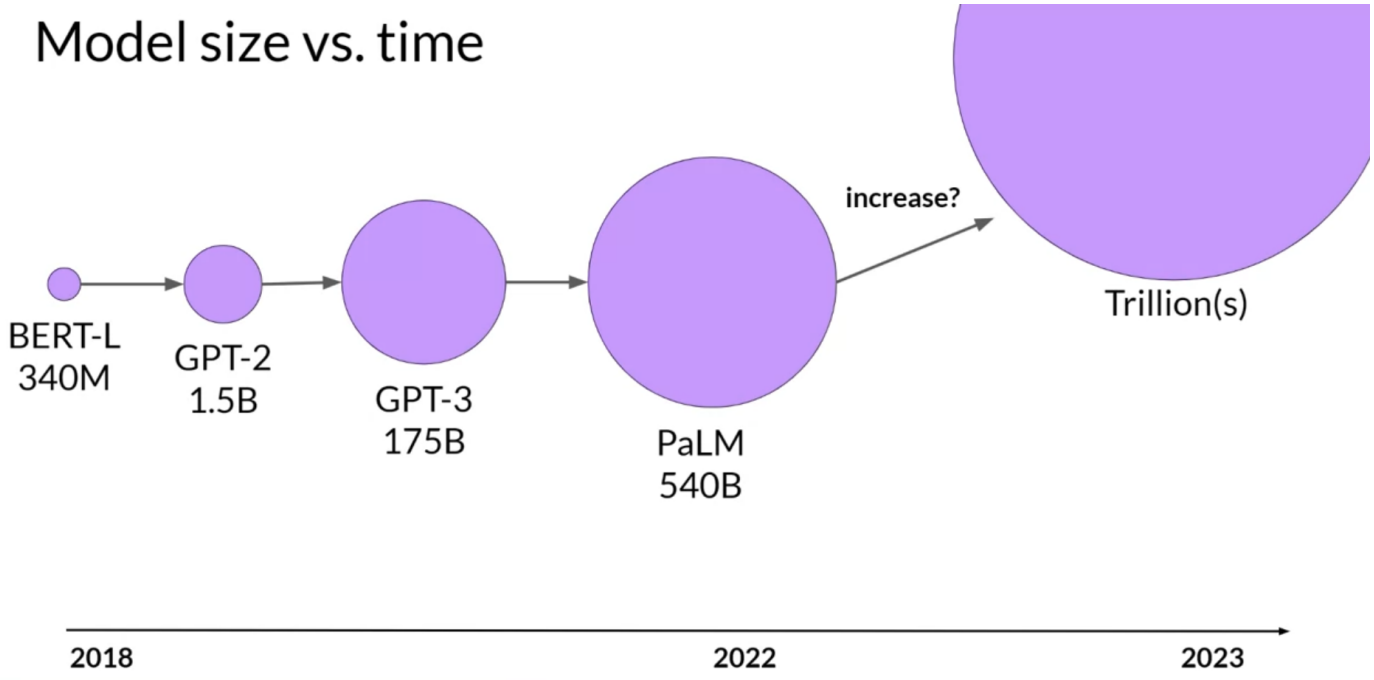
Good use cases:

- Translation
- Text summarization
- Question answering

Example models:

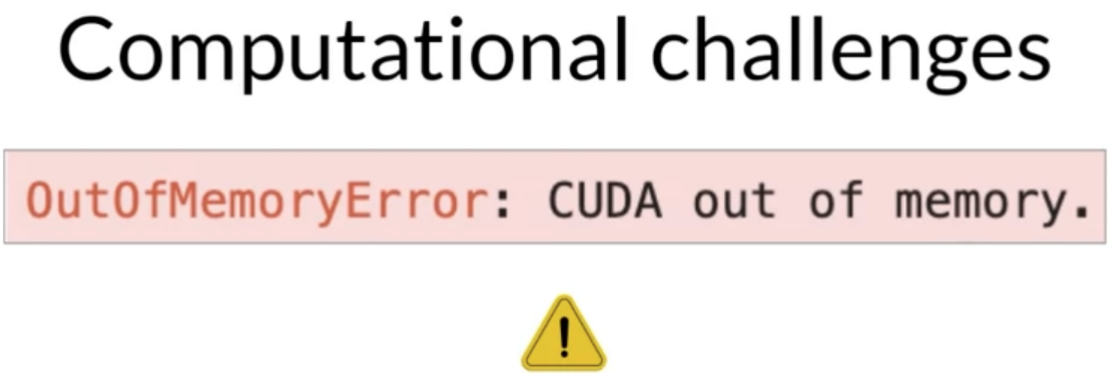
- T5
- BART





Computational Challenges

Out of Memory



Compute Unified Device Architecture

Approximate GPU RAM needed to store 1B parameters

1 parameter = 4 bytes (32-bit float)
1B parameters = 4 x 10^9 bytes = 4GB

Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, https://github.com/facebookresearch/bitsandbytes

Additional GPU RAM needed to train 1B parameters

	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter
Adam optimizer (2 states)	+8 bytes per parameter
Gradients	+4 bytes per parameter
Activations and temp memory (variable size)	+8 bytes per parameter (high-end estimate)
TOTAL	**=4 bytes per parameter +20 extra bytes per parameter**

Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, https://github.com/facebookresearch/bitsandbytes

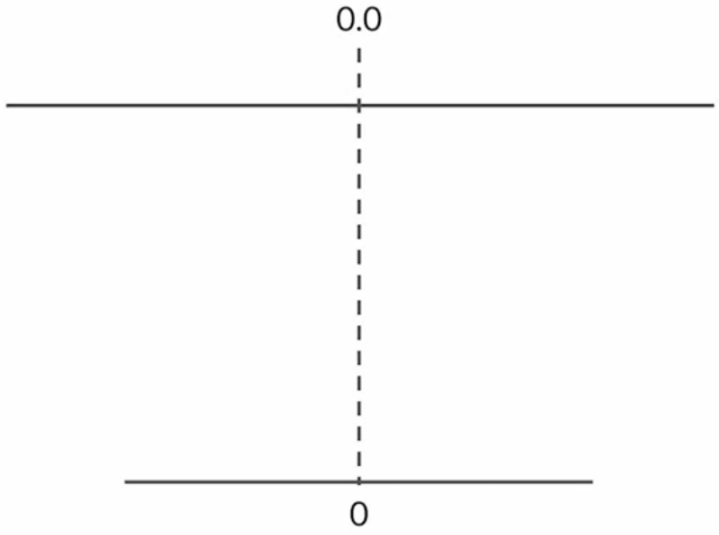
Approximate GPU RAM needed to train 1B-params

Memory needed to store model

4GB @ 32-bit full precision

Memory needed to train model

Quantization



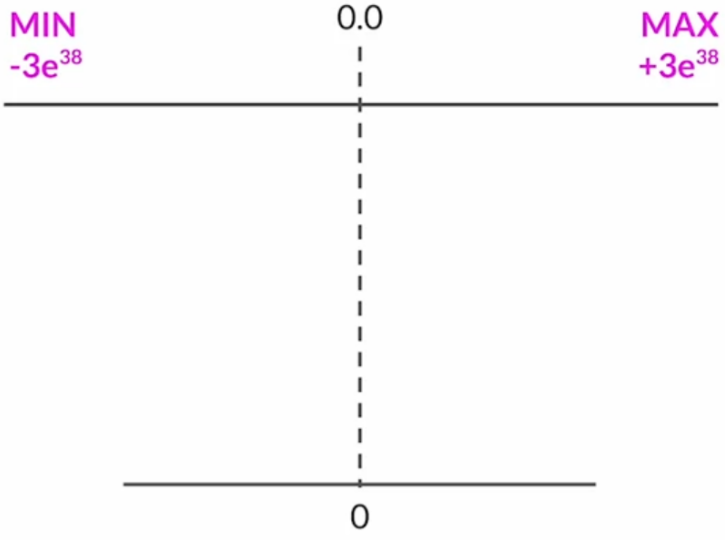
FP32

32-bit floating point

FP16 | BFLOAT16 | INT8

16-bit floating point | 8-bit integer

Quantization



FP32

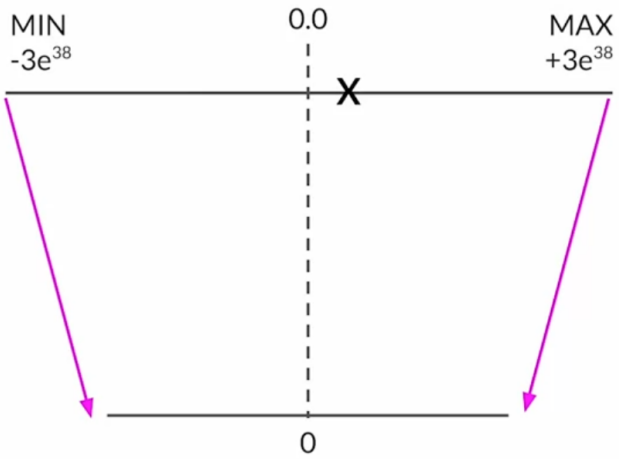
32-bit floating point

Range:
From **-3e^{38}** to **+3e^{38}**

FP16 | BFLOAT16 | INT8

16-bit floating point | 8-bit integer

Quantization: FP32



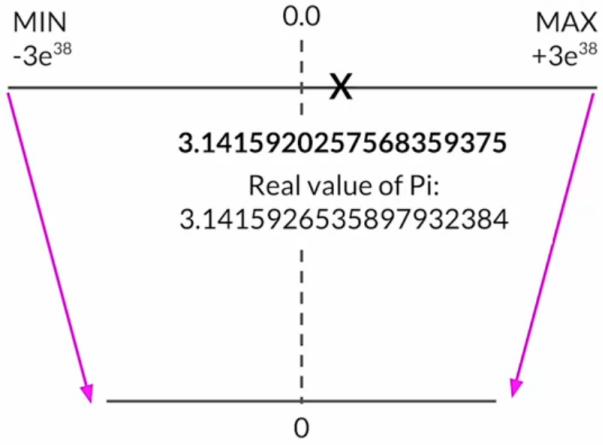
Let's store Pi: 3.141592

FP32

0	10000000	10010010000111111011000
Sign 1 bit	**Exponent** 8 bits	**Fraction** 23 bits

Mantissa / Significand
= Precision

Quantization: FP32



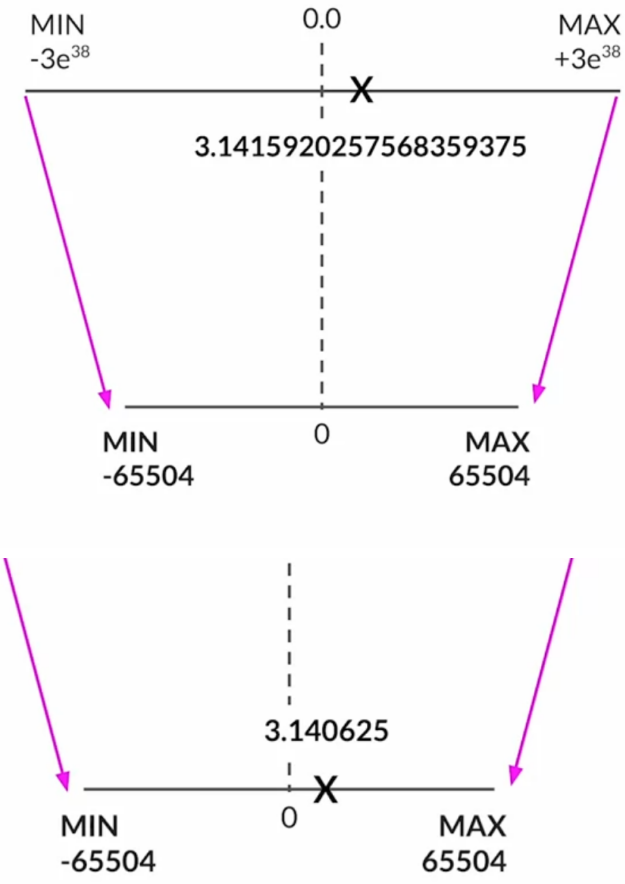
Let's store Pi: 3.141592

FP32

0	10000000	10010010000111111011000
Sign	Exponent	Fraction
1 bit	8 bits	23 bits

Mantissa / Significand
= Precision

Quantization: FP16



Let's store Pi: 3.141592

FP32

0	10000000	10010010000111111011000
Sign	Exponent	Fraction
1 bit	8 bits	23 bits

FP16

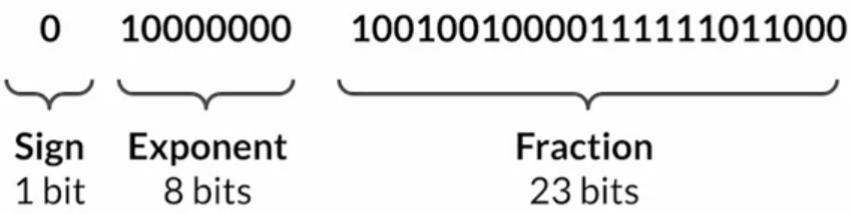
0	10000	1001001000
Sign	Exponent	Fraction
1 bit	5 bits	10 bits

Sign	Exponent	Fraction
1 bit	8 bits	23 bits

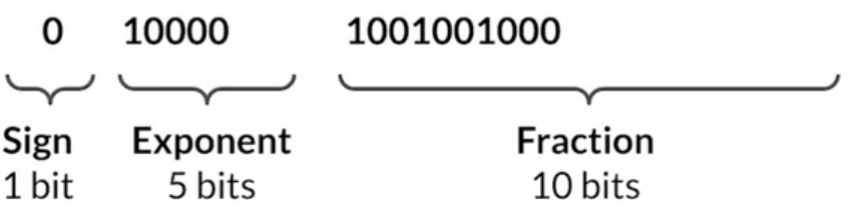
FP16

0	10000	1001001000
Sign	Exponent	Fraction
1 bit	5 bits	10 bits

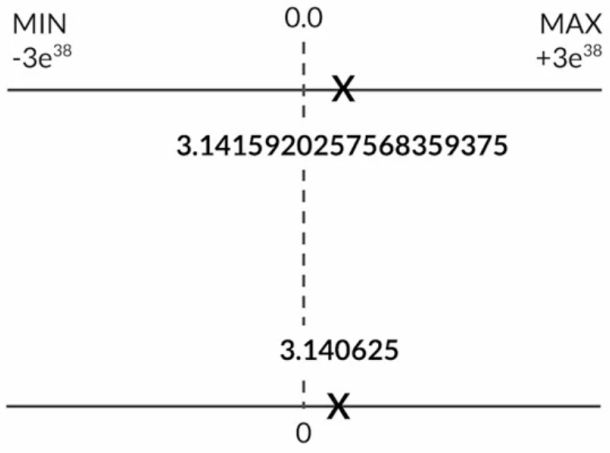
FP32 4 bytes memory



FP16 2 bytes memory

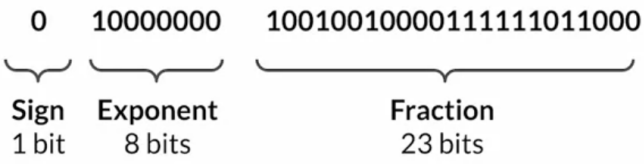


Quantization: BFLOAT16

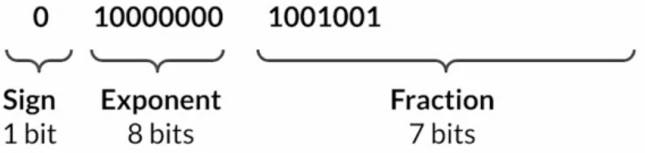


Let's store Pi: 3.141592

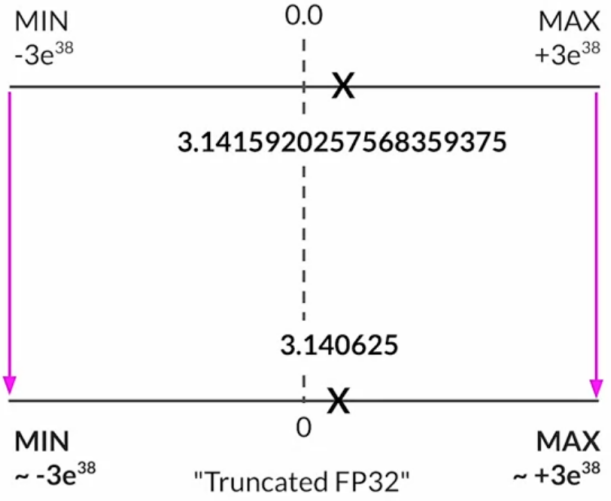
FP32 4 bytes memory



BFLOAT16 | BF16 2 bytes memory

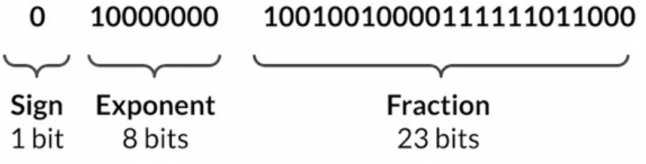


Quantization: BFLOAT16

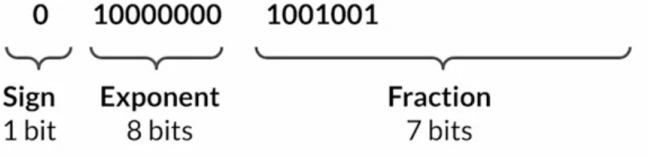


Let's store Pi: 3.141592

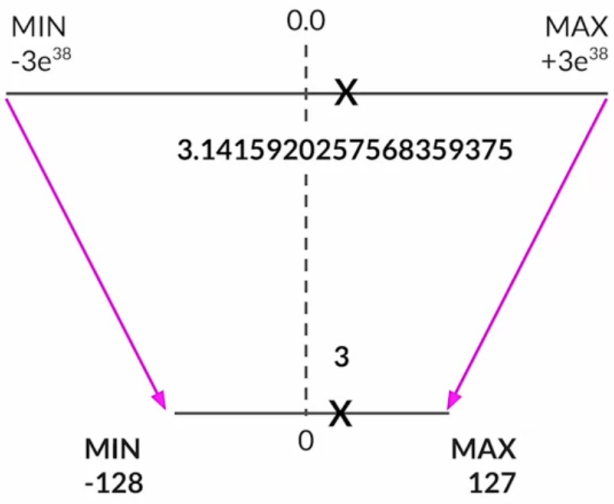
FP32 4 bytes memory



BFLOAT16 | BF16 2 bytes memory



Quantization: INT8



Let's store Pi: 3.141592

FP32 4 bytes memory

0 | 10000000 | 10010010000111111011000

Sign 1 bit | Exponent 8 bits | Fraction 23 bits

INT8 1 byte memory

0 | 0000011

Sign 1 bit | Fraction 7 bits

Quantization: Summary

	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	-/-	7	1 byte

- Reduce required memory to store and train models

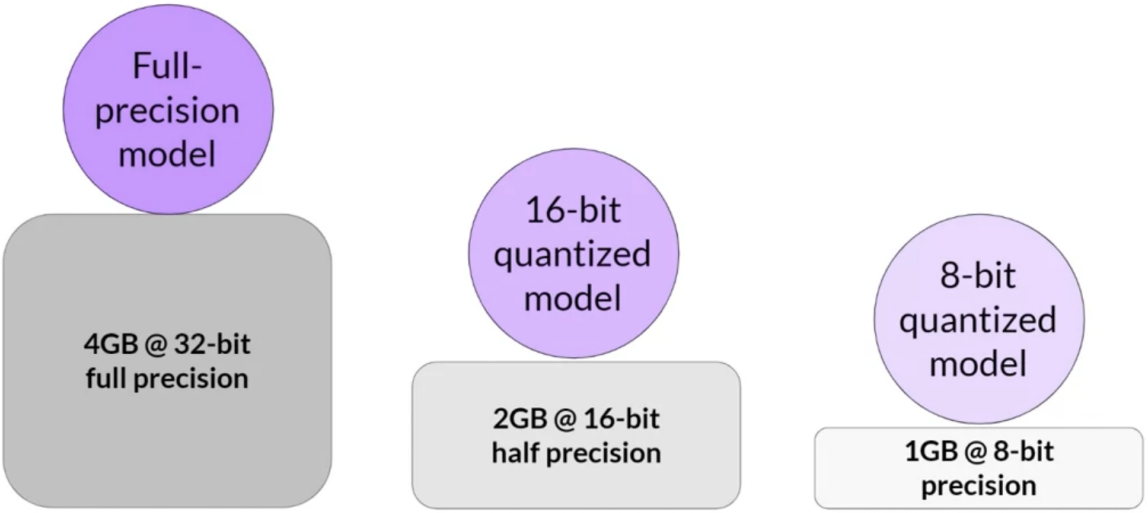
- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training

	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	–/–	7	1 byte

FLAN T5

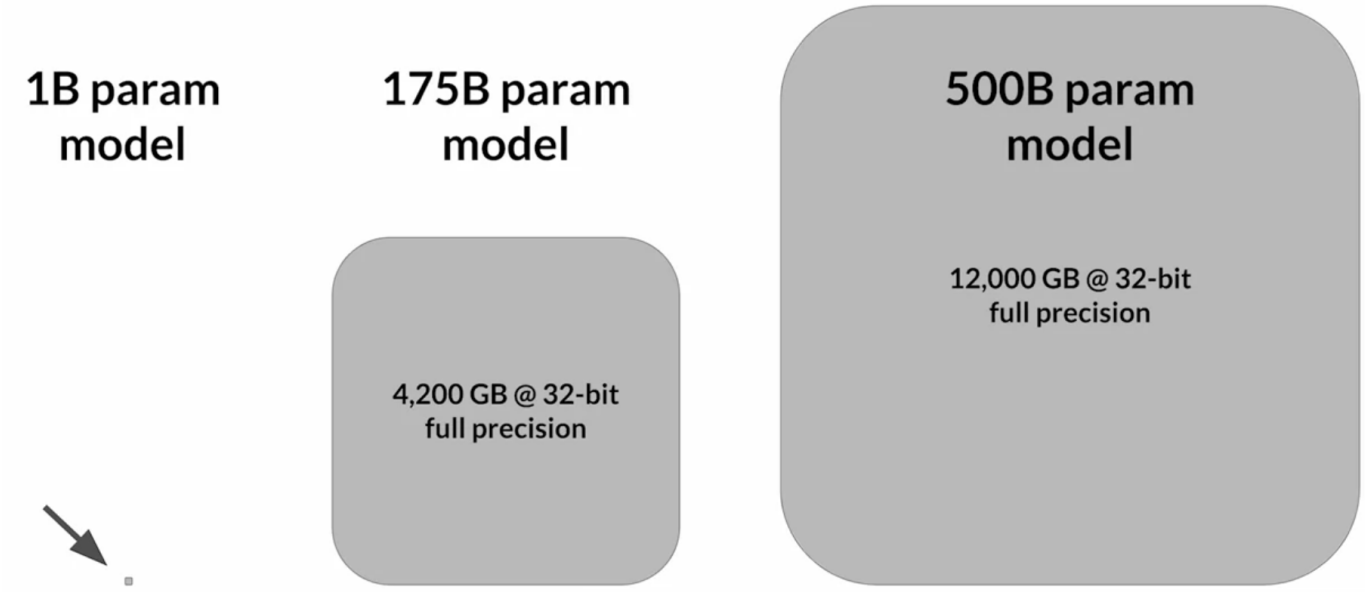
- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training
- BFLOAT16 is a popular choice

Approximate GPU RAM needed to store 1B parameters



Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, https://github.com/facebookresearch/bitsandbytes

GPU RAM needed to train larger models



As model sizes get larger, you will need to split your model across multiple GPUs for training