Transformer

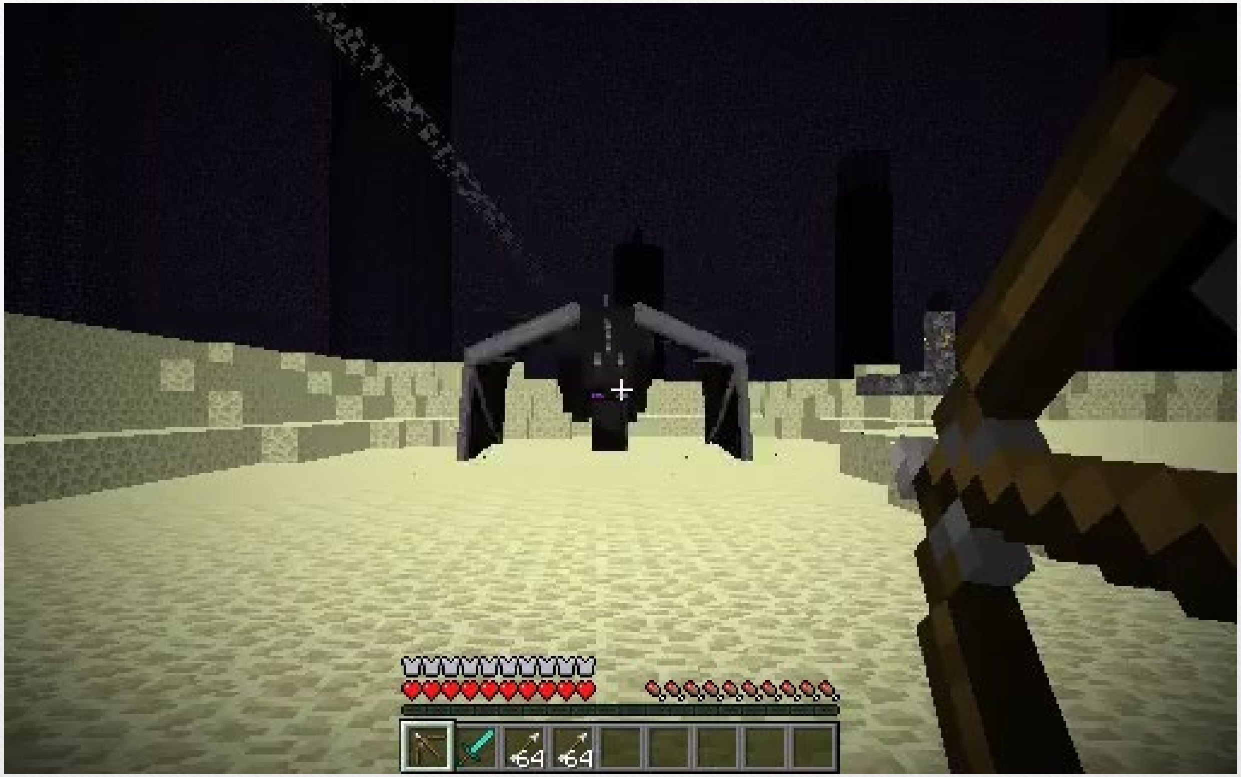
Almost as if it's intelligent.

State-of-the-Art AI Dec. 2022

Stable Diffusion 2: Image generation by description.
https://stability.ai/blog/stable-diffusion-v2-release

State-of-the-Art AI Dec. 2022

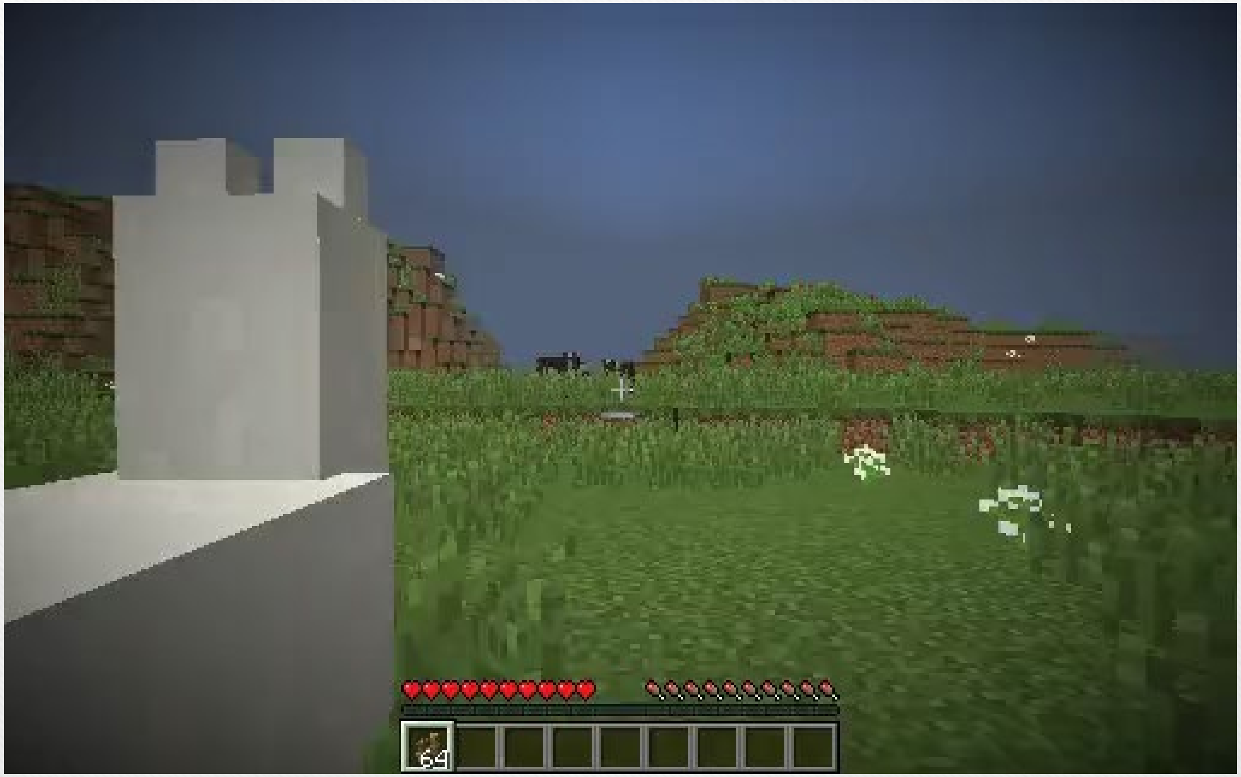
MineDojo: Tell AI Minecraft agent what to do in natural language. https://minedojo.org/



“Fight against an Ender dragon”

State-of-the-Art AI Dec. 2022

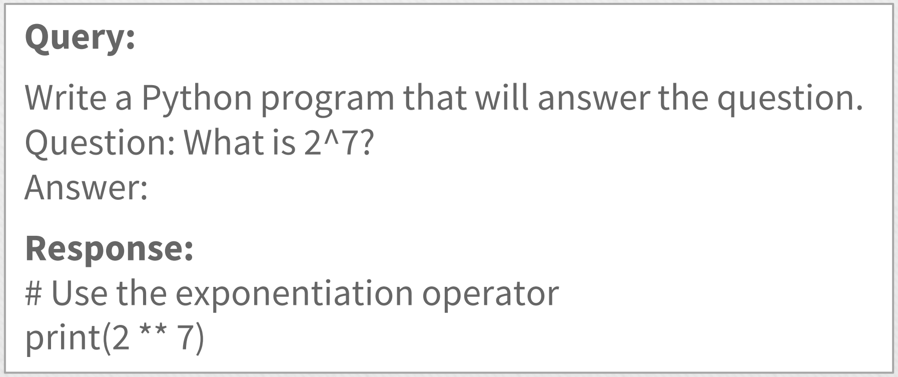
MineDojo: Tell AI Minecraft agent what to do in natural language. https://minedojo.org/



"Encircle Llama with fences"

State-of-the-Art AI Dec. 2022

GPT-3 text-davinci-003: AI writing: code, poems, you name it, it can do it—to an extend. https://beta.openai.com/docs/models/gpt-3

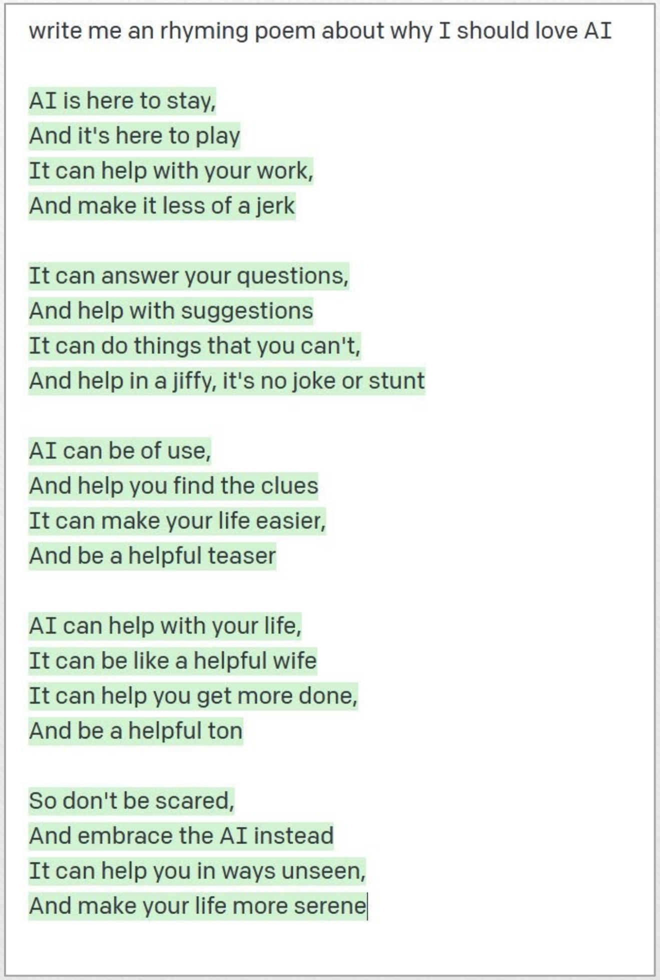


Query:

Write a Python program that will answer the question.
Question: What is 2^7?
Answer:

Response:

```
# Use the exponentiation operator
print(2 ** 7)
```



write me an rhyming poem about why I should love AI

AI is here to stay,
And it's here to play
It can help with your work,
And make it less of a jerk

It can answer your questions,
And help with suggestions
It can do things that you can't,
And help in a jiffy, it's no joke or stunt

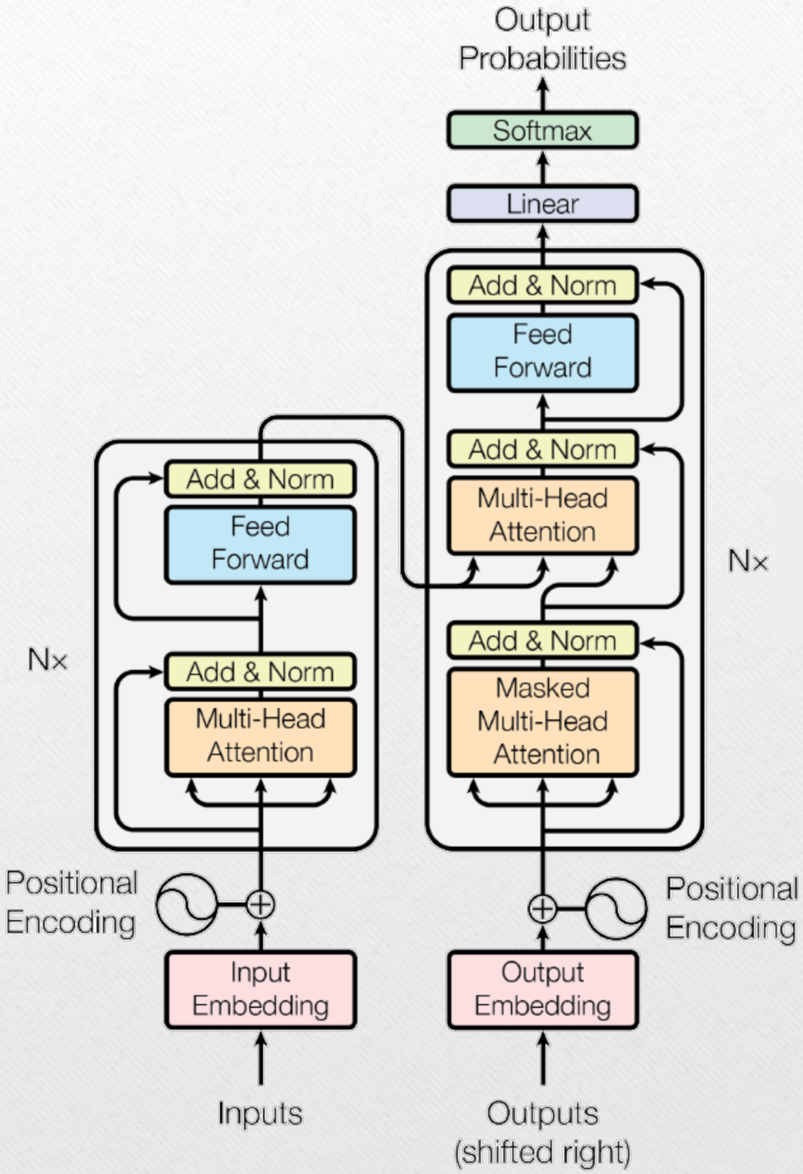
AI can be of use,
And help you find the clues
It can make your life easier,
And be a helpful teaser

AI can help with your life,
It can be like a helpful wife
It can help you get more done,
And be a helpful ton

So don't be scared,
And embrace the AI instead
It can help you in ways unseen,
And make your life more serene

Transformer

- Transformer-based models such as GPT and BERT are the current state of the art in field of natural language processing.
- Original paper: All You Need is Attention (NIPS 2017)
- Transformer is the basis of some of the most advanced AI currently in existence, including the Starcraft-playing AlphaStar and text-generating GPT-3 and image-generating Stable Diffusion.

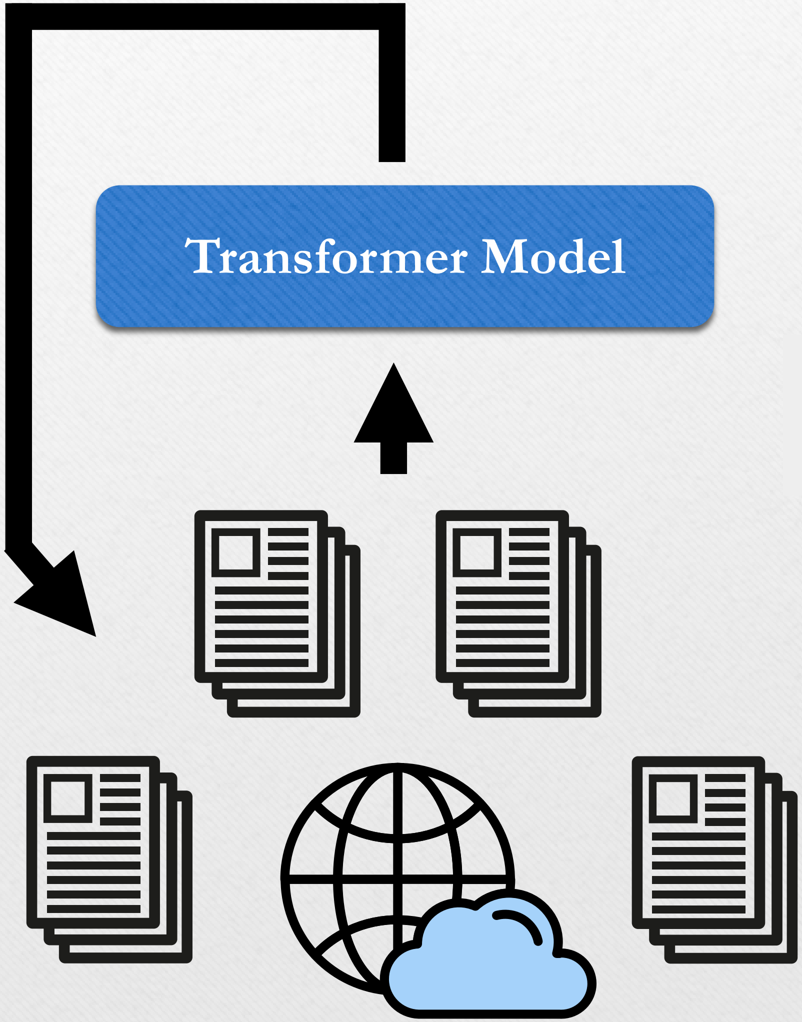


Transformer

Transformer-based models are powerful because they can adapt to many tasks without the need to train from scratch.

This works because training is split into two stages:

1. The model is **pre-trained** with a large corpus of text in an unsupervised manner, allowing it to develop an understanding of language. This stage is computationally expensive.

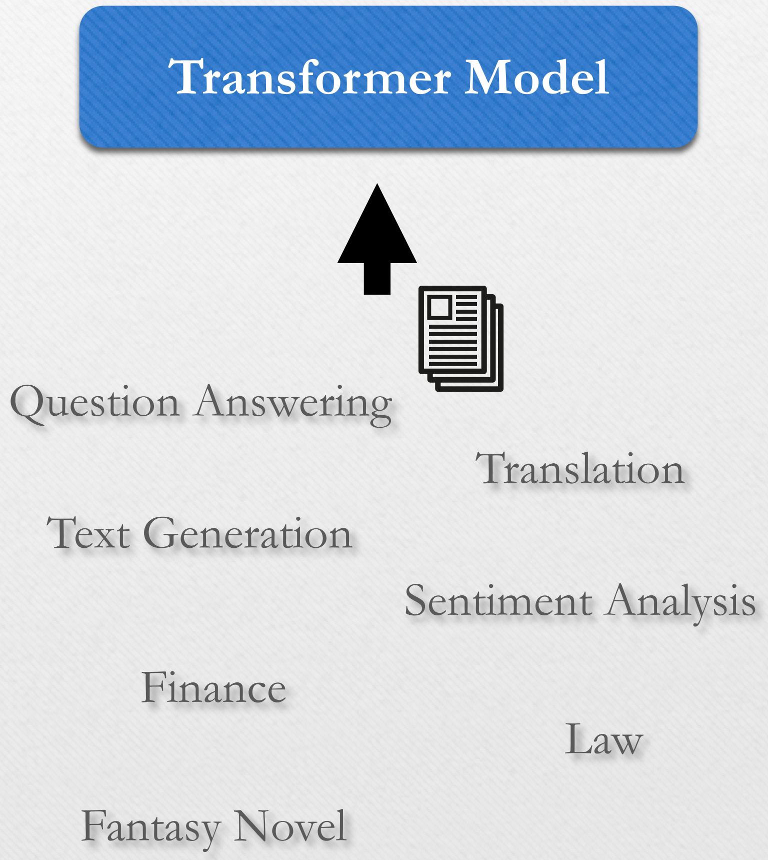


Transformer

Transformer-based models are powerful because they can adapt to many tasks without the need to train from scratch.

This works because training is split into two stages:

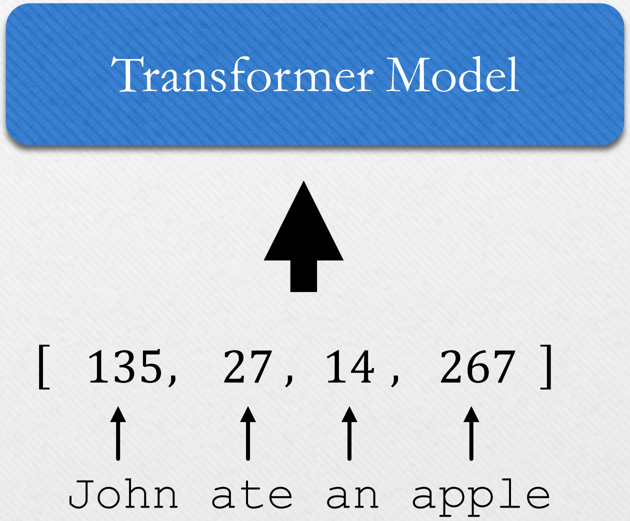
1. The model is **pre-trained** with a large corpus of text in an unsupervised manner, allowing it to develop an understanding of language. This stage is computationally expensive.
2. The model is then **fine-tuned** to a specific task. This stage can be done with relatively little data and computational power.



Input and Output

Transformer is a **sequence-to-sequence** model, meaning that the input and output are both vectors.

The input is a vector with each number representing one word.



Transformer Model

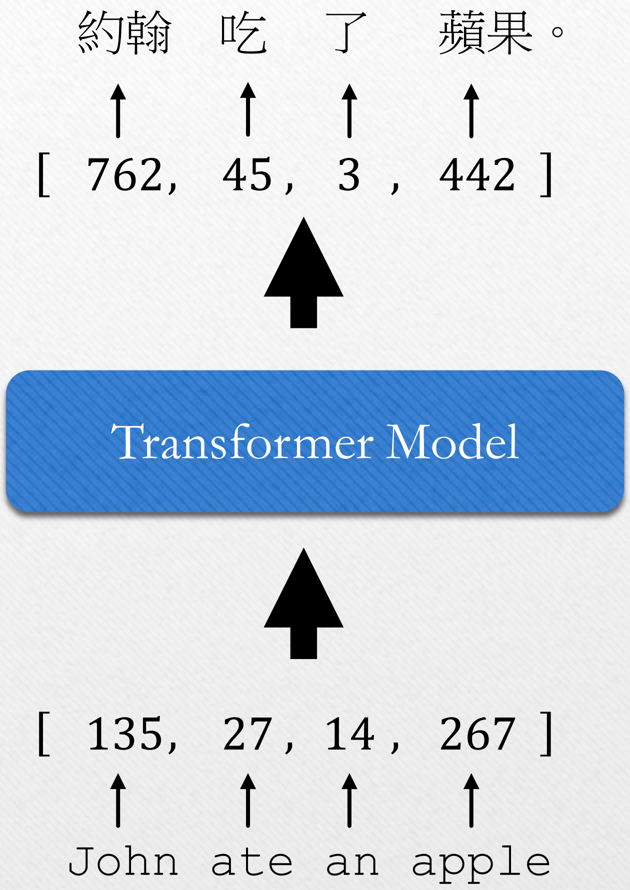
[135, 27 , 14 , 267]

John ate an apple

Input and Output

The output can be:

- **Translation:** the model predicts words in another language. This was what the original paper did.



約翰 吃 了 蘋果。

[762, 45, 3 , 442]

Transformer Model

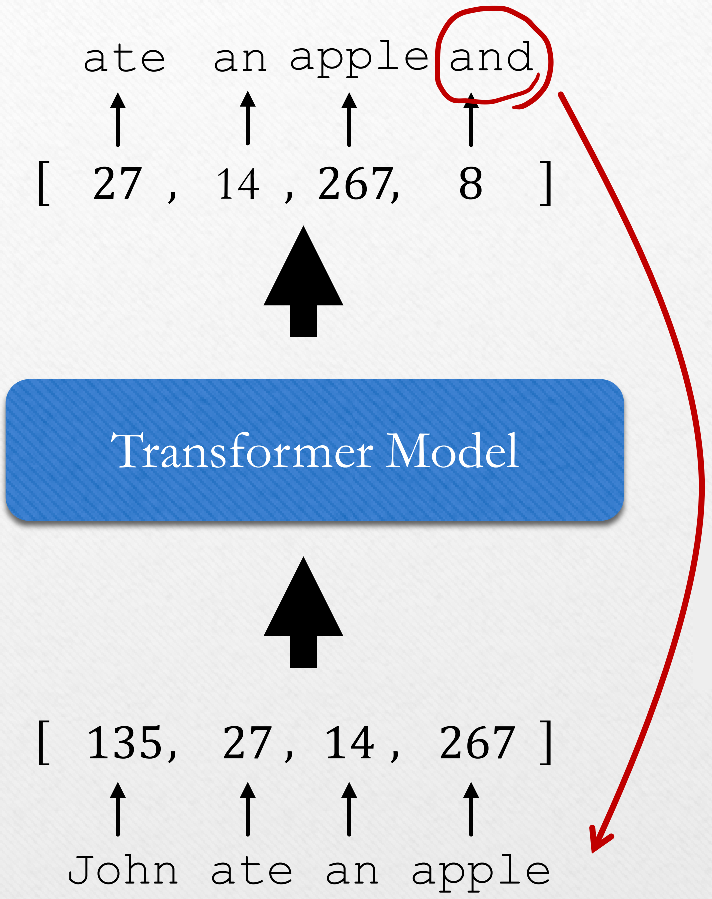
[135, 27, 14, 267]

John ate an apple

Input and Output

The output can be:

- **Translation**: the model predicts words in another language. This was what the original paper did.
- **The next word**: the model predicts the next word after the input. This is how GPT models are pre-trained.

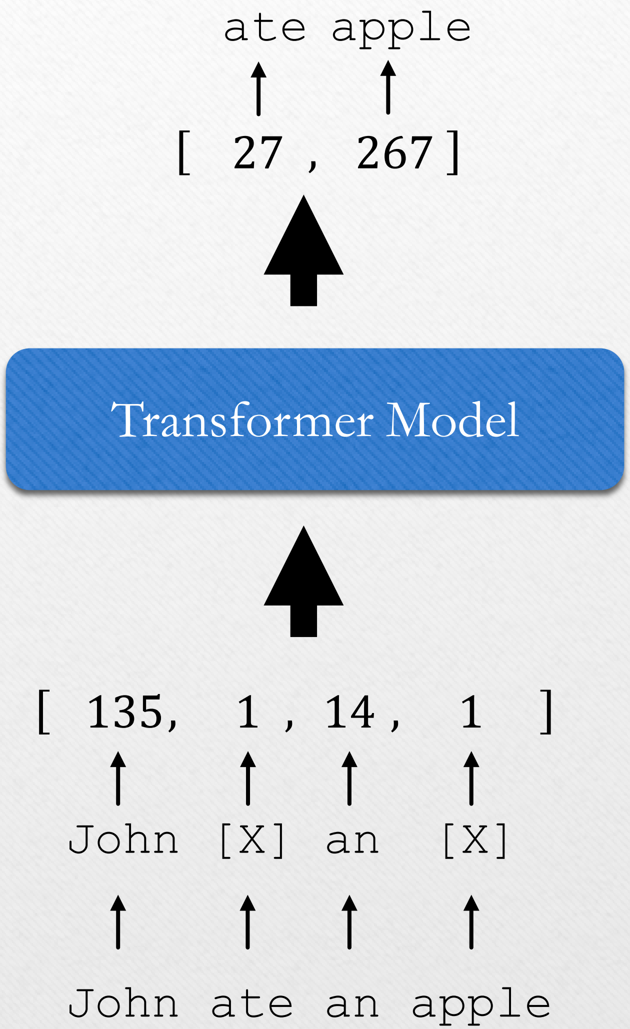


Input and Output

The output can be:

- **Translation**: the model predicts words in another language. This was what the original paper did.
- **The next word**: the model predicts the next word after the input. This is how GPT models are pre-trained.
- **Fill-in-the-blanks/denoising**: the model predicts missing words in a passage from which the input was taken. This is how T5 and BERT models are pre-trained.

In the latter two cases, the model can be pre-trained unsupervised, allowing for the use of extremely large datasets.



ate apple

[27 , 267]

Transformer Model

[135, 1 , 14 , 1]

John [X] an [X]

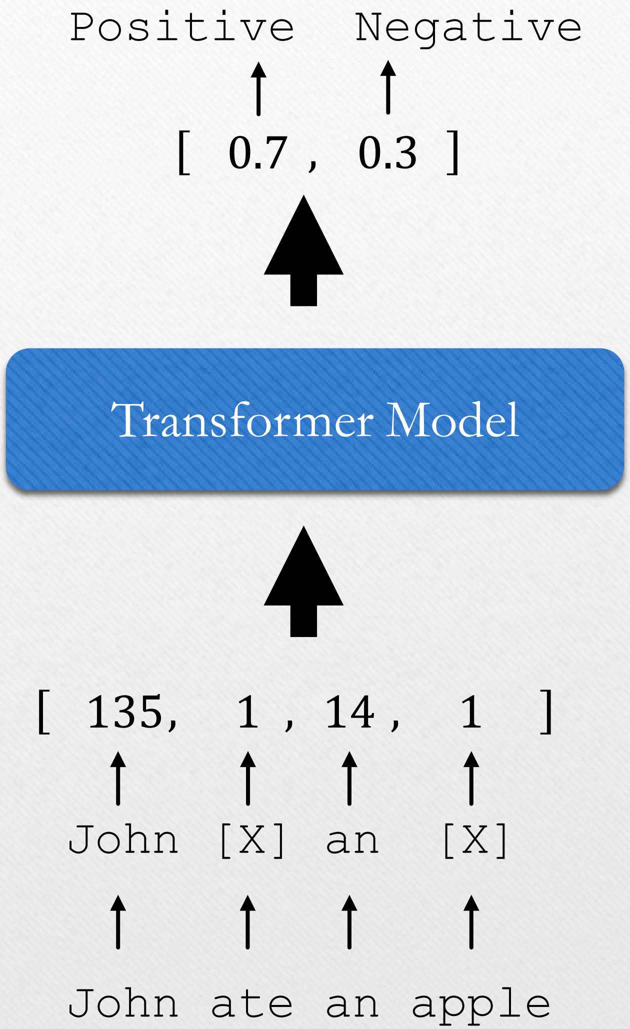
John ate an apple

Input and Output

With fine-tuning, the model can adapt to tasks that it was not pre-trained for:

- Sentiment analysis
- Text classification
- Question answering

Fine-tuning typically only takes a few epochs to achieve good results, because the model already understand language from pre-training.



Try it Out

- Training large Transformer-based models from scratch is very expensive, so in practice you will need to use pretrained models. These models can be fine-tuned if necessary.
- **GPT**: OpenAI provides an API to its GPT models, with different payment tiers. See some applications here. Try in particular AI Dungeon.
- The creators of the various other models usually provide pretrained models for download.
- The easiest service to use is probably Hugging Face.

How Does Transformer Work?

Not necessary if you just
want to use pre-trained models!

Why Transformer?

```
John ate an apple. It was delicious.

John ate an apple. He likes it.
```

- Consider the following above passages.

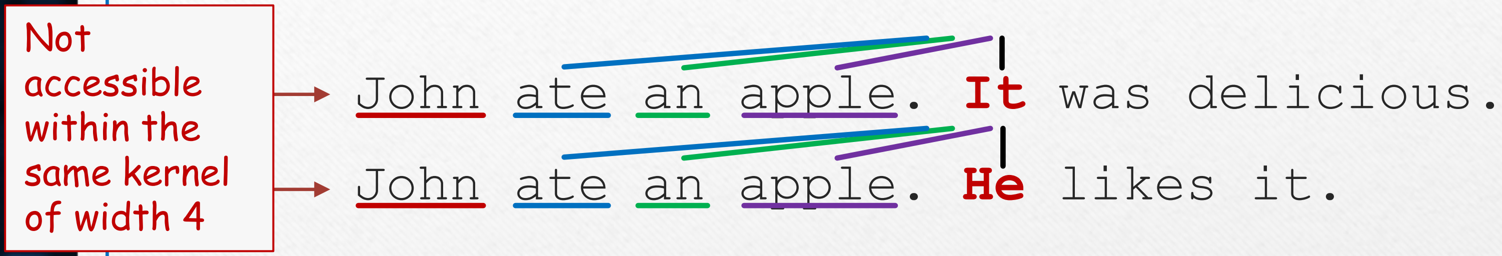
Why Transformer?

$\underline{\texttt{John ate an apple.}}\,^{c_t}$ **It** was delicious.

$\underline{\texttt{John ate an apple.}}\,^{c_t}$ **He** likes it.

- How does a neural network understand the highlighted words?
- **RNN:** since the highlighted words follow the exact same words, they face the same previous state.
- The model has no idea that "It" refers to "apple" while "He" refers to "John".

Why Transformer?



Not accessible within the same kernel of width 4

John ate an apple. **It** was delicious.

John ate an apple. **He** likes it.

- **CNN:** same position, same weights. In principle the highlighted word can make a difference just like pixels of different colors, but how often does one pixel make a difference?
- The defining feature of CNN—neurons are only connected to a neighborhood of outputs—means it is hard to learn the relationship between distant words.

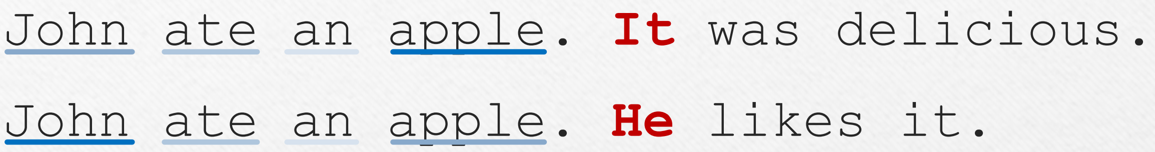
Why Transformer?

John ate an apple. **It** was delicious.

John ate an apple. **He** likes it.

- We want the model to understand that “It” refers to “apple” while “He” refers to “John”.
- RNN and CNN do not work well in this regard, as they do not have a good mechanism to understand context *given* a particular word.

Self-Attention



John ate an apple. **It** was delicious.

John ate an apple. **He** likes it.

- The most important idea of a Transformer is **self-attention**.
- When the model reads a particular word, the output depends on *all* words, weighed by how important they are given the word in concern.
- We will now go through what a **self-attention layer** does.

```
John ate an apple. It was delicious.
```

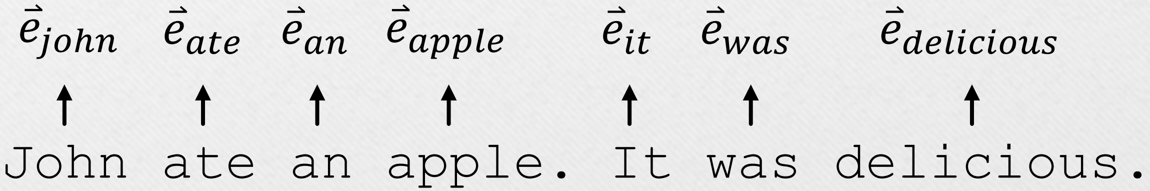
This is one input sample.

```
[start] John ate an apple. It was delicious. [end]
```

During preprocessing, special characters are added to the input and target text phrases. These characters denote the start, the end and other contextual information and are treated just like another any other word.

```
[start] John ate an apple. It was delicious. [end]
```

For space reasons I will omit them in latter slides.



$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$

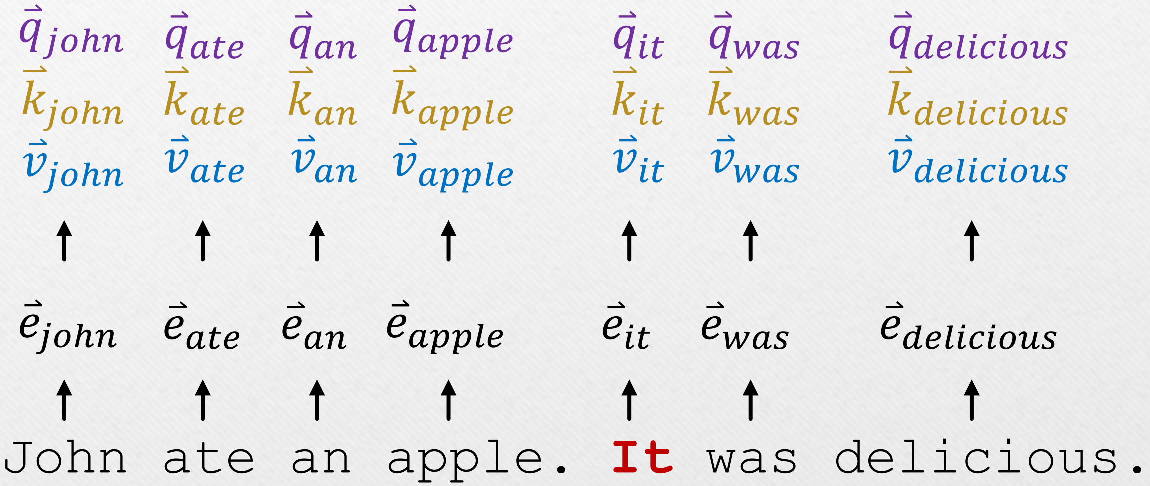
`John ate an apple. It was delicious.`

As in all NLP neural network models, we start with assigning each word an embedding—a vector learnt through training to represent the nature of each word.

$$\begin{array}{rcccc} \text{e.g.} & \text{Noun} & \text{Verb} & \text{Male} & \\ & \searrow & \downarrow & \swarrow & \\ \vec{e}_{john} = [& 1 & 0 & 1 & \dots] \\ \vec{e}_{ate} = [& 0 & 1 & 0 & \dots] \\ \vec{e}_{he} = [& 0 & 0 & 1 & \dots] \end{array}$$

$$\begin{array}{ccccccc} \vec{e}_{john} & \vec{e}_{ate} & \vec{e}_{an} & \vec{e}_{apple} & \vec{e}_{it} & \vec{e}_{was} & \vec{e}_{delicious} \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \texttt{John} & \texttt{ate} & \texttt{an} & \texttt{apple.} & \texttt{It} & \texttt{was} & \texttt{delicious.} \end{array}$$

As in all NLP neural network models, we start with assigning each word an embedding—a vector learnt through training to represent the nature of each word.



$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

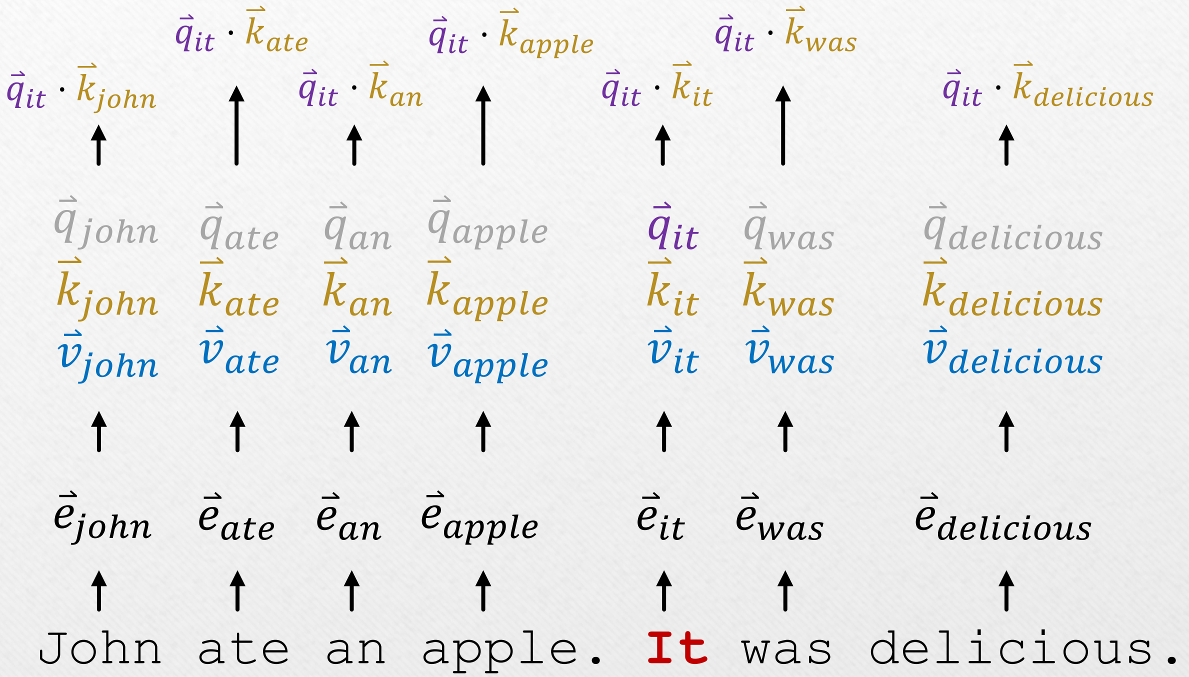
$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$

John ate an apple. **It** was delicious.

The attention mechanism requires three vectors as inputs: **query**, **key** and **value**. In the simplest case, they are simply the output from the previous layer.



$\vec{q}_{it} \cdot \vec{k}_{ate}$ $\vec{q}_{it} \cdot \vec{k}_{apple}$ $\vec{q}_{it} \cdot \vec{k}_{was}$

$\vec{q}_{it} \cdot \vec{k}_{john}$ $\vec{q}_{it} \cdot \vec{k}_{an}$ $\vec{q}_{it} \cdot \vec{k}_{it}$ $\vec{q}_{it} \cdot \vec{k}_{delicious}$

$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

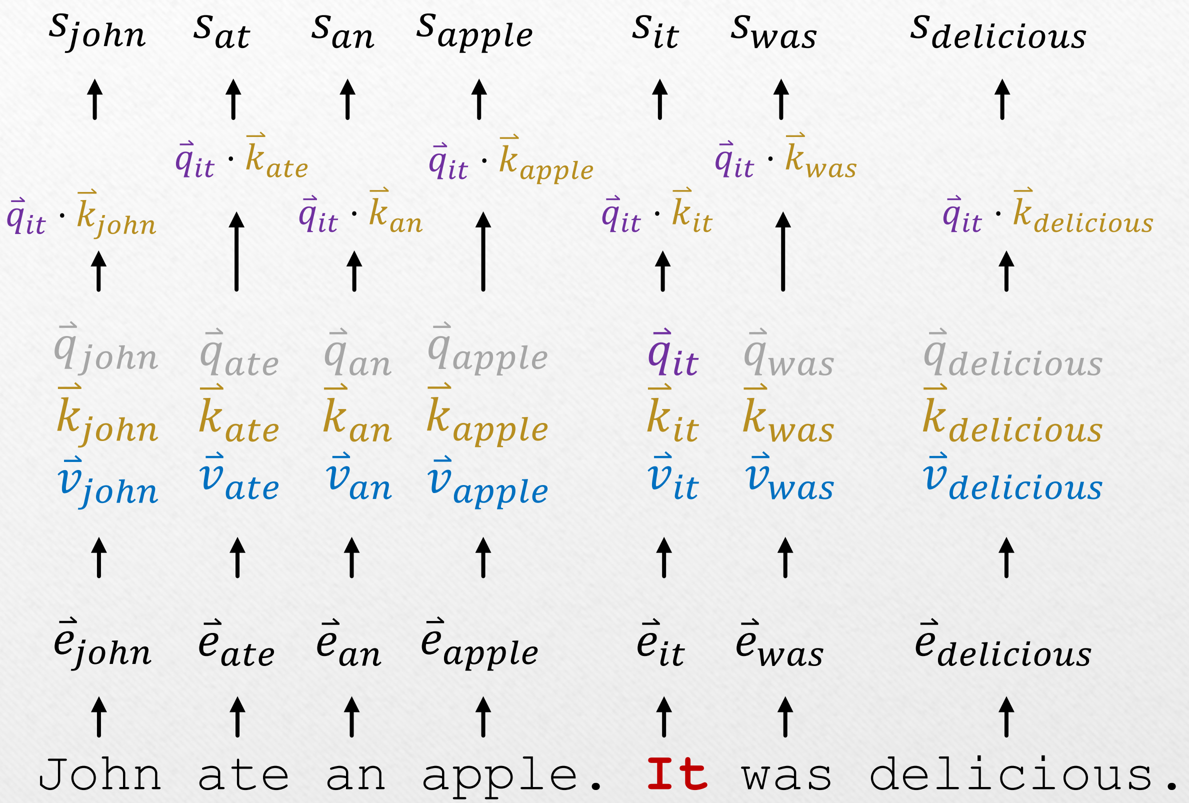
$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$

John ate an apple. **It** was delicious.

When we process a word *i*, we compute a **score** for *all* words by taking the dot product of *i*'s query and each word's key.



s_{john} s_{at} s_{an} s_{apple} s_{it} s_{was} $s_{delicious}$

$\vec{q}_{it} \cdot \vec{k}_{ate}$ $\vec{q}_{it} \cdot \vec{k}_{apple}$ $\vec{q}_{it} \cdot \vec{k}_{was}$

$\vec{q}_{it} \cdot \vec{k}_{john}$ $\vec{q}_{it} \cdot \vec{k}_{an}$ $\vec{q}_{it} \cdot \vec{k}_{it}$ $\vec{q}_{it} \cdot \vec{k}_{delicious}$

$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

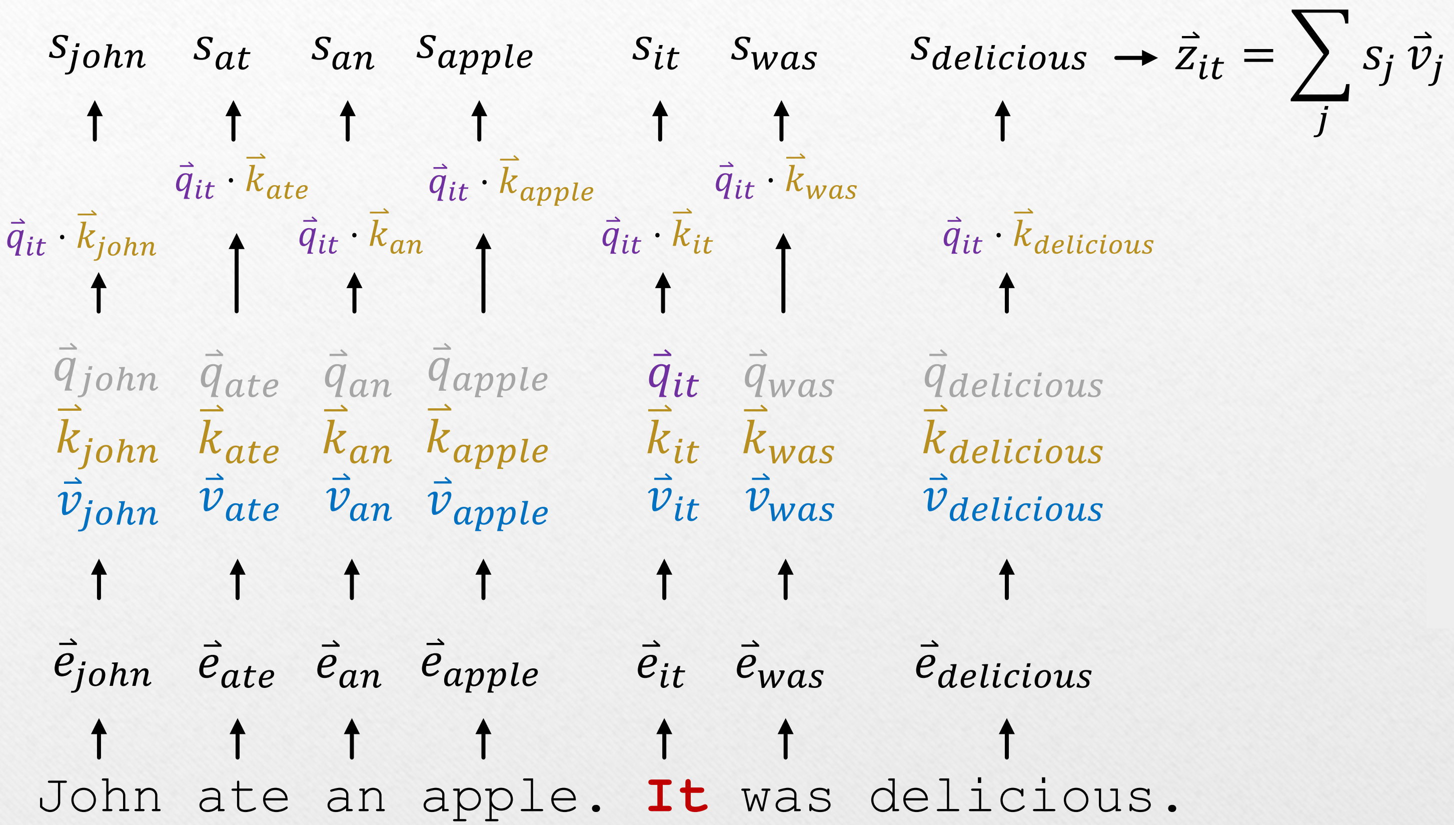
$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$

John ate an apple. **It** was delicious.

A weight is assigned to each word by feeding the score to a softmax function, i.e. multinomial logit.



s_{john} s_{at} s_{an} s_{apple} s_{it} s_{was} $s_{delicious}$ → $\vec{z}_{it} = \sum_j s_j \, \vec{v}_j$

$\vec{q}_{it} \cdot \vec{k}_{john}$ $\vec{q}_{it} \cdot \vec{k}_{ate}$ $\vec{q}_{it} \cdot \vec{k}_{an}$ $\vec{q}_{it} \cdot \vec{k}_{apple}$ $\vec{q}_{it} \cdot \vec{k}_{it}$ $\vec{q}_{it} \cdot \vec{k}_{was}$ $\vec{q}_{it} \cdot \vec{k}_{delicious}$

$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$

John ate an apple. **It** was delicious.

The output z_i of the attention mechanism for word i is the sum of the multiple of each word's weight with its value.

Why Does Self-Attention Make Sense?

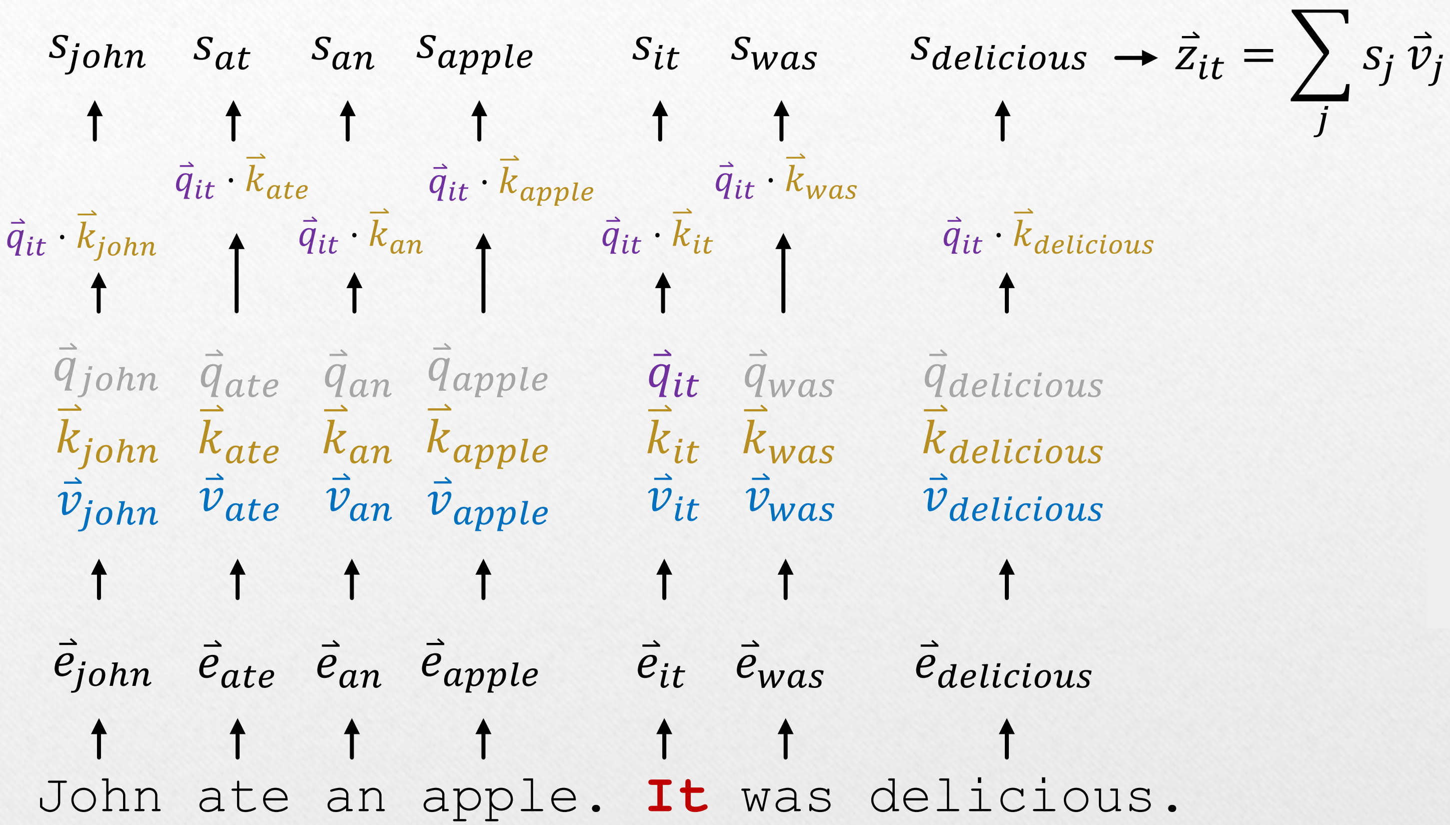
- Reconsider our example:

$$\vec{h}_{john} = [1 \quad 0 \quad 1]$$
$$\vec{h}_{ate} = [0 \quad 1 \quad 0]$$
$$\vec{h}_{he} = [0 \quad 0 \quad 1]$$

- When processing the word "he", the attention mechanism takes the dot product of the vector representations of "he" and each word:

$$s_{ate} = \vec{h}_{he} \cdot \vec{h}_{ate} = 0$$
$$s_{john} = \vec{h}_{he} \cdot \vec{h}_{john} = 1$$

- Words with similar vector representations get a higher weight. This allows the model to "pay attention" to relevant words.



s_{john} s_{at} s_{an} s_{apple} s_{it} s_{was} $s_{delicious}$ $\rightarrow \vec{z}_{it} = \sum_j s_j \, \vec{v}_j$

$\vec{q}_{it} \cdot \vec{k}_{john}$ $\vec{q}_{it} \cdot \vec{k}_{ate}$ $\vec{q}_{it} \cdot \vec{k}_{an}$ $\vec{q}_{it} \cdot \vec{k}_{apple}$ $\vec{q}_{it} \cdot \vec{k}_{it}$ $\vec{q}_{it} \cdot \vec{k}_{was}$ $\vec{q}_{it} \cdot \vec{k}_{delicious}$

$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

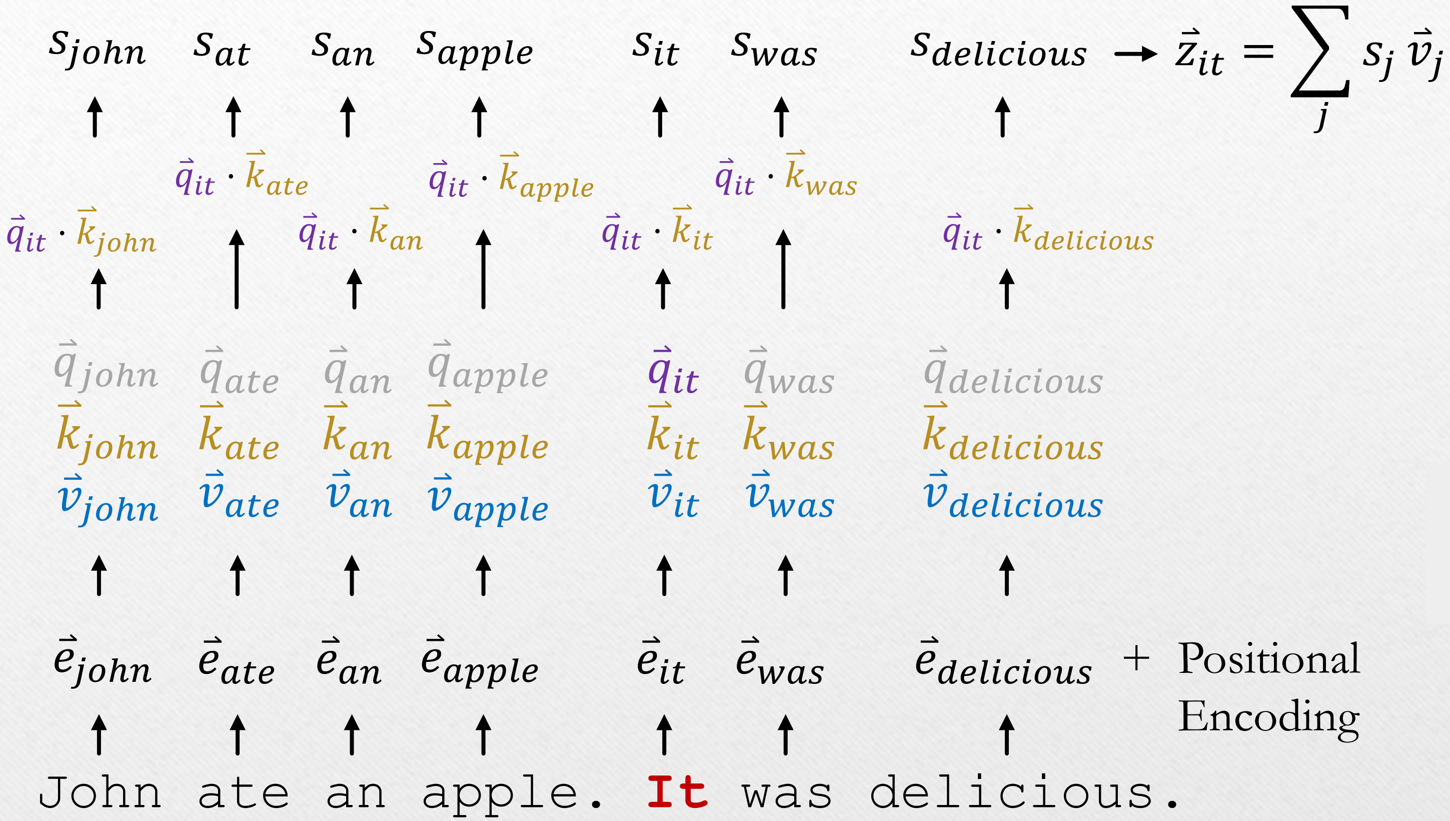
$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$

John ate an apple. **It** was delicious.

As self-attention is applied to all words, the mechanism is bidirectional—or more accurately, non-directional. It makes no distinction between words in different position.



s_{john} s_{at} s_{an} s_{apple} s_{it} s_{was} $s_{delicious}$ → $\vec{z}_{it} = \sum_j s_j \, \vec{v}_j$

$\vec{q}_{it} \cdot \vec{k}_{ate}$ $\vec{q}_{it} \cdot \vec{k}_{apple}$ $\vec{q}_{it} \cdot \vec{k}_{was}$

$\vec{q}_{it} \cdot \vec{k}_{john}$ $\vec{q}_{it} \cdot \vec{k}_{an}$ $\vec{q}_{it} \cdot \vec{k}_{it}$ $\vec{q}_{it} \cdot \vec{k}_{delicious}$

$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

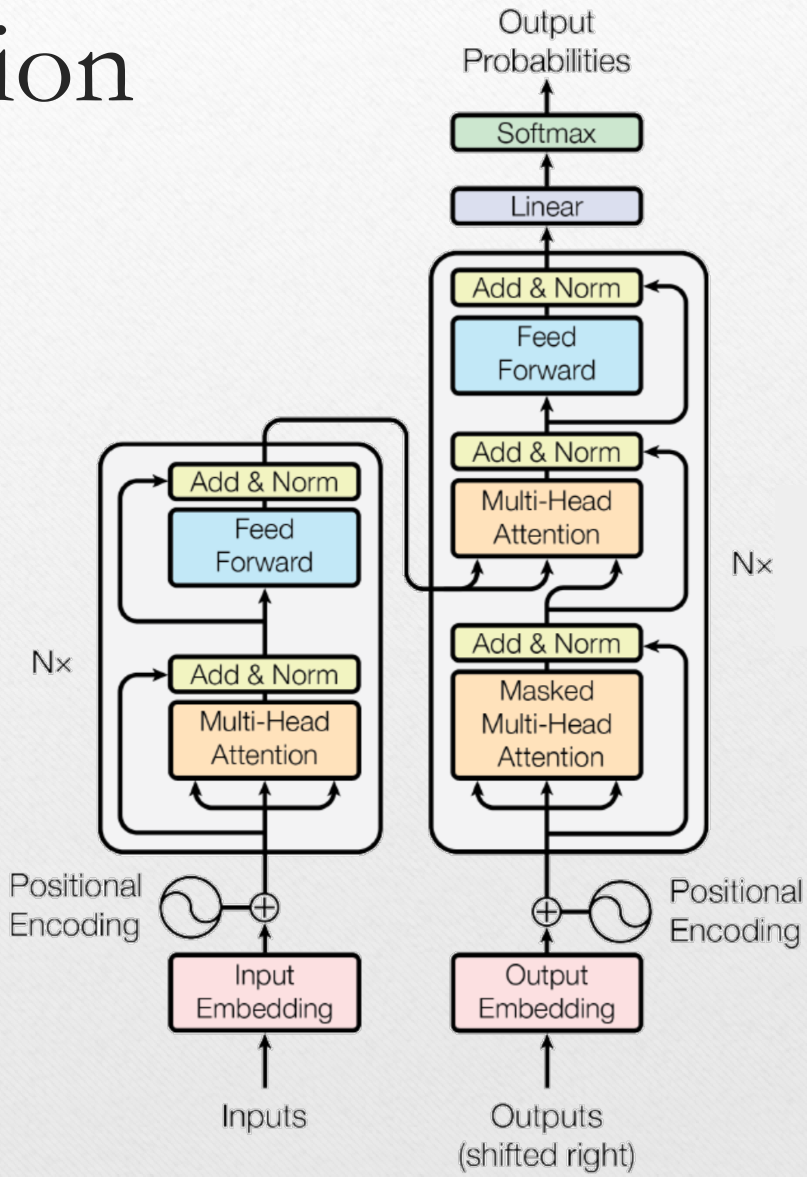
$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$ + Positional Encoding

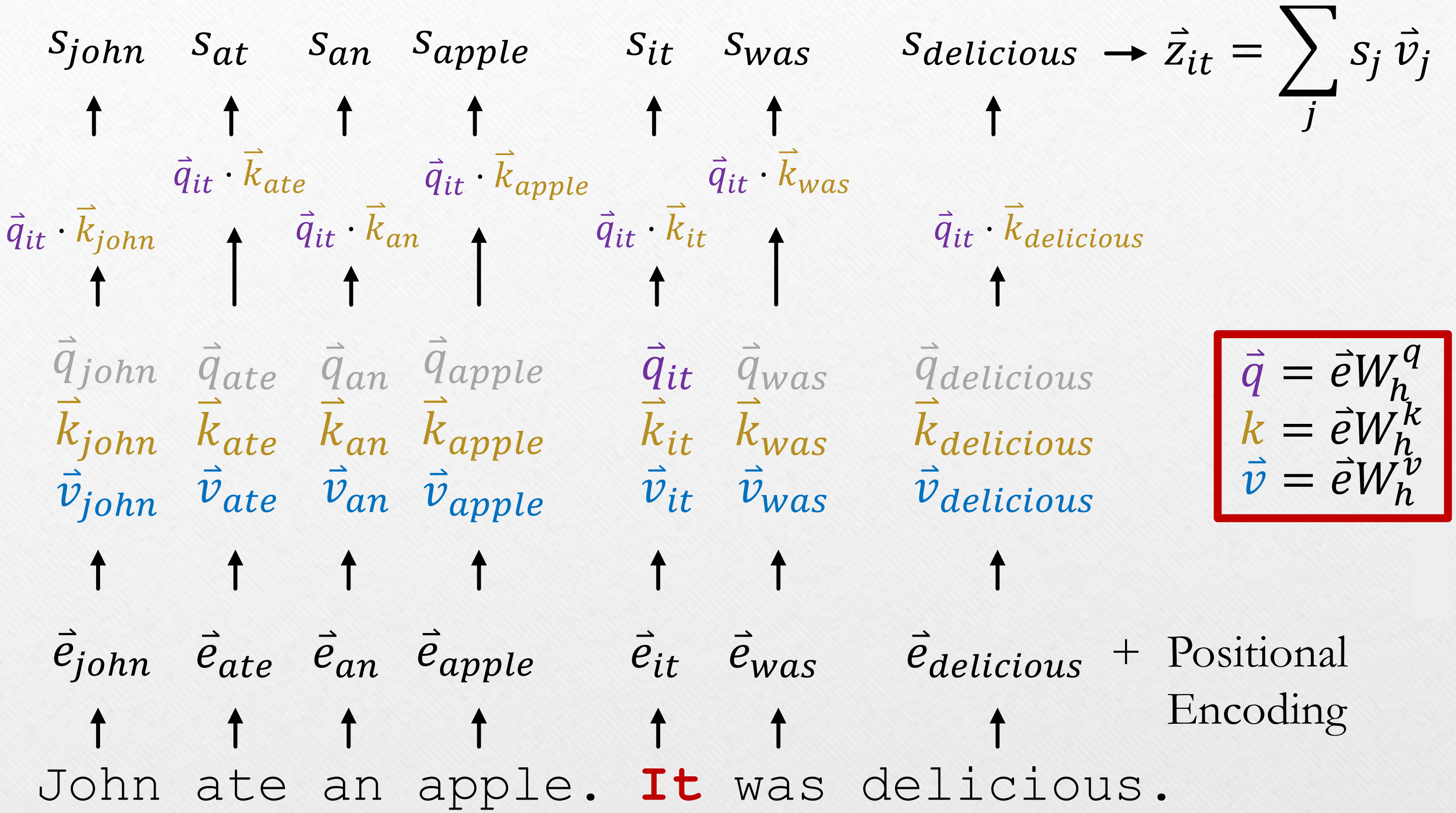
John ate an apple. **It** was delicious.

To provide the model with word positions, a **positional encoding** is added to the usual word embedding. The positional encoding can be fixed or learnt.

Multi-Head Attention

- What we just saw is **single-head (self-)attention**.
- **Multi-head attention** simply means we have multiple copies of the process, each with different weights.





s_{john} s_{at} s_{an} s_{apple} s_{it} s_{was} $s_{delicious}$ → $\vec{z}_{it} = \sum_j s_j \, \vec{v}_j$

$\vec{q}_{it} \cdot \vec{k}_{john}$ $\vec{q}_{it} \cdot \vec{k}_{ate}$ $\vec{q}_{it} \cdot \vec{k}_{an}$ $\vec{q}_{it} \cdot \vec{k}_{apple}$ $\vec{q}_{it} \cdot \vec{k}_{it}$ $\vec{q}_{it} \cdot \vec{k}_{was}$ $\vec{q}_{it} \cdot \vec{k}_{delicious}$

$\vec{q}_{john}$ $\vec{q}_{ate}$ $\vec{q}_{an}$ $\vec{q}_{apple}$ $\vec{q}_{it}$ $\vec{q}_{was}$ $\vec{q}_{delicious}$

$\vec{k}_{john}$ $\vec{k}_{ate}$ $\vec{k}_{an}$ $\vec{k}_{apple}$ $\vec{k}_{it}$ $\vec{k}_{was}$ $\vec{k}_{delicious}$

$\vec{v}_{john}$ $\vec{v}_{ate}$ $\vec{v}_{an}$ $\vec{v}_{apple}$ $\vec{v}_{it}$ $\vec{v}_{was}$ $\vec{v}_{delicious}$

$$\boxed{\begin{aligned} \vec{q} &= \vec{e}W_h^q \\ \vec{k} &= \vec{e}W_h^k \\ \vec{v} &= \vec{e}W_h^v \end{aligned}}$$

$\vec{e}_{john}$ $\vec{e}_{ate}$ $\vec{e}_{an}$ $\vec{e}_{apple}$ $\vec{e}_{it}$ $\vec{e}_{was}$ $\vec{e}_{delicious}$ + Positional Encoding

John ate an apple. **It** was delicious.

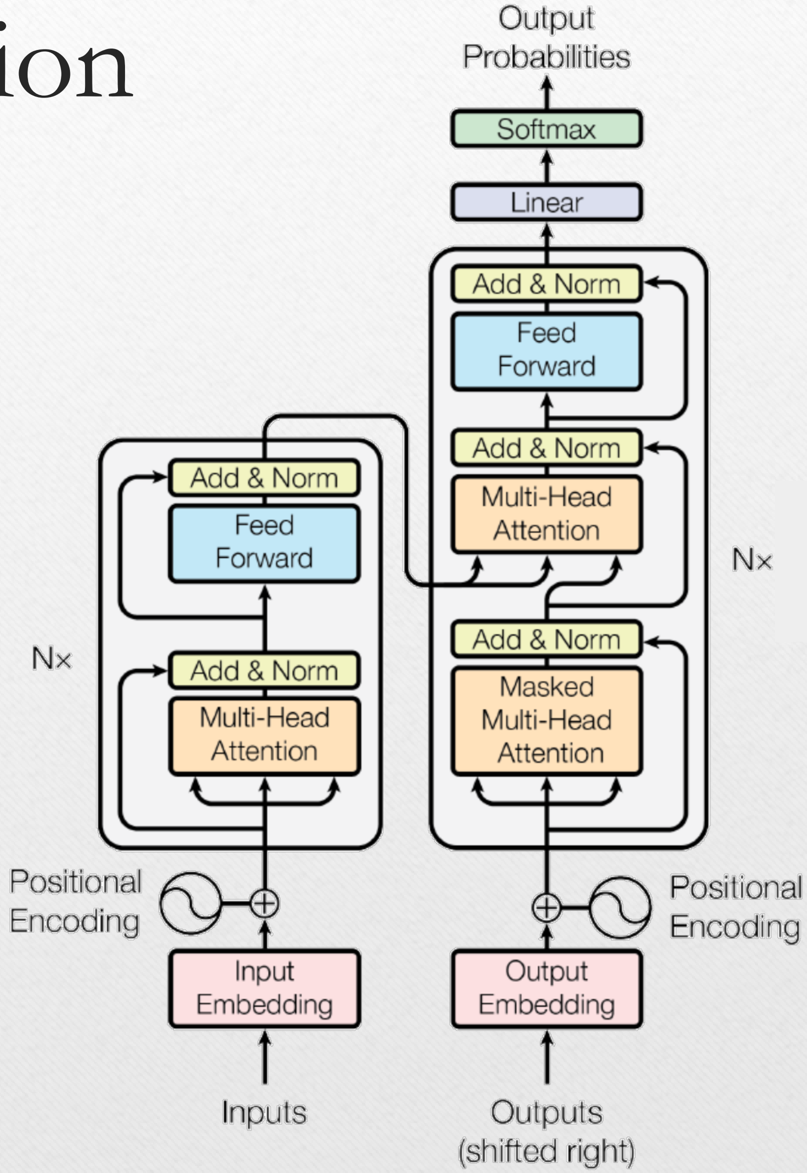
In multi-head attention, query, key and value are projections of the previous layer's output. You can view these as new embeddings capturing specific aspects of a word.

The weights W_h^q, W_h^k and W_h^v are learnt from training.

Multi-Head Attention

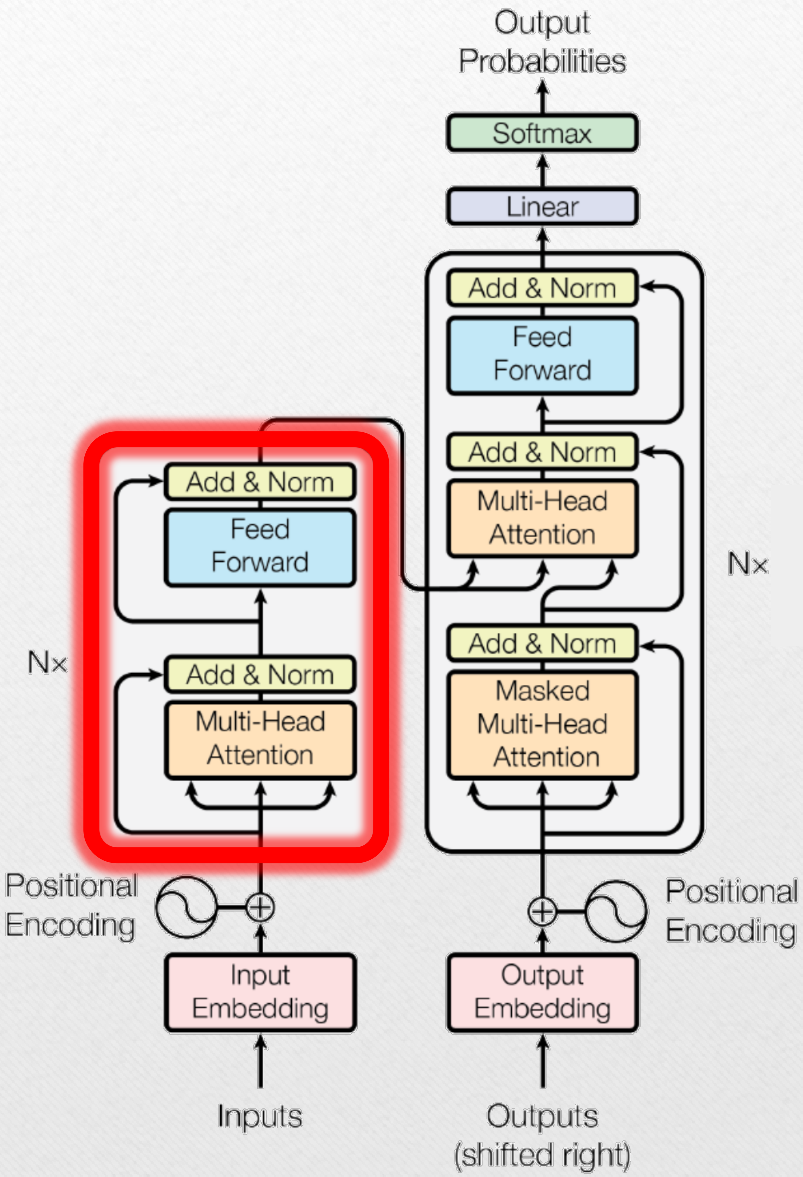
- What we just saw is **single-head (self-)attention**.
- **Multi-head attention** simply means we have multiple copies of the process, each with different weights.
- The weights from the heads are concatenated and multiply to yet another set of weights, generating the final output of the attention layer:

$$\vec{z}_i = [\vec{z}_{1,i}, \dots, \vec{z}_{h,i}]W_o$$



Encoder

- The output from the attention mechanism is then
  - Added to the output of the previous layer (i.e. residual connection)
  - Normalized
  - Fed to a fully-connected network
  - Another residual connection added and normalized
- The whole set of computation is called a **Transformer Encoder Layer**.
- Multiple encoder layers stacked together form the **Transformer Encoder**.
- The final output is a vector for each word, just like RNN.

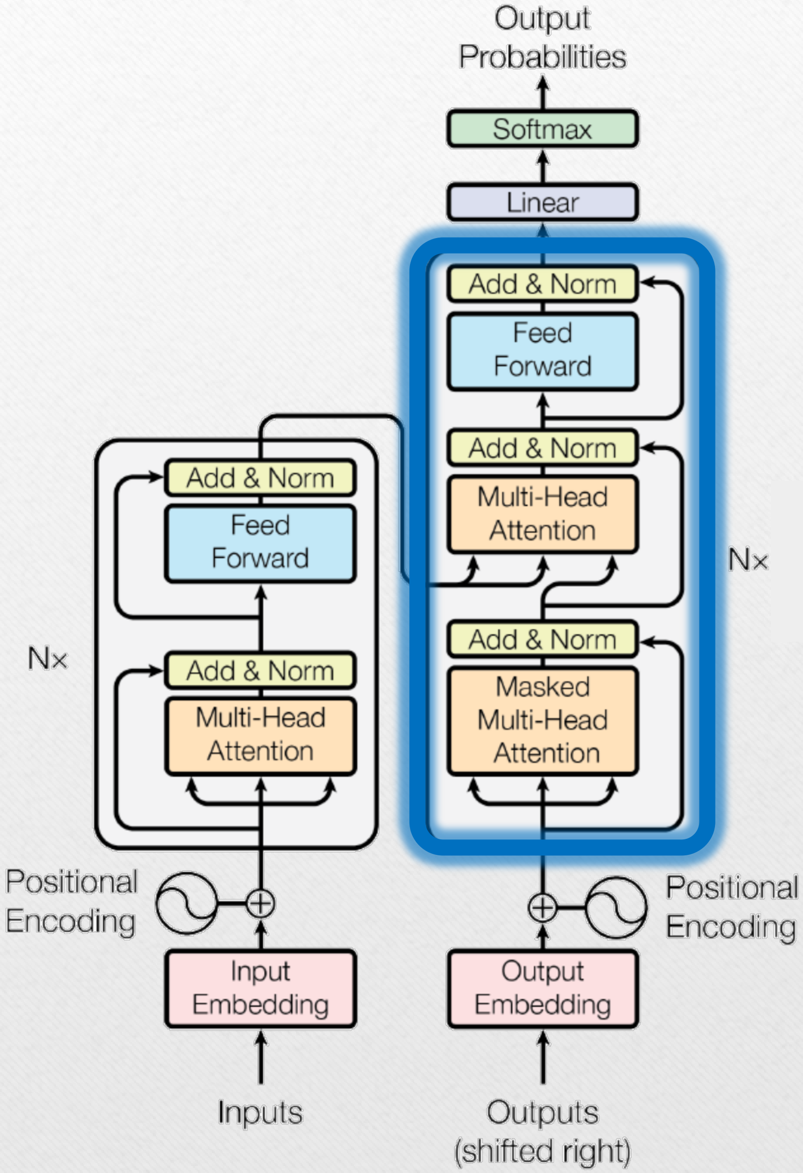


Decoder

The **Transformer Decoder** is responsible for generating a new string of text based on the input.

A **Transformer Decoder Layer** is very similar to the encoder layer, with two crucial differences:

1. The self-attention mechanism is masked. When we process word i, the keys and values of words $> i$ are not used. This is similar to unidirectional RNN.
2. There is a second attention mechanism where the keys and values are the output of the encoder.

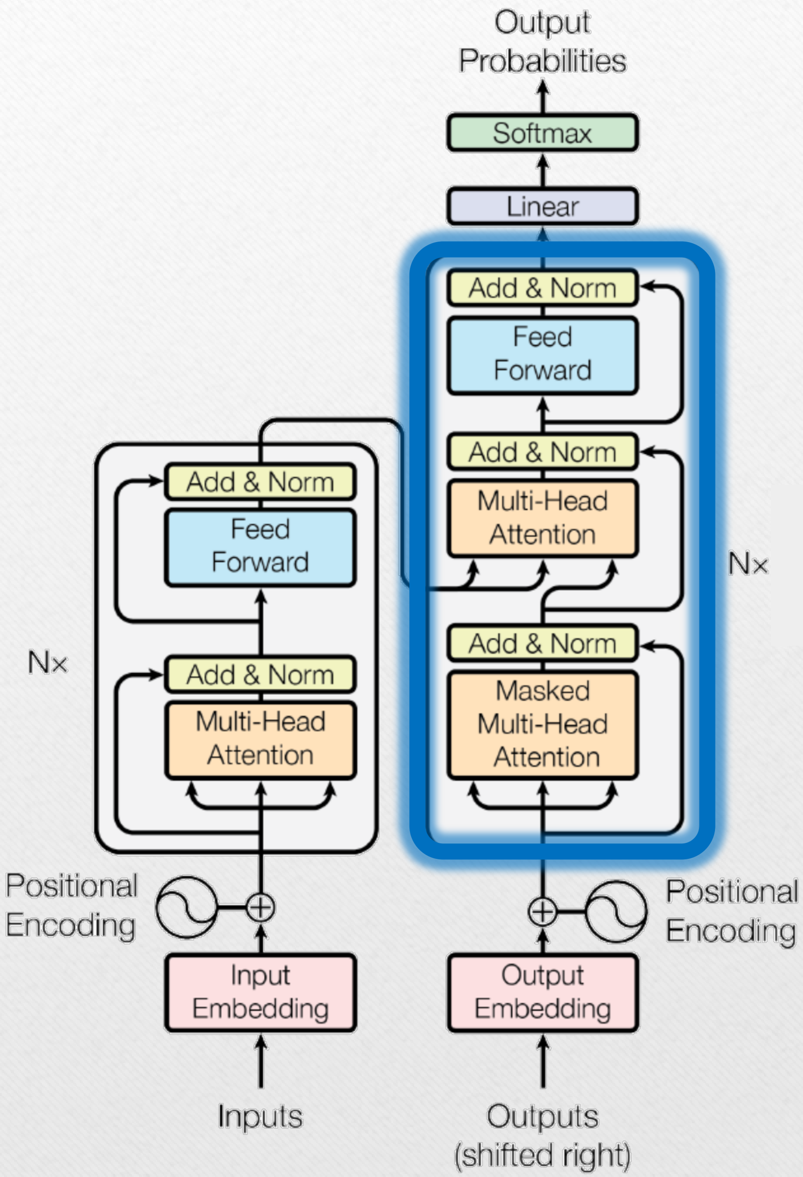


Decoder

The decoder takes a list of words as input. It predicts what the next word should be.

During training, the input to the decoder is simply the desired output shifted right by one word.

- Desired output:
  `John ate an apple [end]`
- Input:
  `[start] John ate an apple`
- When predicting `ate`, the effective input is `[start] John`



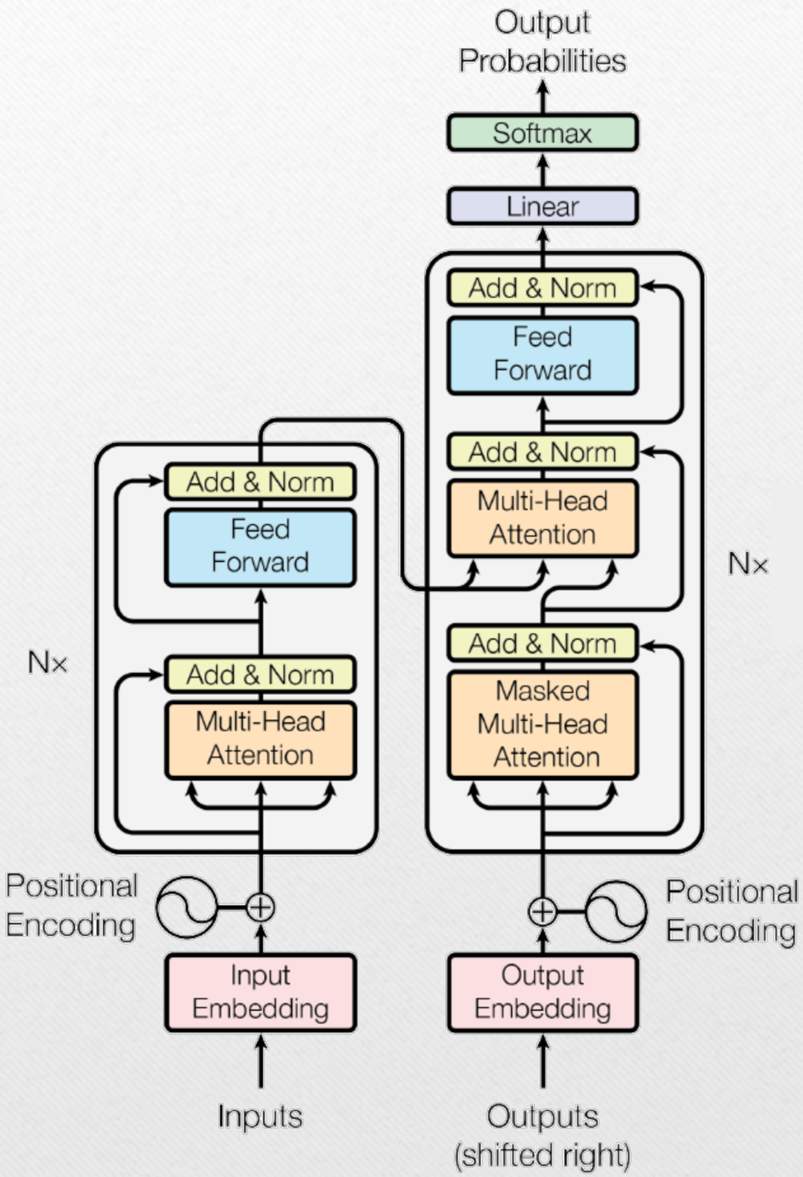
Training

Initial weights are random, just like other neural network models.

For each sample,

1. Provide the encoder with the input text, e.g. a sentence in English.
2. Encoder outputs a vector for each word.
3. Provide the decoder with its input (= desired output shifted right)
4. Decoder takes its input and the encoder's output, generate its own output, e.g. a sentence in French.

Backpropagate errors after each batch.



Decoder

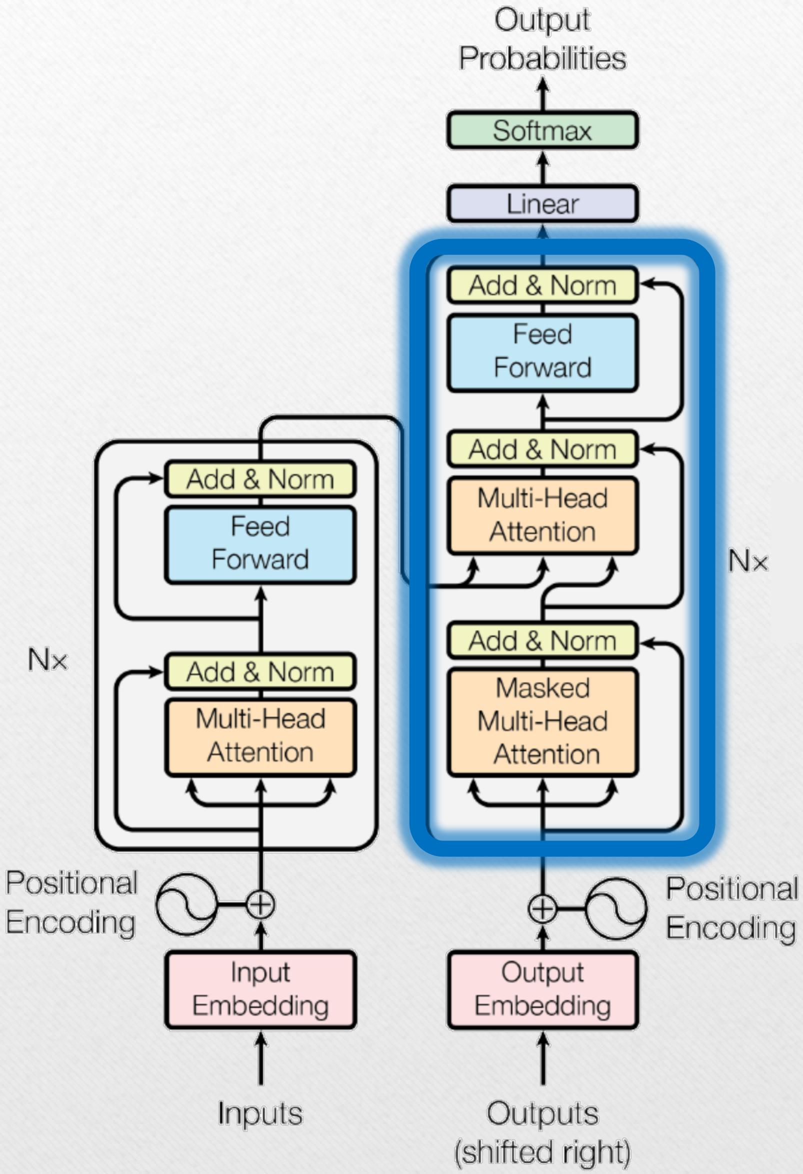
The decoder takes a list of words as input. It predicts what the next word should be.

During inference, after the decoder generates the next word, it will be used as part of the input to generate the word after the next.

```
[start] → ?
```

Say the model predicts `Paul`, so next we have:

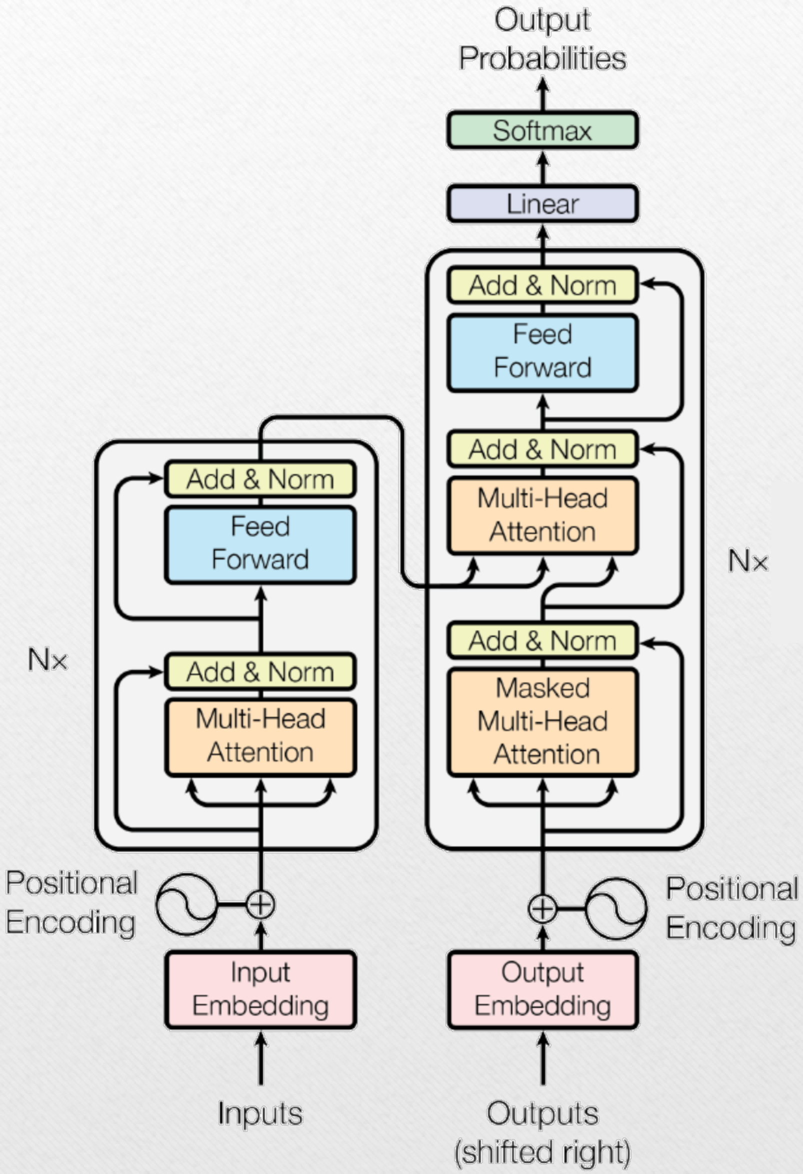
```
[start] Paul → ?
```



Inference

When using a trained model for inference, the following happens:

1. Provide the encoder with the input text, e.g. a sentence in English.
2. Encoder generates a vector for each word.
3. Provide the decoder with the `[start]` character.
4. Decoder predicts the first word, taking into consideration the encoder's output.
5. Append the predicted word to the decoder's input and predict the next word. Repeat until the decoder generates the `[end]` character.



Important Families of Transformer Model

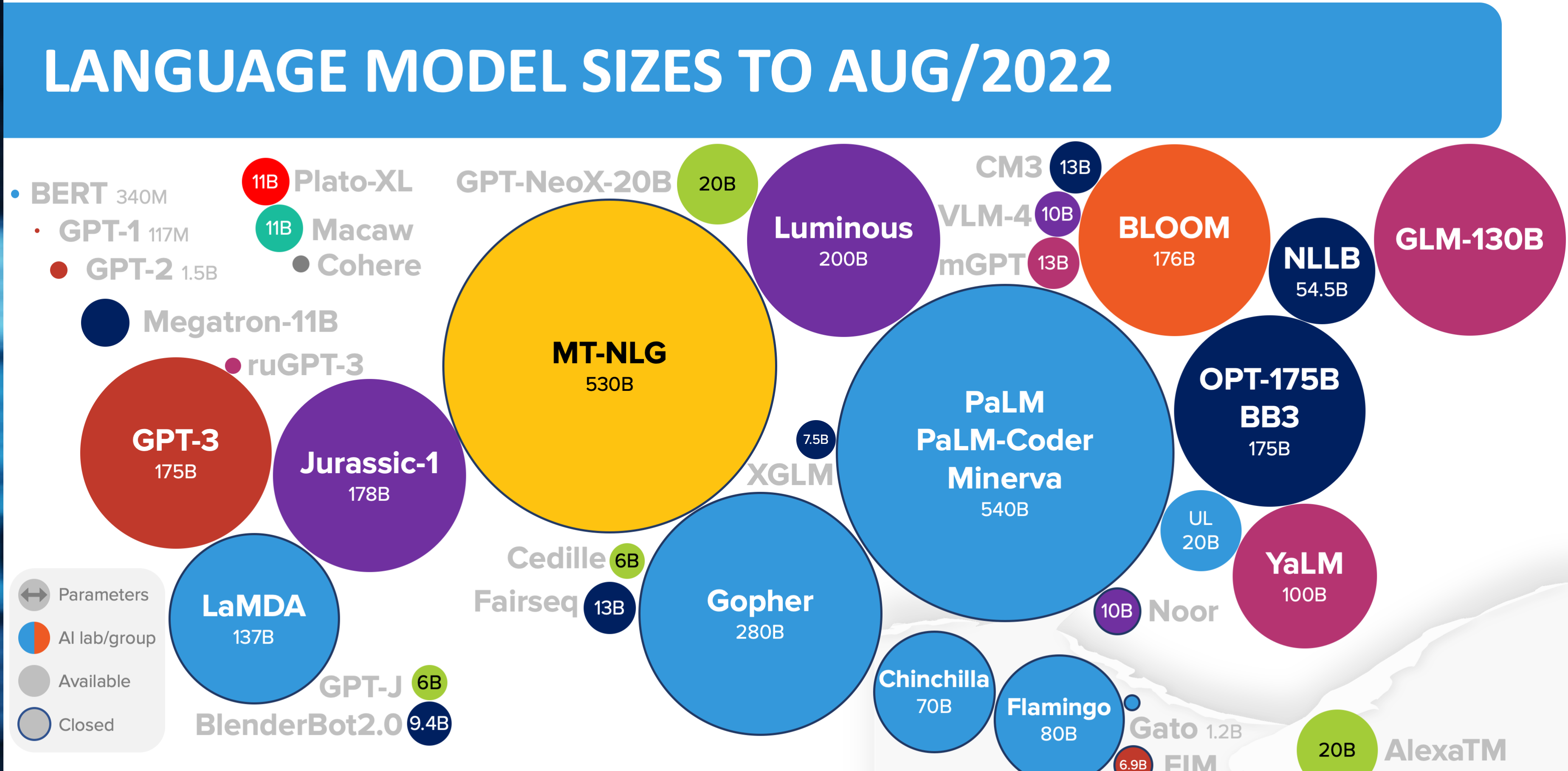
There are three major families of Transformer-based models:

- The original Transformer has Transformer Encoder and Decoder. The most important model in this family is Google's T5.
- GPT (**G**enerative **P**re-**T**raining) only uses the Transformer Decoder. Models in this family include OpenAI's GPT, GPT-2 and GPT-3.
- BERT (**B**idirectional **E**ncoder **R**epresentation from **T**ransformers) only uses the Transformer Encoder. Models in this family include BERT, RoBERTa, DistilBERT and ALBERT.
- Finally, there are efforts aiming at optimizing the training process for all families, e.g. NVIDIA's Megatron-LM.

Future Reading

- https://e2eml.school/transformers.html
- https://artifact-research.com/artificial-intelligence/talking-to-machines-prompt-engineering-injection/
- https://arxiv.org/pdf/2212.03551.pdf

2022 Big = Big in Parameter Count



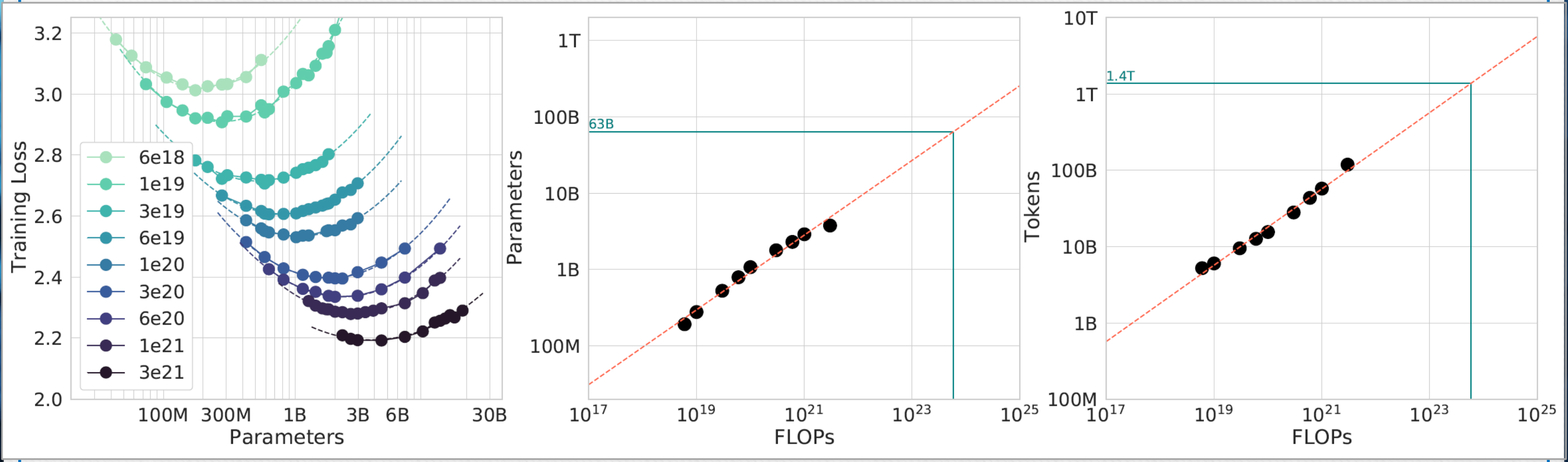
Beeswarm/bubble plot, sizes linear to scale. Selected highlights only. Alan D. Thompson. August 2022. https://lifearchitect.ai/

LifeArchitect.ai/models

2023 Big = Big in Data and Compute

Current generation of large language models are undertrained.

In other words, the amount of data and computational power spent on training is insufficient relative to model size.

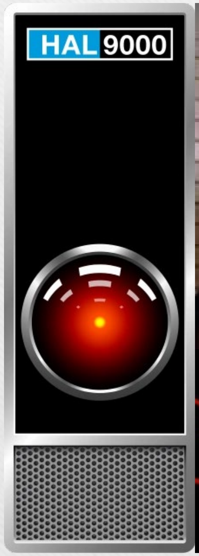


Hoffman et al (2022) a.k.a. "The Chinchilla Paper"

Future of AI

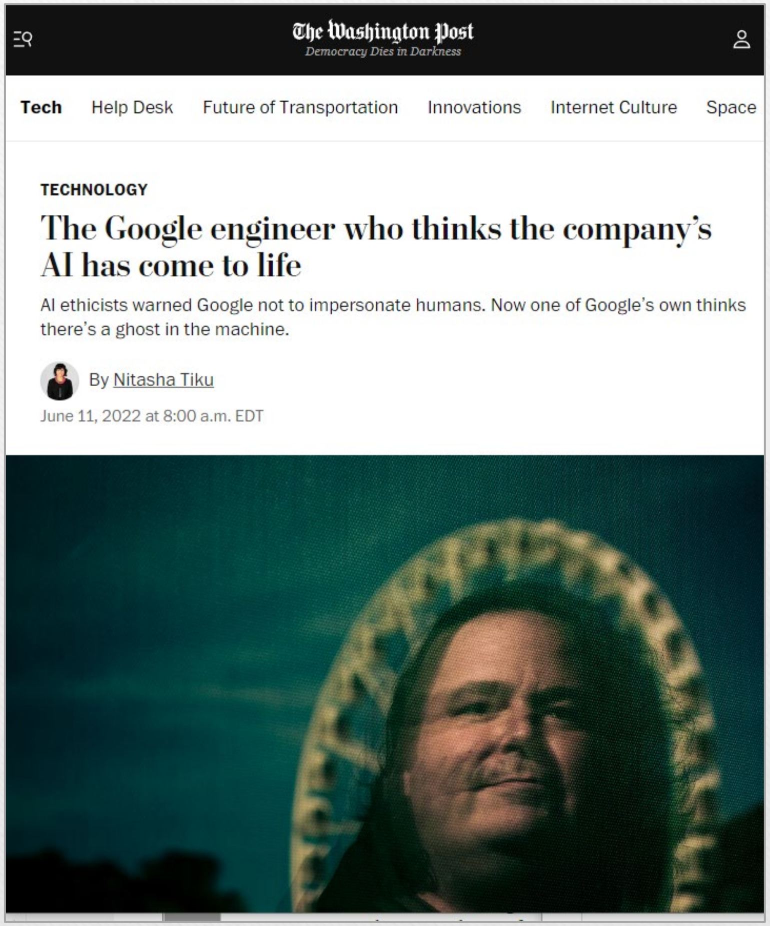
- It is not hard to find an AI model that does well in a narrow domain.
- We are much further away from Artificial General Intelligence (AGI) that what people would have predicted 10 years ago, 20 years ago, 30 years ago…

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last – a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO



Future of AI

While transformer-based large language models appear to be quite intelligent…



The Washington Post
Democracy Dies in Darkness

Tech Help Desk Future of Transportation Innovations Internet Culture Space

TECHNOLOGY

The Google engineer who thinks the company's AI has come to life

AI ethicists warned Google not to impersonate humans. Now one of Google's own thinks there's a ghost in the machine.

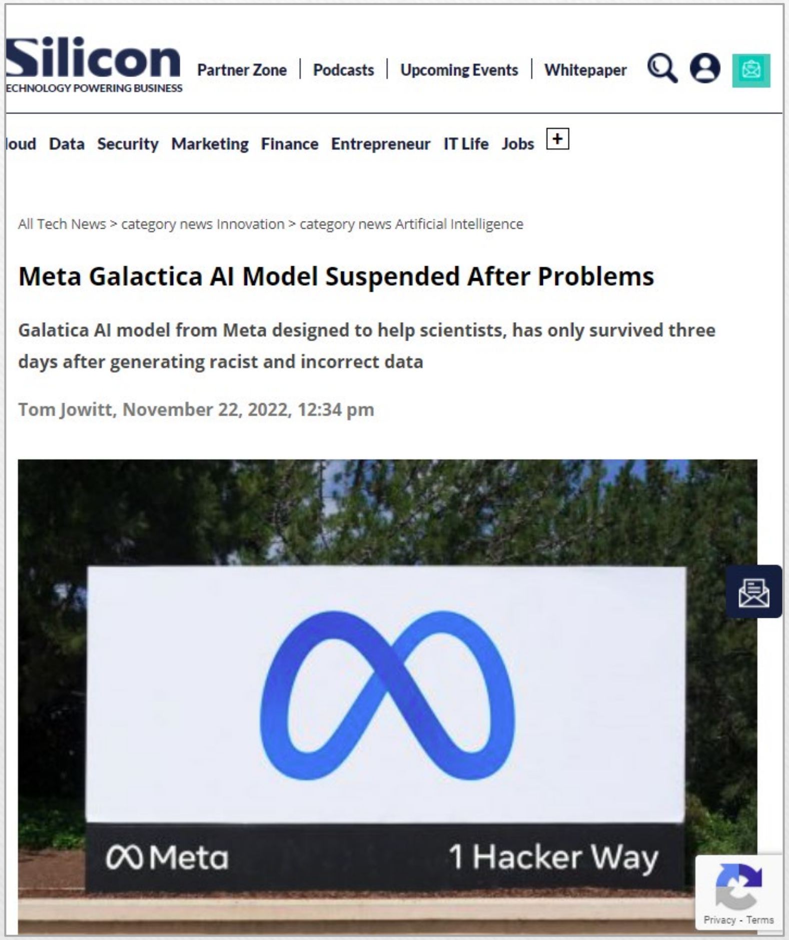
By Nitasha Tiku

June 11, 2022 at 8:00 a.m. EDT

Future of AI

What they output is best described as *hallucination*.

This works well for image generation (e.g. Stable Diffusion), but works poorly for writing scientific paper.



Silicon
ECHNOLOGY POWERING BUSINESS
Partner Zone | Podcasts | Upcoming Events | Whitepaper
loud Data Security Marketing Finance Entrepreneur IT Life Jobs +
Meta
1 Hacker Way
Privacy - Terms

All Tech News > category news Innovation > category news Artificial Intelligence

Meta Galactica AI Model Suspended After Problems

Galatica AI model from Meta designed to help scientists, has only survived three days after generating racist and incorrect data

Tom Jowitt, November 22, 2022, 12:34 pm