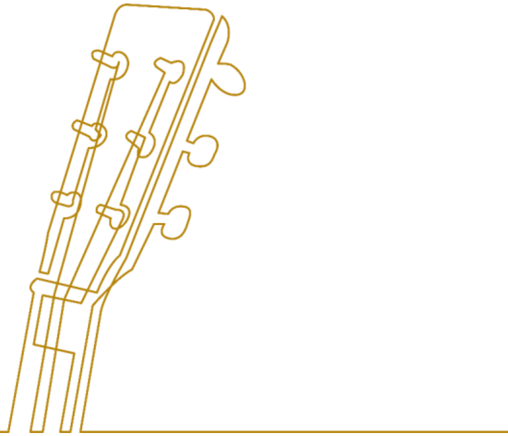
VIENNA SCIENTIFIC CLUSTER

Foundations of LLM Mastery: Fine-tuning with one GPU

22 January 2025
ONLINE



EURO AUSTRIA

Large Language Models on Supercomputers

A brief overview

Speaker: Simeon Harrison
Trainer at EuroCC Austria

EURO
AUSTRIA

EuroCC

Fully funded EU project

- EuroCC is EU-funded international initiative aimed to support the uptake of AI and High-Performance Computing (HPC) in Europe
- Set up of 32 National Competence Centres (NCCs) across Europe
- EuroCC Austria is one of them
- Service Provider for AI, HPC and HPDA

EURO
AUSTRIA

Need More Compute-Power?

LUMI

- Third most powerful supercomputer in Europe and the 8th globally (Nov 2024)
- Sustained computing power (HPL) is 380 petaflops
- Over 262 000 AMD EPYC CPU cores
- Equipped with AMD Radeon Instinct MI250X GPUs

https://www.lumi-supercomputer.eu/

Leonardo

- 4th most powerful supercomputer in Europe and the 9th globally (Nov 24)
- Sustained computing power (HPL) is 239 petaflops
- Intel new gen Sapphire Rapids 56 cores
- Equipped with custom NVIDIA A100 SXM6 64GB GPUs

https://leonardo-supercomputer.cineca.eu/

European HPC Landscape

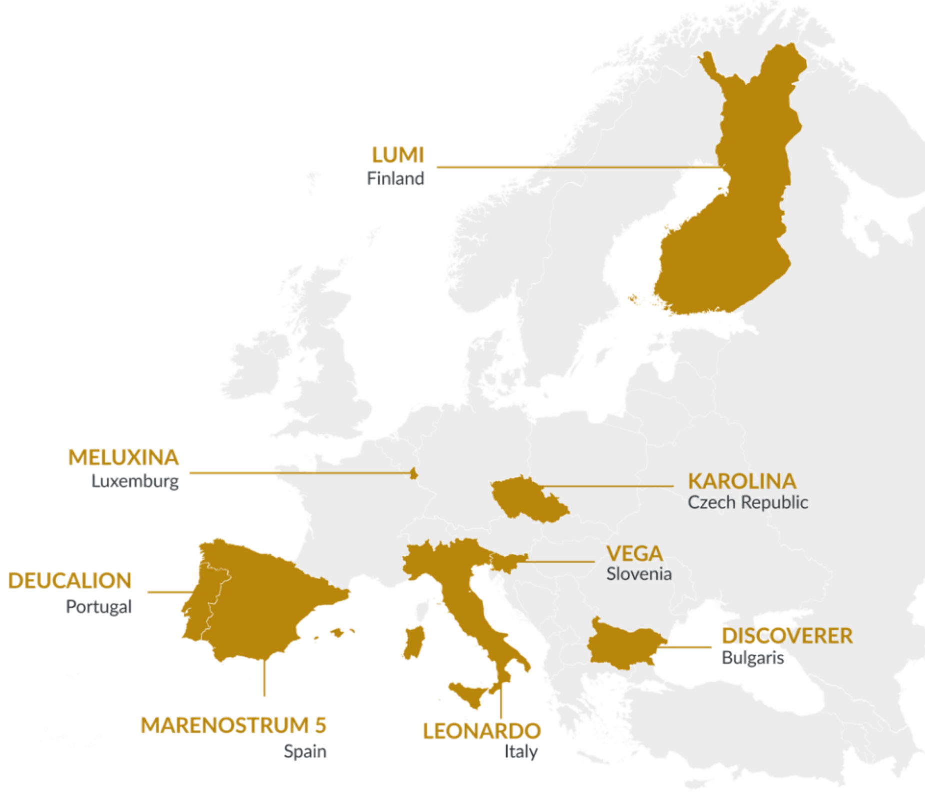
EURO AUSTRIA

EuroHPC JU systems

Different access modes:
Calls for Proposals

EuroHPC development access:
Opportunity to test the system

Applicants can request a small number of node hours to get acquainted with the supercomputers to further develop their software.



What can LLMs be used for?

Many different use-cases

- Made possible by the transformer architecture
- Choose your model according to the use-case
- Is there a pre-trianed model for your use-case?

Conversation

Text Generation

Translation

Text to Speech

Question Answering

Transcription of Videos

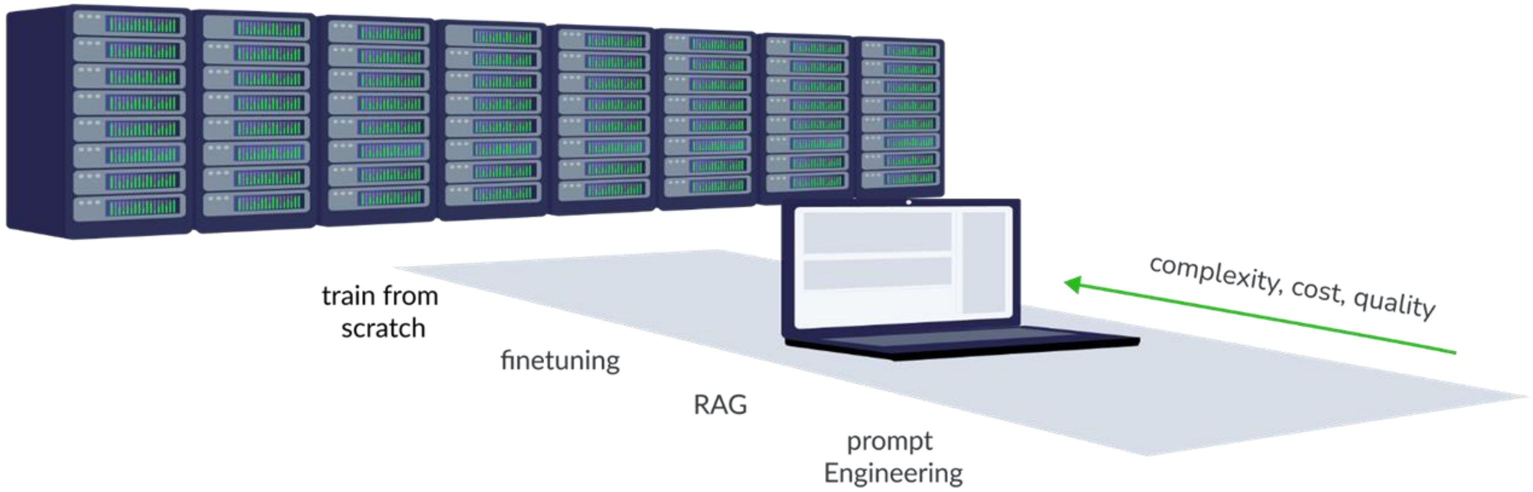
Text Classification

Sentiment Analysis

Similarity Detection

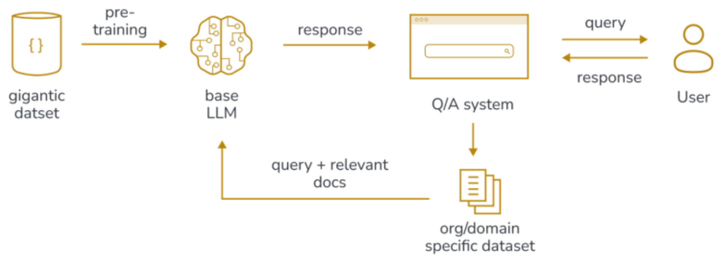
Summarization

How can you influence LLMs?



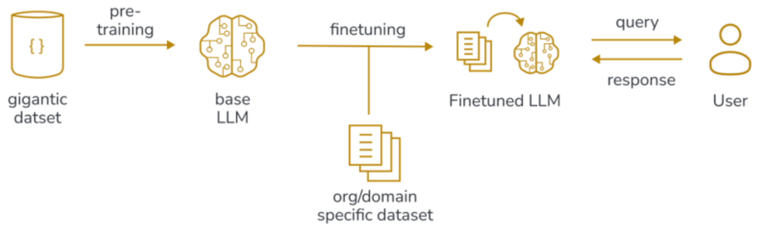
How can you use LLMs with your data?

RAG:
Retrieval Augmented Generation



- Ideal for tapping into company‘s knowledge DBs
- Minimises hallucinations by grounding response on retrieved evidence
- Can quickly adapt to changing data
- Makes it easier to interpret result

Finetuning



- Ideal if plenty of labelled data is available
- Teaches model domain specific vocabulary
- Company‘s writing/answer style is „baked“ into model through fine-tuned parameters

Prepare your Data

Garbage in – garbage out

- Most underrated aspect of AI
- Most time consuming aspect of AI. Time spent in data preparation reflects in the quality of the product
- For fine-tuning you need labelled data
- Remember, that you are going to change the models parameters with your data

The Hugging Face Ecosystem

From 🤗Transformers to the 🤗Hub

Speaker: Simeon Harrison
Trainer at EuroCC Austria

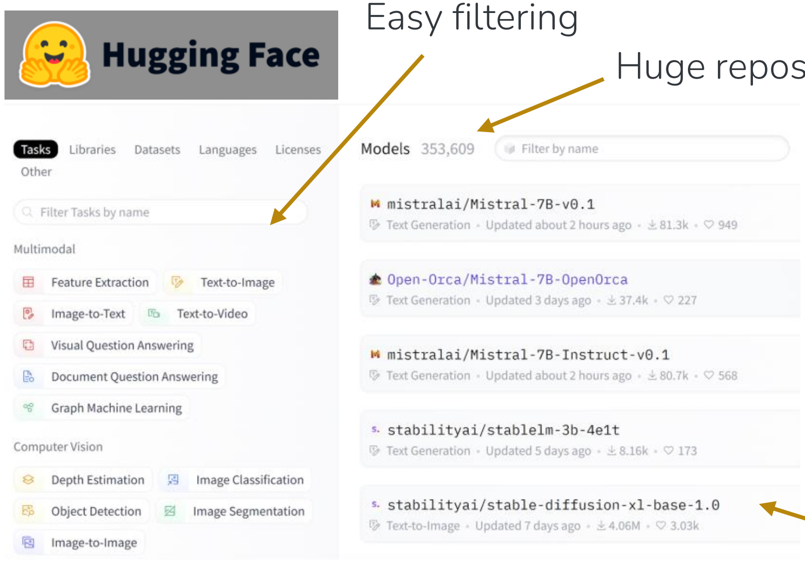
EURO
AUSTRIA

Transformer Models

Spoilt for Choice at https://huggingface.co/

Easy filtering

Huge repository



mistralai/Mistral-7B-v0.1
Text Generation · Updated about 2 hours ago · 81.3k · 949

Open-Orca/Mistral-7B-OpenOrca
Text Generation · Updated 3 days ago · 37.4k · 227

mistralai/Mistral-7B-Instruct-v0.1
Text Generation · Updated about 2 hours ago · 80.7k · 568

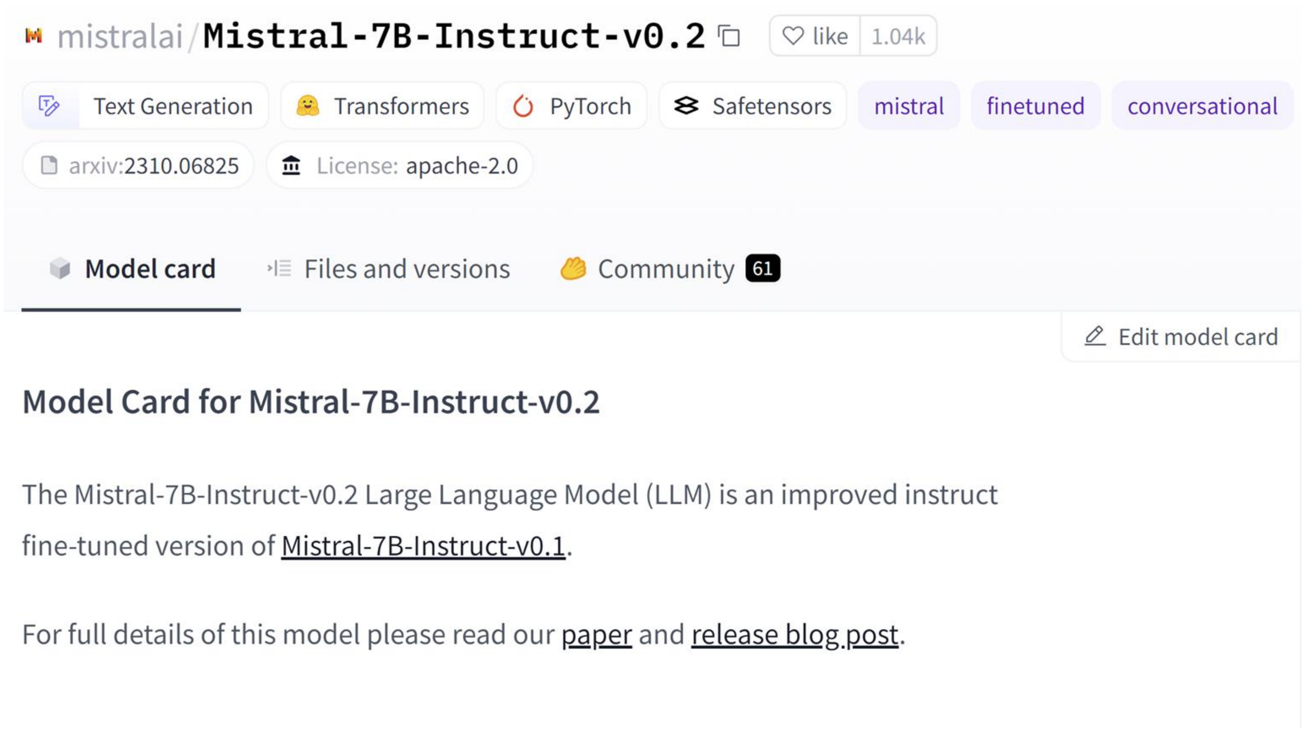
stabilityai/stablelm-3b-4e1t
Text Generation · Updated 5 days ago · 8.16k · 173

stabilityai/stable-diffusion-xl-base-1.0
Text-to-Image · Updated 7 days ago · 4.06M · 3.03k

All the relevant info

Source: https://huggingface.co/

Pick the Right Model

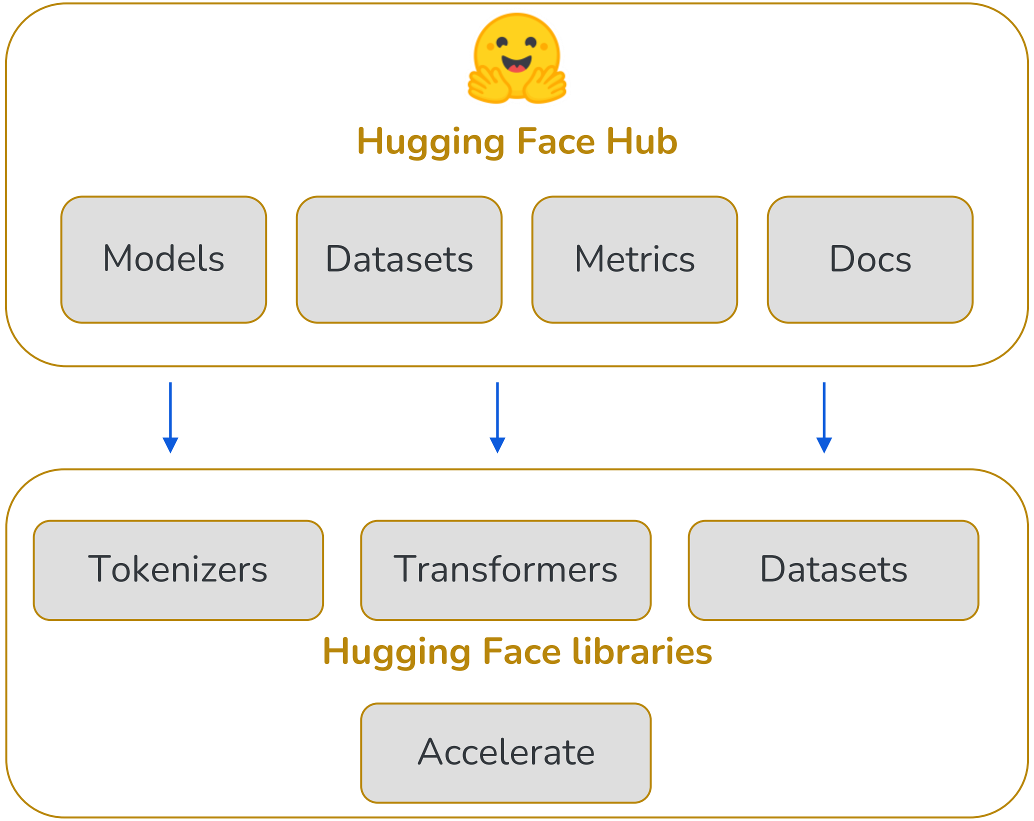

mistralai/Mistral-7B-Instruct-v0.2

Model Card for Mistral-7B-Instruct-v0.2

The Mistral-7B-Instruct-v0.2 Large Language Model (LLM) is an improved instruct fine-tuned version of Mistral-7B-Instruct-v0.1.

For full details of this model please read our paper and release blog post.

Hub and libraries



Libraries

Tokenizers

Very fast at tokenizing text, thanks to the Rust bakend.

Takes care of pre- and postprocessing, such as normalization and transformation to required format.

You can load tokenizers just like you load models.

Transformers

Provides a standardized interface to a wide range of transformer models.

Supports PyTorch, TensorFlow and JAX and makes sitching between them easy.

Provides task-specific heads to fine-tune LLMs on downstream tasks

Libraries

Datasets

Simplifies the process of downloading and transforming data into required format.

Interface to thousands of datasets that can be found on the hub.

Interoperable with most common data wrangling tools and frameworks such as Pandas, NumPy etc.

Accelerate

Adds a layer of abstraction to training loops with the goal of making code portable (local laptop to cluser)

Makes training on multi-GPU, multi-node easier to accomplish, as you can switch between parallelization strategies.

Transformer Anatomy

Attention is really all you need?

Speaker: Simeon Harrison
Trainer at EuroCC Austria

EURO
AUSTRIA

Transformer Anatomy

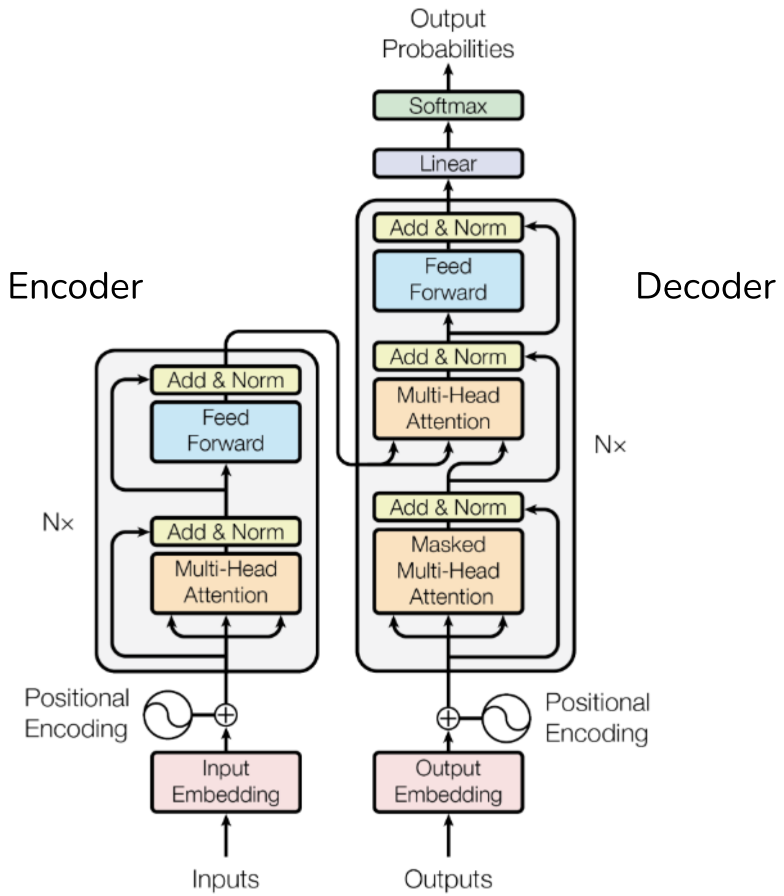
The original architecture

A transforer consists of a encoder and/or decoder block.

Words (tokens) are input as numerical representations (embeddings).

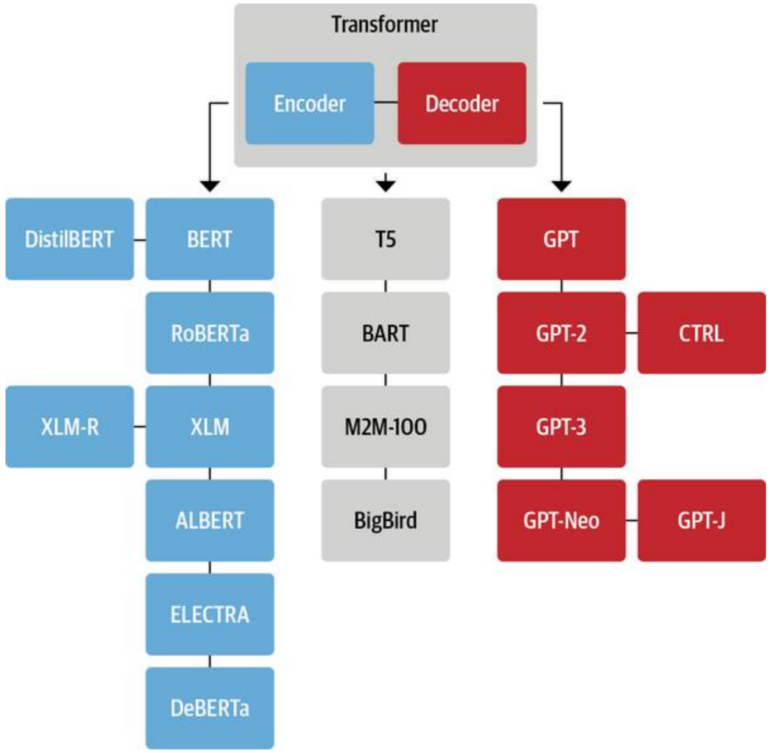
About 1/3 of all parameters are in the multi-head attention blocks

About 2/3 of all parameters are in the feed forward networks (also known as multi layer perceptron)



Source: “Attention Is All You Need”, Vaswani et al.

Transformer Family



Source: Natural Language Processing with Transformers, O'Reilly

Encoder only:
These models ecxel at text classification, named entity recognition, and question answering

Decoder only:
Very good at predicting the next word in a sequence, therefore mostly used for text generation

Encoder-Decoder:
These models are often used for machine translation or summerization tasks.

Context Is All You Need

Embeddings

Here, we will refer to “word” instead of “token”, as it makes the content easier to explain.

A word embedding comes as a multi dimensional vector (e.g. 12.000 dim).

The initial word embedding in all of the examples of the word „mole“ is the same.

The European mole is a mammal

Take a biopsy of the mole

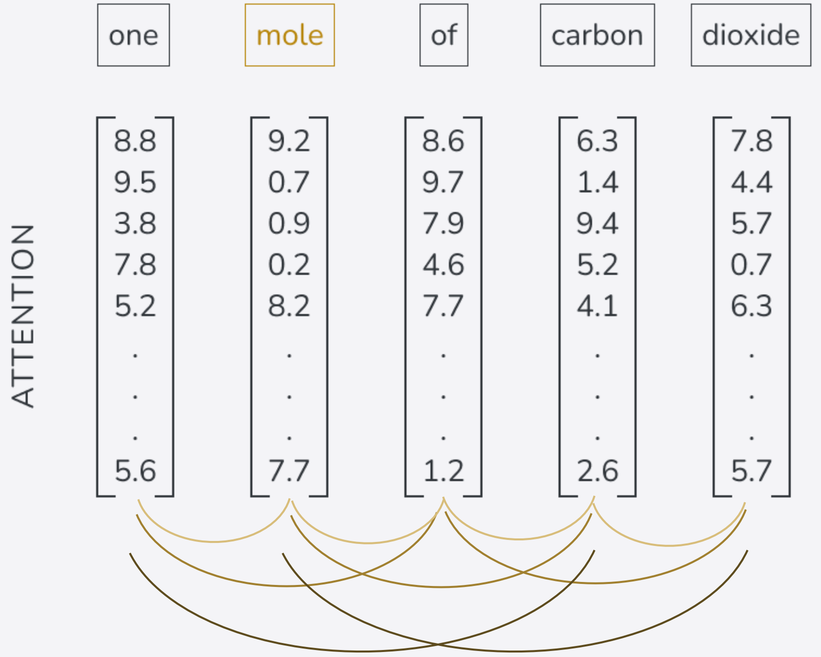
One $\overbrace{\text{mole}}^{6.02 \times 10^{23}}$ of carbon dioxide

Context Is All You Need

Attention

The word „mole“ should be represented by a **unique vector** in the embedding space, depending on ist **context.**

An **attention** block should **compute the vectors that you need to add** to the original, generic vector to get it to the correct, meaningfull, rich representation, depending on the context in which the word is used.

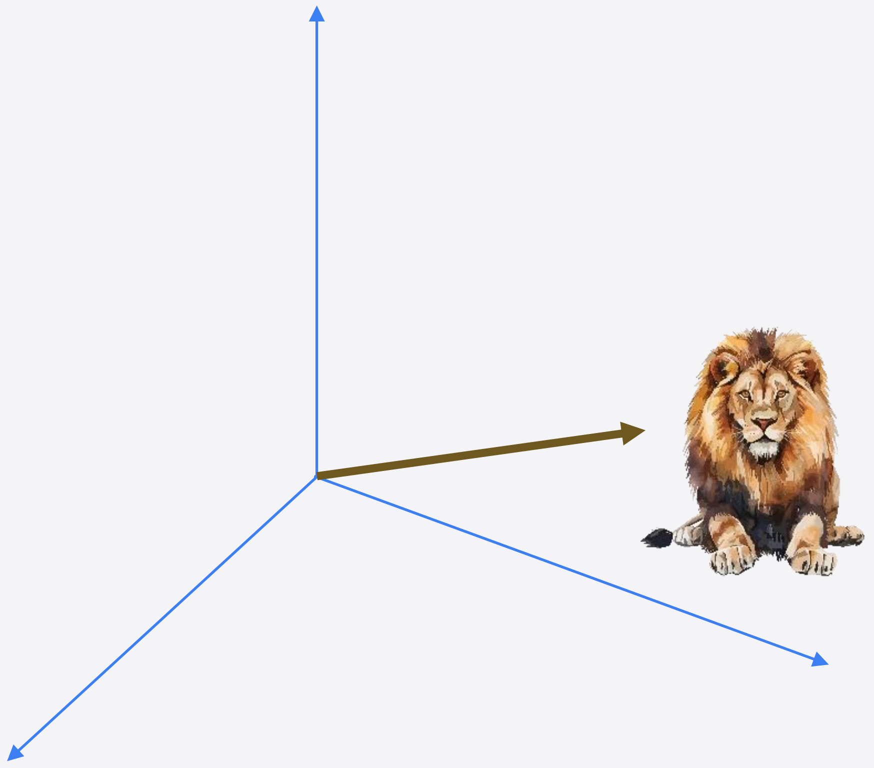


Context Is All You Need

Lion

We associate the word „lion“ with a big cat, living wild on the African continent.
We probably imagine a majestic predator with a big mane.

The embedding of the word „lion“ is a vector with a certain length and direction within the embedding space.



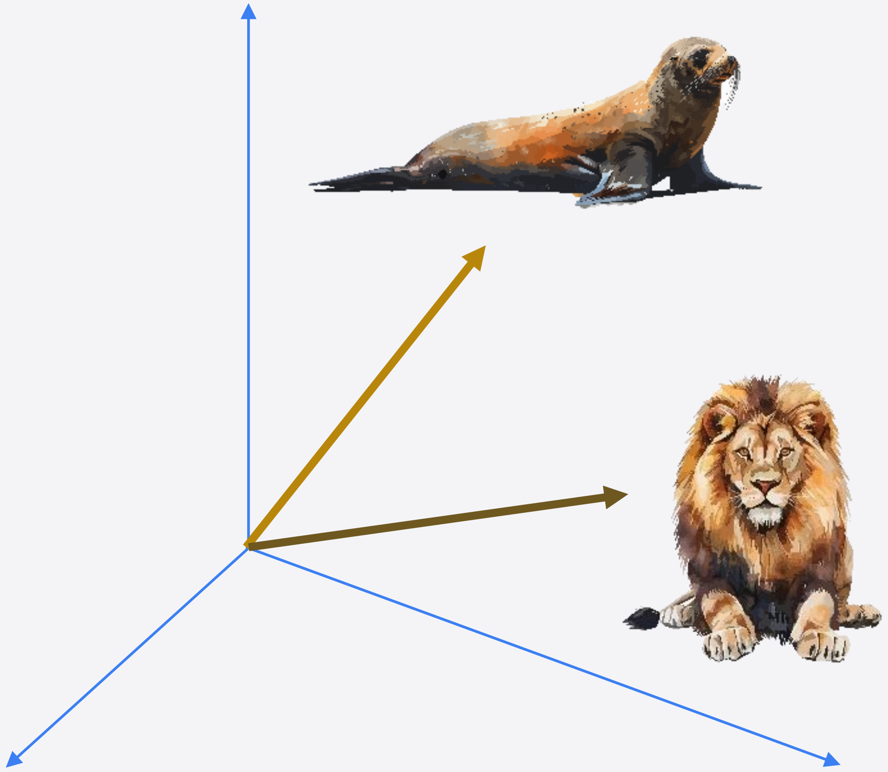
EURO AUSTRIA

Context Is All You Need

Sea Lion

However, as soon we add the word „sea“ infront of „lion“ we imagine a totally different animal.

The same goes for the embedding. The attiontion mechanism needs to update the direction and length of the vector so that it represents the animal in question correctly.



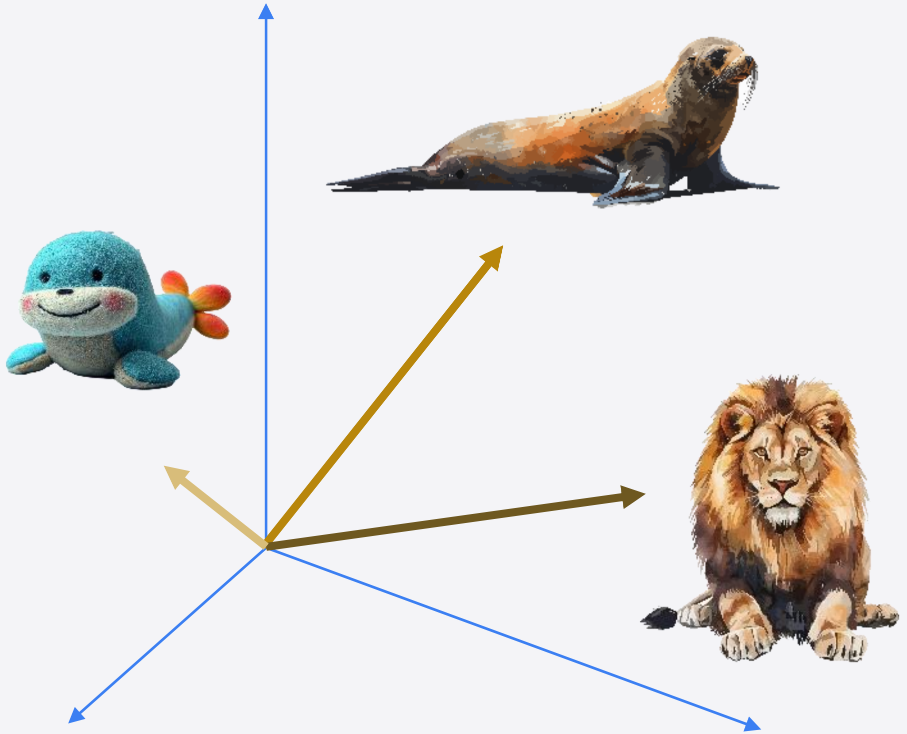
Context Is All You Need

Sea Lion Cuddly Toy

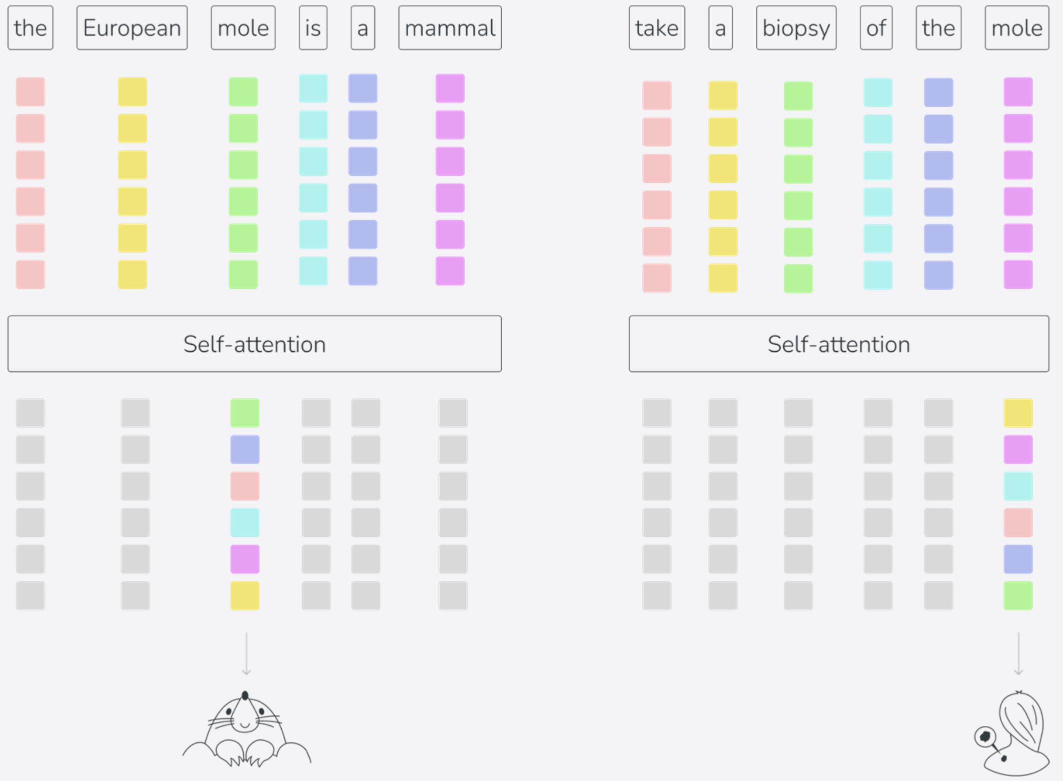
The context depends on more than just the immediate words to the left and right.

The embedding of „sea lion cuddly toy“ will certainly be very different of just „lion“.

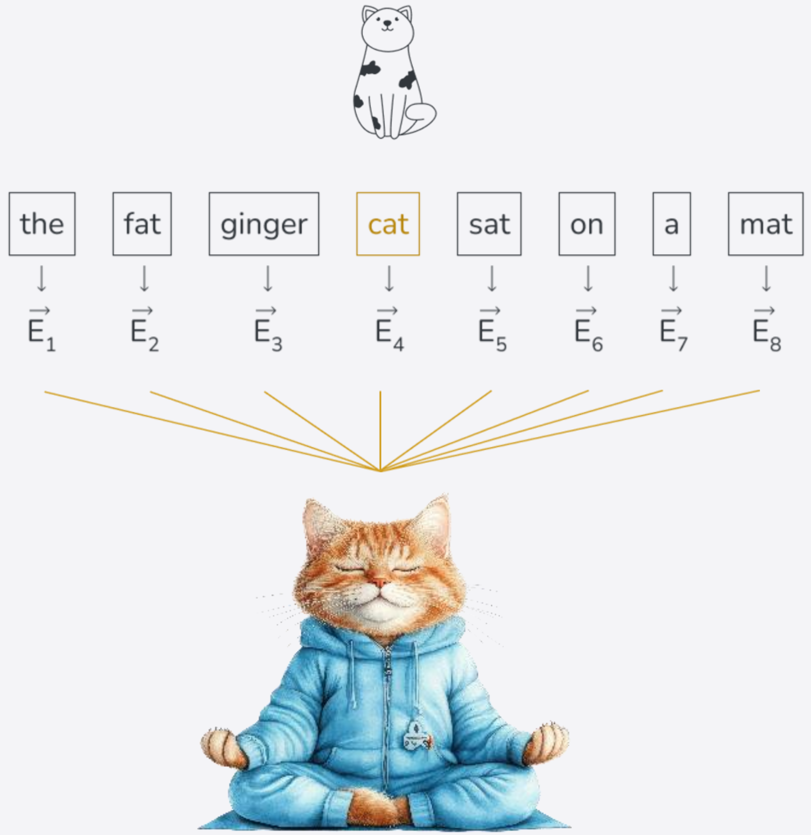
In order to achieve that the vector for „lion“ needs to attend to all the other words in the input (context size).



Self Attention



Self Attention



Attention! Queries, Keys and Values

The Attention Score

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

Q.....query matrix

K.....key matrix

V.....value matrix

d.....dimension of (smaller) query-key space

EURO
AUSTRIA

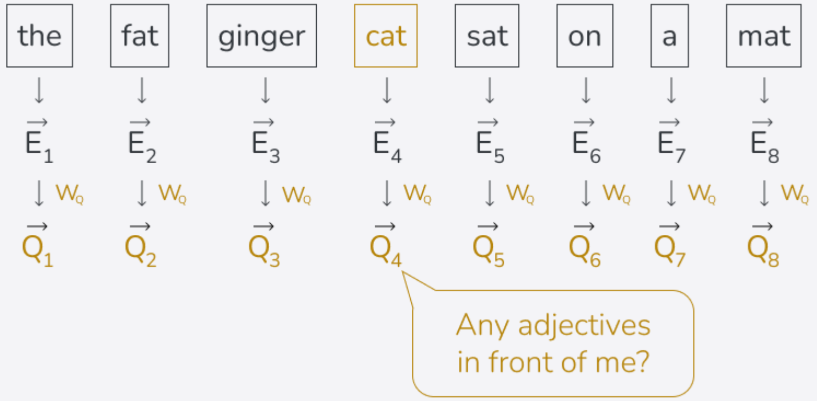
Attention! Queries, Keys and Values

Query

The query vector is obtained by multiplying a matrix full of trainable parameters with the embedding vector of that word (token).
Each word has its own query vector:

$$\overrightarrow{Q_i} = W_Q \overrightarrow{E_i}$$

It mappes the embedding vector to a much smaller dimensional space (e.g. 128 dimensions)


the
fat
ginger
cat
sat
on
a
mat
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
W_Q
$\vec{Q}_1$ $\vec{Q}_2$ $\vec{Q}_3$ $\vec{Q}_4$ $\vec{Q}_5$ $\vec{Q}_6$ $\vec{Q}_7$ $\vec{Q}_8$
Any adjectives in front of me?

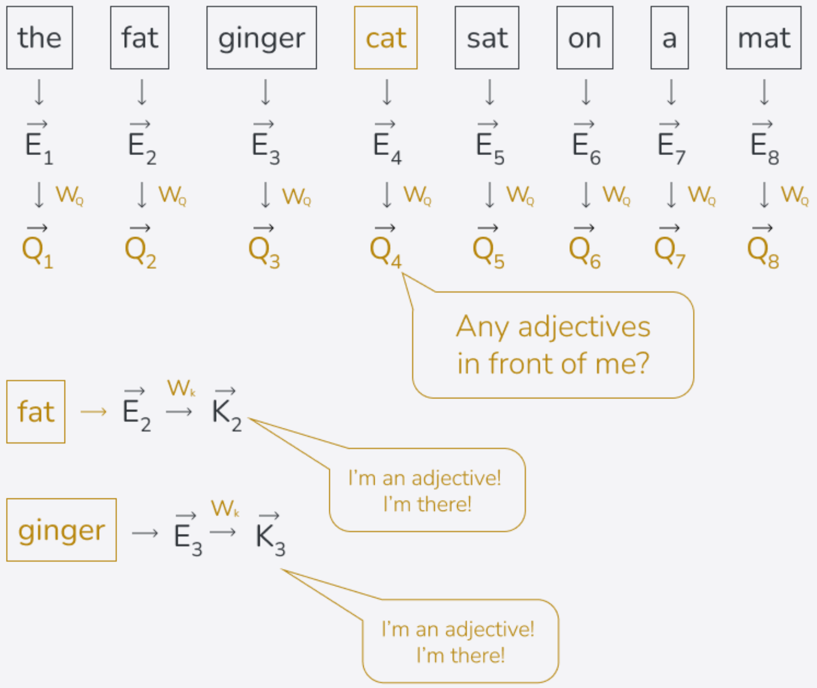
Imagine it like this:
One particular attention head is focussing on nouns. The query vector is like looking for labels with „adjective“ on them to better understand the noun.
In reality, this is much more abstract.

Attention! Queries, Keys and Values

Key

The key vector is also obtained by multiplying a matrix full of trainable parameters with the embedding vector of that word (token). Each word also has its own key vector: $\overrightarrow{K_i} = W_K\overrightarrow{E_i}$

Imagine it like this:
We are still dealing with the particular attention head that is focussing on nouns. The key is answering the question the query raised.

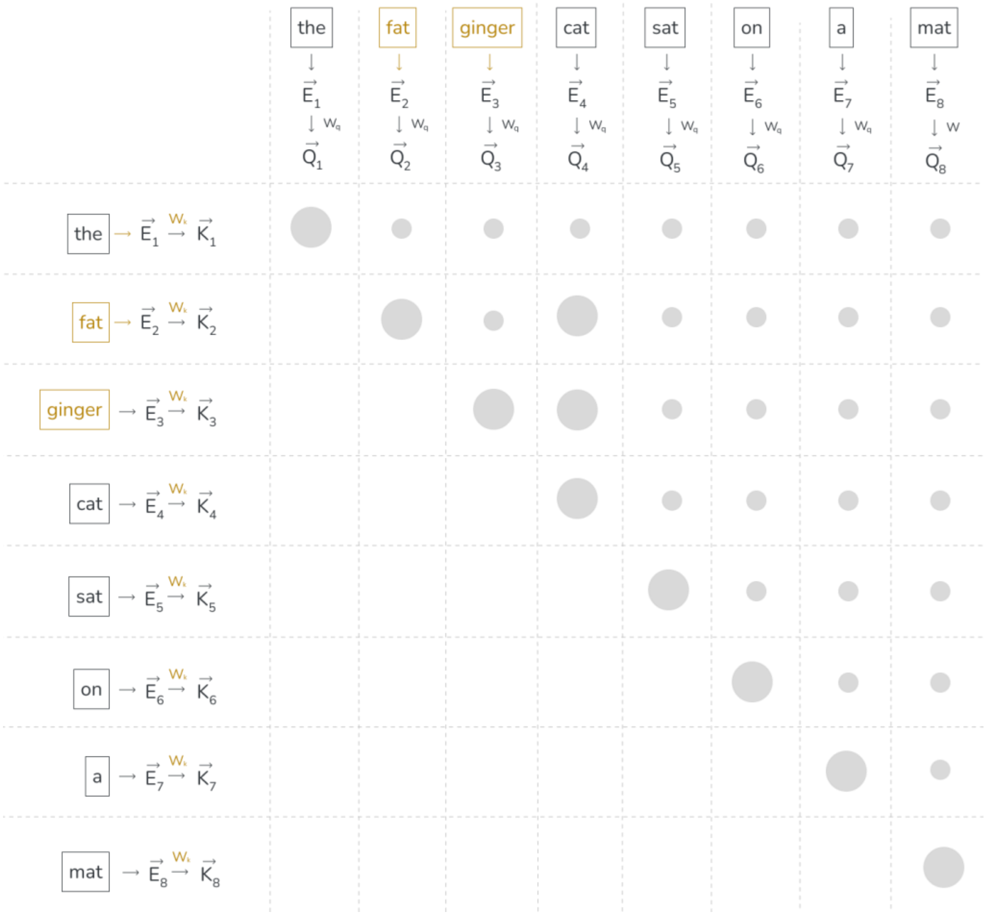

the
fat
ginger
cat
sat
on
a
mat
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
W_Q
$\vec{Q}_1$ $\vec{Q}_2$ $\vec{Q}_3$ $\vec{Q}_4$ $\vec{Q}_5$ $\vec{Q}_6$ $\vec{Q}_7$ $\vec{Q}_8$
Any adjectives in front of me?
fat $\rightarrow \vec{E}_2 \xrightarrow{W_k} \vec{K}_2$
I'm an adjective! I'm there!
ginger $\rightarrow \vec{E}_3 \xrightarrow{W_k} \vec{K}_3$
I'm an adjective! I'm there!

Attention! Queries, Keys and Values

Query – Key

Compute the dot product with each query-key pair, to determine how well the key matches the query. Where the queries and keys align, the dot product is larger.

Imagine it like this:
With our previous example, the dot product of the key vectors of „fat" and „ginger" with the vector of „cat" yields the largest result. The embeddings of „fat" and „ginger" attend to „cat".



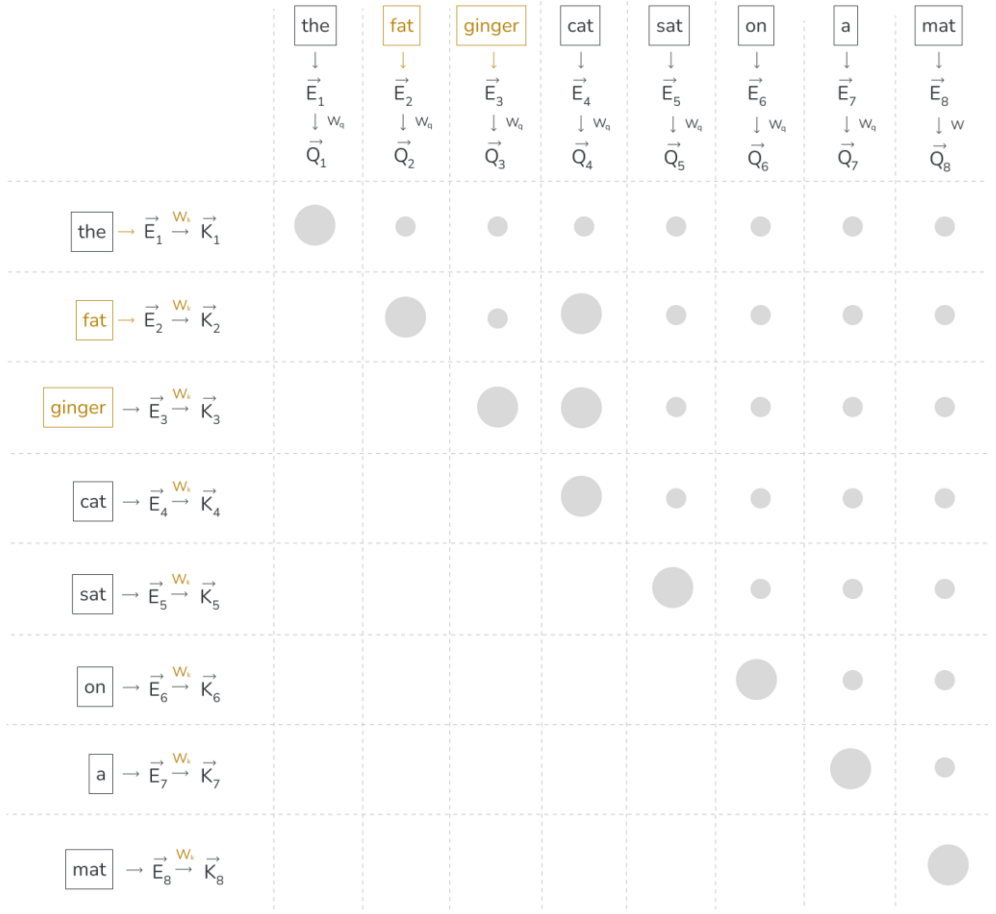
the
fat
ginger
cat
sat
on
a
mat
$\vec{E}_1$ W_q $\vec{Q}_1$
$\vec{E}_2$ W_q $\vec{Q}_2$
$\vec{E}_3$ W_q $\vec{Q}_3$
$\vec{E}_4$ W_q $\vec{Q}_4$
$\vec{E}_5$ W_q $\vec{Q}_5$
$\vec{E}_6$ W_q $\vec{Q}_6$
$\vec{E}_7$ W_q $\vec{Q}_7$
$\vec{E}_8$ W $\vec{Q}_8$
the $\vec{E}_1$ W_k $\vec{K}_1$
fat $\vec{E}_2$ W_k $\vec{K}_2$
ginger $\vec{E}_3$ W_k $\vec{K}_3$
cat $\vec{E}_4$ W_k $\vec{K}_4$
sat $\vec{E}_5$ W_k $\vec{K}_5$
on $\vec{E}_6$ W_k $\vec{K}_6$
a $\vec{E}_7$ W_k $\vec{K}_7$
mat $\vec{E}_8$ W_k $\vec{K}_8$

Attention! Queries, Keys and Values

Attention Pattern

Lower left dot products are masked, as we want the model to predict every next word during training. To prevent data leakage, future words should not influence previous ones.

The size of the attention pattern is the context size squared.
This is why the context size can be a substantial bottleneck.



the
fat
ginger
cat
sat
on
a
mat
$\vec{E}_1$ … $\vec{E}_8$
W_q
$\vec{Q}_1$ … $\vec{Q}_8$
W_k
$\vec{K}_1$ … $\vec{K}_8$

Attention! Queries, Keys and Values

Attention

So far, we have determined which word is relevant to which other word (dot product of query and key).

We would like to use this as a score for how relevant every word is to update the meaning of other words.

For numerical stability the dot product is devided by the square root of the dimension of the query-key space.

To normalize the numbers to be between 0 and 1 we apply softmax.

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

EURO AUSTRIA

Attention! Queries, Keys and Values

Value

The value matrix multiplied by the embedding of the preceding word results in the value vector.

$$\vec{V_i} = W_K \vec{E_i}$$

	the	fat	ginger	cat	sat	on	a	mat
	$\downarrow$ $\vec{E}_1$	$\downarrow$ $\vec{E}_2$	$\downarrow$ $\vec{E}_3$	$\downarrow$ $\vec{E}_4$	$\downarrow$ $\vec{E}_5$	$\downarrow$ $\vec{E}_6$	$\downarrow$ $\vec{E}_7$	$\downarrow$ $\vec{E}_8$
$\vec{E}_1 \xrightarrow{W_v} \vec{V}_1$	$1.0\ \vec{V}_1$	$0.0\ \vec{V}_1$	$0.0\ \vec{V}_1$	$0.0\ \vec{V}_1$	$0.0\ \vec{V}_1$	$0.0\ \vec{V}_1$	$0.0\ \vec{V}_1$	$0.0\ \vec{V}_1$
	+	+	+	+	+	+	+	+
$\vec{E}_2 \xrightarrow{W_v} \vec{V}_2$	$0.0\ \vec{V}_2$	$1.0\ \vec{V}_2$	$0.0\ \vec{V}_2$	$0.42\ \vec{V}_2$	$0.0\ \vec{V}_2$	$0.0\ \vec{V}_2$	$0.0\ \vec{V}_2$	$0.0\ \vec{V}_2$
	+	+	+	+	+	+	+	+
$\vec{E}_3 \xrightarrow{W_v} \vec{V}_3$	$0.0\ \vec{V}_3$	$0.0\ \vec{V}_3$	$1.0\ \vec{V}_3$	$0.58\ \vec{V}_3$	$0.0\ \vec{V}_3$	$0.0\ \vec{V}_3$	$0.0\ \vec{V}_3$	$0.0\ \vec{V}_3$
	+	+	+	+	+	+	+	+
$\vec{E}_4 \xrightarrow{W_v} \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$	$0.0\ \vec{V}_4$
	+	+	+	+	+	+	+	+
$\vec{E}_5 \xrightarrow{W_v} \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$	$0.0\ \vec{V}_5$
	+	+	+	+	+	+	+	+
$\vec{E}_6 \xrightarrow{W_v} \vec{V}_6$	$0.0\ \vec{V}_6$	$0.0\ \vec{V}_6$	$0.0\ \vec{V}_6$	$0.0\ \vec{V}_6$	$0.99\ \vec{V}_6$	$1.0\ \vec{V}_6$	$0.0\ \vec{V}_6$	$0.0\ \vec{V}_6$
	+	+	+	+	+	+	+	+
$\vec{E}_7 \xrightarrow{W_v} \vec{V}_7$	$0.0\ \vec{V}_7$	$0.0\ \vec{V}_7$	$0.0\ \vec{V}_7$	$0.0\ \vec{V}_7$	$0.0\ \vec{V}_7$	$0.0\ \vec{V}_7$	$1.0\ \vec{V}_7$	$1.0\ \vec{V}_7$
	+	+	+	+	+	+	+	+
$\vec{E}_8 \xrightarrow{W_v} \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$	$0.0\ \vec{V}_8$
	$\parallel$ $\Delta\vec{E}_1$	$\parallel$ $\Delta\vec{E}_2$	$\parallel$ $\Delta\vec{E}_3$	$\parallel$ $\Delta\vec{E}_4$	$\parallel$ $\Delta\vec{E}_5$	$\parallel$ $\Delta\vec{E}_6$	$\parallel$ $\Delta\vec{E}_7$	$\parallel$ $\Delta\vec{E}_8$

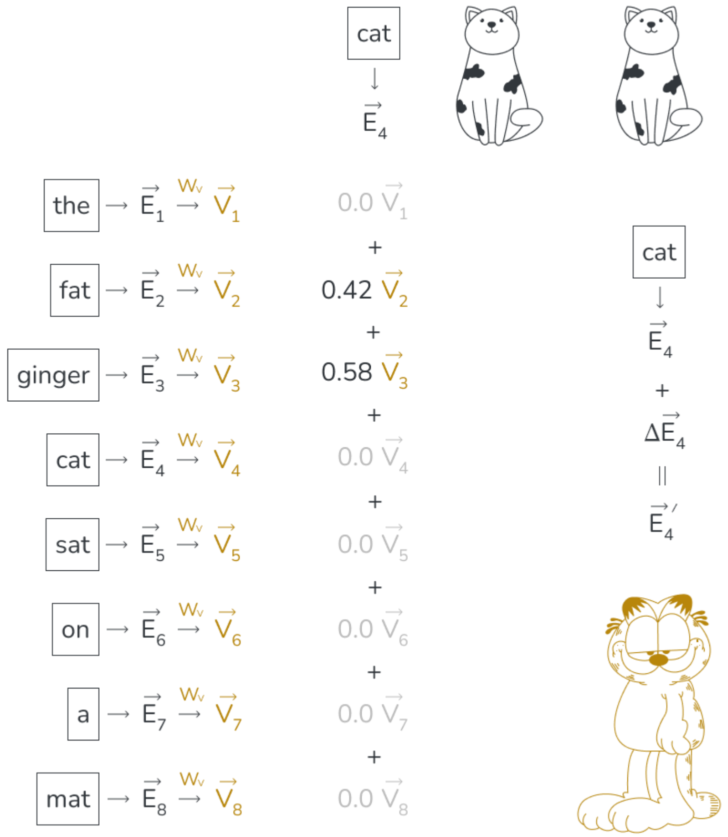
Attention! Queries, Keys and Values

Value

Eeach value vector is then multiplied by the corresponding weight for this word and the results summed up.

The result $\Delta\vec{E_i}$ is the change, that needs to be added to the origial embedding to get an updated, richer meaning.

Since this happens to every word in the sequence, we end up with a set of more refined embeddings.



cat
$\vec{E}_4$
the $\rightarrow \vec{E}_1 \xrightarrow{W_v} \vec{V}_1$
fat $\rightarrow \vec{E}_2 \xrightarrow{W_v} \vec{V}_2$
ginger $\rightarrow \vec{E}_3 \xrightarrow{W_v} \vec{V}_3$
cat $\rightarrow \vec{E}_4 \xrightarrow{W_v} \vec{V}_4$
sat $\rightarrow \vec{E}_5 \xrightarrow{W_v} \vec{V}_5$
on $\rightarrow \vec{E}_6 \xrightarrow{W_v} \vec{V}_6$
a $\rightarrow \vec{E}_7 \xrightarrow{W_v} \vec{V}_7$
mat $\rightarrow \vec{E}_8 \xrightarrow{W_v} \vec{V}_8$
$0.0\,\vec{V}_1 + 0.42\,\vec{V}_2 + 0.58\,\vec{V}_3 + 0.0\,\vec{V}_4 + 0.0\,\vec{V}_5 + 0.0\,\vec{V}_6 + 0.0\,\vec{V}_7 + 0.0\,\vec{V}_8$
cat
$\vec{E}_4 + \Delta\vec{E}_4 = \vec{E}_4'$

EURO AUSTRIA

Attention

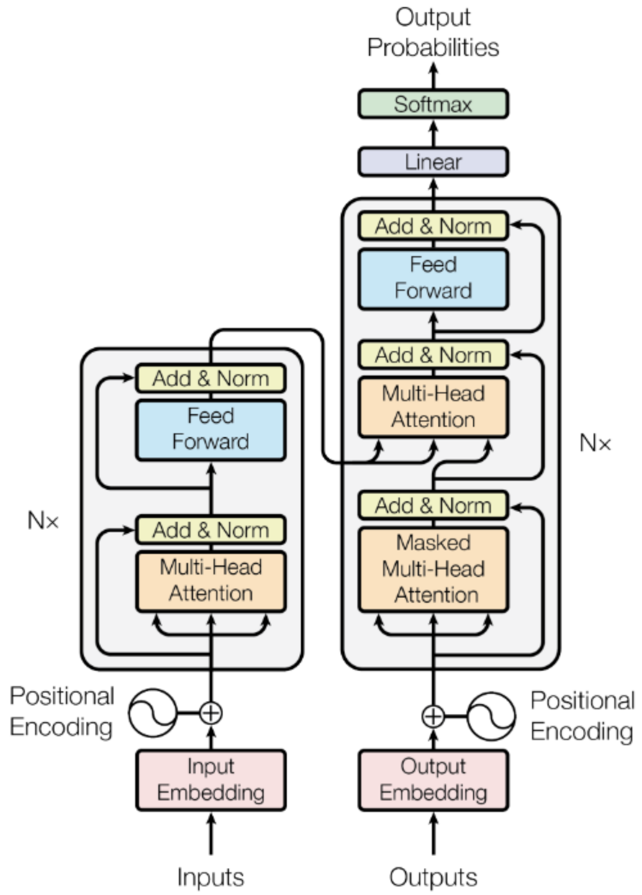
Multi-Head Attention

The attention mechanism we just looked at, is done several times in parallel.

Each attention head is focussing on different features of the embeddings and computes ist own $\Delta\overrightarrow{E_i}^{(j)}$

The resulting $\Delta\overrightarrow{E_i}^{(j)}$ vectors, each suggesting the necessary change, are added to the orginal embedding.

This can be done in parallel on GPUs.



Source: “Attention Is All You Need”, Vaswani et al.

Attention

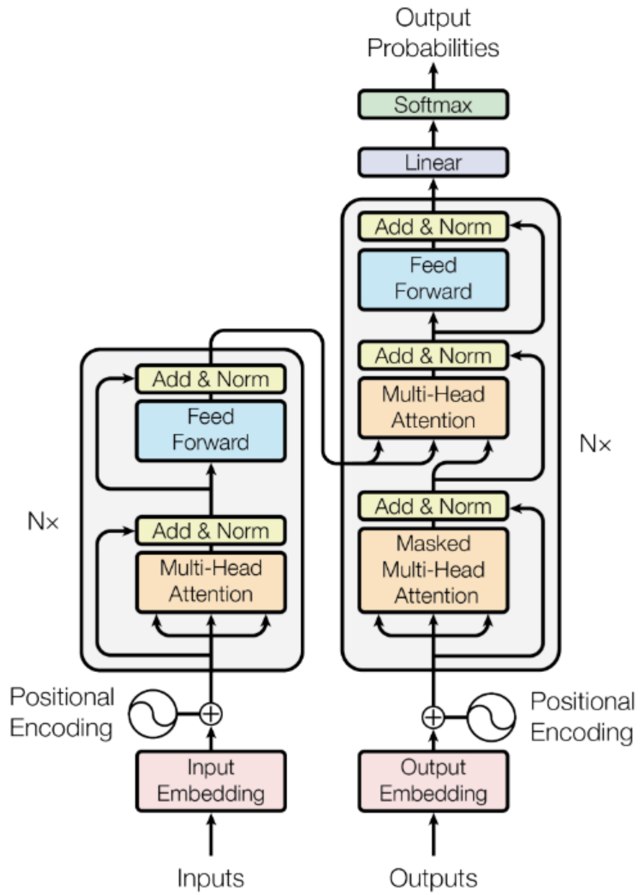
Cross Attention

Cross attention is almost the same as self attention.

Difference:

- query and key maps act on different data sets (e.g. 2 different languages in machine translation)
- no masking, since there is no issue of later words affecting earlier ones.

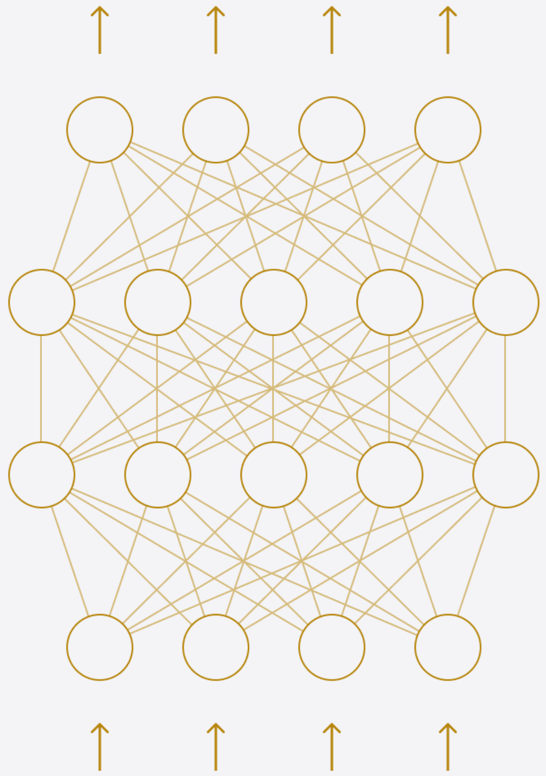
Source: “Attention Is All You Need”, Vaswani et al.



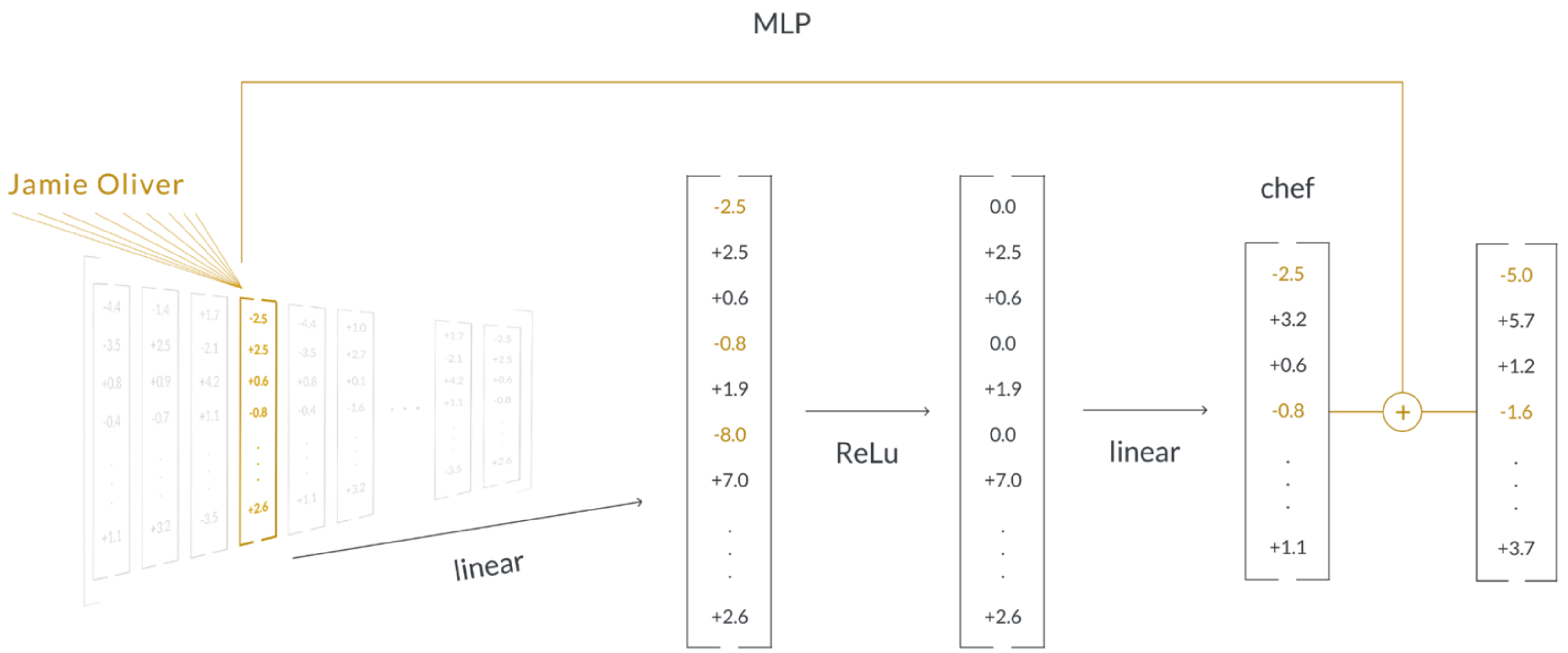
Feed Forward Network

Multi Layer Perceptron (MLP)

- Home to approx. 2/3s of all parameters
- This is where „knowledge“ is baked in
- Source of halluzinations
- Facts that are associated with input embeddings are added to the input embeddings
- Each embedding vector can be processed independently -> parallelization!



Feed Forward Network



Tokenization & Embeddings

Turning words into numbers

Speaker: Simeon Harrison
Trainer at EuroCC Austria

From Text to Tokens

Character Tokenization

- Only needs 256 integers
- Good to tokenize rare words
- Helps with misspellings
- Looses linguistic structure
- Rarely used in practice

L a r g e l a n g u a g e m
o d e l s o n s u p e r c o m
p u t e r s

EURO
AUSTRIA

From Text to Tokens

Word Tokenization

- Model does not have to learn words from characters
- Each word has specific ID
- Size of vocabulary explodes
- Model needs to learn different tokens for e.g. singular and plural

Large lanugage models on supercomputers

From Text to Tokens

Subword Tokenization

- Best of both worlds
- Deal with complex words easily
- Frequent words remain as one token

Large lanugage models
on super com put ers

From Tokens to Vectors

Embeddings

Tokens are mapped to unique integers accoding to the vocabulary size of the tokenizer.

Now, the tokens need to be embedded, which means turned into a vector representation.

This is done by an embedding layer of a model. The model takes each token ID and looks it up in an embedding matrix. The embedding matrix is a learned set of weights that maps each token ID to a corresponding high-dimensional vector (embedding).

$$\text{Woman} \longrightarrow \begin{bmatrix} 1.2 \\ 0.4 \\ 3.6 \\ 5.0 \\ 3.9 \\ 8.5 \\ 2.1 \\ 0.6 \\ 7.0 \\ 8.2 \\ 4.2 \\ 9.8 \\ 3.0 \\ \vdots \\ 1.1 \\ 4.3 \\ 2.7 \\ 3.3 \\ 0.0 \end{bmatrix}$$

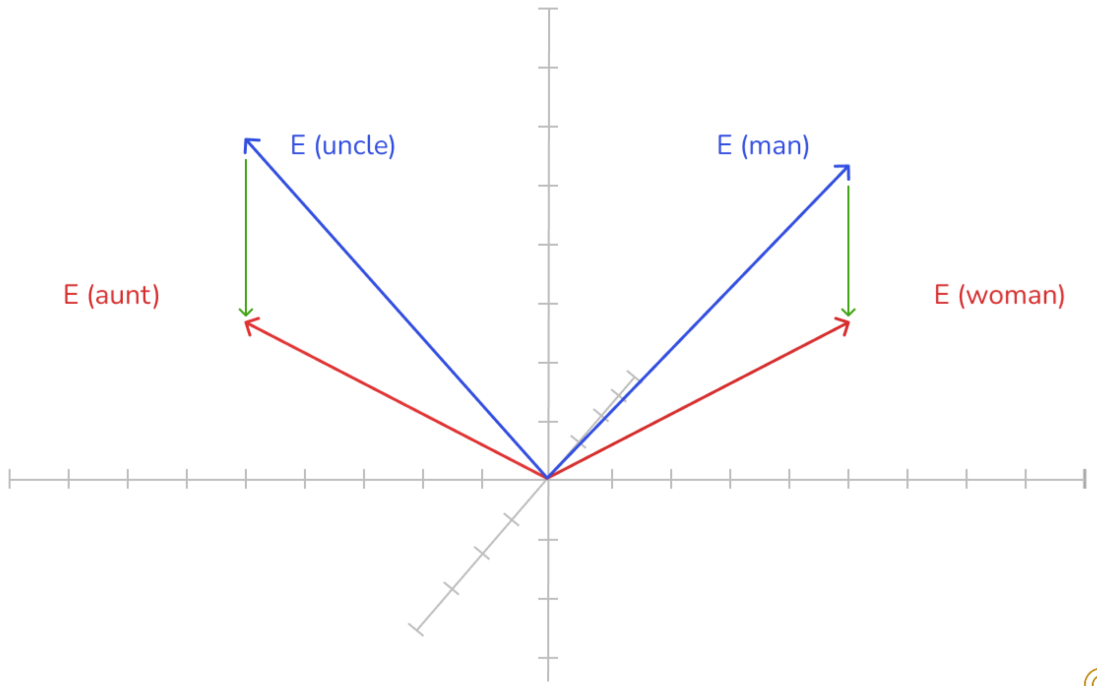
From Tokens to Vectors

Embeddings

Each word/token has a unique direction in the embedding space

Similar words point in a similar direction.

$$E\,(\text{aunt}) - E\,(\text{uncle}) \approx E\,(\text{woman}) - E\,(\text{man})$$



THANK YOU

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101101903. The JU receives support from the Digital Europe Programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia