Module 1

Transformers

Attention and the Transformer Architecture

Learning Objectives

By the end of this module you will:

- Describe and build the inner workings of a transformer model
- Compare and contrast the different types of transformer architectures
- Recognize the importance and mechanics of attention
- Apply the different types of transformer models to solve problems

A *Transformative* Technology

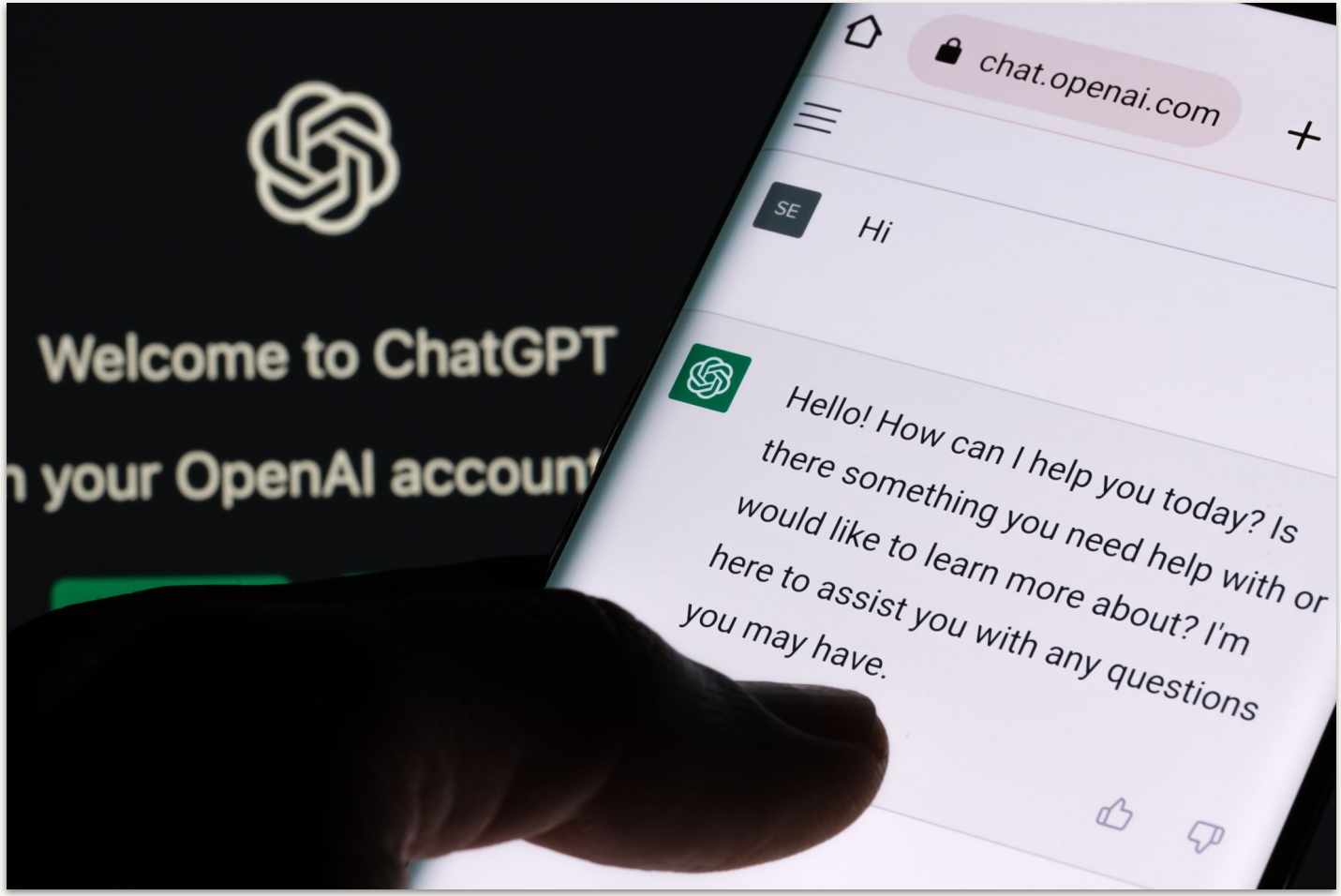
A breakthrough in Natural Language Processing

Let's Chat

A new technology paradigm has arrived.

Less than a year old, ChatGPT is the fastest adopted technology in human history.

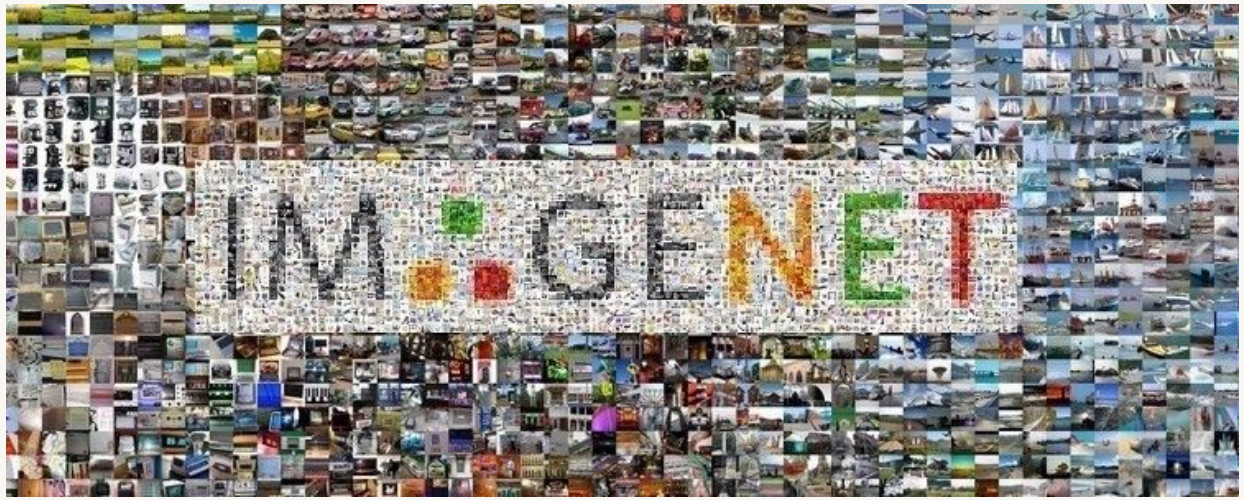
LLMs like ChatGPT have ushered in a new **golden era of AI**.

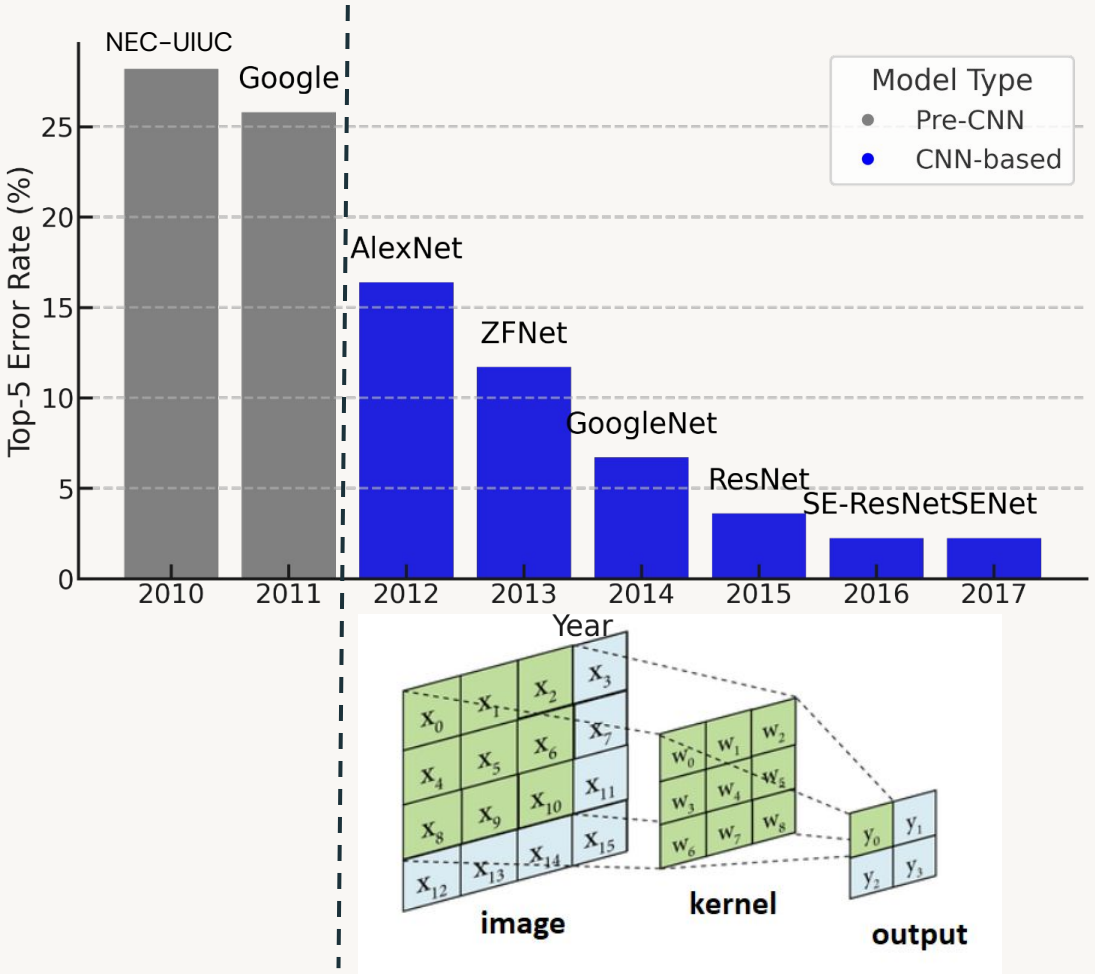


Source: Brookings

We've *seen* this before

The Convolution Layer led to models that dominated the vision field



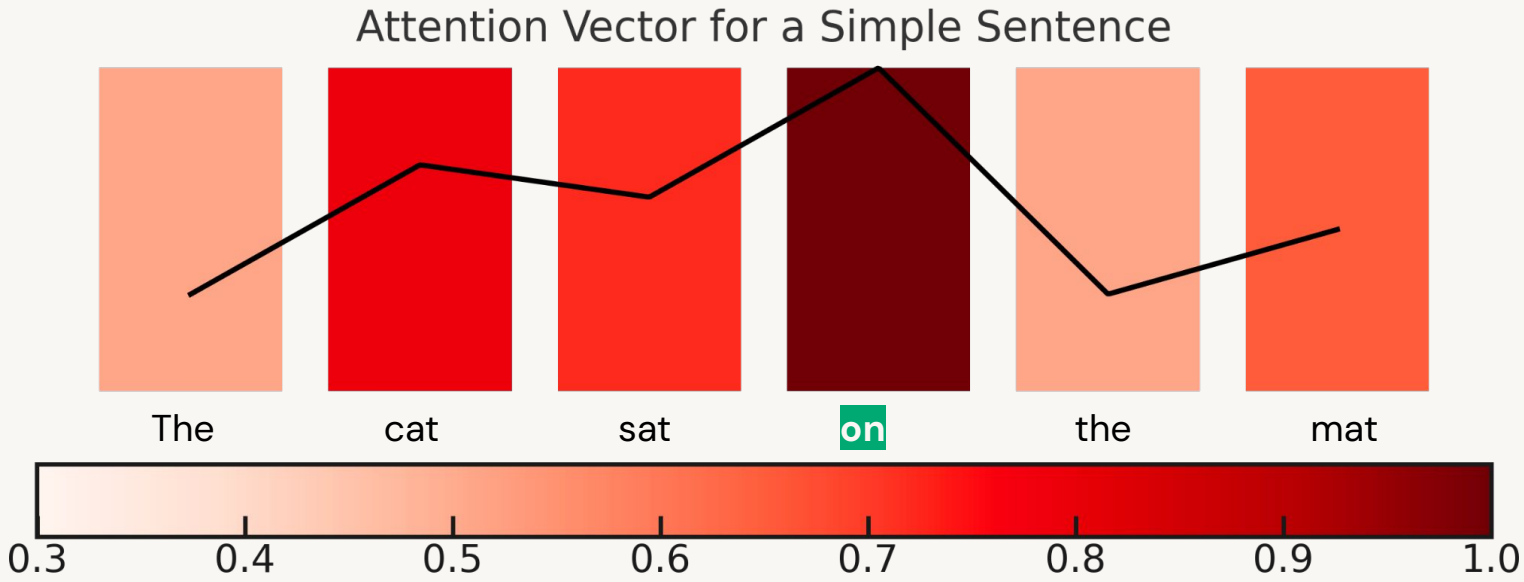


Source: ResearchGate

We weren’t paying attention before

Attention: “the convolution layer of language”

The attention mechanism measures how words interrelate



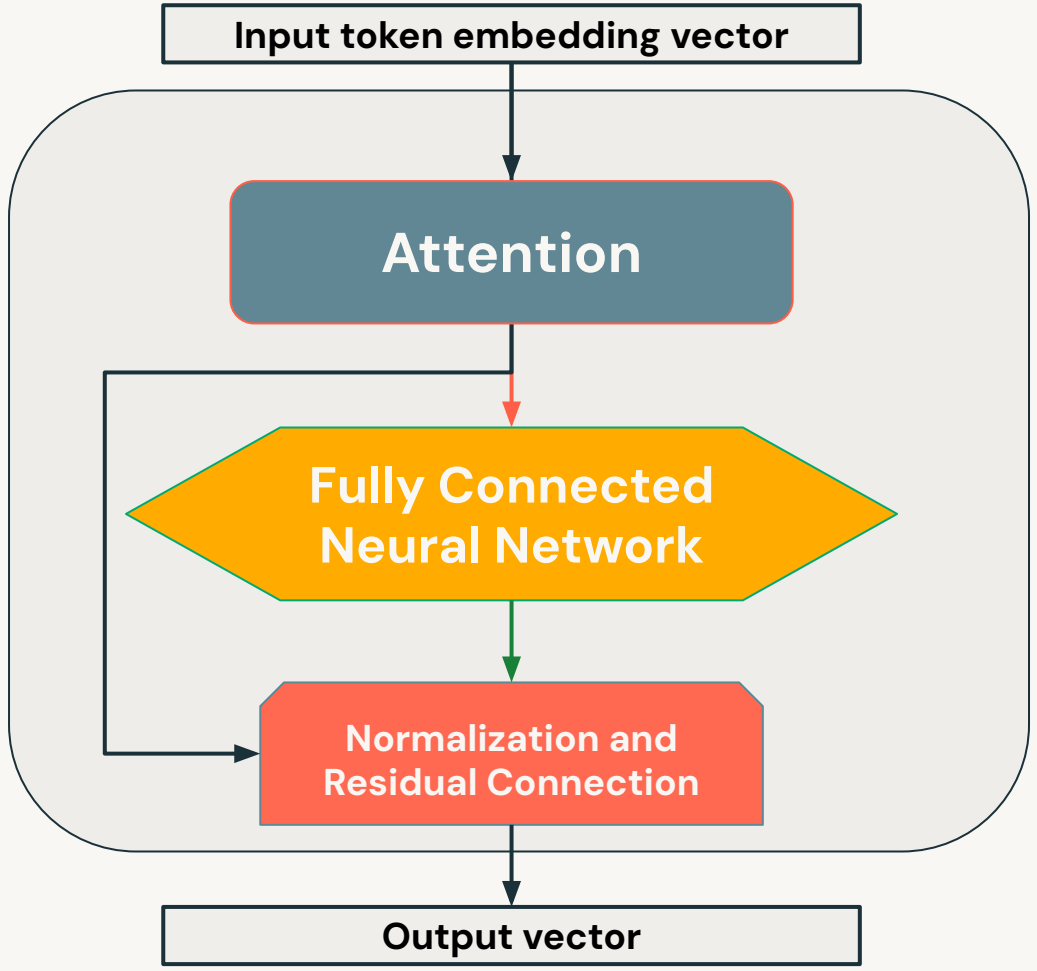
Attention is (not) all you need

A new type of deep learning architecture

Attention is a linear, matrix operation.

Where is the neural network?

Let's look at **The Transformer Block**



The Transformer Block

Building up the most important AI advancement in years

Transforming a sequence

What is the goal of a transformer block?

Transformer blocks:

1. Process input vectors with attention to enrich
2. Add nonlinear transformation with NN
3. Apply enriched vectors to select the correct token from the model's vocabulary

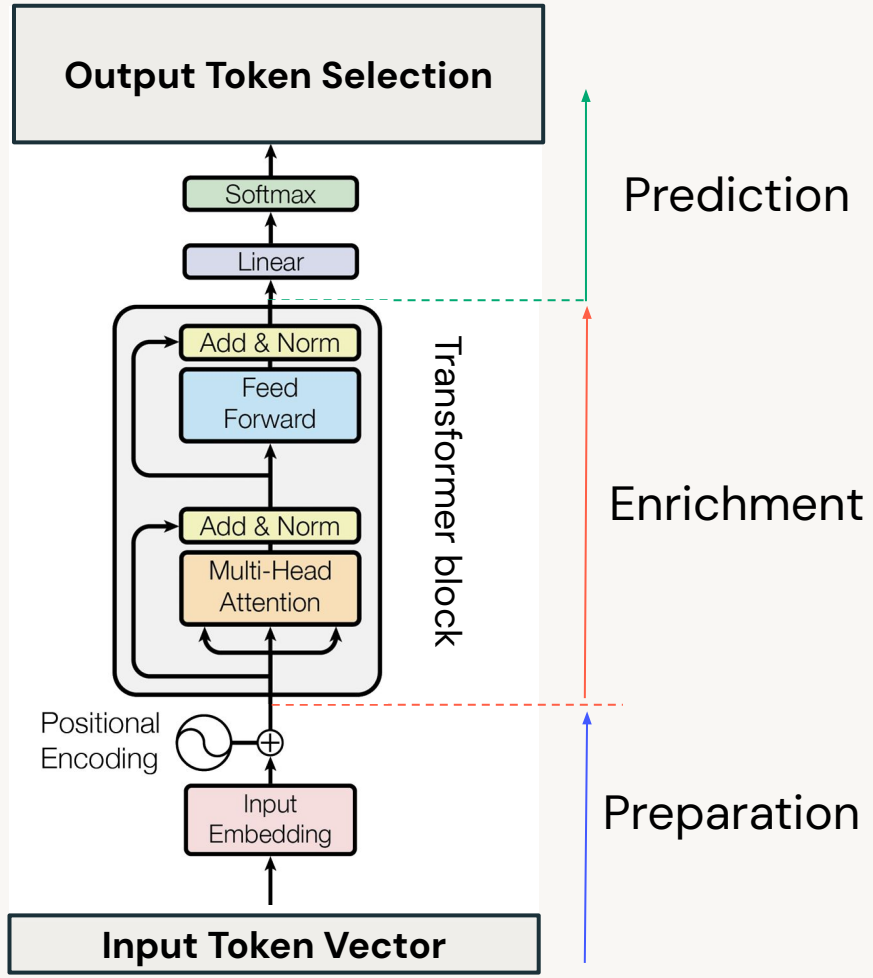
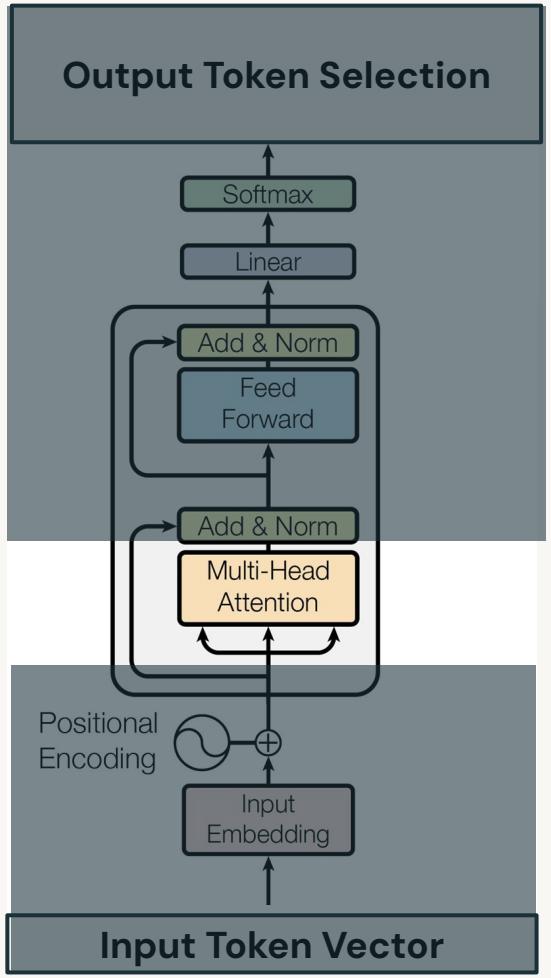


*Image modified from original transformer paper.

Source: ArXiv

Attention Mechanism

How important is each word to each other?

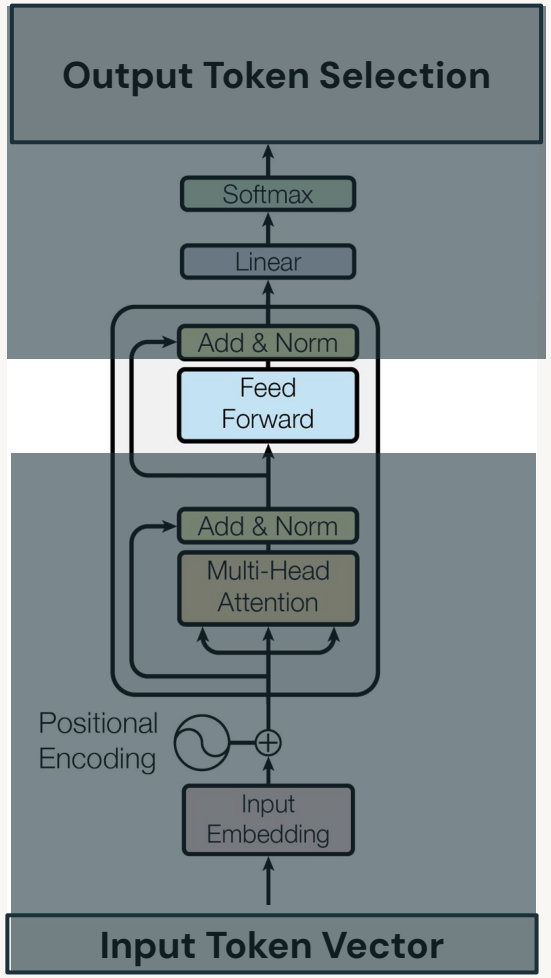


The role of attention is to:

- Measure the importance and relevance of each word relative to each other
- Allow for the building-up of enriched vectors with more context and logic

Position-wise Feed-Forward Networks

Adding nonlinearity to understanding the input.

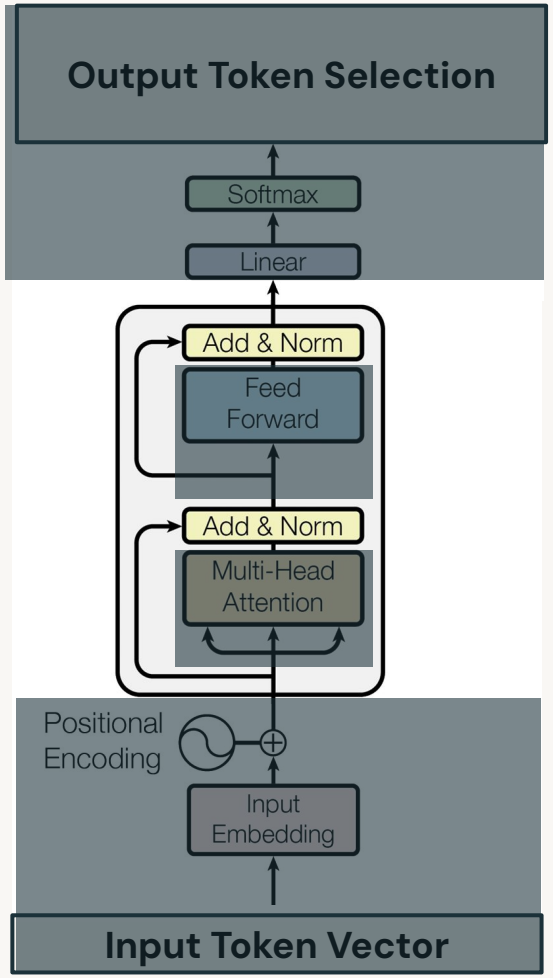


The role of Position-wise Feed-Forward Network is to:

- Use the information gathered from the attention stage for each element in the sequence and translate it to a further enriched state
- Allow for nonlinear transformations of each element in the sequence and builds upon itself in subsequent layers

Residual Connections & Layer Normalization

Ensuring robust and reliable training.



Residual Connections

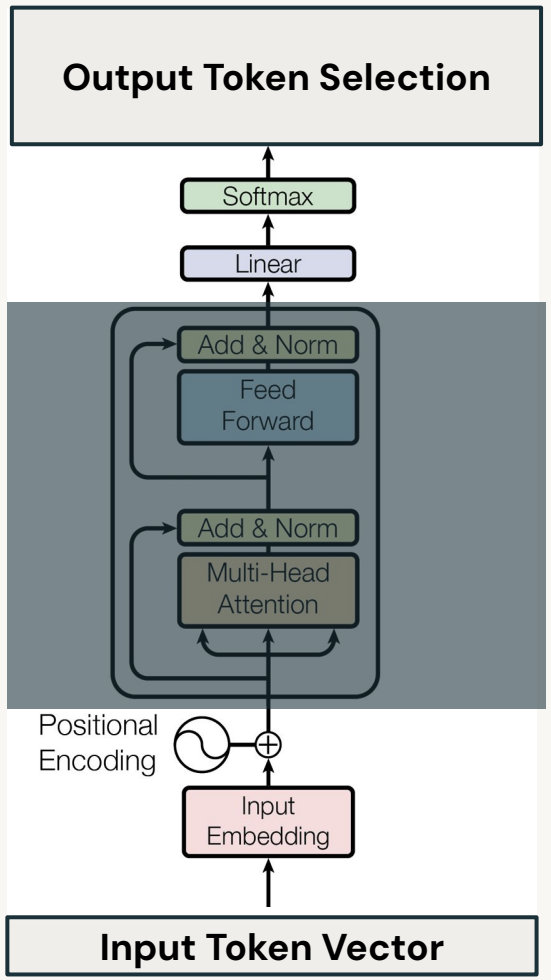
- Shortcuts in the network that allow information to flow directly from earlier layers to later layers.
- Help mitigate the vanishing gradient problem in deep neural networks.

Layer Normalization

- Normalizes the inputs across the features rather than the batch.
- Stabilizes the network's training, ensuring consistent input distribution, crucial in tasks with varying sequence lengths.

Into and out of the transformer

Input transformations and output transformations



Output from the Transformer

- Exiting the last Transformer block, is a sequence of context-aware vectors.
- Each vector in this sequence represents an input token, but now its representation is deeply influenced by its interactions with all other tokens in the sequence.
- Different models use the output from the transformer differently.

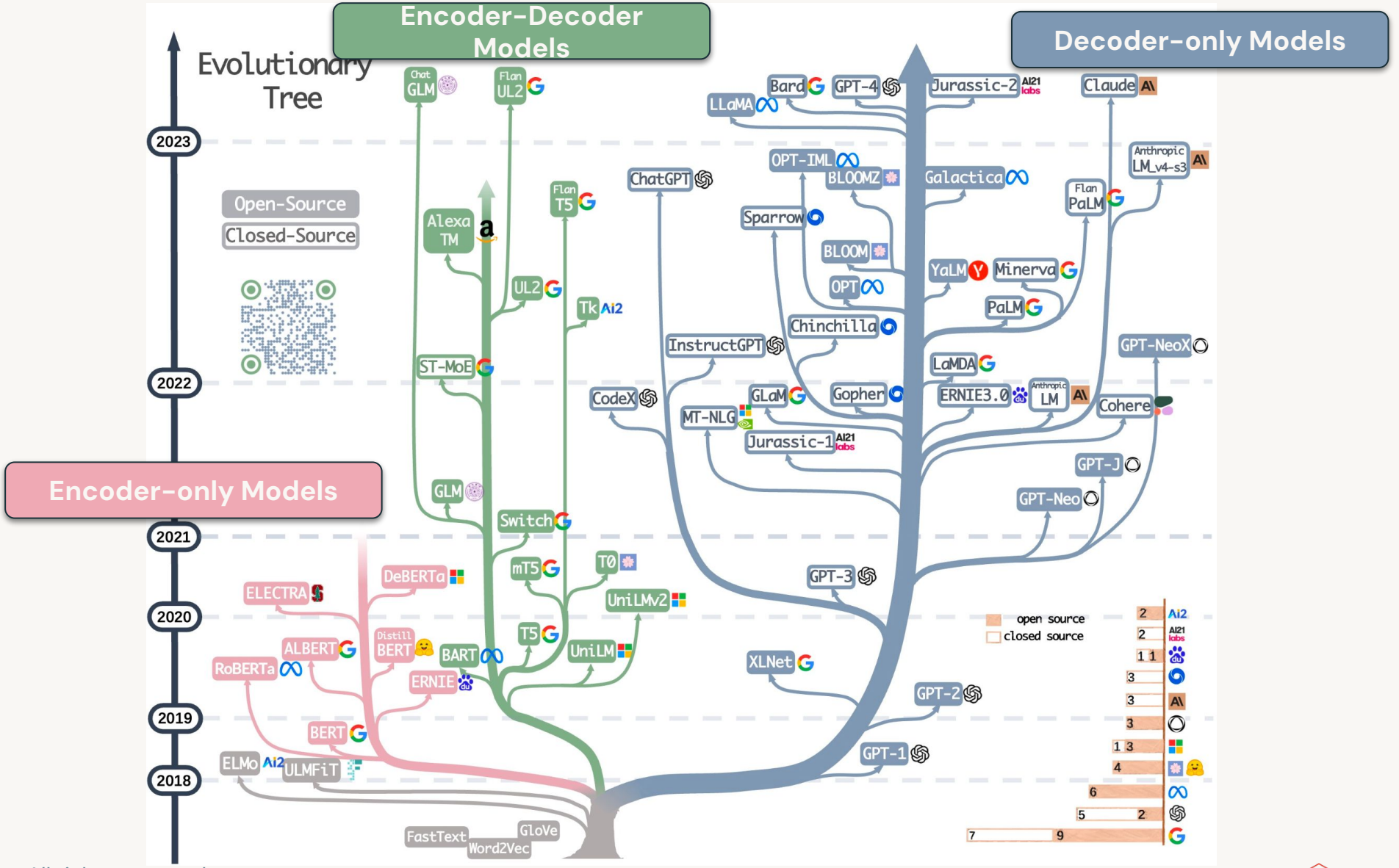
Input to the Transformer

- The initial input to a Transformer model is a sequence of word tokens.
- These tokens are converted into word embeddings.
- Positional encodings are added to these embeddings, providing the model with necessary positional information.
- This new sequence, with each element a dense vector, is then passed to the first transformer block.

Transformer Architectures

“Encoders, decoders, T5, oh my!”

The Transformer Family Tree



Source: GitHub

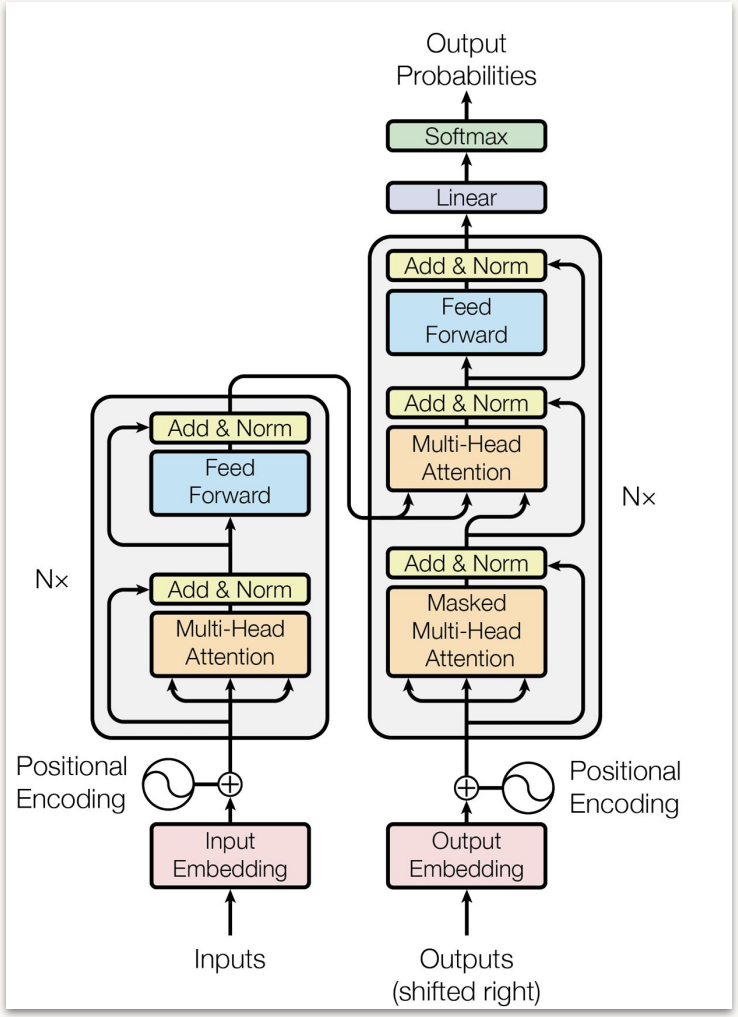
The Original Transformer

An encoder-decoder model

Defining Features:

- Two sets of transformer blocks
  - Encoder blocks
  - Decoder blocks
  - Cross-attention

- Typical Uses:
  - Translation
  - Conversion



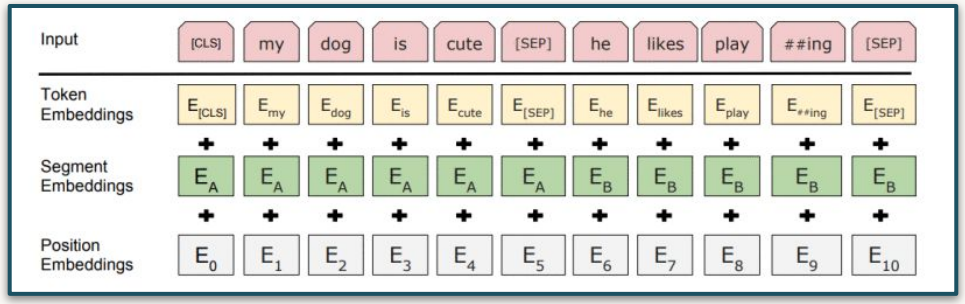
Source: ArXiv

B is for BERT

Encoder-only models

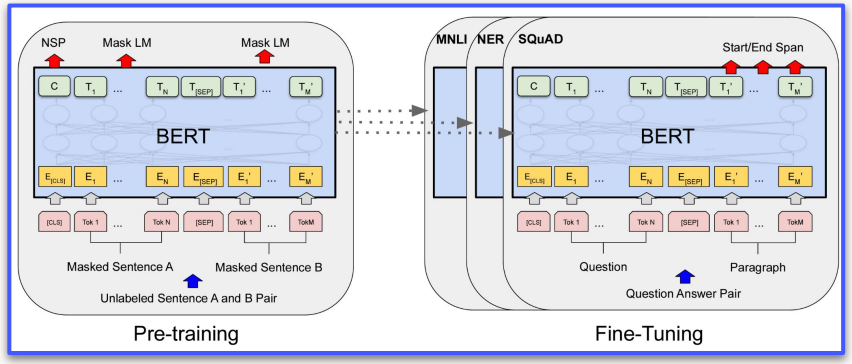
Novel Features of BERT:

- Segment embeddings ([CLS] and [SEP] special tokens)
- Trained with Masked Language Modeling & Next Sentence Prediction
- Excellent Fine-Tuning Performance



Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext



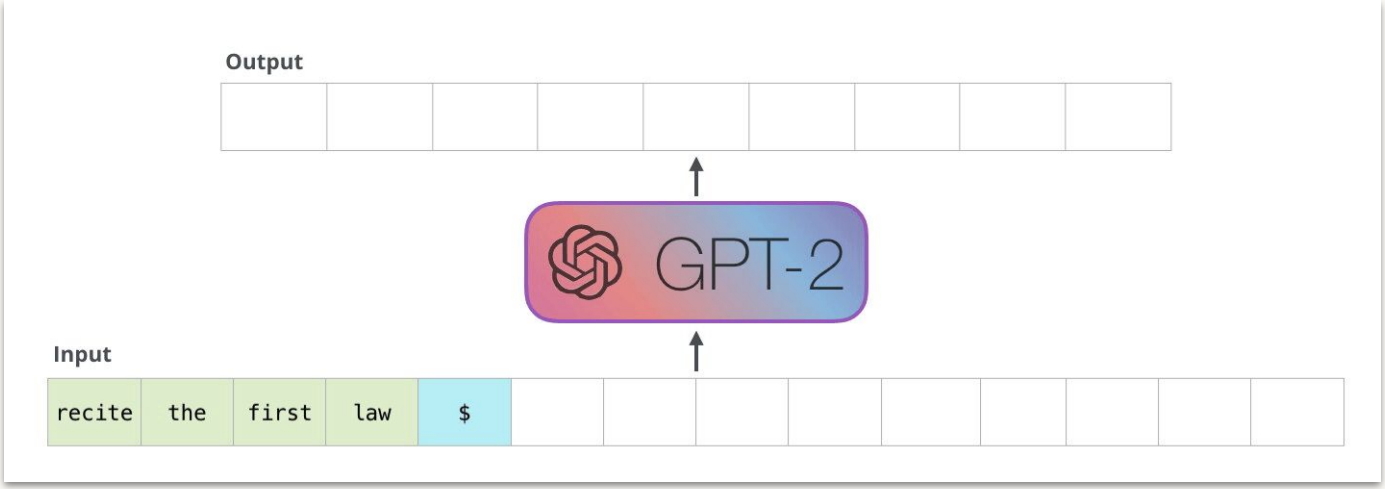
Source: ArXiv

Generating text with GPT

Decoder-only models

Decoder models have a single task:

- Look over the sequence and predict the next word (or token).



Decoder models products: **ChatGPT, Bard, Claude, LLaMA, MPT**

Source: IllustratedTransformer

Important variables in Transformers

Input

- **Vocabulary Size (V)**:
  The number of unique tokens that the model recognizes.
- **Embedding/Model Size (D)**:
  The dimensionality of the word embeddings, also known as the hidden size.
- **Sequence/Context Length (L)**:
  The maximum number of tokens that the model can process in a single pass.

Internal

- **Number of Attention Heads (H):**
  In the multi-head attention mechanism, the input is divided into H different parts.
- **Intermediate Size (I):**
  The feed-forward network has an intermediate layer whose size is typically larger than the embedding size.
- **Number of Layers (N):**
  The number of Transformer blocks/layers.

Training

- **Batch Size (B):**
  The number of examples processed together in one forward/backward pass during training.
- **Tokens Trained on (T):**
  The total number of tokens that a model sees during training. This is normally reported more than the number of epochs.

Time to Pay *Attention*

The secret that unlocked the power of LLMs

The inner workings of attention

Learning the weights of attention.

We use three, large (millions of elements), matrices to create the Query, Key, and Value vectors in each layer.

Query Matrix: W_Q, Key Matrix: W_K, Value Matrix: W_V

The Attention equation can be expressed as an operation over all the vectors:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The inner workings of attention

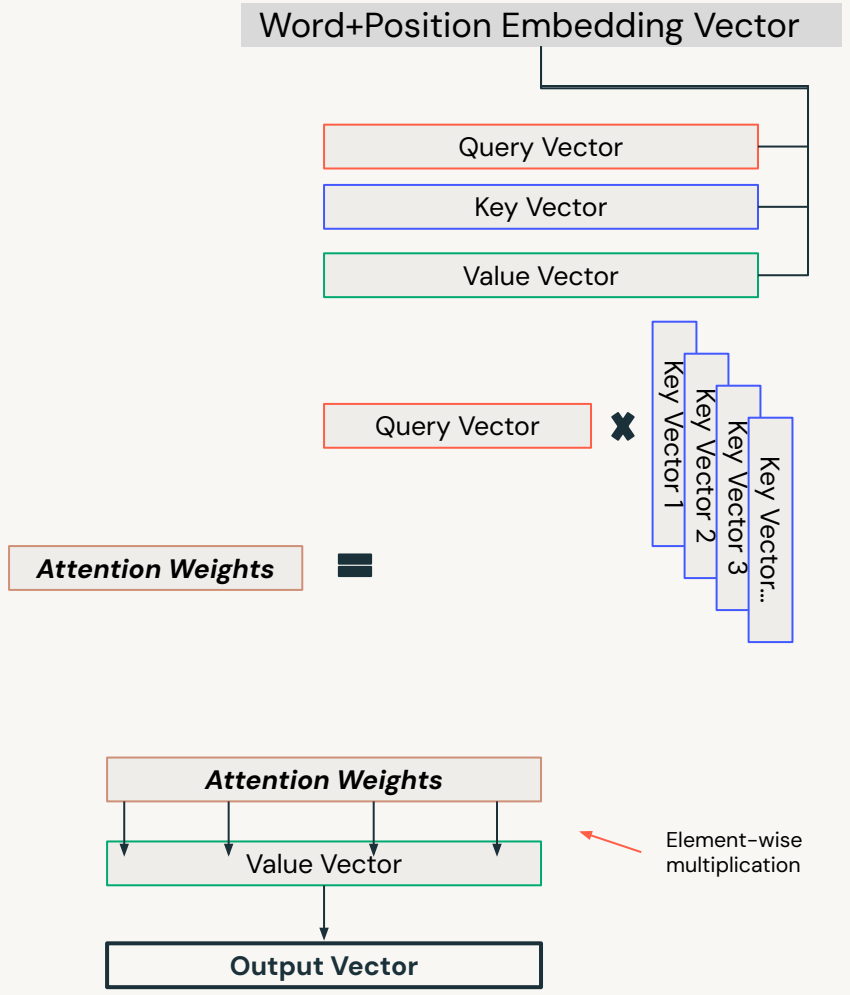
How do we calculate attention?

The mechanism can be broken down into three steps:

1) The input vector (in the first layer, this is the word embedding vector (with the position information), is used to create three new vectors:
   a) the Query (Q) vector - made from the current token
   b) the Key (K) vector - made from all other tokens in the sequence
   c) the Value (V) vector - made from all tokens in the sequence

2) A scaled dot-product is then performed on the Query and Key vectors, this is the attention score. This also uses a softmax function to produce a final set of ***Attention Weights*** that are scaled 0-1.

3) Each Value vector is multiplied by its corresponding attention weight and then all are summed up to generate the output for the self-attention layer.



Building Base/Foundation Models

Training transformers, what does it take?

Foundation Model Training - Getting Started

Choosing the right options to build your model.

A **foundation**, or **base**, model is a transformer that is trained from scratch.

- Model Architecture - decoder? encoder-decoder?
  - Tailored to the task/problem
  - Different sizes: embedding dimensions, number of transformer blocks, etc.
- Available Data
  - Task-relevant data
  - Good language coverage
- Available Compute Resources
  - Time to allow for training
  - Hardware available

Foundation Model Training - Architecture

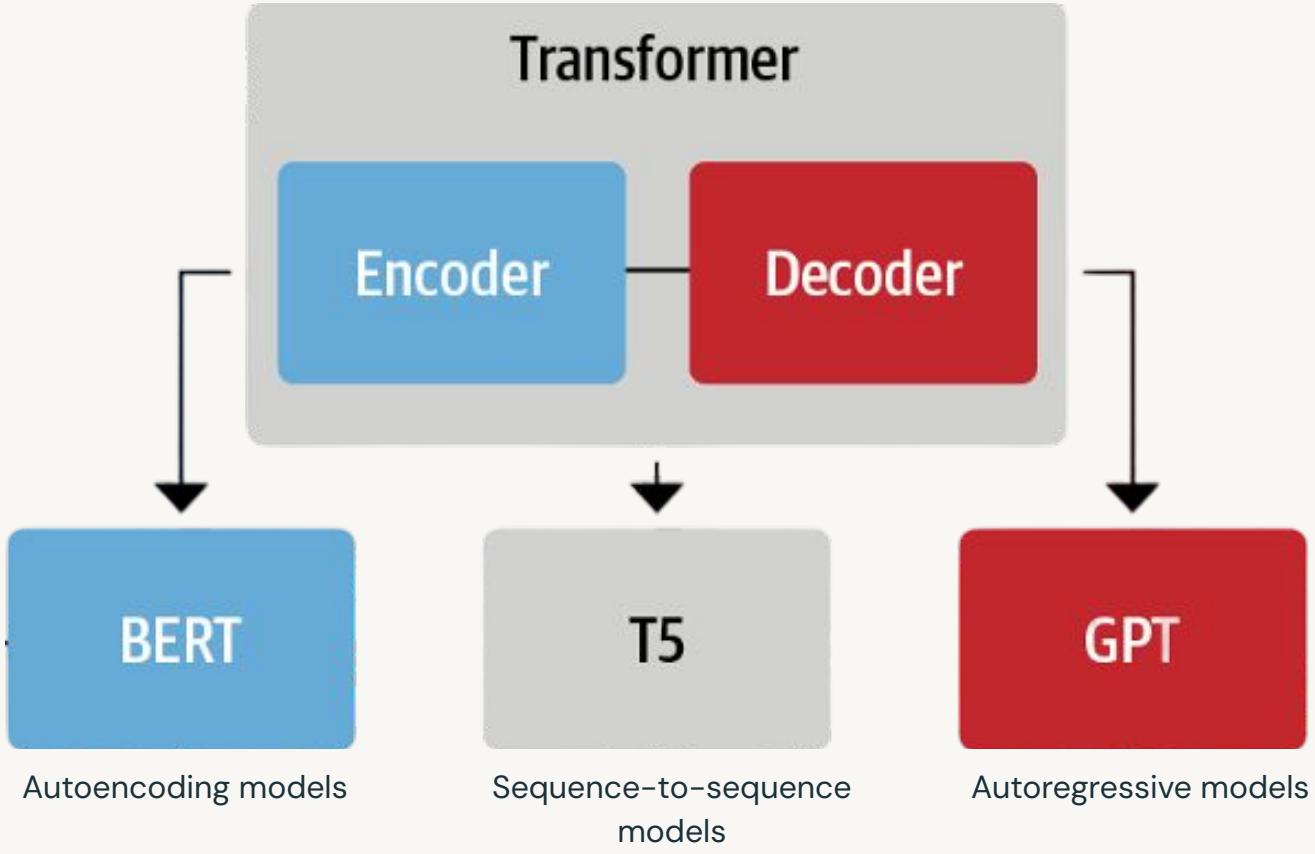
Which transformer flavor is right?

Start with your task:

- Classification?
- Generation?
- Translation?

Architecture:

- Layer count
- Context size



Source: GitHub

Foundation Model Training – Data

It's all about the data

Datasets for foundation LLM:

- Web Text
- Code
- Images
- Digitized Text
- Transcriptions

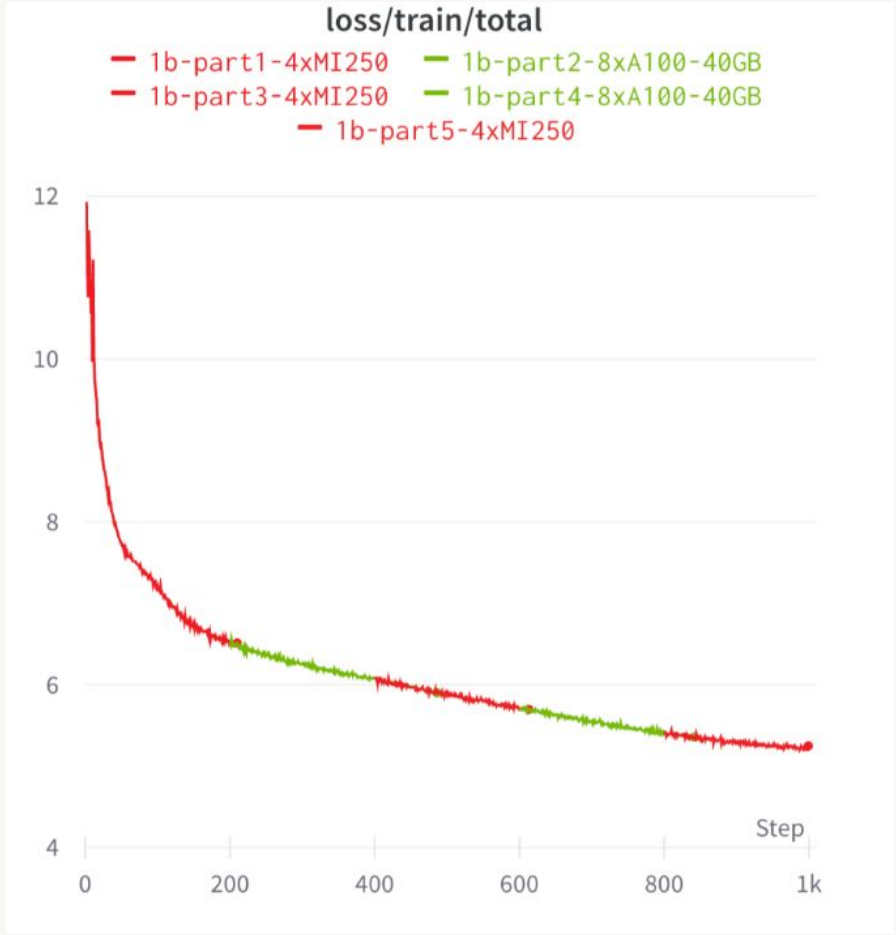
Component	Raw Size	Weight	Epochs	Effective Size	Mean Document Size
Pile-CC	227.12 GiB	18.11%	1.0	227.12 GiB	4.33 KiB
PubMed Central	90.27 GiB	14.40%	2.0	180.55 GiB	30.55 KiB
Books3†	100.96 GiB	12.07%	1.5	151.44 GiB	538.36 KiB
OpenWebText2	62.77 GiB	10.01%	2.0	125.54 GiB	3.85 KiB
ArXiv	56.21 GiB	8.96%	2.0	112.42 GiB	46.61 KiB
Github	95.16 GiB	7.59%	1.0	95.16 GiB	5.25 KiB
FreeLaw	51.15 GiB	6.12%	1.5	76.73 GiB	15.06 KiB
Stack Exchange	32.20 GiB	5.13%	2.0	64.39 GiB	2.16 KiB
USPTO Backgrounds	22.90 GiB	3.65%	2.0	45.81 GiB	4.08 KiB
PubMed Abstracts	19.26 GiB	3.07%	2.0	38.53 GiB	1.30 KiB
Gutenberg (PG-19)†	10.88 GiB	2.17%	2.5	27.19 GiB	398.73 KiB
OpenSubtitles†	12.98 GiB	1.55%	1.5	19.47 GiB	30.48 KiB
Wikipedia (en)†	6.38 GiB	1.53%	3.0	19.13 GiB	1.11 KiB
DM Mathematics†	7.75 GiB	1.24%	2.0	15.49 GiB	8.00 KiB
Ubuntu IRC	5.52 GiB	0.88%	2.0	11.03 GiB	545.48 KiB
BookCorpus2	6.30 GiB	0.75%	1.5	9.45 GiB	369.87 KiB
EuroParl†	4.59 GiB	0.73%	2.0	9.17 GiB	68.87 KiB
HackerNews	3.90 GiB	0.62%	2.0	7.80 GiB	4.92 KiB
YoutubeSubtitles	3.73 GiB	0.60%	2.0	7.47 GiB	22.55 KiB
PhilPapers	2.38 GiB	0.38%	2.0	4.76 GiB	73.37 KiB
NIH ExPorter	1.89 GiB	0.30%	2.0	3.79 GiB	2.11 KiB
Enron Emails†	0.88 GiB	0.14%	2.0	1.76 GiB	1.78 KiB
The Pile	**825.18 GiB**			**1254.20 GiB**	**5.91 KiB**

Source: Stanford CS324

Foundation Model Training – Training

Optimizing LLM Losses

- LLMs train just like other deep learning models.
- Loss functions are typically **cross-entropy.**
- Familiar optimizers like **AdamW**, are used.
- Training LLMs with 100's of billions of parameters, can take **months** on **hundreds** of GPUs.



Source: MosaicML

Now what?

What do you use a foundation LLM for?

Foundation LLMs suffer from “alignment” problems:

- Bias/toxicity in the training data
- Lack of specific task focus -> for models like GPT it just focuses on selected the next correct word

Fine tuning and other methods are required to ensure task success.

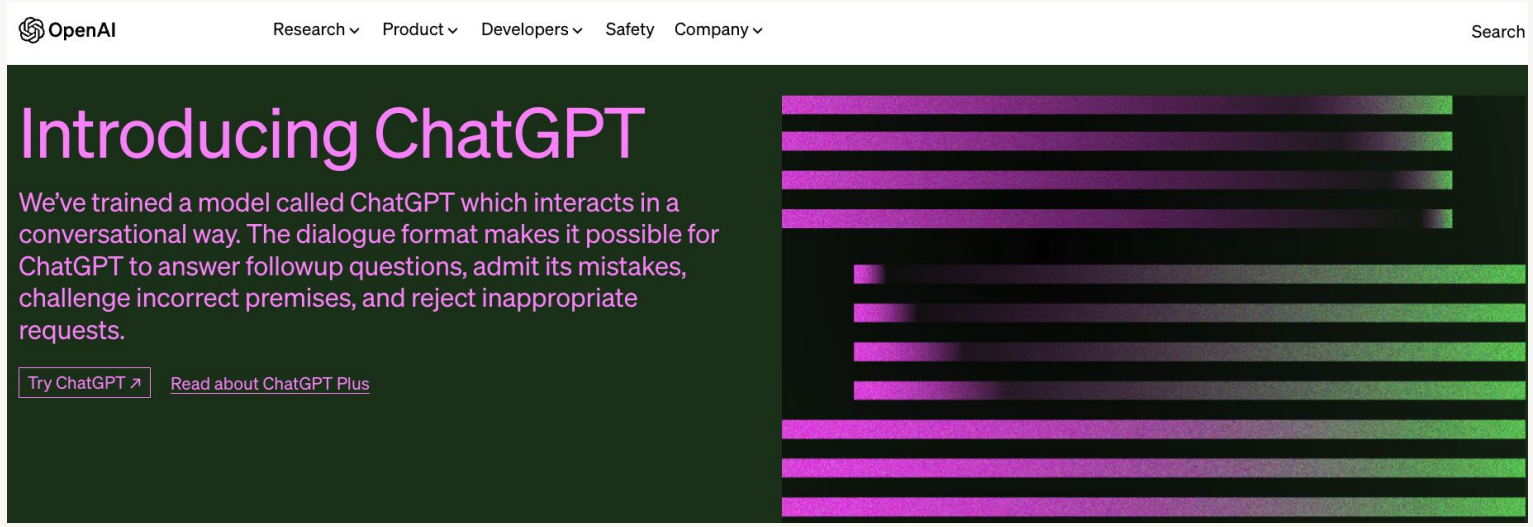
Generative Pretrained Transformer

A journey to discover how GPT-4 and ChatGPT were built.

The journey to ChatGPT

What is GPT?

- ChatGPT was originally built atop a large language model known as GPT-3.5.



- GPT-3.5 is a type of decoder-based transformer model.
- Let's see what exactly GPT is.

Source: OpenAI

Generative Pre-trained Transformers (GPT)

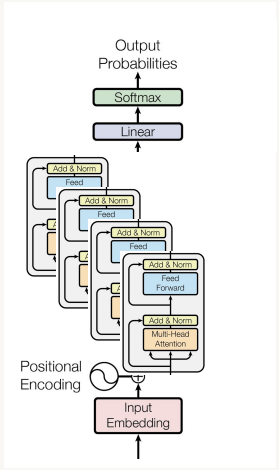
Decoder-based transformers

- The first GPT model, introduced in 2018 was just the decoder part of the original transformer.

- GPT-2/-3/-4 have mostly just been larger versions. With the key differences coming from training data and training processes.

GPT/GPT-1

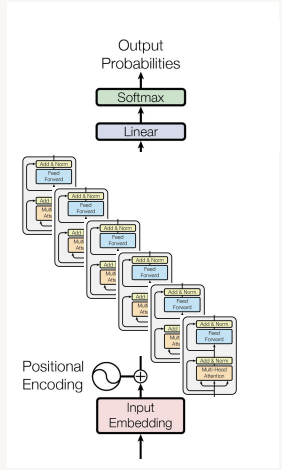
12 x



512 dimension embeddings

GPT-2

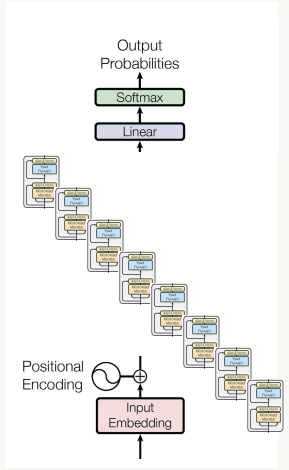
48 x



1024 dimension embeddings

GPT-3

96 x



2048 dimension embeddings

Decoder Models only use the second half of the original transformer architecture. No cross/encoder attention.

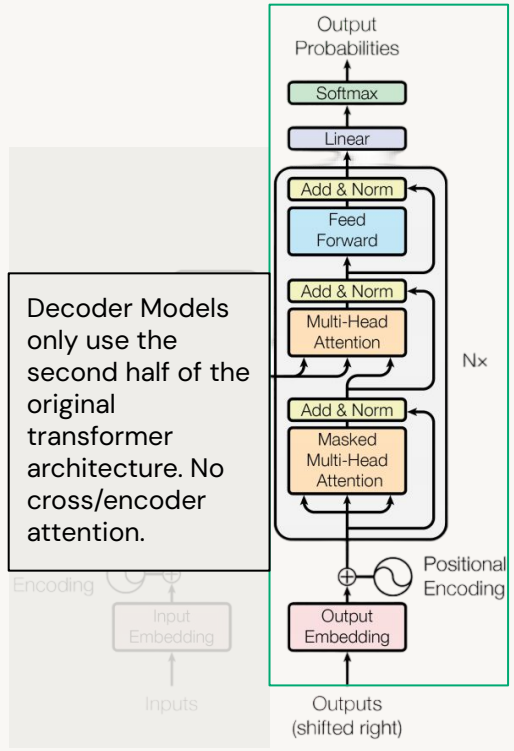


Figure 1: The Transformer - model architecture.

Source: LinkedIn

Generative Pre-trained Transformers (GPT)

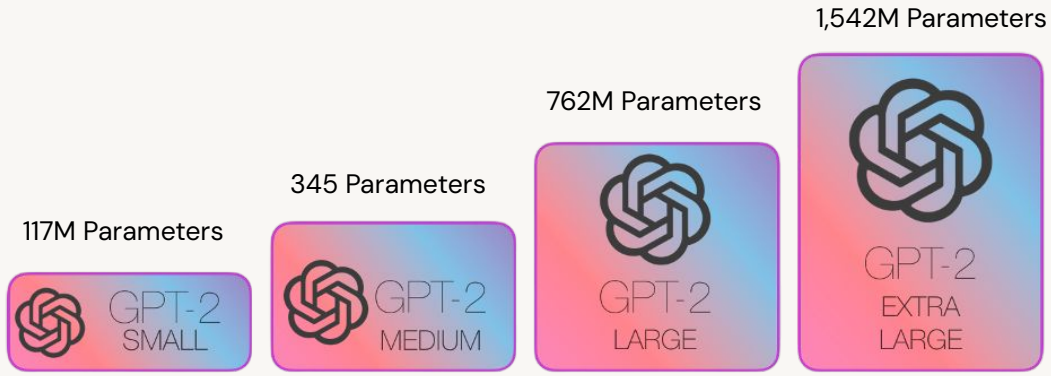
Generational Improvements

GPT (2018):

It was pre-trained on the **BooksCorpus** dataset and contained 117 million parameters. GPT was an autoregressive model based on the Transformer architecture and employed a unidirectional language modeling objective.

GPT-2 (2019):

pre-trained on a much larger dataset called **WebText**, which contained text from web pages with a total of 45 terabytes of data. GPT-2 came in four different sizes: 117M (small), 345M (medium), 774M (large), and 1.5B (extra-large) parameters.



Source: IllustratedGPT2

Generative Pre-trained Transformers (GPT)

Generational Improvements

GPT-3 (2020):

Pre-trained on the **WebText2** dataset, an even larger portion of the internet, 45 terabytes of text.

GPT-3 came with a staggering 175 billion parameters

Exceptional **few-shot** and **zero-shot** learning capabilities, allowing it to perform well on various tasks with minimal or no fine-tuning.

GPT-4 (2023):

Specific details about the architecture, dataset, and parameter count for GPT-4 have not been officially released. Likely ~1T parameters in an ensemble of smaller models of ~220B parameters each.

GPT Architecture

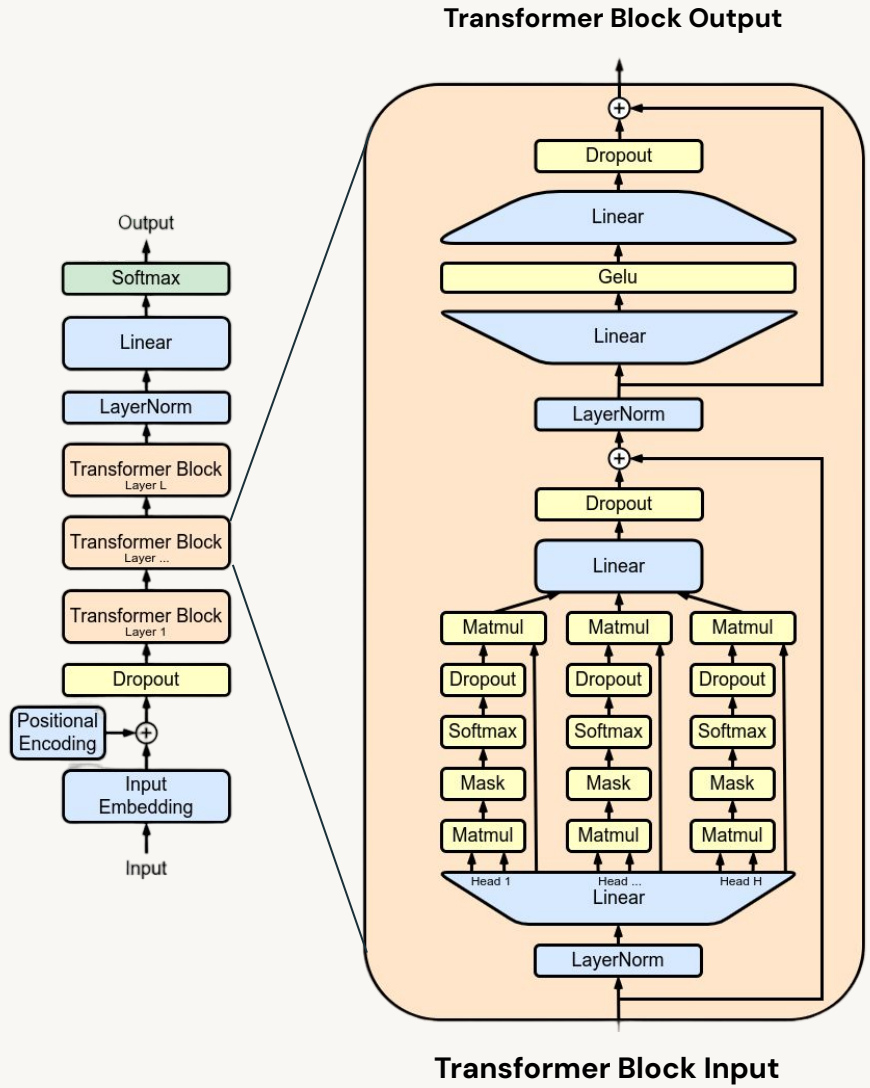
So many layers?

Why GPTs keep getting bigger

The main component of each layer: **multi-headed attention block**.

Like in convolutional layers, the earlier features that are learned might be edges and lines, and the later layers are more complex visual artifacts, attention layers work in a similar fashion:

1. **Early Attention**: short-range dependencies, relationships between adjacent or close tokens.
   - Word order
   - Part-of-Speech
   - Basic sentence structure.
2. **Middle Attention**: overall context of the input sequence.
   - Semantic information.
   - Meaning.
   - Relationships between phrases.
   - Roles of different words within the sentence.
3. **Late Attention**: integrates the lower layers and generates coherent and contextual outputs.
   - High-level abstractions.
   - Discourse structure.
   - Sentiment.
   - Complex long-range dependencies.

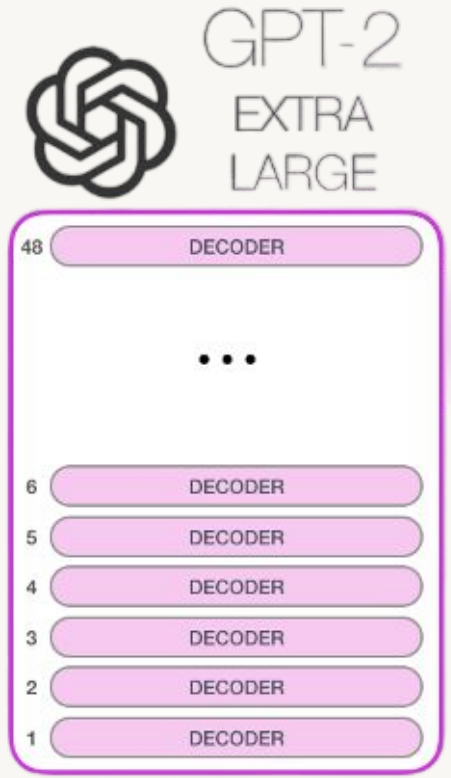


Source: Wikipedia

Why so many parameters?

Parameter	GPT-2 Small	GPT-2 Extra Large	Difference Extra Large / Small
Layers	12	48	3x
Model Dimensionality	768	1600	2x
Attention Heads per Layer	12	25	2.5x
Total	**117 Million**	**1.5 Billion**	**12.8x**

117M parameters



1.5B parameters

Source: IllustratedGPT2

Training GPT

Training Data

The magic behind the model

GPT-1: BooksCorpus

- Over 7,000 unpublished books spanning various genres.
- 800 million words, covering a wide range of topics and styles.

GPT-2: WebText

- Much larger than BookCorpus
- A collection of text from web pages, consists of 45 terabytes of data.
- Filtered out web pages with low quality content

GPT-3: WebText2

- Even larger than WebText and more diverse
- The increased size and diversity of the WebText2 dataset contribute to GPT-3's impressive performance on various NLP tasks.

2.0

Comparing LLM Architectures

BERT vs. GPT vs. T5

Which type of LLM is best?

	Encoder-only (eg. BERT)	Decoder-only (eg. GPT)	Seq-2-Seq (eg. T5)
Pros	• Pre-trained on a masked language modeling task bidirectional context for a deeper understanding of input sequences. Leads to strong feature extraction capabilities. • Typically several orders of magnitude smaller than decoder or seq-2-seq model.	• Autoregressive nature makes them well-suited for text generation tasks. • Can generate coherent and contextually relevant text.	• Encoder-decoder architecture allows for better handling of complex, structured input-output relationships. • Attention mechanisms help weigh the importance of different parts of the input when generating the output.
Cons	• Not ideal for text generation tasks due to their bidirectional nature. • Can have higher computational costs compared to decoder-only models.	• Only capture unidirectional context (left-to-right), which can limit their contextual understanding. • Autoregressive generation can be slow due to sequential token prediction.	• Can require more training data and computational resources compared to encoder-only or decoder-only models. • Can be more complex to train and fine-tune due to the two-part architecture.

Module Summary

Transformers - What have we learned

- Transformers are built from a number of transformer blocks
- Transformer blocks use attention and neural networks to enrich vectors
- Transformer models can be encoder, decoder, or encoder-decoder
- The evolution of GPT required changes in architecture, and in data
- Base/Foundation models require fine tuning to solve most tasks

Time for some code!