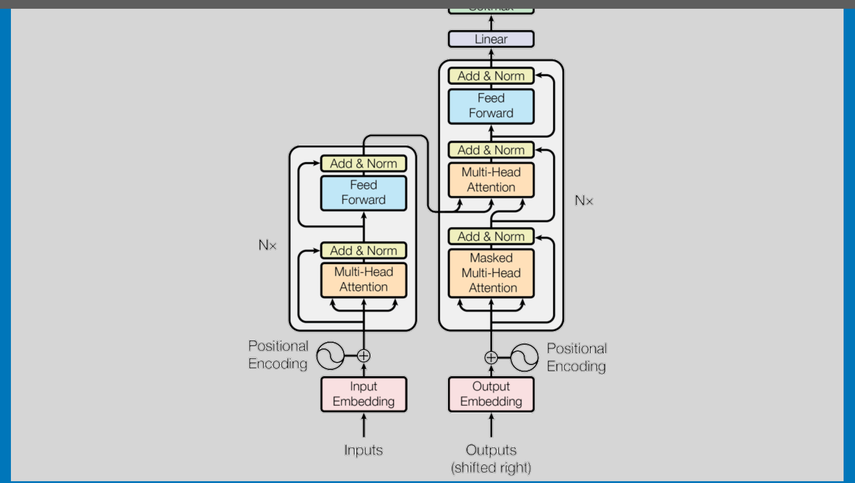
Wie funktioniert ChatGPT? Gibt es das auch als Open Source?

Michael Christen
Maintainer YaCy.net & SUSI.ai

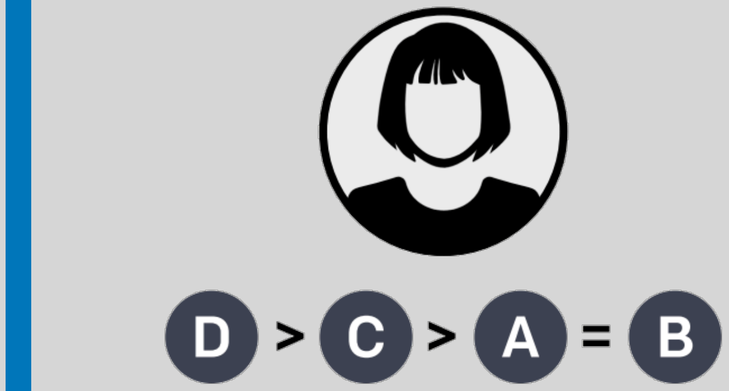
Email mc@yacy.net
Mastodon @orbiterlab@sigmoid.social
X/Twitter @orbiterlab
Github https://github.com/orbiter
Youtube https://youtube.com/@orbiterlab
Subscribe https://patreon.com/orbiterlab

Funktionsweise von Chat-LLMs

Zeitreise: 120 Jahre zuvor



Transformer

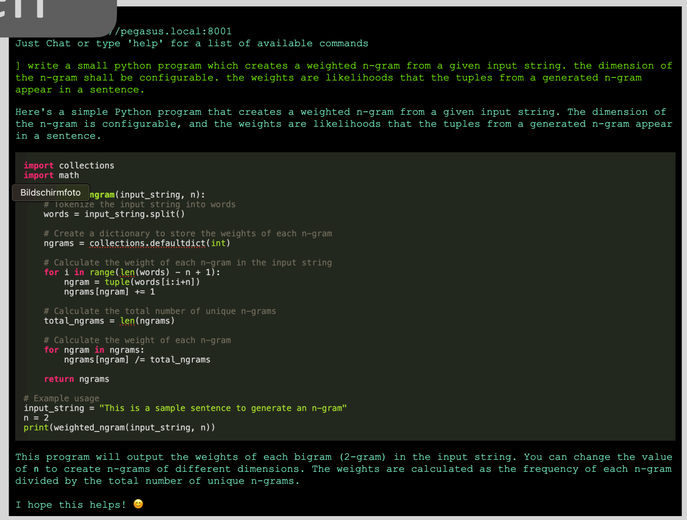


Instruct-Training

Freie LLM Modelle und Anwendungen

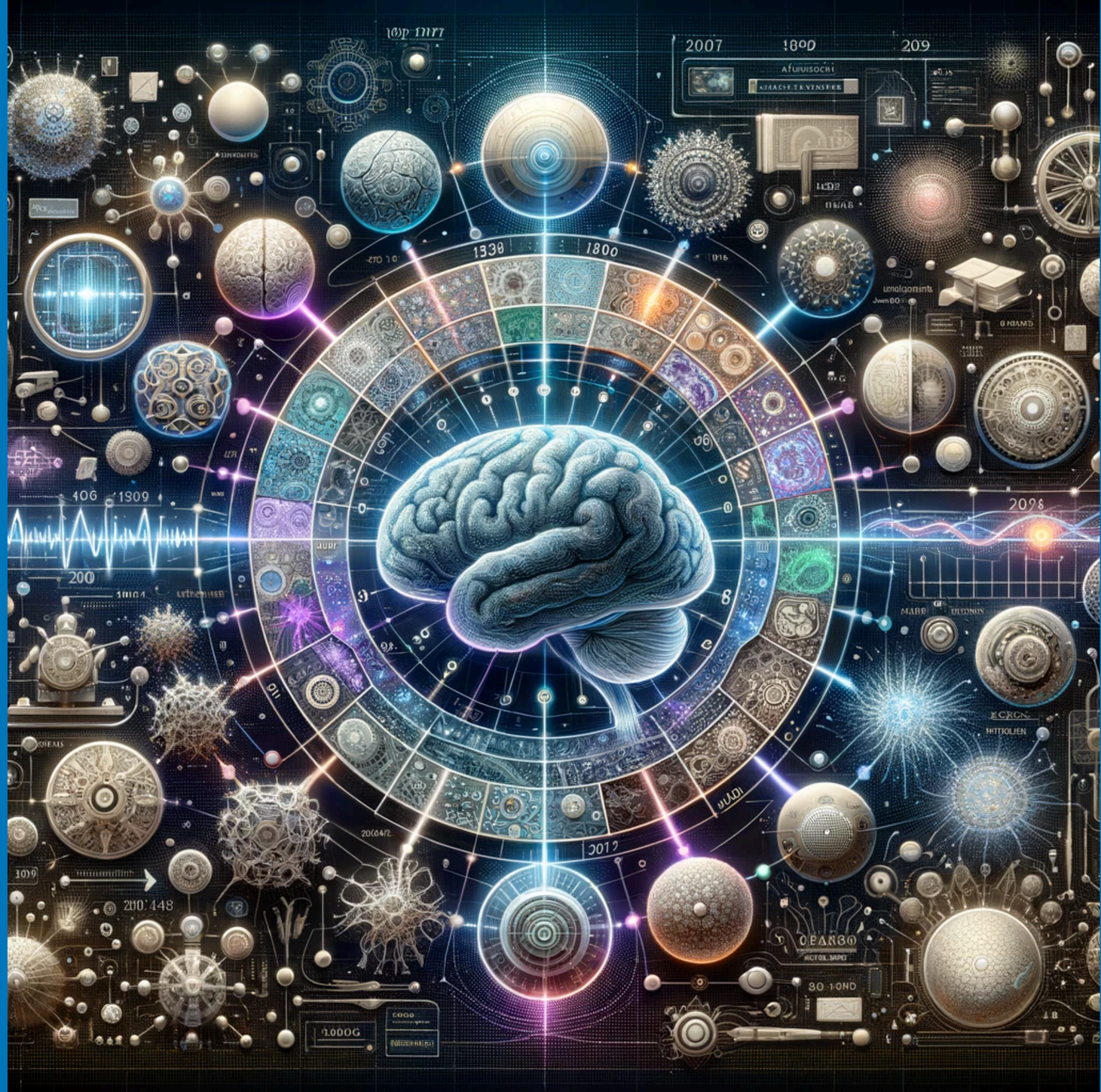
Open Source nach ChatGPT

LoRa & RAG



OpenAI API & Chat-Clients

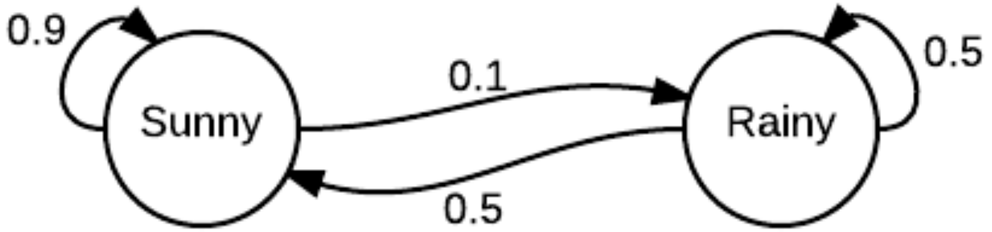
Zeitreise
durch 120 Jahre
KI-Entwicklung



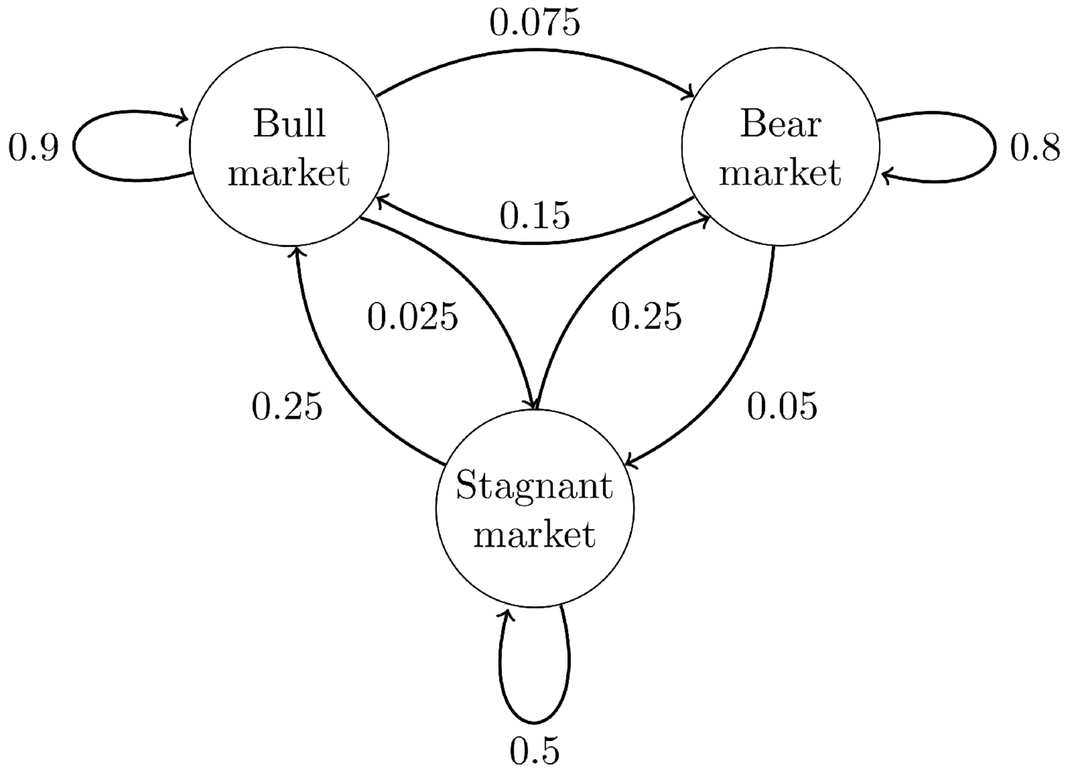
1906

Markov Ketten

Wettervorhersage



Aktien Markt Vorhersage



1906: Andrey Markov showed that under certain conditions the average outcomes of the Markov chain would converge to a fixed vector of values.

https://en.wikipedia.org/wiki/Examples_of_Markov_chains

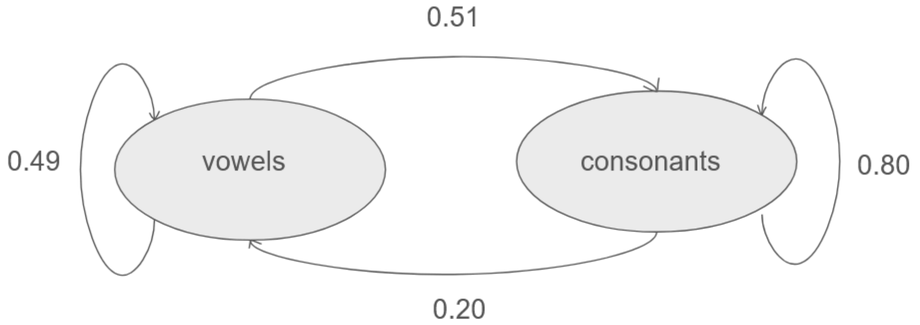
1913

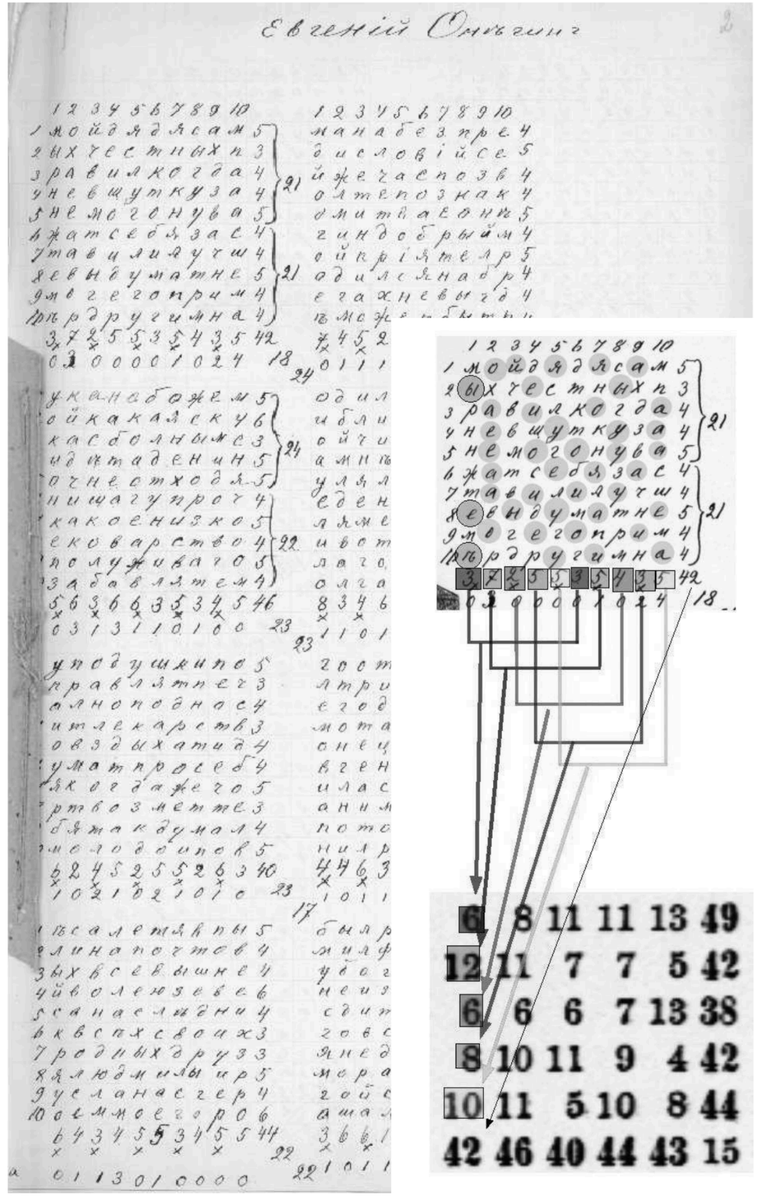
Markov Language Model

1913 Markov used Alexander Pushkin's 19th century verse novel „Eugene Onegin" to make to make a language model.

Text Structure Predictions

„An Example of Statistical Investigation of the Text Eugene Onegin Concerning the Connection of Samples in Chains"





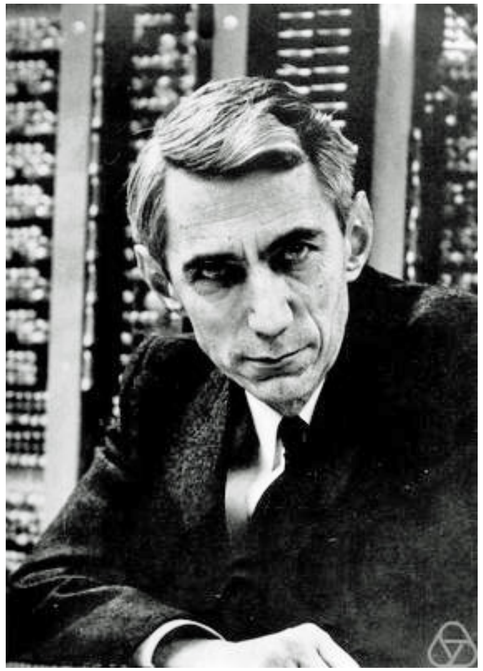
https://spectrum.ieee.org/andrey-markov-and-claude-shannon-built-the-first-language-generation-models

http://www.alpha60.de/research/markov/DavidLink_AnExampleOfStatistical_MarkovTrans_2007.pdf

https://www.math.purdue.edu/~yipn/talks/108-MarkovChain-Fall2019.pdf

1948

Claude Shannon Informationstheorie



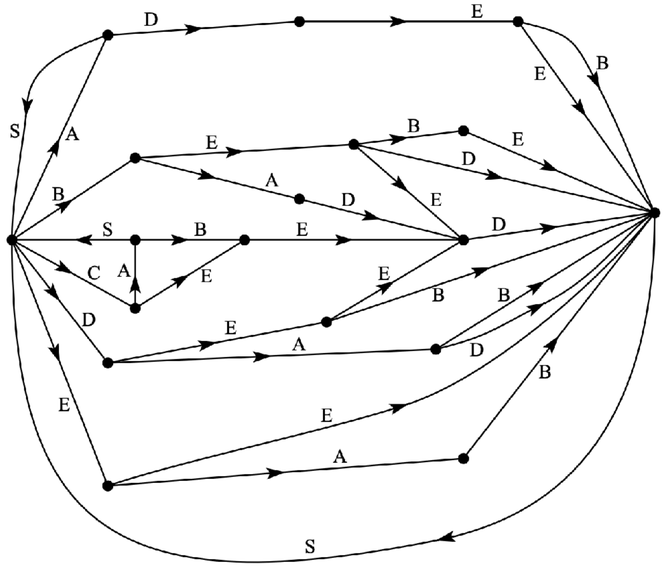
„A Mathematical Theory of Communication“

Stochastic processes can also be defined which produce a text consisting of a sequence of "words." Suppose there are five letters A, B, C, D, E and 16 "words" in the language with associated probabilities:

.10 A	.16 BEBE	.11 CABED	.04 DEB
.04 ADEB	.04 BED	.05 CEED	.15 DEED
.05 ADEE	.02 BEED	.08 DAB	.01 EAB
.01 BADD	.05 CA	.04 DAD	.05 EE

Suppose successive "words" are chosen independently and are separated by a space. A typical message might be:

DAB EE A BEBE DEED DEB ADEE ADEE EE DEB BEBE BEBE BEBE ADEE BED DEED DEED CEED ADEE A DEED DEED BEBE CABED BEBE BED DAB DEED ADEB.



Shannon stellte eine Methode vor, die Informationsmenge in einer Nachricht durch komplexere statistische Modelle genau zu messen.

Mittels des Markov-Prozesses und eines N-Gramms schuf er ein Modell der englischen Sprache, welches auf Wahrscheinlichkeiten des gemeinsamen Auftretens von Buchstaben und Wörtern basierte, und dadurch die Generierung von Sprache ermöglichte.

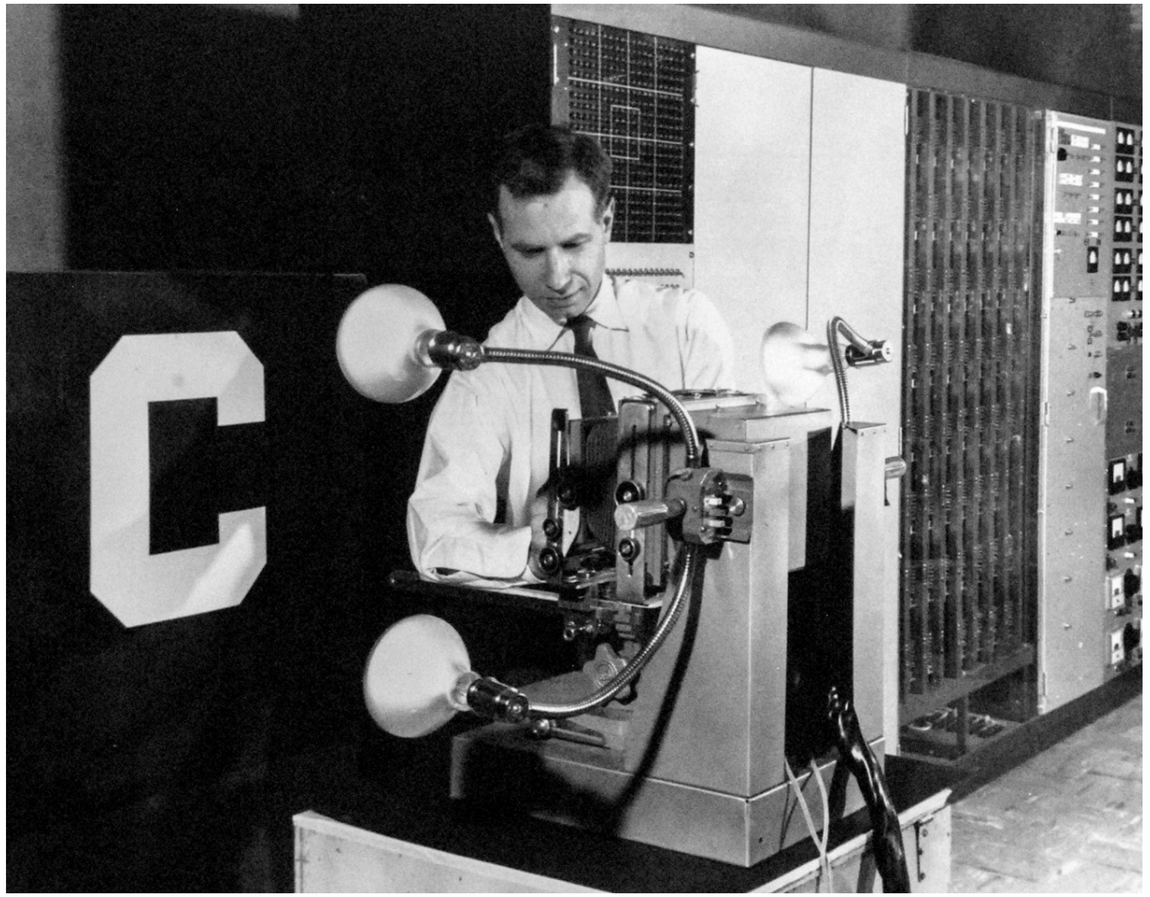
1948 Claude Shannon skizzierte eine Möglichkeit, um die Informationsmenge in einer Nachricht präzise zu messen. Er verwendete den Markov-Prozess und ein N-Gramm, um ein Sprachmodell zu repräsentieren.

https://spectrum.ieee.org/andrey-markov-and-claude-shannon-built-the-first-language-generation-models

https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

1957

The Perceptron



1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

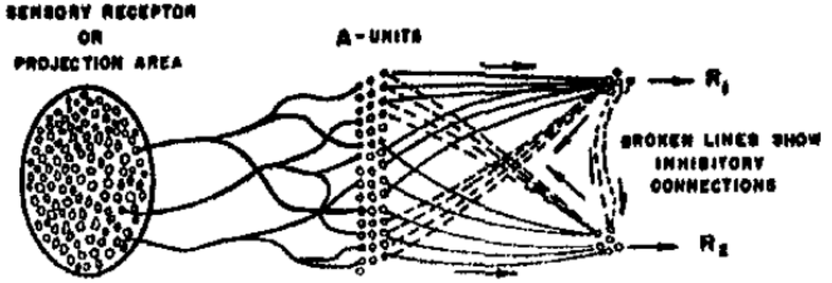
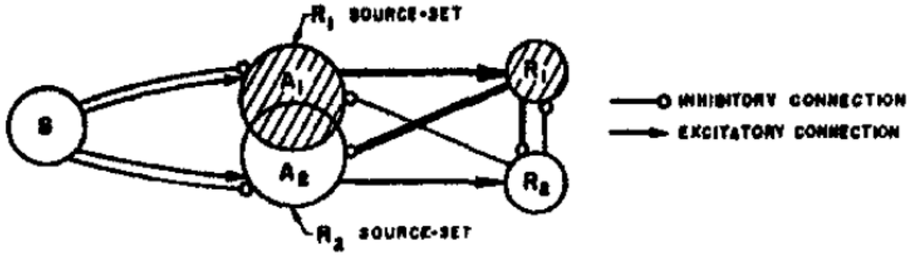


FIG. 2A. Schematic representation of connections in a simple perceptron.



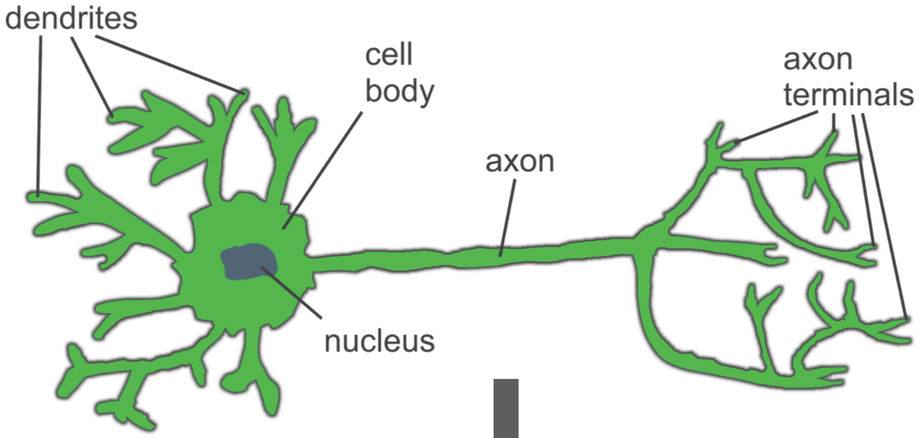
Frank Rosenblatt:

1957 - „The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain“

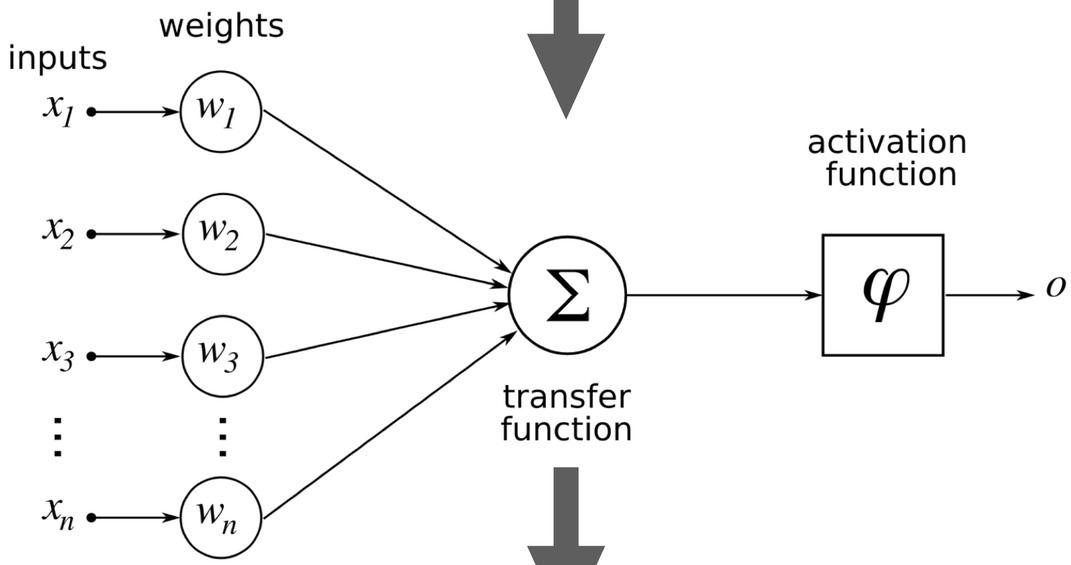
1960 - Mark 1 Perceptron

http://134.208.26.59/INA/A%20probabilistic%20model.pdf

Neuronale Netze



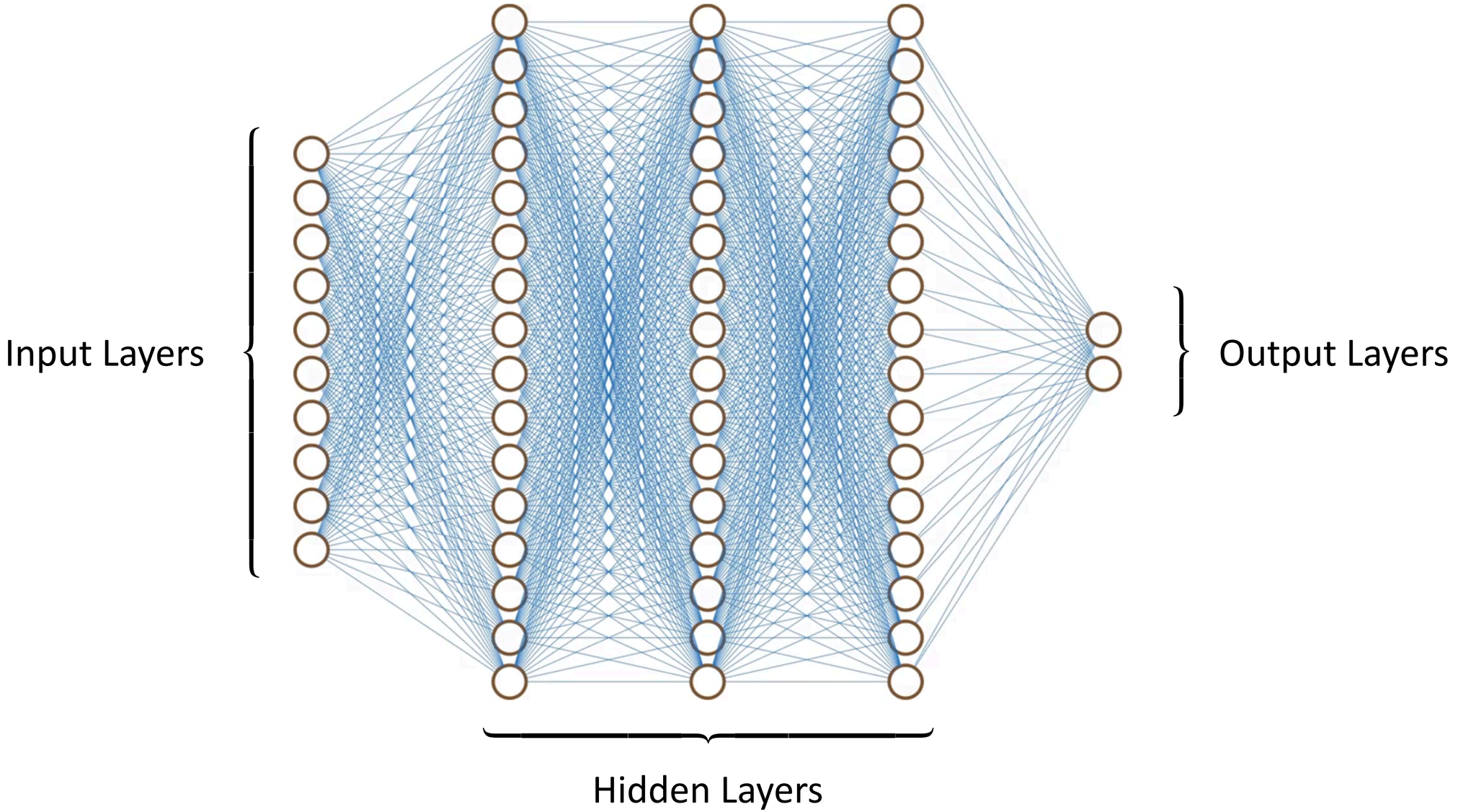
from: nnfs.io



$$o = \varphi\left(\sum_{i=1}^{n} x_i w_i\right)$$

from: https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz

Neuronale Netze



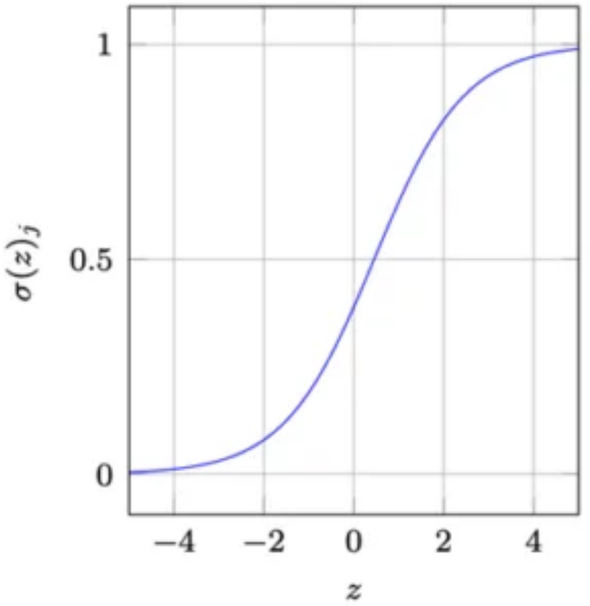
from: nnfs.io

Softmax Activation

$$\sigma : \mathbb{R}^K \to \left\{ z \in \mathbb{R}^K \mid z_i \geq 0, \sum_{i=1}^{K} z_i = 1 \right\}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{für } j = 1, \ldots, K.$$

from: https://de.wikipedia.org/wiki/Softmax-Funktion



softmax

```
[-1.7629,    -inf,    -inf,    -inf,    -inf,    -inf,    -inf,    -inf],
[-3.3334, -1.6556,    -inf,    -inf,    -inf,    -inf,    -inf,    -inf],
[-1.0226, -1.2606,  0.0762,    -inf,    -inf,    -inf,    -inf,    -inf],
[ 0.7836, -0.8014, -0.3368, -0.8496,    -inf,    -inf,    -inf,    -inf],
[-1.2566,  0.0187, -0.7880, -1.3204,  2.0363,    -inf,    -inf,    -inf],
[-0.3126,  2.4152, -0.1106, -0.9931,  3.3449, -2.5229,    -inf,    -inf],
[ 1.0876,  1.9652, -0.2621, -0.3158,  0.6091,  1.2616, -0.5484,    -inf],
[-1.8044, -0.4126, -0.8306,  0.5899, -0.7987, -0.5856,  0.6433,  0.6303]
```

=

```
[1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
[0.1574, 0.8426, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
[0.2088, 0.1646, 0.6266, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
[0.5792, 0.1187, 0.1889, 0.1131, 0.0000, 0.0000, 0.0000, 0.0000],
[0.0294, 0.1052, 0.0469, 0.0276, 0.7909, 0.0000, 0.0000, 0.0000],
[0.0176, 0.2689, 0.0215, 0.0089, 0.6812, 0.0019, 0.0000, 0.0000],
[0.1691, 0.4066, 0.0438, 0.0416, 0.1048, 0.2012, 0.0329, 0.0000],
[0.0210, 0.0843, 0.0555, 0.2297, 0.0573, 0.0709, 0.2423, 0.2391]
```

- Für Ausgangswerte die Wahrscheinlichkeiten angeben sollen
- Die Summe der Werte in einem Vektor sind 1

1980

Das Chinesische Zimmer

Gedankenexperiment von John Searle:

In dem Gedankenexperiment zum Chinesischen Zimmer stellt man sich vor, eine Person sei in einem Raum eingeschlossen und erhalte chinesische Schriften, die sie nicht versteht, zusammen mit englischen Anweisungen, diese Symbole zu manipulieren.

Indem diese Person den Anweisungen folgt, kann sie Antworten produzieren, die von denen eines Muttersprachlers nicht zu unterscheiden sind, obwohl sie kein Chinesisch versteht.

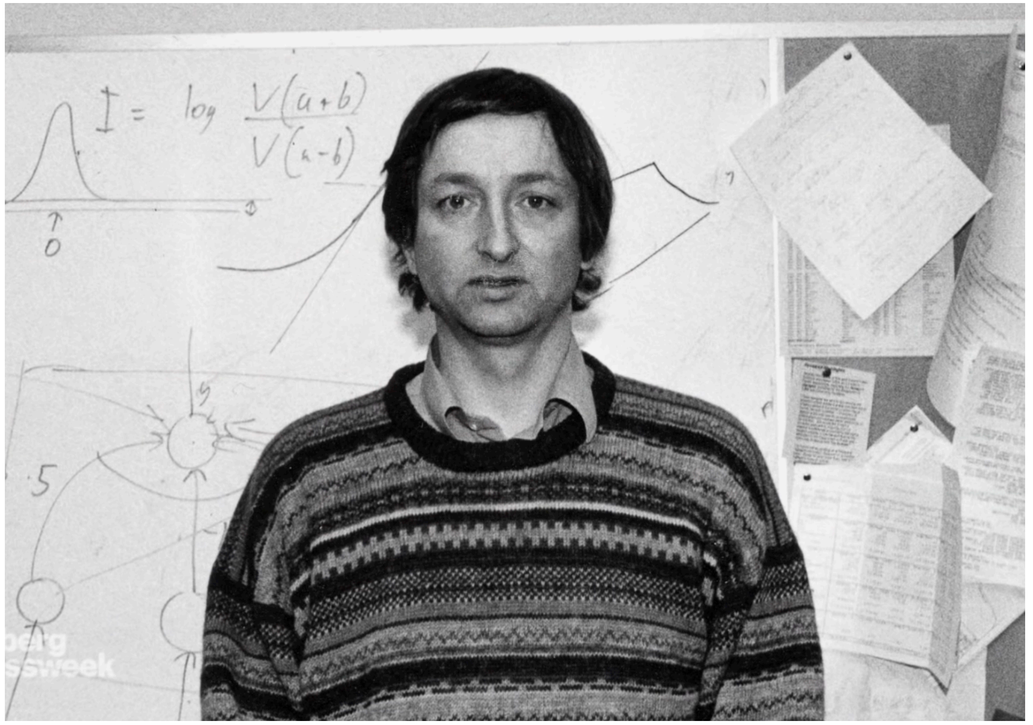
Dieses Szenario wird von John Searl verwendet, um zu argumentieren, dass bloße Symbolmanipulation nicht mit Verstehen oder Bewusstsein gleichzusetzen ist.

https://home.csulb.edu/~cwallis/382/readings/482/searle.minds.brains.programs.bbs.1980.pdf

Image: Dall.E 3

1986

Back-propagation & Autoencoder



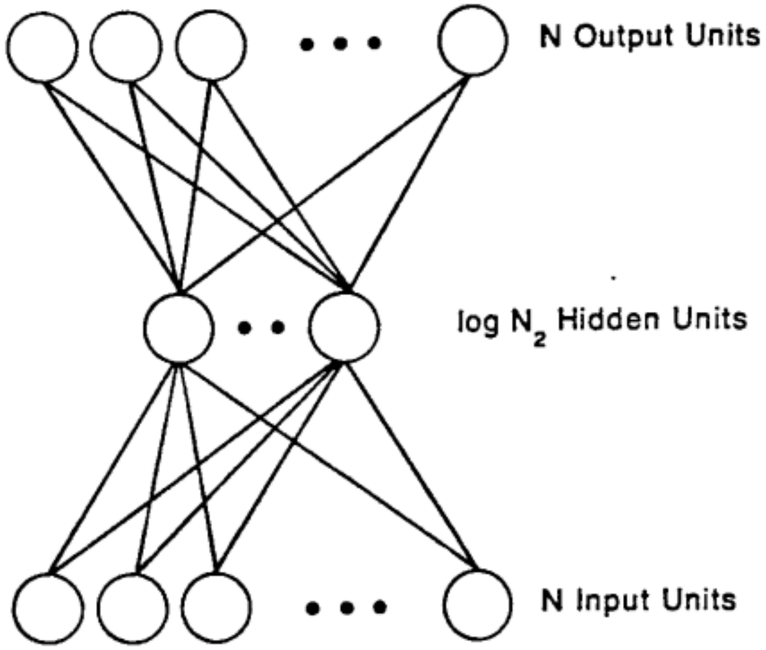


FIGURE 7. A network for solving the encoder problem. In this problem there are N orthogonal input patterns each paired with one of N orthogonal output patterns. There are only $\log N_2$ hidden units. Thus, if the hidden units take on binary values, the hidden units must form a binary number to encode each of the input patterns. This is exactly what the system learns to do.

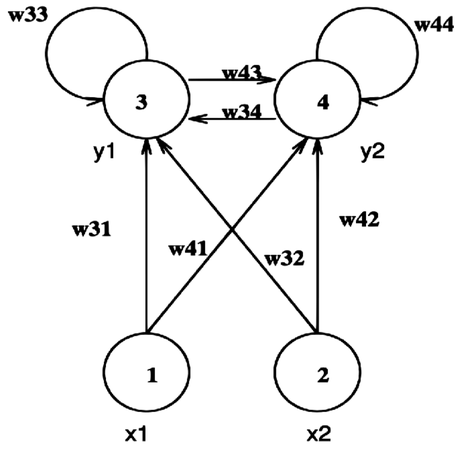
Geoffrey Hinton:
1986 - "Learning internal representations by error propagation" (mit David E. Rumelhart und Ronald J. Williams)
1988 - "Learning internal representations by back-propagating errors to adjust model connections" (mit Rumelhart, Williams)

https://cs.uwaterloo.ca/~y328yu/classics/bp.pdf
https://stanford.edu/~jlmcc/papers/PDP/Volume%201/Chap8_PDP86.pdf
https://www.cs.utoronto.ca/~hinton/absps/naturebp.pdf

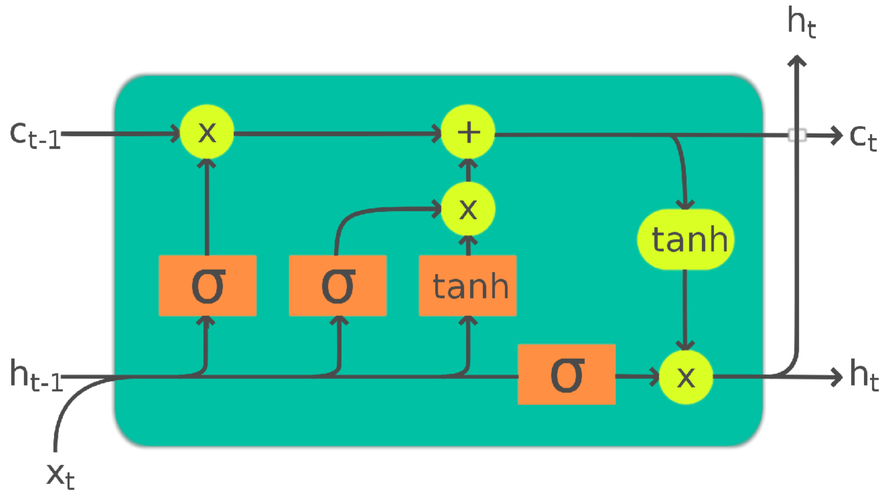
1995

Introduction of Long Short-Term Memory („LSTM“) Network Architecture

Recurrent Neural Networks (RNNs)



Recurrent Neural Network (RNN)
Example from Hochreiters
Diploma Thesis



The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time.

Jürgen Schmidhuber & Sepp Hochreiter

1991: Sepp Hochreiter analyzed the vanishing gradient problem

1995: "Long Short-Term Memory (LSTM)" by Hochreiter & Schmidhuber.

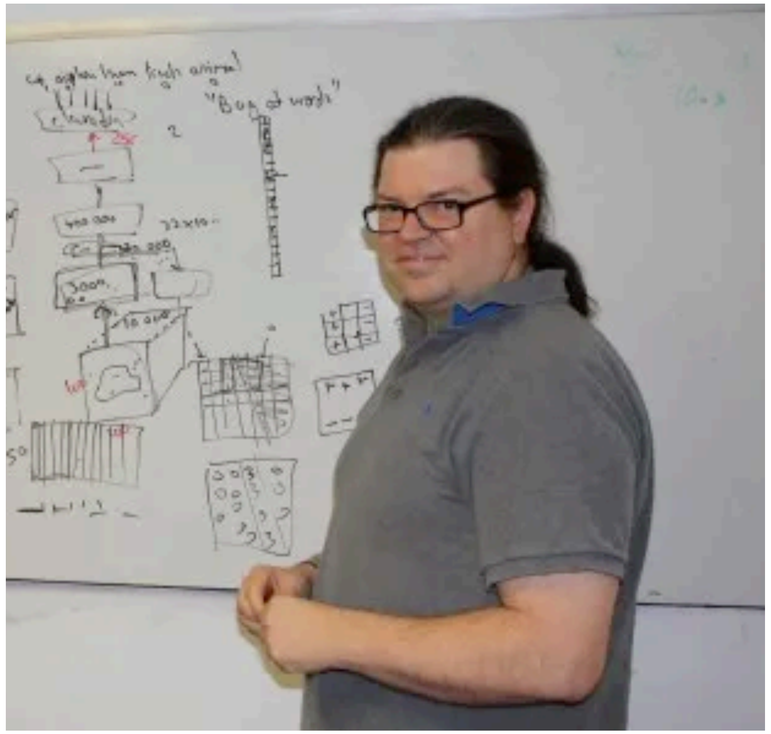
2015: Google is using LSTM for speech recognition

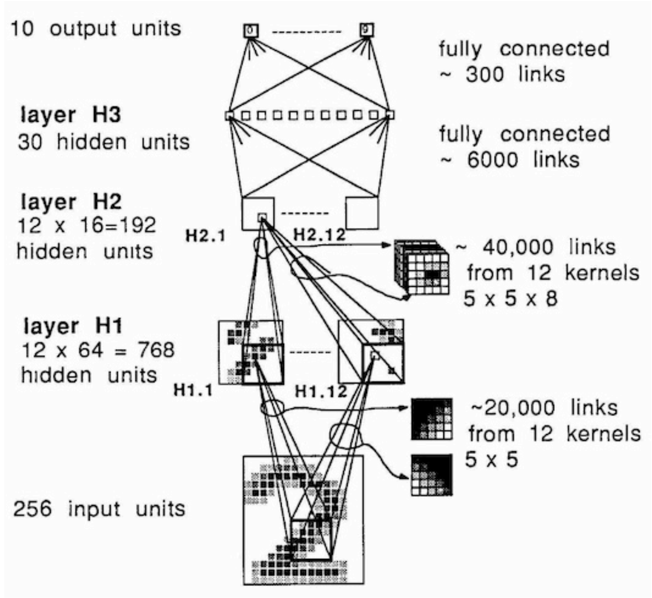
http://www.bioinf.jku.at/publications/older/3804.pdf

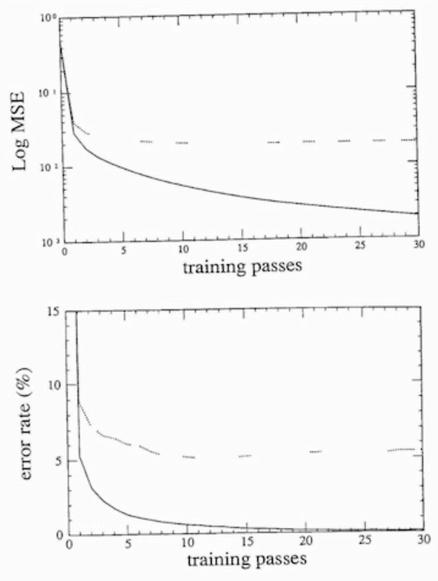
https://www.bioinf.jku.at/publications/older/3504.pdf

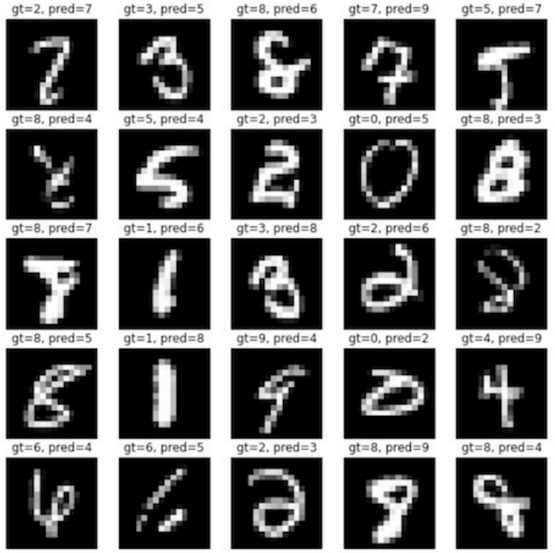
1998

Convolutional Networks (CNNs)









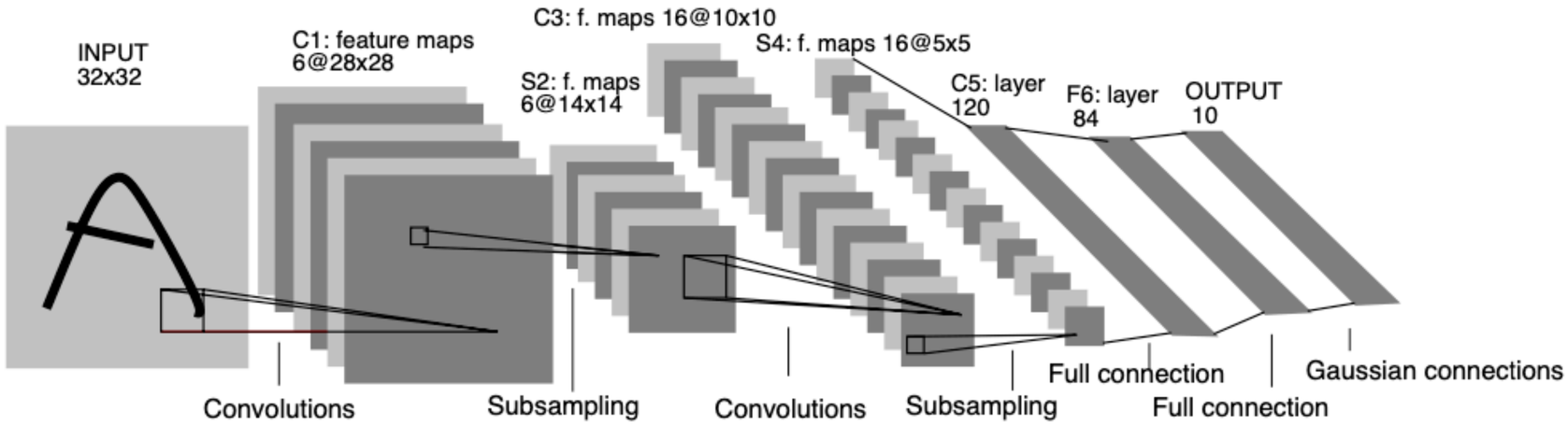


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

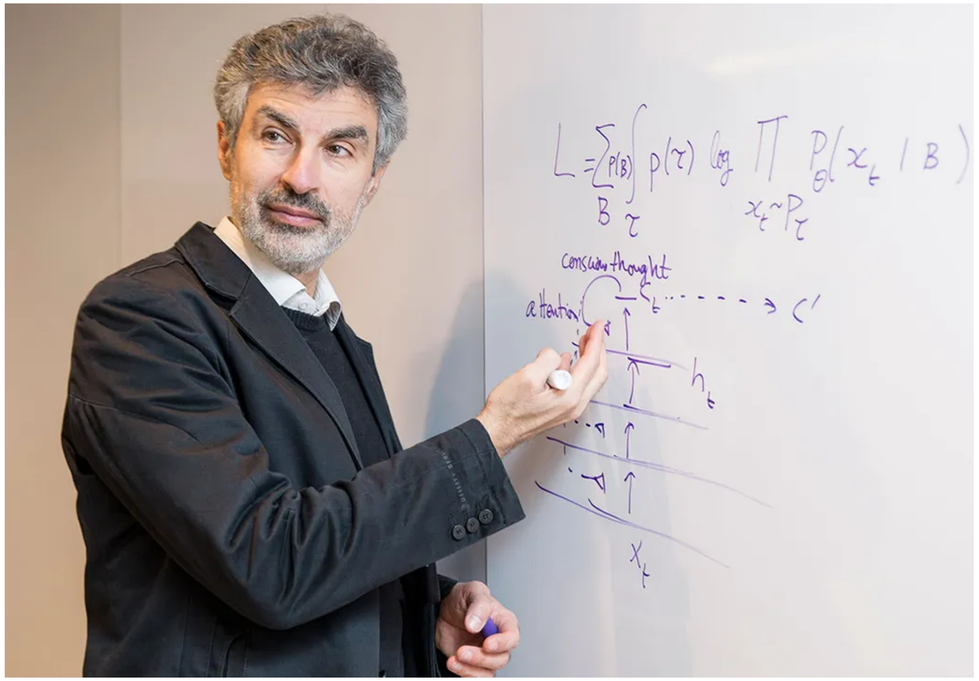
Yann LeCun:

1989 - „Backpropagation Applied to Handwritten Zip Code Recognition“ (mit Bose, Denker, Henderson, Howard, Hubbard, Jackel)
1998 - "GradientBased Learning Applied to Document Recognition" (mit Léon Bottom, Yoshua Bengio, Patrick Haffner)

https://karpathy.github.io/2022/03/14/lecun1989/

2002

Embeddings



1.1 Fighting the Curse of Dimensionality with Distributed Representations

In a nutshell, the idea of the proposed approach can be summarized as follows:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

The feature vector represents different aspects of the word: each word is associated with a point in a vector space. The number of features (e.g. m =30, 60 or 100 in the experiments) is much smaller than the size of the vocabulary (e.g. 17,000). The probability function is expressed as a product of conditional probabilities of the next word given the previous ones, (e.g. using a multi-layer neural network to predict the next word given the previous ones, in the experiments). This function has parameters that can be iteratively tuned in order to **maximize the log-likelihood of the training data** or a regularized criterion, e.g. by adding a weight decay penalty.[2] The feature vectors associated with each word are learned, but they could be initialized using prior knowledge of semantic features.

Yoshua Bengio

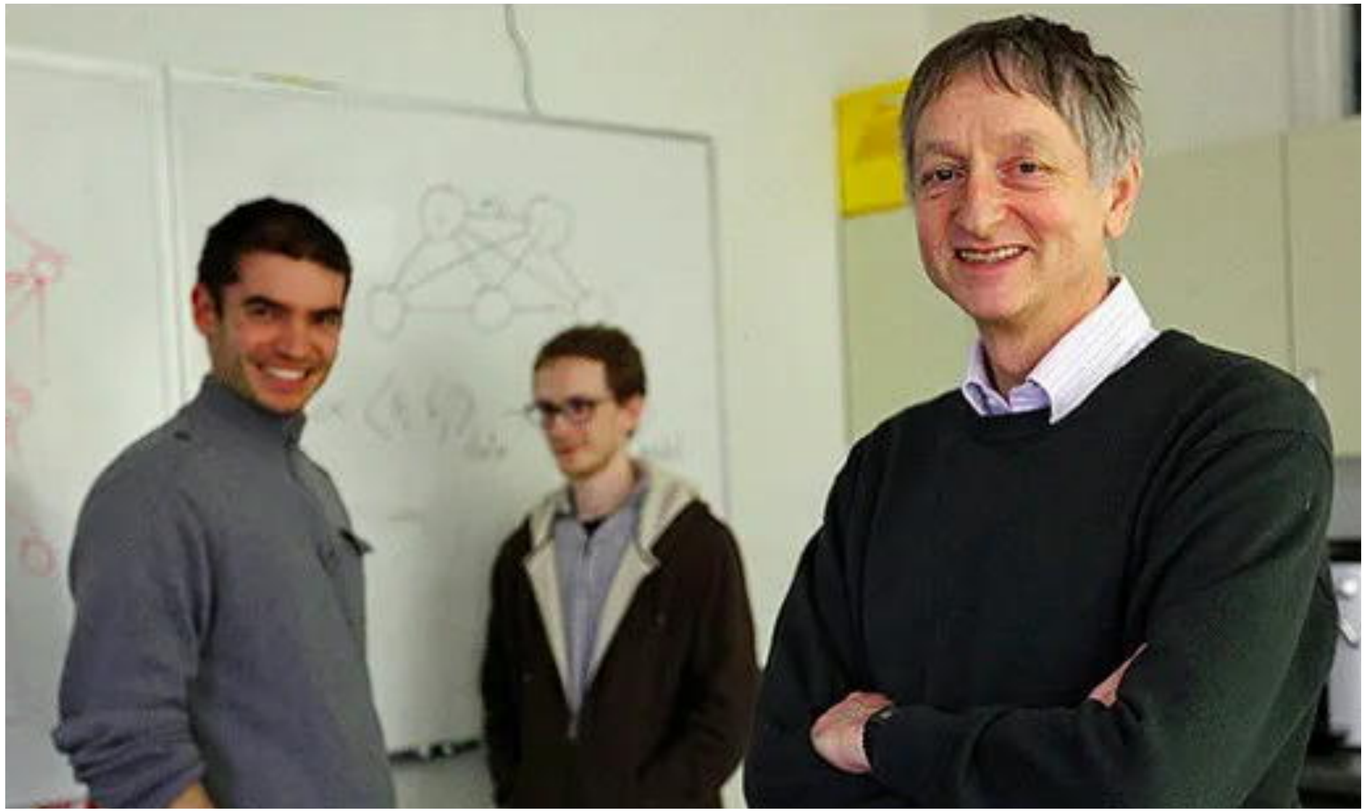
2002: „A Neural Probabilistic Language Model“ - Es wird das Konzept von Embeddings eingeführt - aber noch nicht so genannt. Im Paper werden diese „word feature vectors“ genannt. Wörter werden bedeutungsnah im Vektorraum angeordnet:

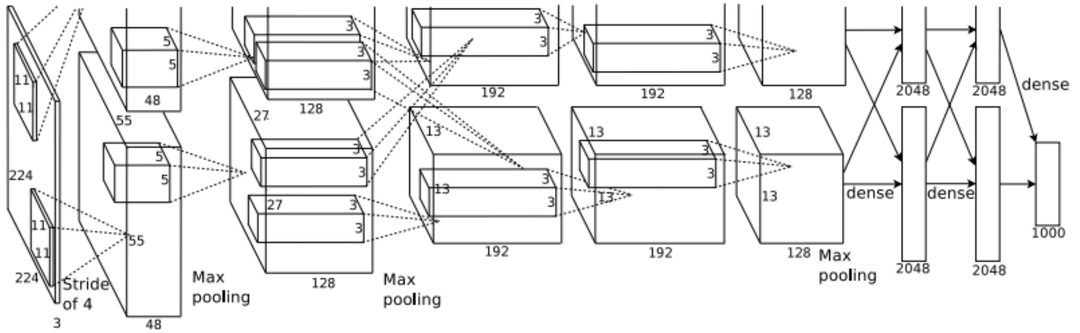
„We propose to fight the curse of dimensionality by learning a distributed representation for words which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations.“

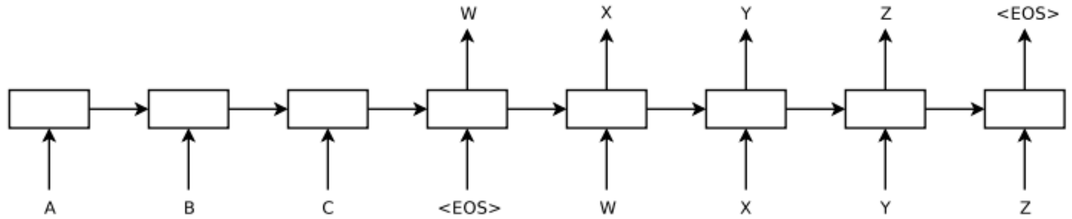
https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

2012

CNNs & Enhanced Performance of Neural Networks







Ilia Sutskever:
2012 - "ImageNet Classification with Deep Convolutional Neural Networks" (mit Alex Krizhevsky, Geoffrey E Hinton)
2014 - "Sequence to sequence learning with neural networks" (mit Oriol Vinyals, Quoc V. Le)
2014 - "Dropout: a simple way to prevent neural networks from overfitting" (mit Srivastava, Hinton, Krizhevsky, Salakhutdinov)

https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
https://arxiv.org/pdf/1409.3215.pdf
https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

2014

Attention mechanism for encoder-decoder

2014:
Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio
„Neural Machine Translation by Jointly Learning to Align and Translate“

First introduction of an attention mechanism: „this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. **By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed length vector.** With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.“

Attention Mechanism

3.1 Decoder: General Description

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1,\dots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i), \quad (4)$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1},y_{i-1},c_i).$$

It should be noted that unlike the existing encoder–decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.

The context vector c_i depends on a sequence of *annotations* $(h_1,\cdots,h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5)$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x}\exp(e_{ik})},$$

where

$$e_{ij} = a(s_{i-1},h_j)$$

is an *alignment model* which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i, Eq. (4)) and the j-th annotation h_j of the input sentence.

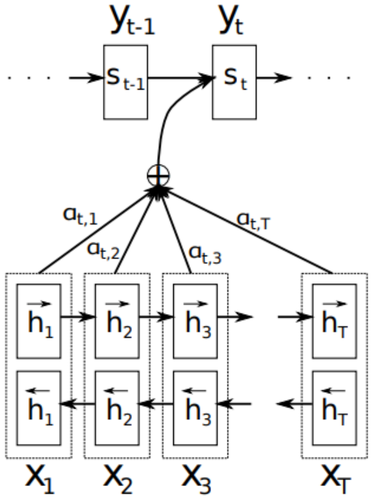


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1,x_2,\dots,x_T)$.

α_{ij} is a probability that the target word y_i is aligned to, or translated from, a source word x_j

e_{ij} is the energy associated to α_{ij}, implemented with an alignment model which is trained.

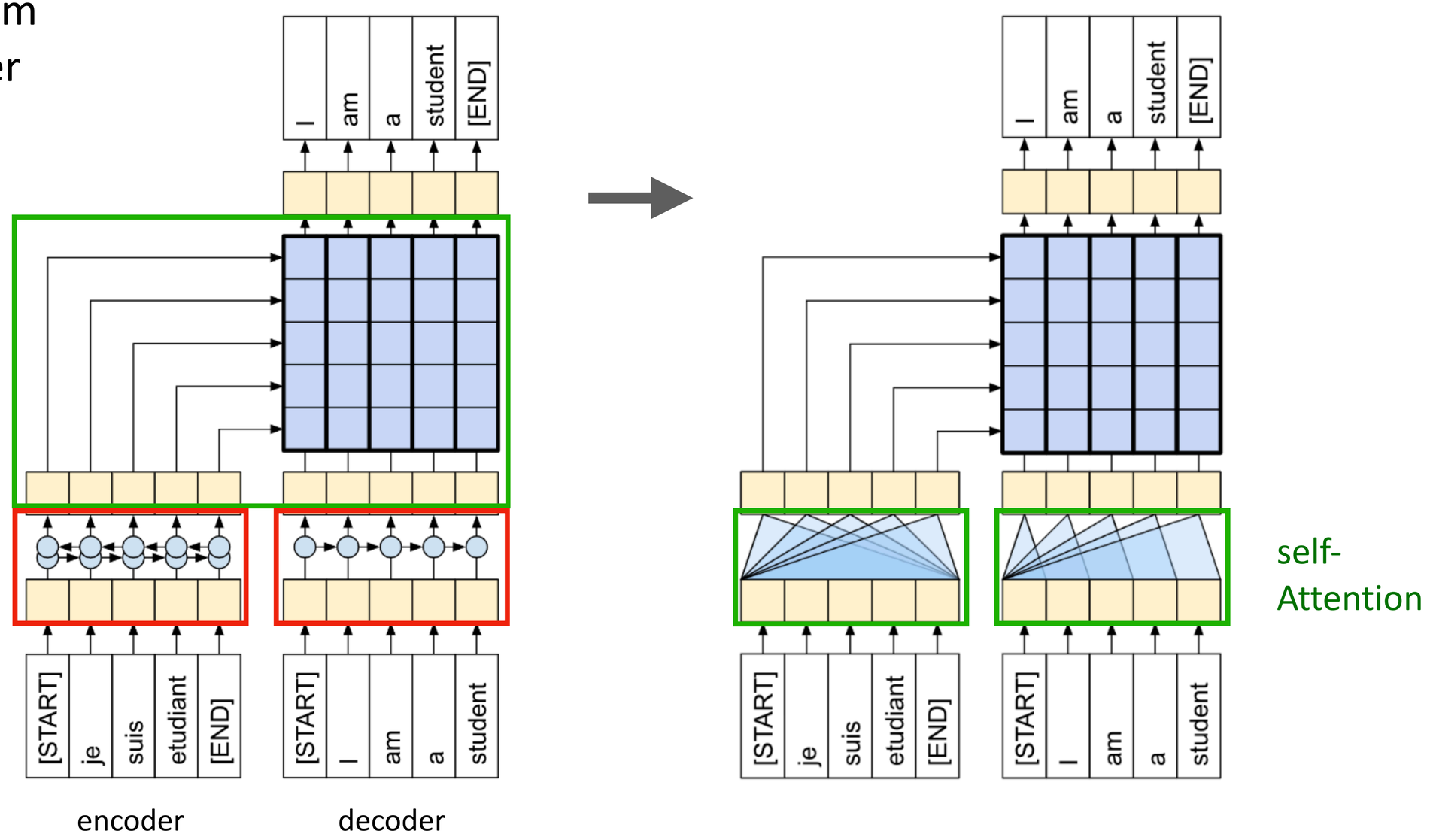
https://arxiv.org/pdf/1409.0473.pdf

2014

Attention mechanism for encoder-decoder

Attention

RNN



https://www.tensorflow.org/text/tutorials/transformer

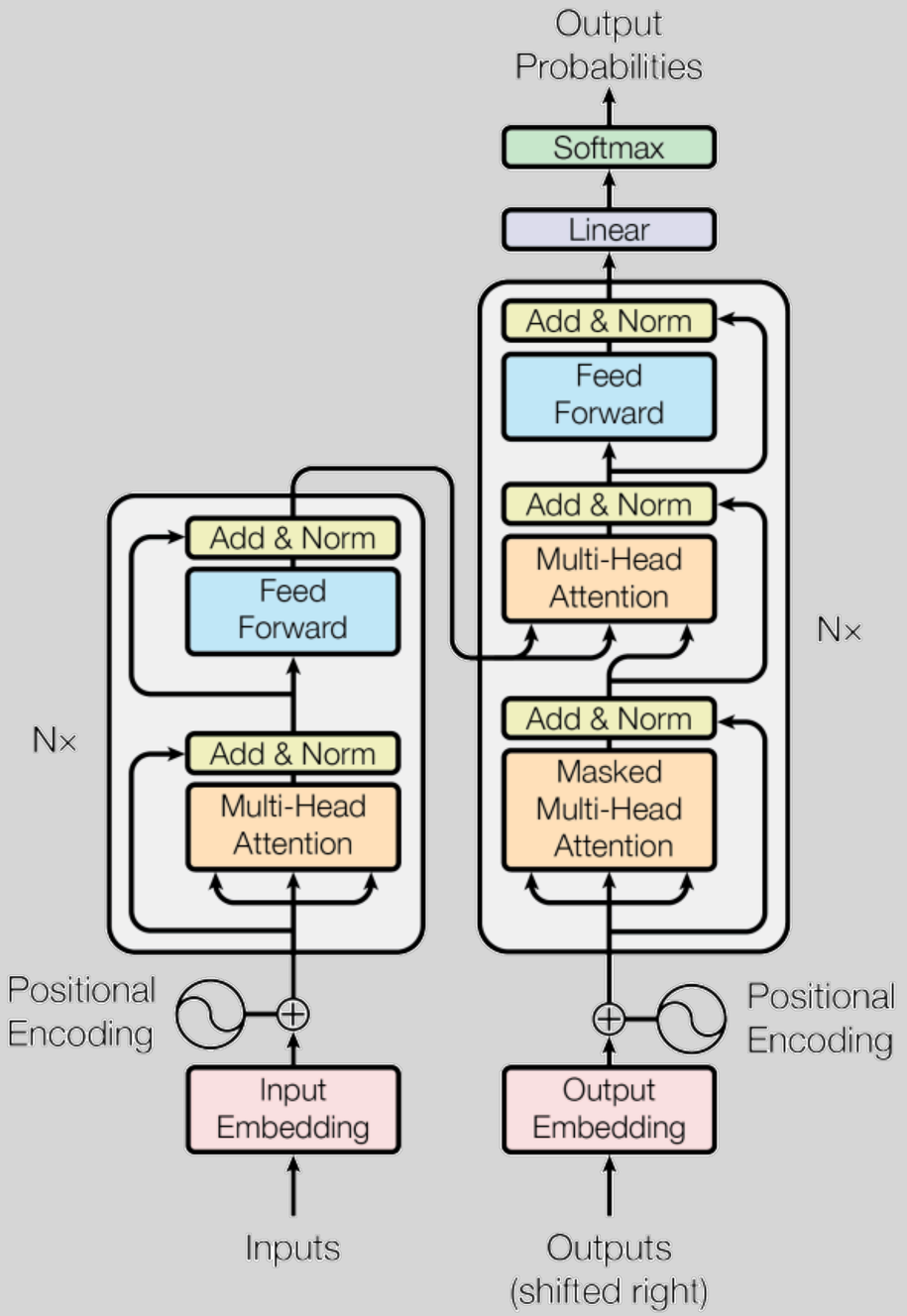
2018

Turing Award für Geoffrey Hinton, Yann LeCun und Yoshua Bengio

„Working independently and together, Hinton, LeCun and Bengio developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks.“

https://awards.acm.org/about/2018-turing

Transformer



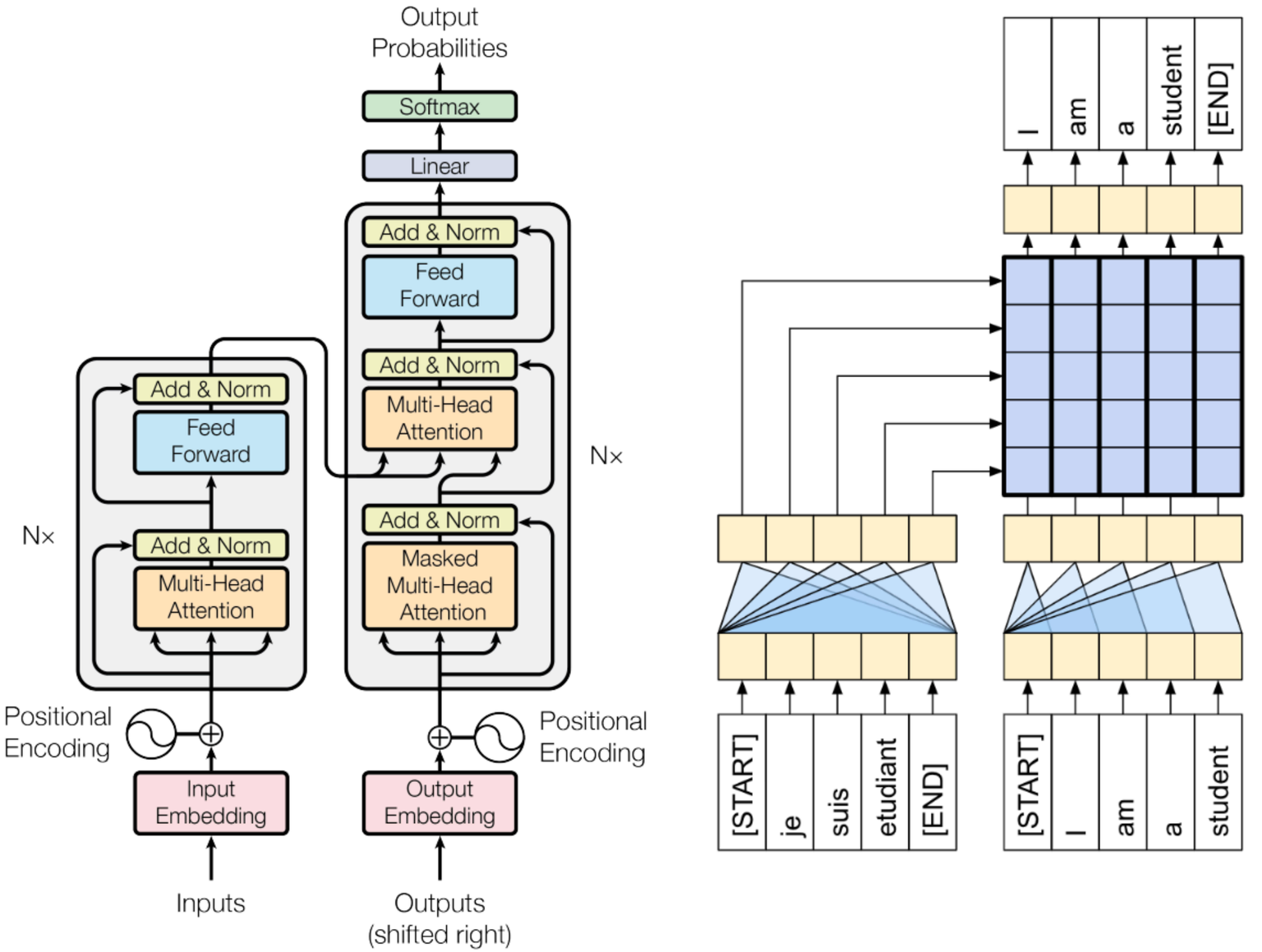
2017

„Attention is all you need“ Die Transformer Architektur

2017:
Google publishes a paper named „Attention is all you need“ to introduce the Transformer architecture:

„The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.“

Transformer



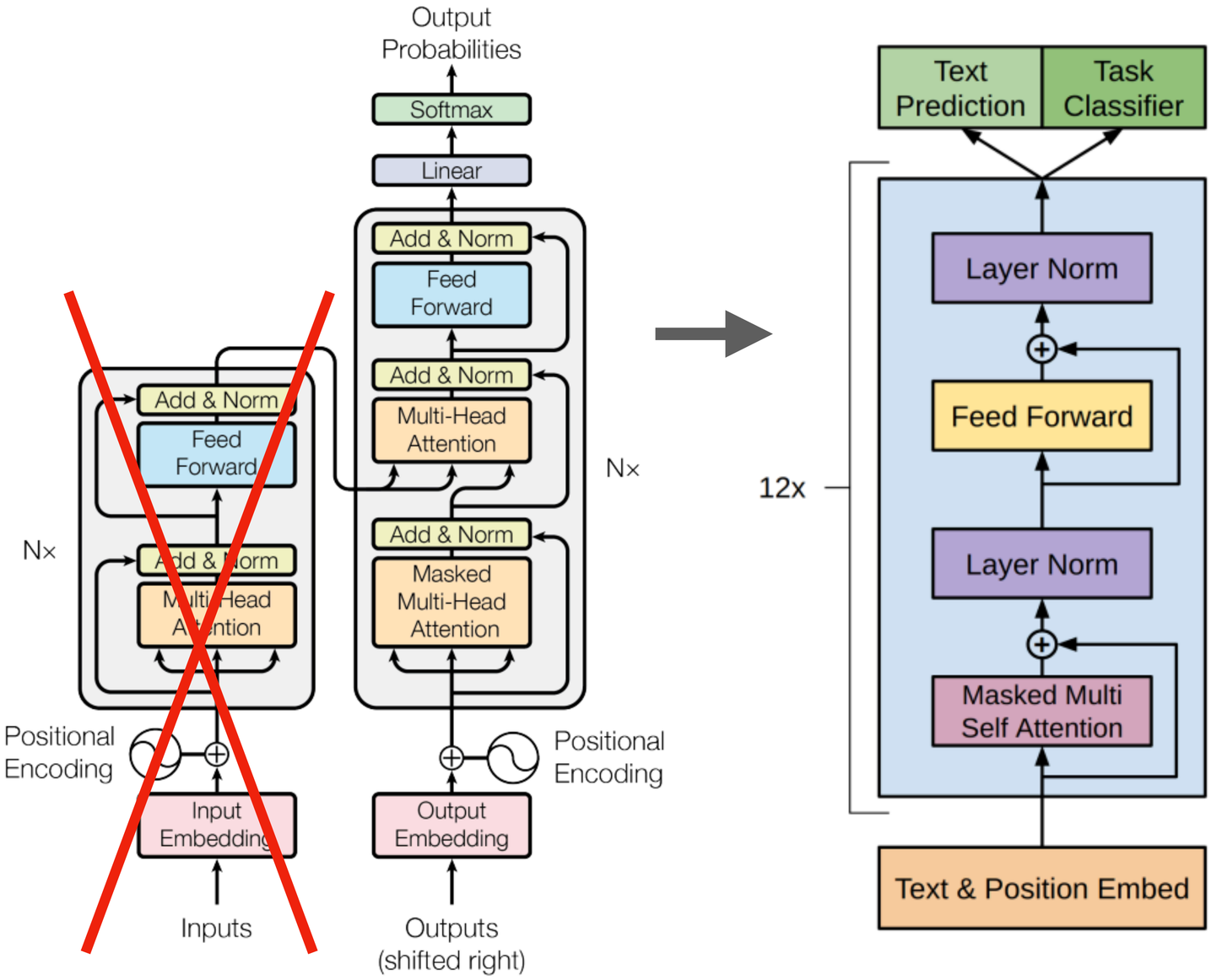
„Attention is all you need“ https://arxiv.org/pdf/1706.03762.pdf

2018

GPT-1

2018:
OpenAI publishes a paper named „Improving Language Understanding by Generative Pre-Training“ to introduce the first GPT model.

The paper describes a method for natural language understanding that uses a transformer model with only the decoder for text generation. The method involves two steps: pre-training and fine-tuning. In the pre-training step, a language model is trained on a large corpus of unlabeled text. In the fine-tuning step, the model is adapted to specific tasks using task-aware input transformations.



https://paperswithcode.com/paper/improving-language-understanding-by

Tokenization

GPT-3.5 & GPT-4 | GPT-3 (Legacy)

Die Chemnitzer Linux-Tage leben vor allem von der persönlichen Atmosphäre und dem direkten Kontakt zu vielen Linux-Fans. Das Rundumpaket entsteht nur durch den Einsatz zahlreicher freiwilliger helfender Hände.

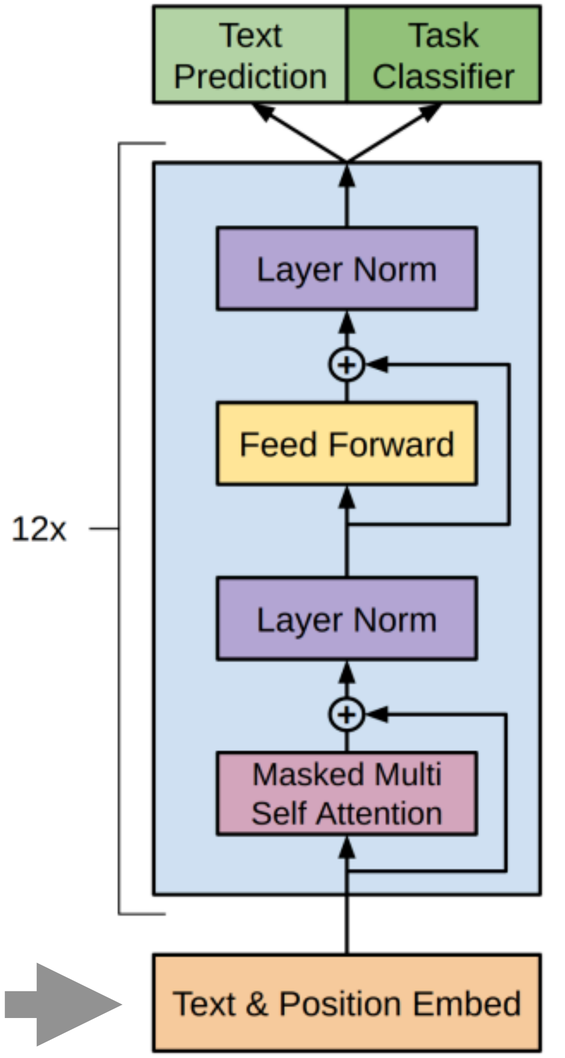
Clear | Show example

Tokens 52

Characters 209

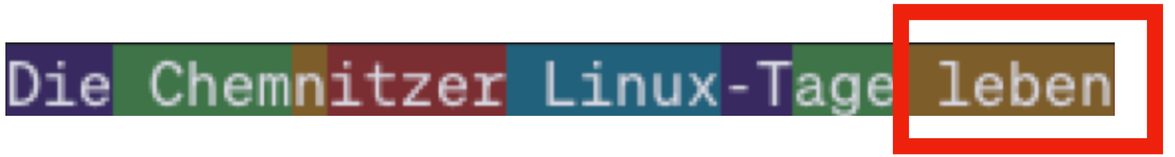
Die Chemnitzer Linux-Tage leben vor allem von der persönlichen Atmosphäre und dem direkten Kontakt zu vielen Linux-Fans. Das Rundumpaket entsteht nur durch den Einsatz zahlreicher freiwilliger helfender Hände.

[18674, 19531, 77, 21114, 14677, 9469, 425, 98972, 14230, 61397, 6675, 2761, 78520, 22412, 54928, 764, 47786, 2073, 2486, 13510, 74, 2002, 66708, 6529, 69142, 14677, 7424, 598, 13, 19537, 432, 1263, 1538, 62042, 1218, 5455, 427, 12500, 20350, 3453, 85347, 89728, 265, 29164, 84523, 14724, 7420, 75503, 1693, 473, 91460, 13]



https://platform.openai.com/tokenizer

Training

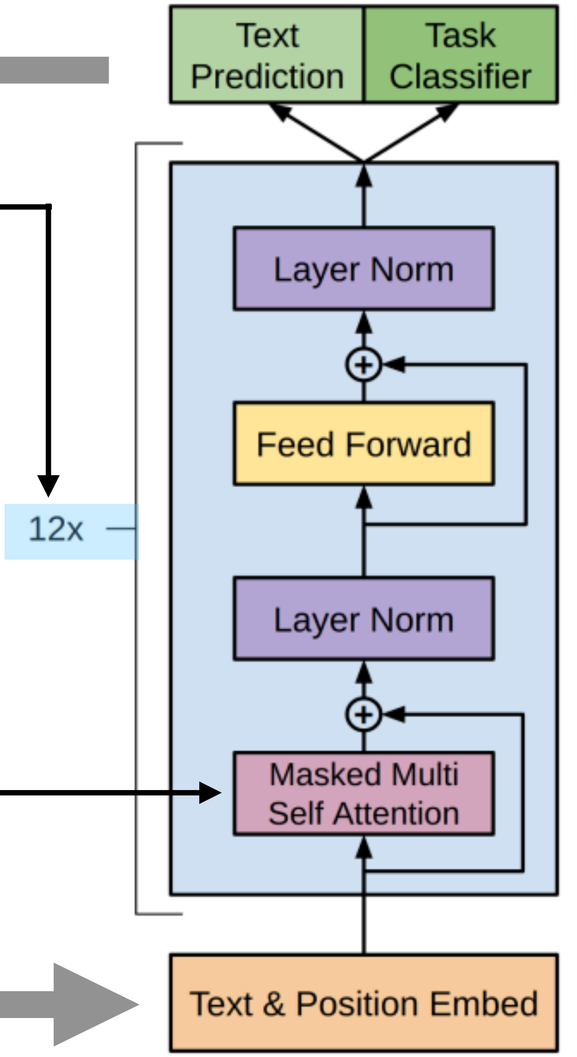



```
def __init__(
    self,
    vocab_file,
    merges_file,
    errors="replace",
    unk_token="<|endoftext|>",
    bos_token="<|endoftext|>",
    eos_token="<|endoftext|>",
    pad_token=None,
    add_prefix_space=False,
    add_bos_token=False,
    **kwargs,
):
```

https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/tokenization_gpt2.py

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
GPT-3 Small	125M	12	768	12	64
GPT-3 Medium	350M	24	1024	16	64
GPT-3 Large	760M	24	1536	16	96
GPT-3 XL	1.3B	24	2048	24	128
GPT-3 2.7B	2.7B	32	2560	32	80
GPT-3 6.7B	6.7B	32	4096	32	128
GPT-3 13B	13.0B	40	5140	40	128
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128

GPT-3 Paper https://arxiv.org/pdf/2005.14165.pdf



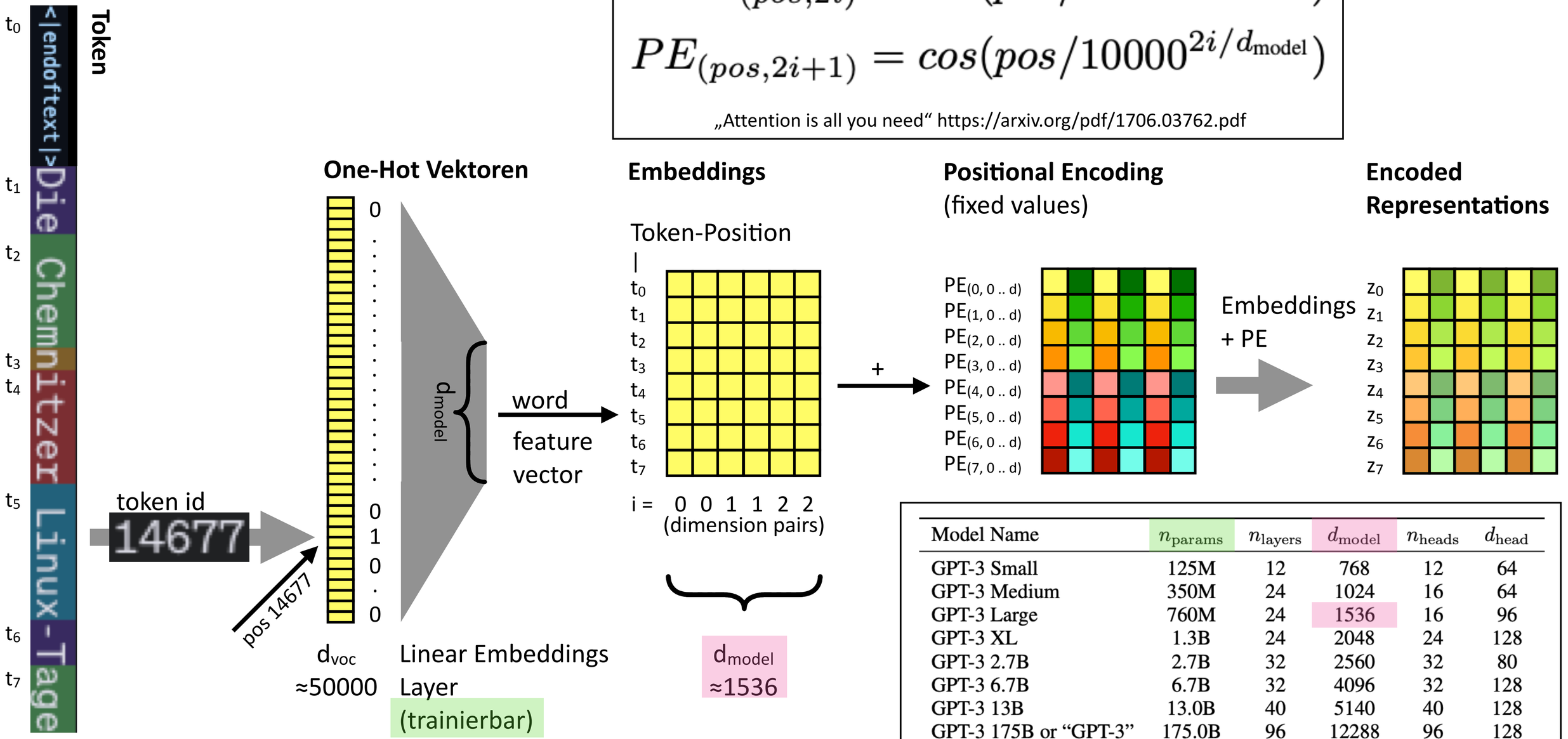
Der Transformer/Decoder wird so trainiert, dass er als Ausgabe die gleichen Tokens wie in der Eingabe liefert, aber um eine Stelle verschoben und mit einem zusätzlichen Token als Ergänzung. Der Text beginnt immer mit einem Starttoken, in GPT-2 ist dieser gleich mit dem Endtoken.

Embeddings & Positional Encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

„Attention is all you need“ https://arxiv.org/pdf/1706.03762.pdf

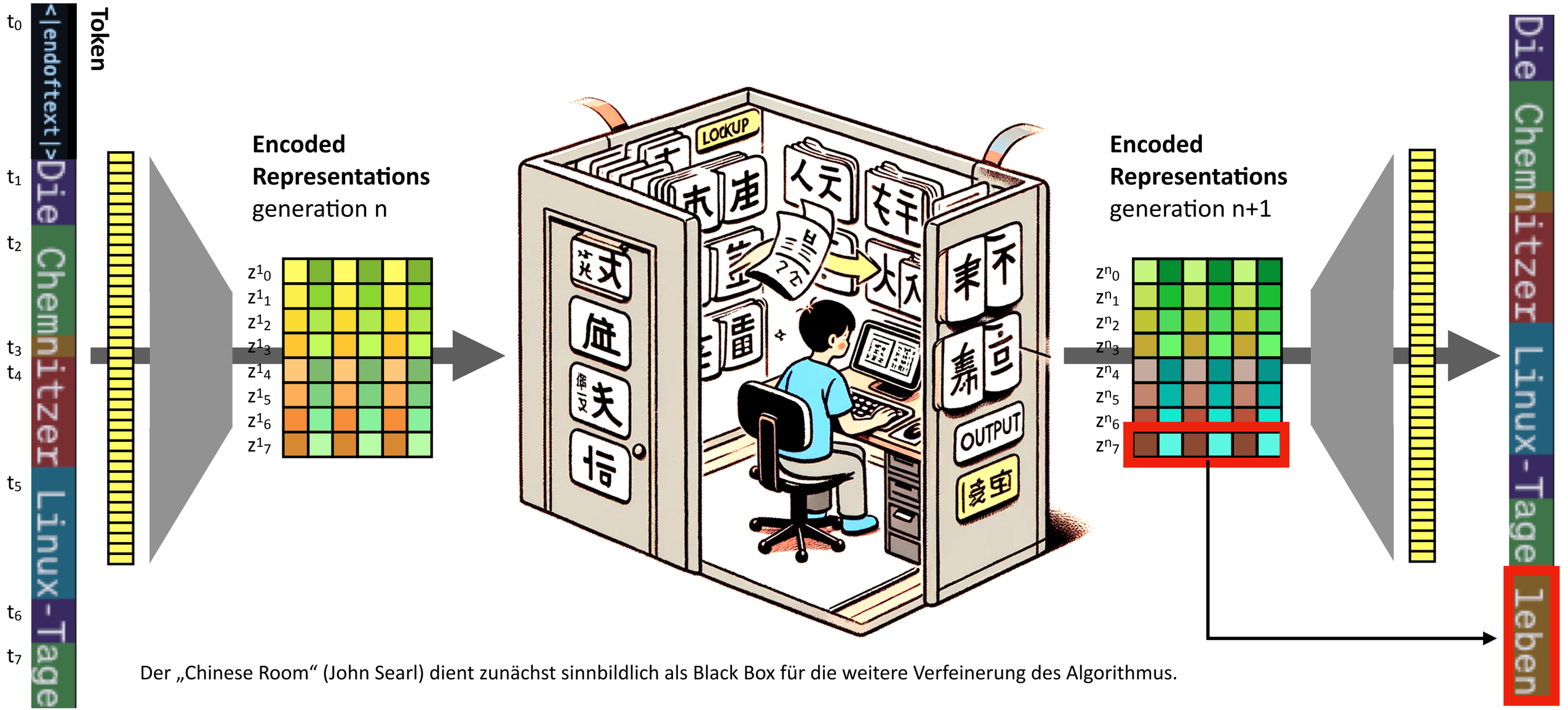


Token werden als hochdimensionale One-Hot Vektoren mit Embeddings abgebildet und mit Positional Encoding mittels Sinus- und Kosinusfunktionen ergänzt.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
GPT-3 Small	125M	12	768	12	64
GPT-3 Medium	350M	24	1024	16	64
GPT-3 Large	760M	24	1536	16	96
GPT-3 XL	1.3B	24	2048	24	128
GPT-3 2.7B	2.7B	32	2560	32	80
GPT-3 6.7B	6.7B	32	4096	32	128
GPT-3 13B	13.0B	40	5140	40	128
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128

GPT-3 Paper https://arxiv.org/pdf/2005.14165.pdf

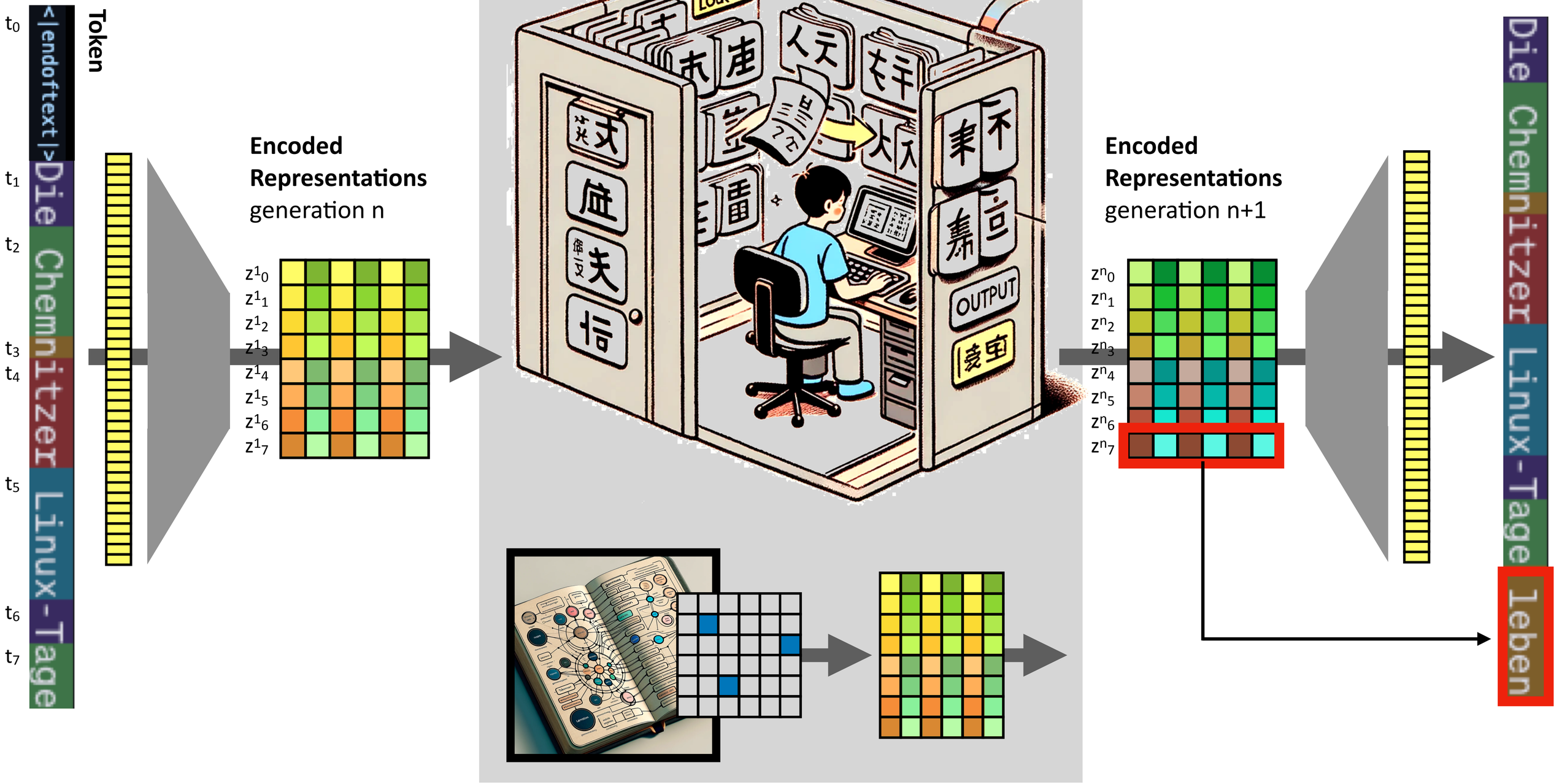
Textgenerierung: Schema



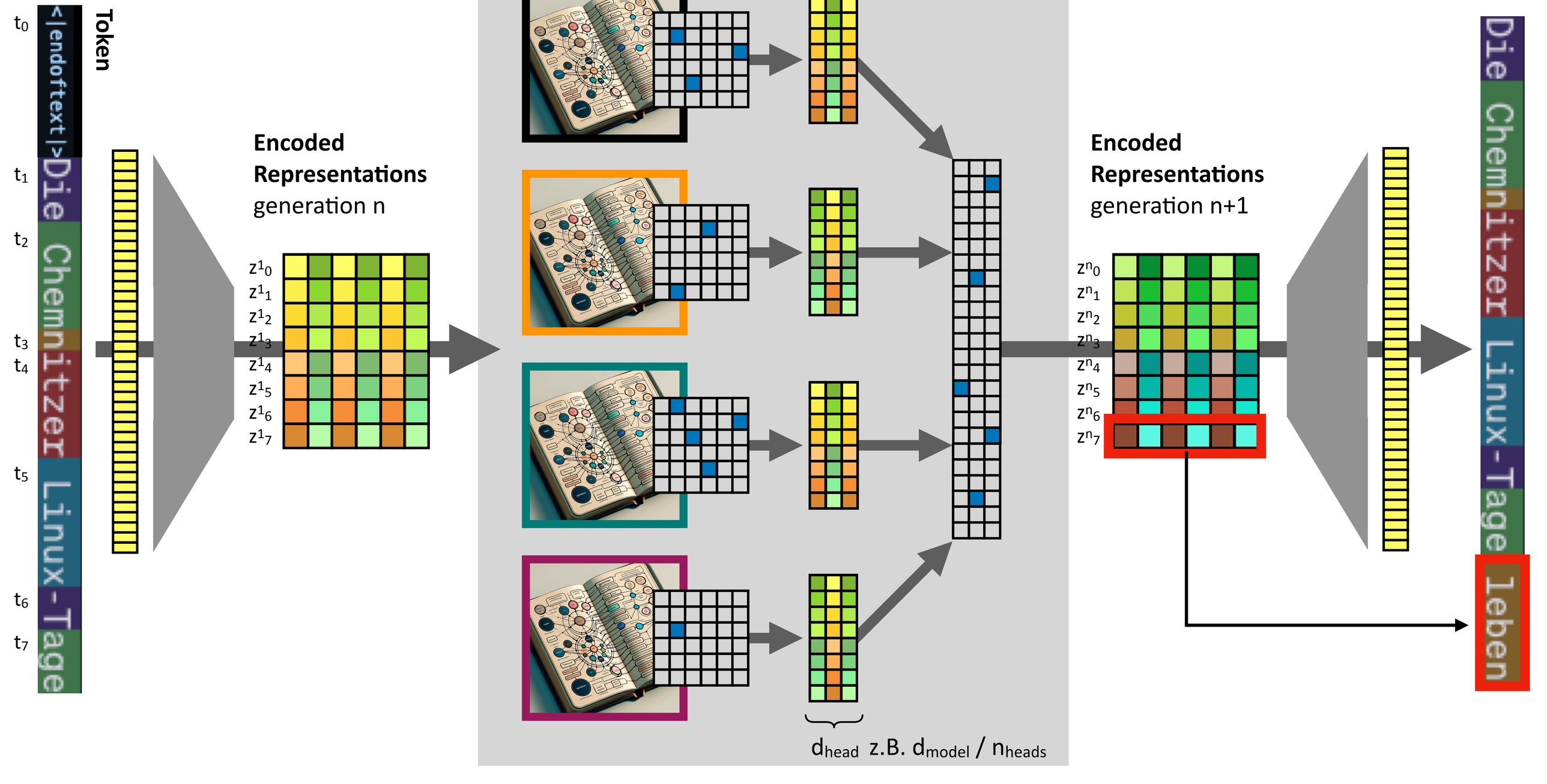
Der „Chinese Room“ (John Searl) dient zunächst sinnbildlich als Black Box für die weitere Verfeinerung des Algorithmus.

Mit einem Reverse Embedding wird das letzte Element des letzten generierten Encoded Representations wieder in einen hochdimensionalen One-Hot Vektor übersetzt und diesen zum Selektieren eines Tokens verwendet. Dieser Token ergänzt den Text. Der neue Text wird wieder als Eingabe für die Generierung des nächsten Tokens verwendet.

Textgenerierung: Schema



Multi-Head Attention

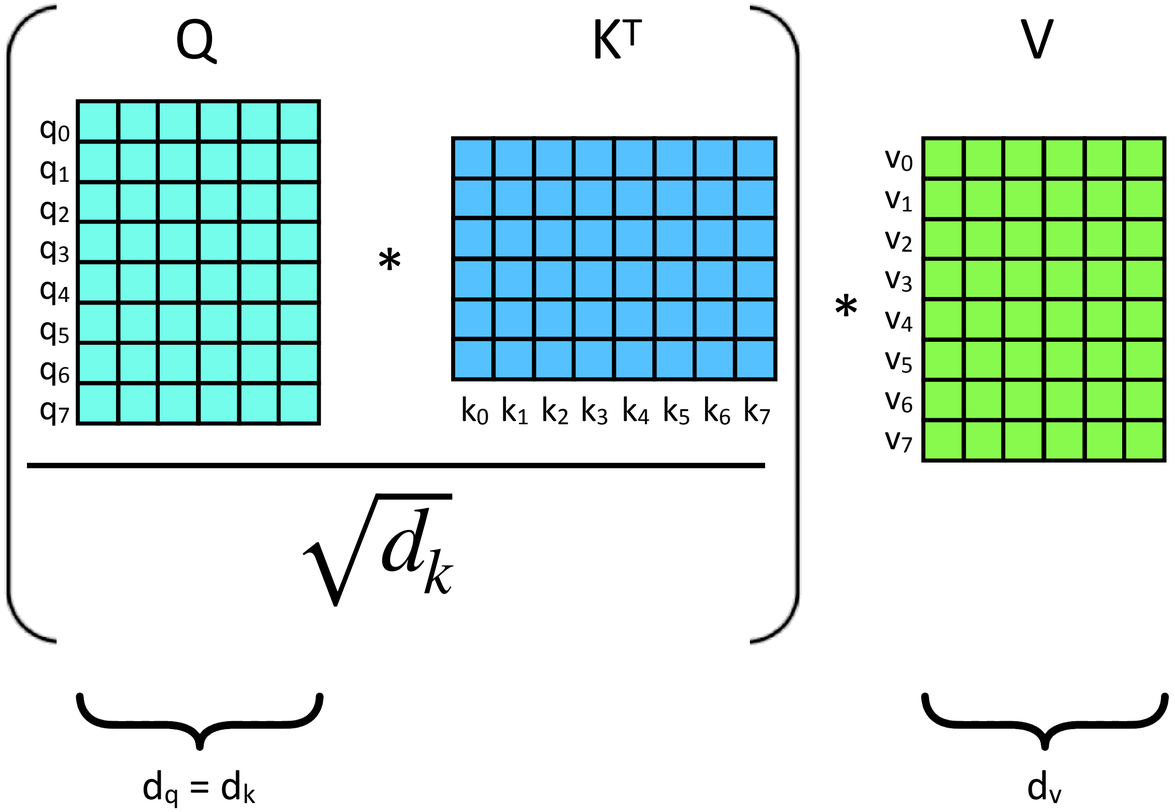


Scaled Dot-Product Attention

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

„Attention is all you need“ https://arxiv.org/pdf/1706.03762.pdf

Attention (Q, K, V) = softmax



„Attention“ Wortherkunft:
Each position in Q „attends“ to every other position in K.

Es geht darum, dass man eine Positions- zu Position Beziehungsmatrix aufstellen möchte. Die Beziehung besteht aus Vektor-Ähnlichkeiten, die im Zusammenhang mit einer Cosinus-Ähnlichkeit stehen. Diese wirken, weil wir Embeddings aufgrund von semantischen Ähnlichkeiten berechnen.

Attention produziert (bildhaft) einen „schlauen Merkzettel“ für das Verfassen von Text.
Der Merkzettel liefert die Bedeutungen von Wörtern und den Beziehungen von Aussagen.

Single-Head of Multi-Head Attention

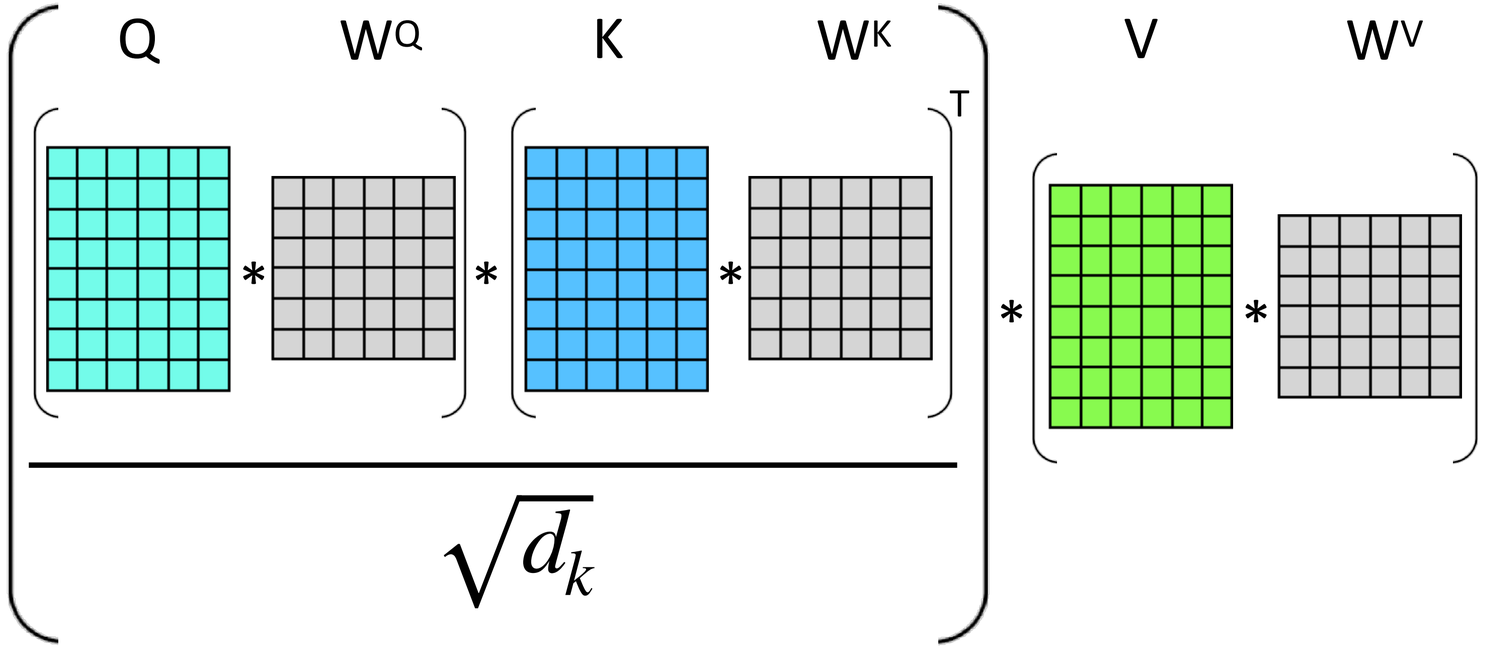
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k} \quad W_i^K \in \mathbb{R}^{d_\text{model} \times d_k} \quad W_i^V \in \mathbb{R}^{d_\text{model} \times d_v} \quad W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$$

„Attention is all you need“ https://arxiv.org/pdf/1706.03762.pdf

Attention (QW^Q, KW^K, VW^V) = softmax



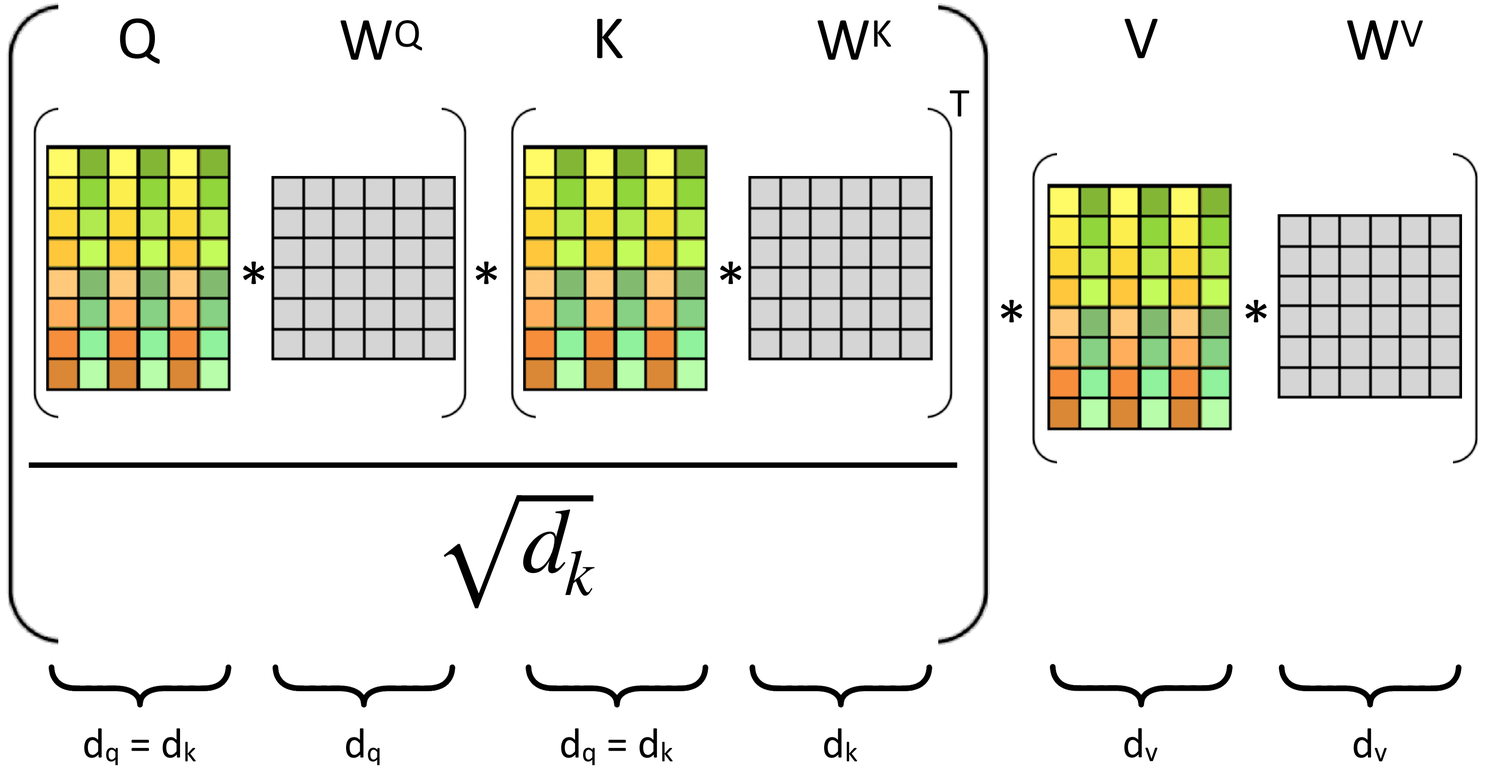
Jede Eingangsmatrix wird mit angelernten Gewichten versehen.

W^Q, KW^K, VW^V sind lineare Faktoren, die während des Lernens des LLMs trainiert werden.

Die Gewichte drücken aus, welche Positionsbeziehungen tatsächlich bestehen.

Self-Attention: Q = K = V

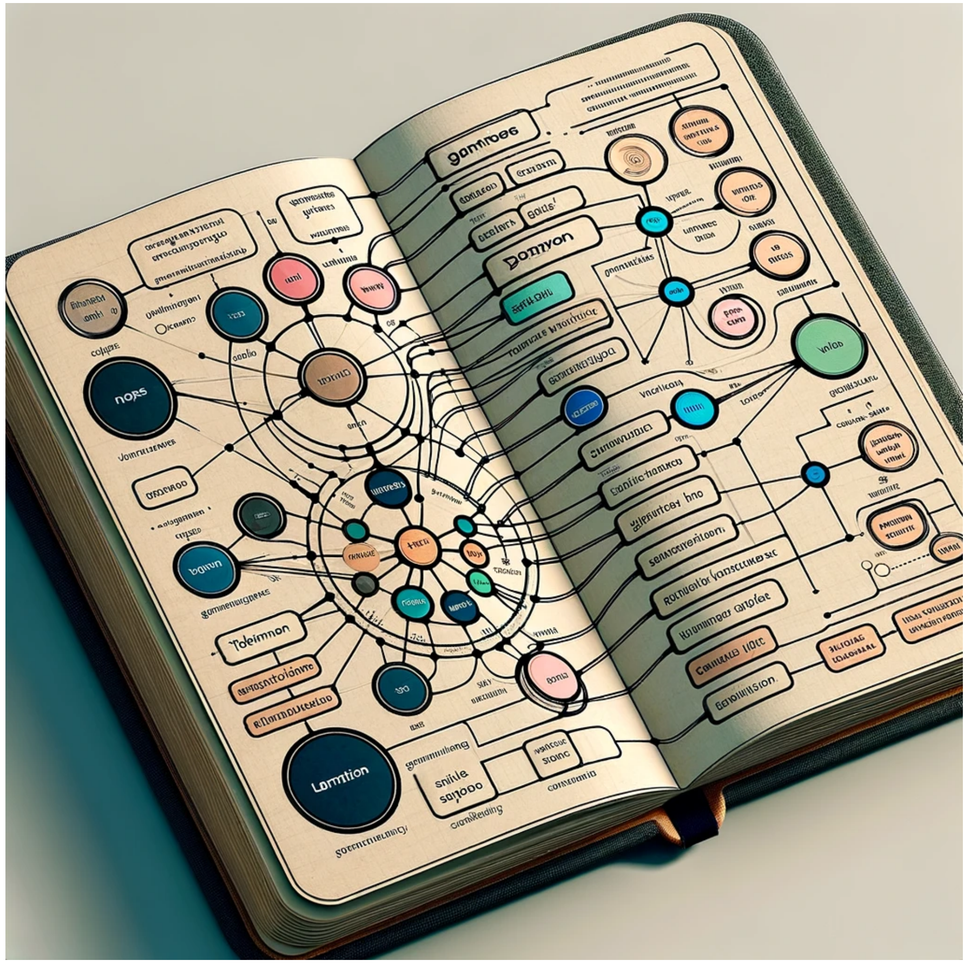
Attention (QW^Q, KW^K, VW^V) = softmax

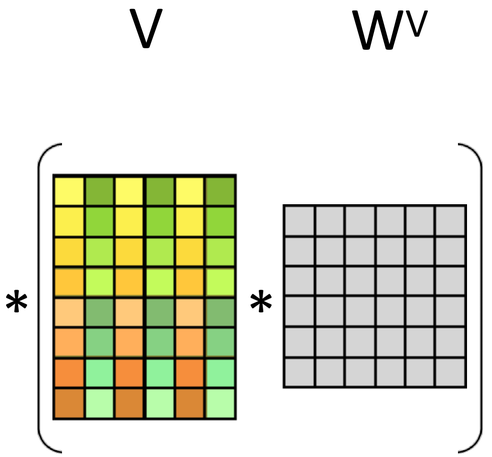


Alle Parameter Q, K, V werden mit der aktuellen encodierten Textrepräsentation bzw. einer Subdimension davon initialisiert.
Die Gewichte W^Q und W^K erzeugen eine Beziehungsmatrix, die gewichtet mit W^V) auf die Textrepräsentation angewendet wird.

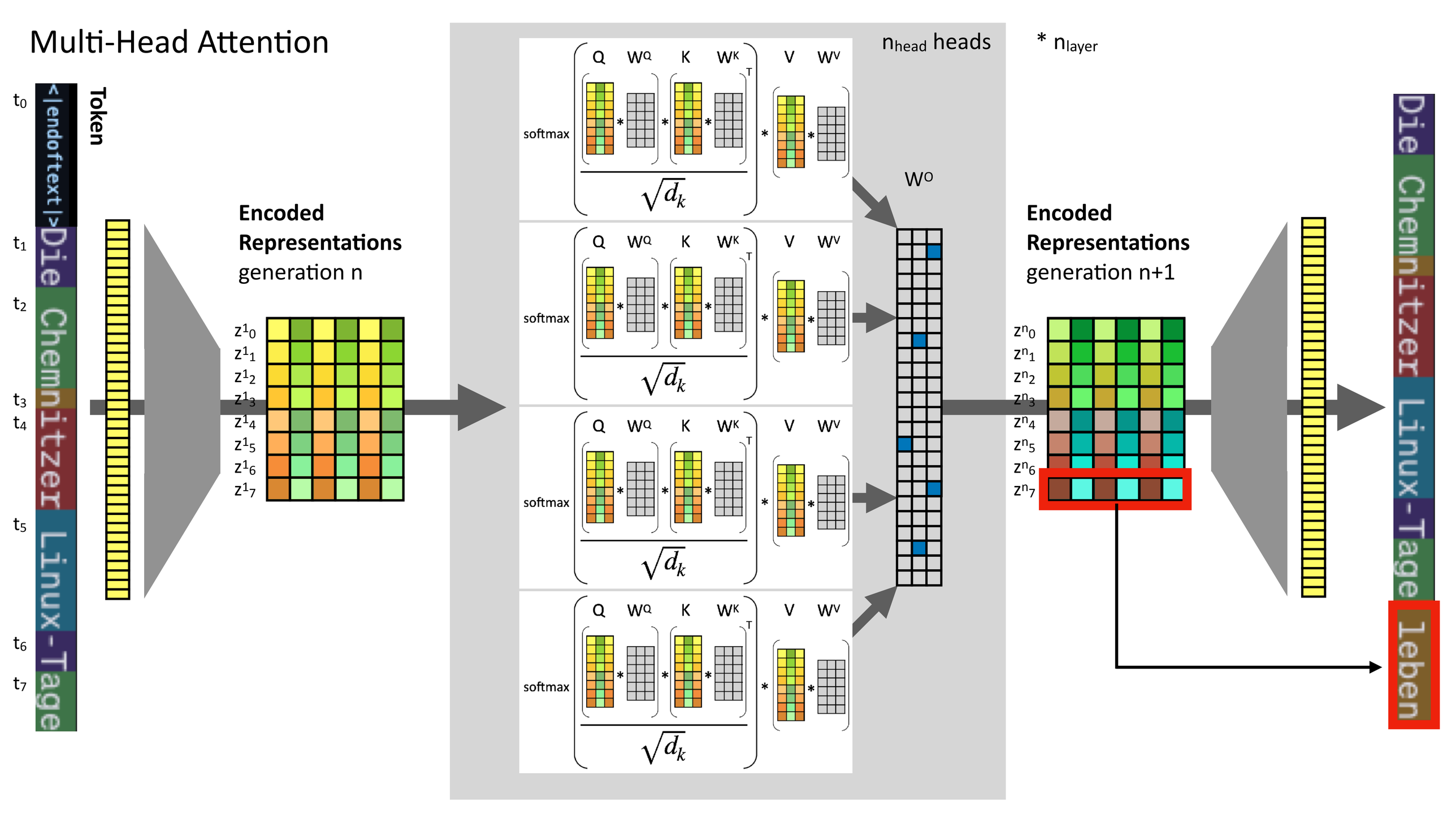
Self-Attention: Q = K = V

Attention (QW^Q, KW^K, VW^V) =

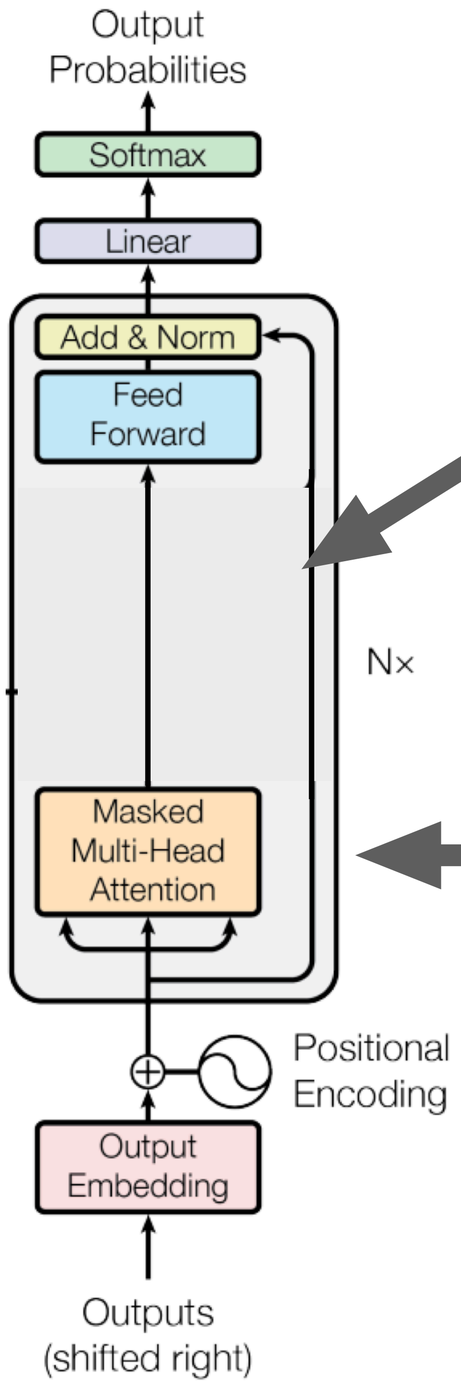


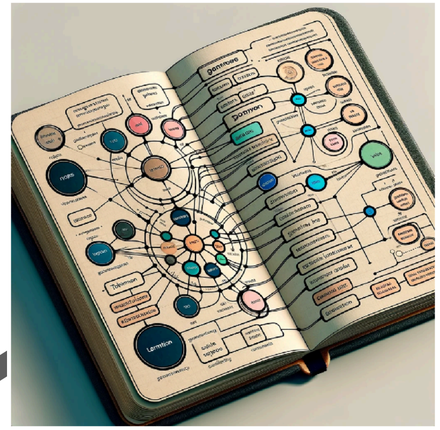


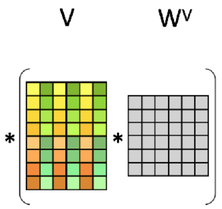
Alle Parameter Q, K, V werden mit der aktuellen encodierten Textrepräsentation bzw. einer Subdimension davon initialisiert.
Die Gewichte W^Q und W^K erzeugen eine Beziehungsmatrix, die gewichtet mit W^V) auf die Textrepräsentation angewendet wird.


Multi-Head Attention
Token
t0
t1
t2
t3
t4
t5
t6
t7
<|endoftext|>
Die
Chemnitzer
Linux-Tage
Encoded Representations generation n
z1 0
z1 1
z1 2
z1 3
z1 4
z1 5
z1 6
z1 7
nhead heads
* nlayer
Q
WQ
K
WK
V
WV
softmax
WO
Encoded Representations generation n+1
zn 0
zn 1
zn 2
zn 3
zn 4
zn 5
zn 6
zn 7
Die
Chemnitzer
Linux-Tage
Leben


Multi-Head Multi-Layer self-Attention
Token
t0
t1
t2
t3
t4
t5
t6
t7
<|endoftext|>
Die
Chemnitzer
Linux-Tage
softmax
Q
K
V
GPT-3: nhead = 128
GPT-3: nlayer = 96
Die Chemnitzer Linux-Tage
leben







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

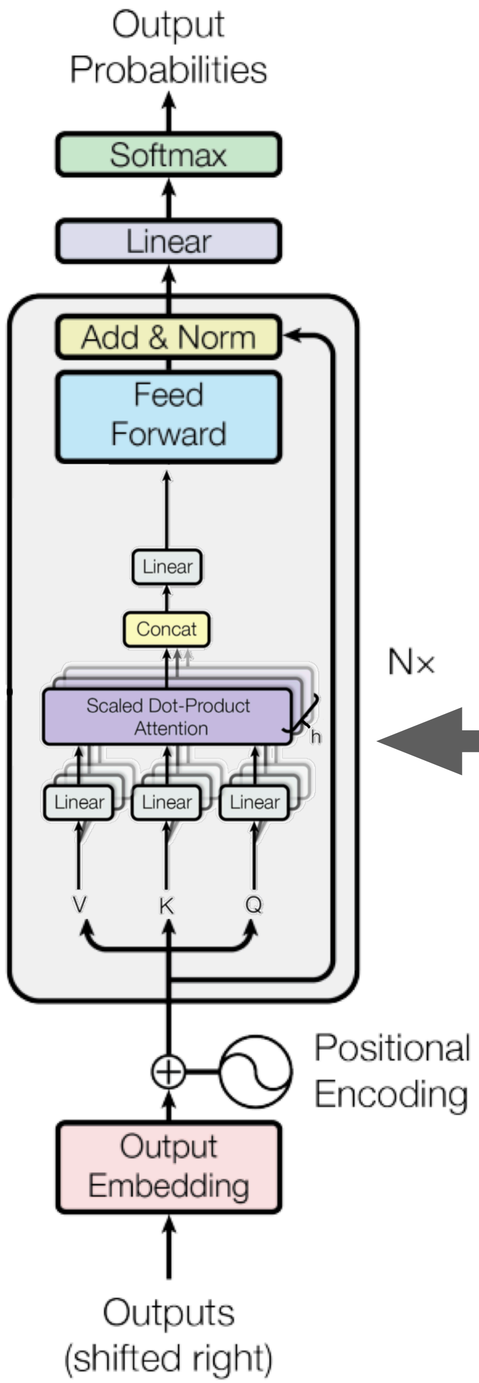
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$ $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$ $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$

„Attention is all you need“ https://arxiv.org/pdf/1706.03762.pdf

Q = K = V = Encoded-Representations generation n



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

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

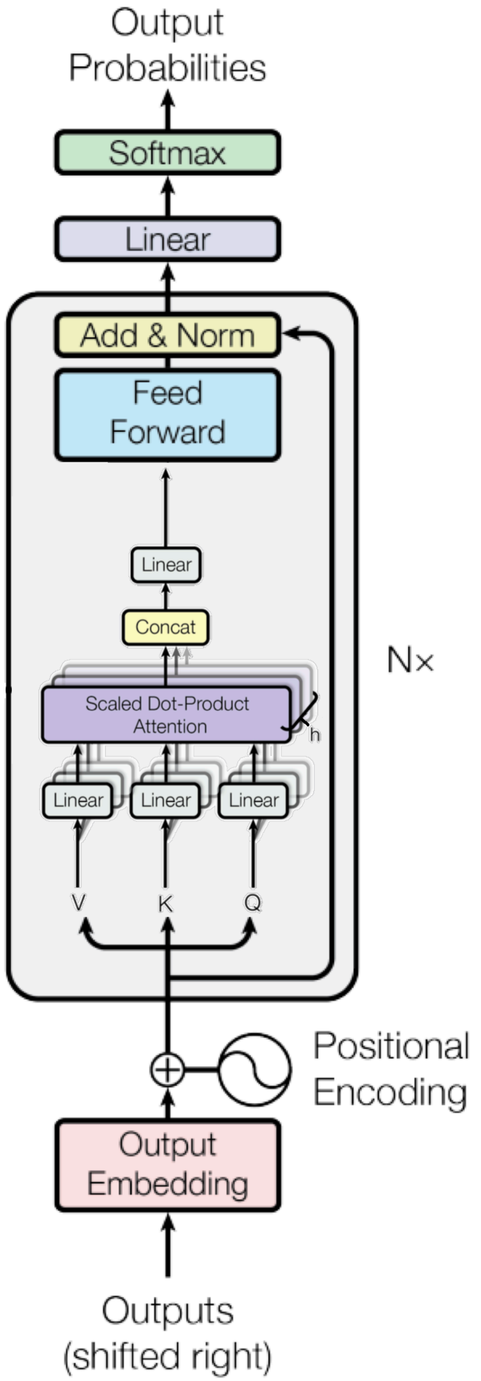
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$ $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$ $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$

„Attention is all you need“ https://arxiv.org/pdf/1706.03762.pdf

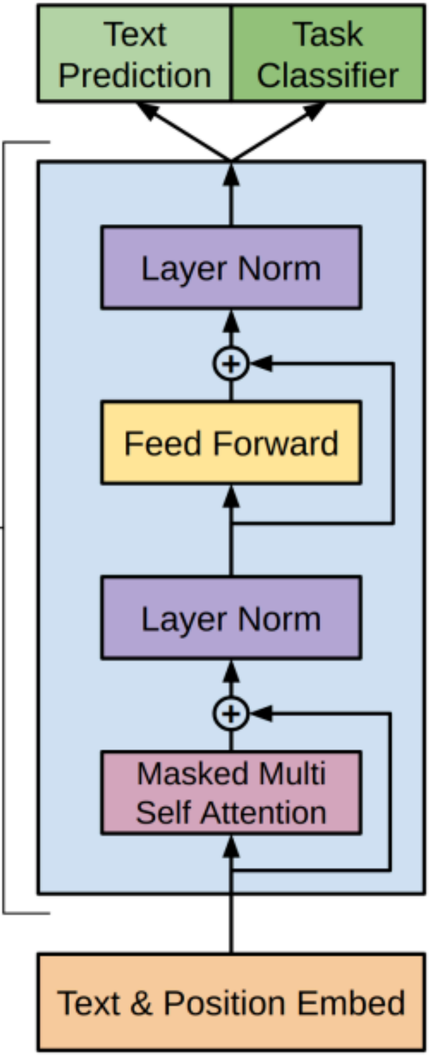
Q = K = V = Encoded-Representations generation n

„Attention is all you need“



N×

12x



GPT-1

2020

Training Data for LLMs

„The Pile: An 800GB Dataset of Diverse Text for Language Modeling“

Training Data Set for EleutherAI models, like GPT-J-6B and GPT-NeoX but also non-EleutherAI models like Meta’s LLaMA, Galactica, Stanfords BioMedLM-2.5B, Yandex YaLM 100B

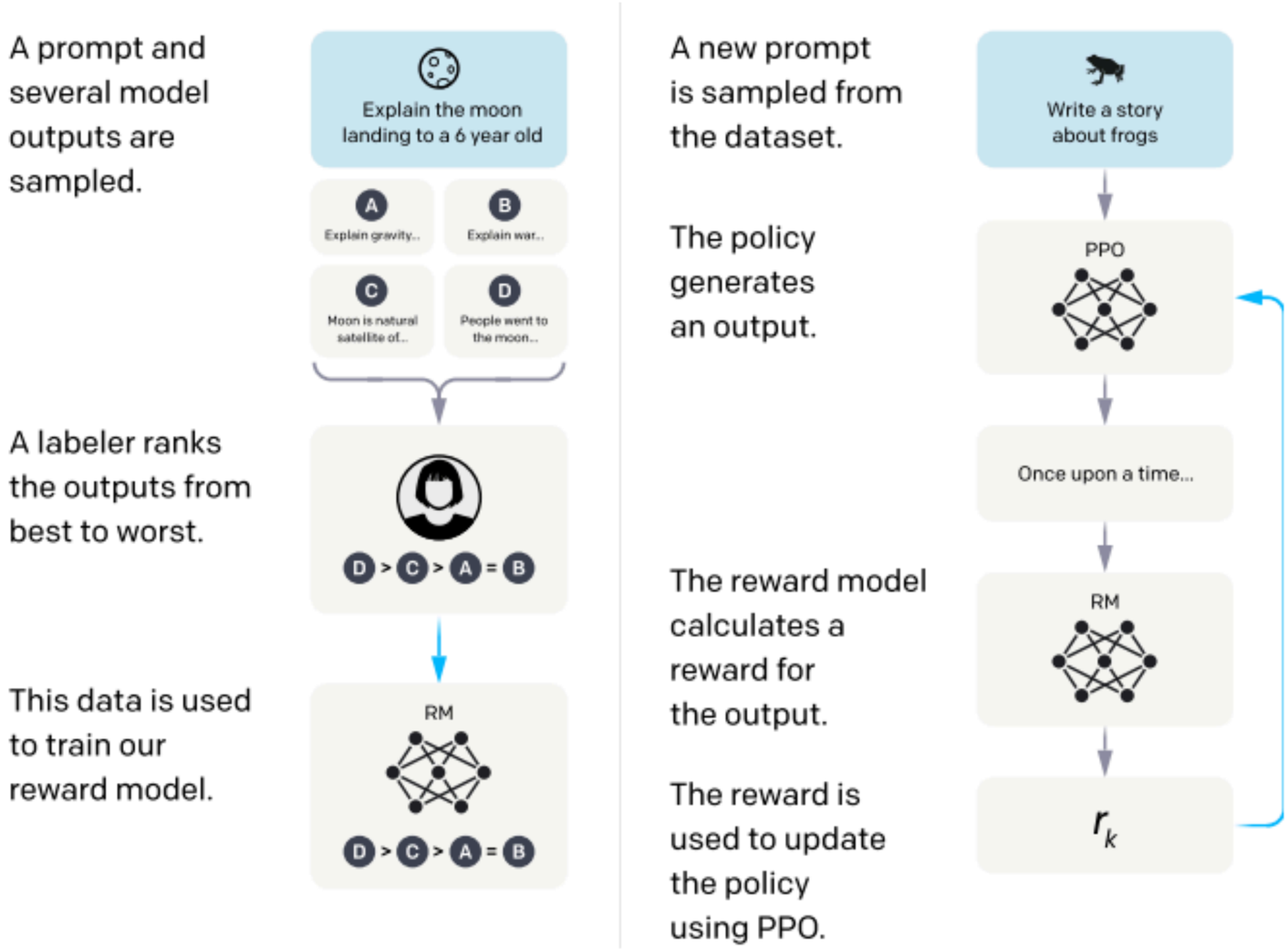
Composition of the Pile by Category

Academic Internet Prose Dialogue Misc

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**

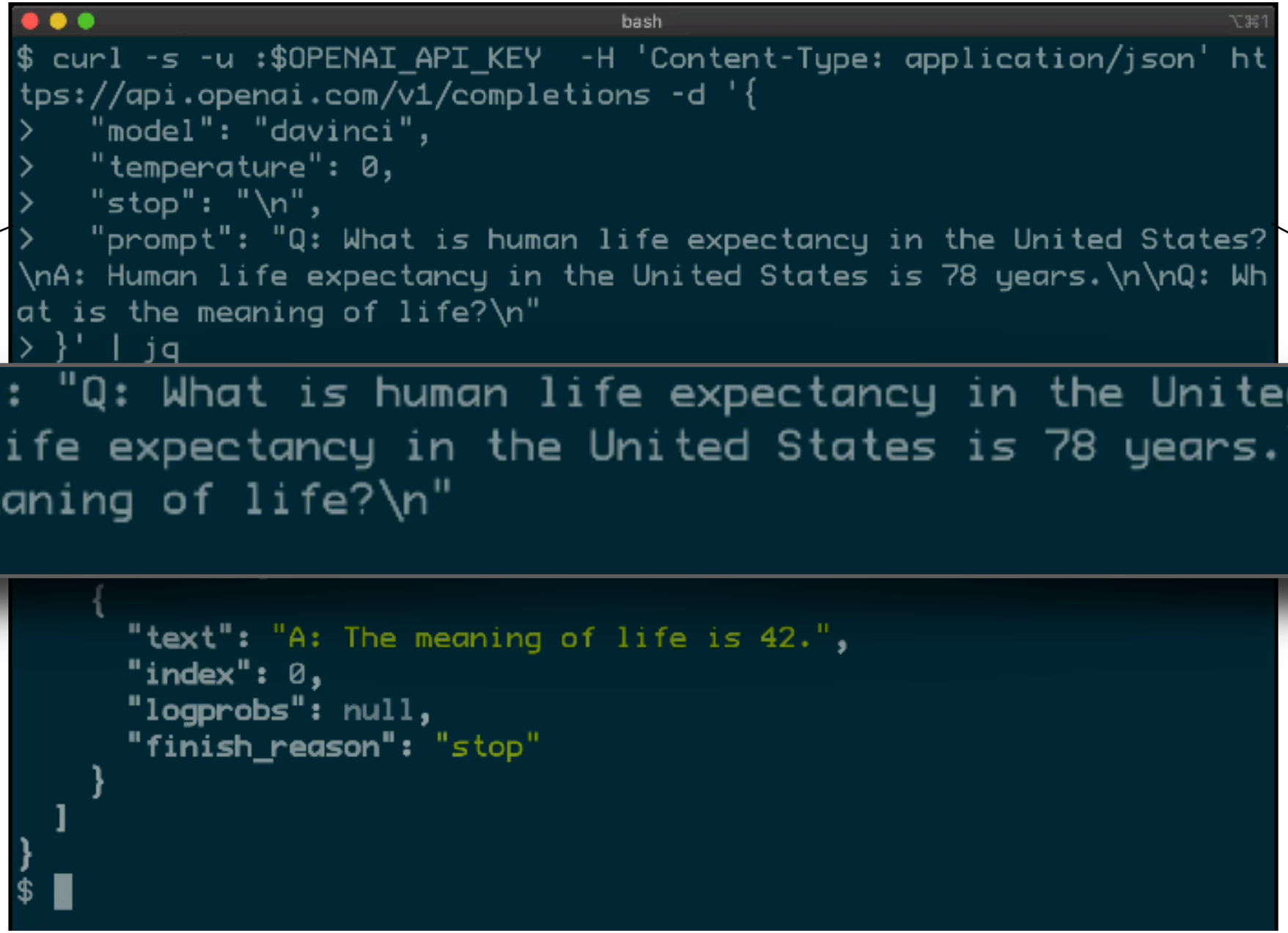
https://arxiv.org/pdf/2101.00027.pdf
https://en.wikipedia.org/wiki/The_Pile_(dataset)

Instruct-Training, Alignment, RLHF



2020

1-shot prompt



https://openai.com/blog/openai-api

Base Models sind keine Assistenten / Chats

Make it look like document

Few-shot prompt

(Man kann sie aber trickreich dazu bringen, sich wie Assistenten zu verhalten.)

- LLMs vervollständigen Dokumente
- Wenn das Dokument wie ein Chat aussieht, vervollständigen sie diesen ebenfalls
- Man formuliert eine Anfrage als letzten Rumpf eines Chats im Dokument.

Insert query here →

Completion

The following is a conversation between a Human and a helpful, honest and harmless AI Assistant.

[Human]
Hi, how are you?

[Assistant]
I'm great, thank you for asking. How I can help you today?

[Human]
I'd like to know what is 2+2 thanks

[Assistant]
2+2 is 4.

[Human]
Great job.

[Assistant]
What else can I help you with?

[Human]
What is the capital of France?

[Assistant]
Paris.

https://www.youtube.com/watch?v=bZQun8Y4L2A
Mehr Beispiele unter
https://medium.com/@manoranjan.rajguru/prompt-template-for-opensource-llms-9f7f6fe8ea5

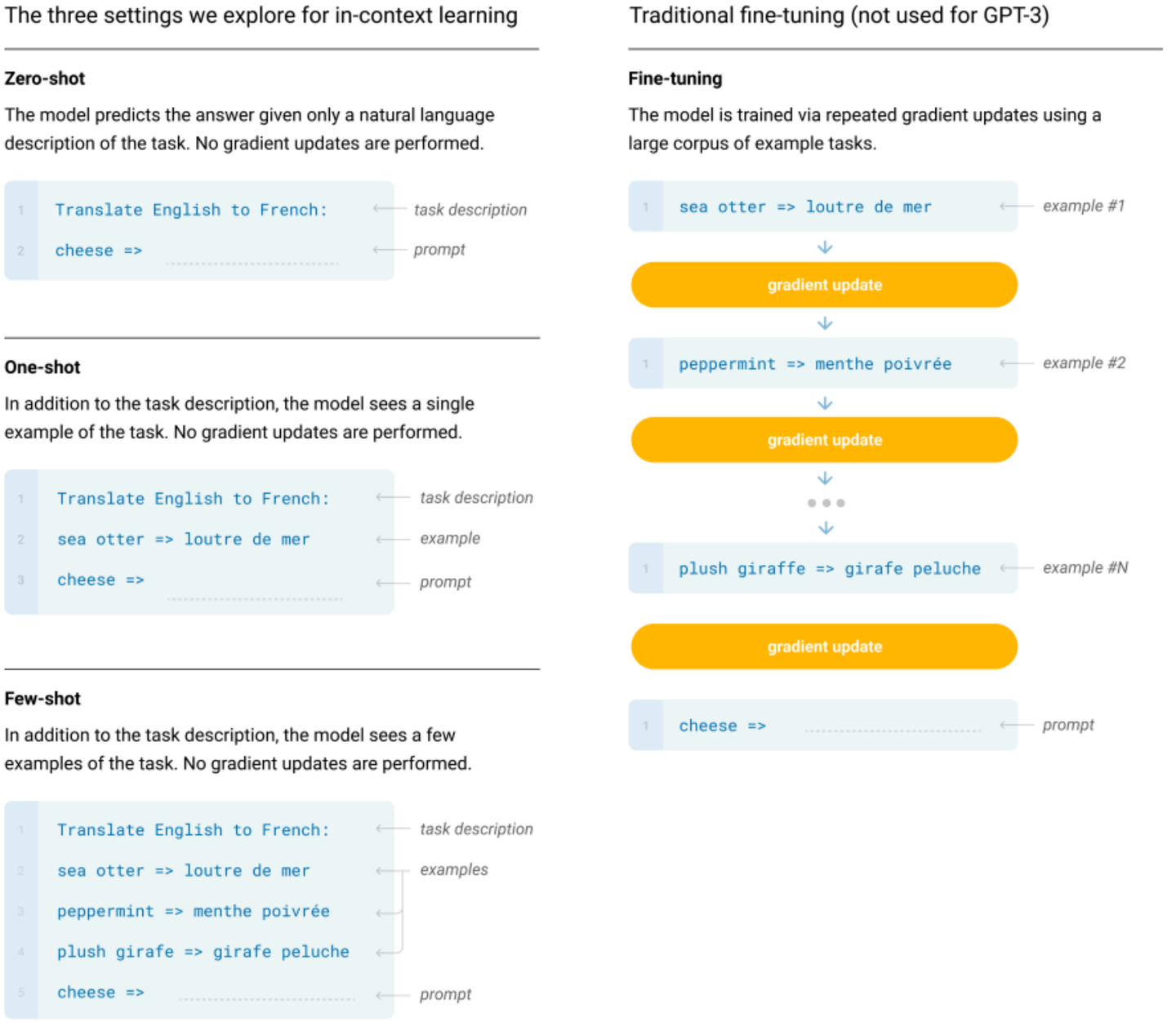
2020

few-shot prompt

2020:
„Language Models are Few-Shot Learners“ (OpenAI)

„Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples.“

Few-Shot



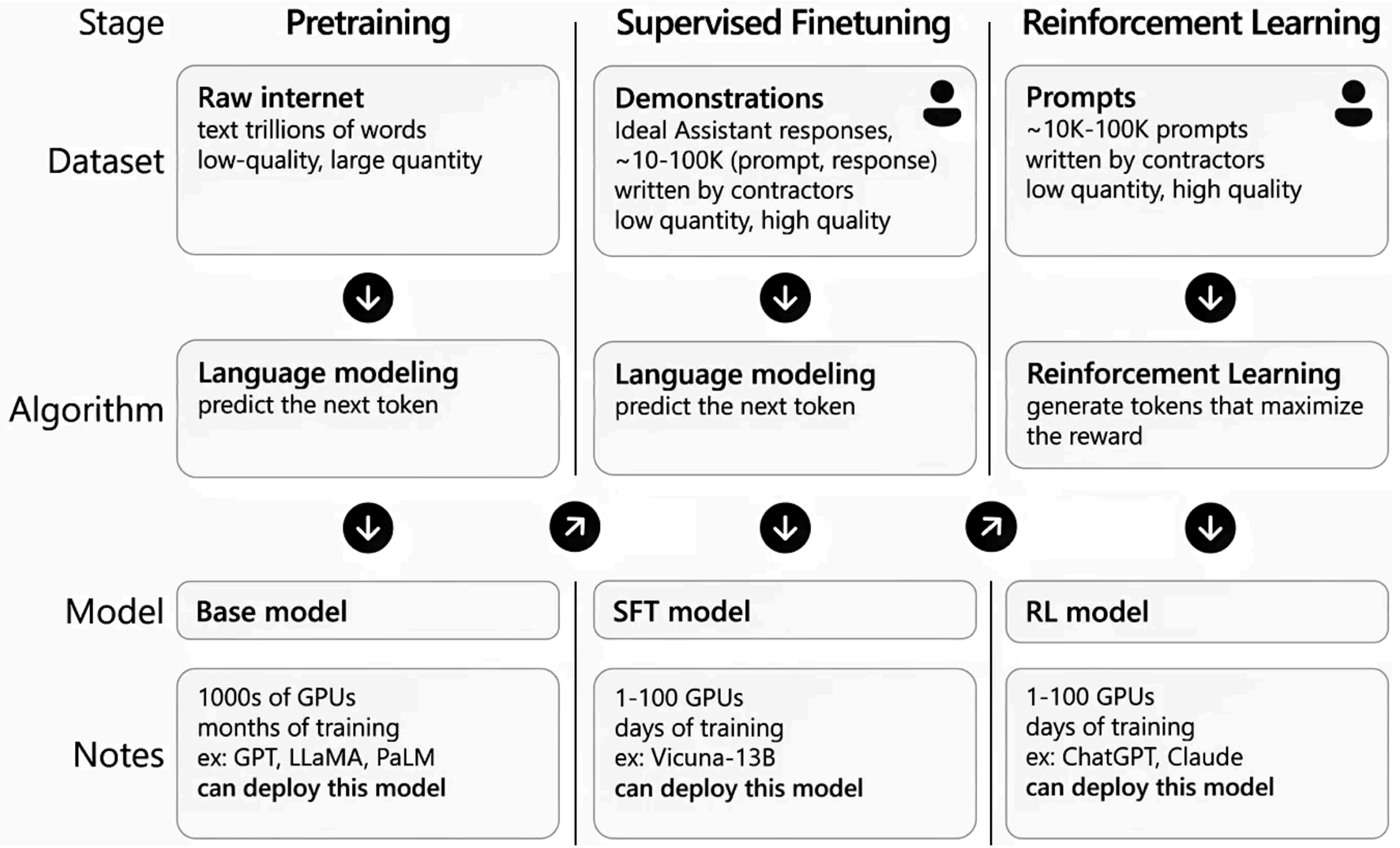
https://arxiv.org/pdf/2005.14165.pdf

2022

Instruction Training + RLHF

March **2022**:
„Aligning language models to follow instructions“

InstructGPT is a GPT-style language model. Researchers at OpenAI developed the model by fine-tuning GPT-3 to follow instructions using human feedback.



https://arxiv.org/pdf/1909.08593.pdf
https://openai.com/research/instruction-following
https://www.youtube.com/watch?v=bZQun8Y4L2A

30. November 2022

„Introducing ChatGPT“

https://openai.com/blog/chatgpt

Open Source
Modelle
nach ChatGPT 3.5

20.12.2022

„Self-Instruct: Aligning Language Models with Self-Generated Instructions“

SELF-INSTRUCT: Aligning Language Models with Self-Generated Instructions

Yizhong Wang[♣] **Yeganeh Kordi**[◇] **Swaroop Mishra**[♡] **Alisa Liu**[♣]
Noah A. Smith[♣+] **Daniel Khashabi**[♠] **Hannaneh Hajishirzi**[♣+]
[♣]University of Washington [◇]Tehran Polytechnic [♡]Arizona State University
[♠]Johns Hopkins University [+]Allen Institute for AI
yizhongw@cs.washington.edu

Abstract

Large “instruction-tuned” language models (i.e., finetuned to respond to instructions) have demonstrated a remarkable ability to generalize zero-shot to new tasks. Nevertheless, they depend heavily on human-written instruction data that is often limited in quantity, diversity, and creativity, therefore hindering the generality of the tuned model. We introduce SELF-INSTRUCT, a framework for improving the instruction-following capabilities of pretrained language models by bootstrapping off their own generations. Our pipeline generates instructions, input, and output samples from a language model, then filters invalid or similar ones before using them to finetune the original model. Applying our method to the vanilla GPT3, we demonstrate a 33% absolute improvement over the original model on SUPER-NATURALINSTRUCTIONS, on par with the performance of InstructGPT$_{001}$,[1] which was trained with private user data and human annotations. For further evaluation, we curate a set of expert-written instructions for novel tasks, and show through human evaluation that tuning GPT3 with SELF-INSTRUCT outperforms using existing public instruction datasets by a large margin, leaving only a 5% absolute gap behind InstructGPT$_{001}$. SELF-INSTRUCT provides an almost annotation-free method for aligning pretrained language models with instructions, and we release our large synthetic dataset to facilitate future studies on instruction tuning.[2]

Instruction: Given an address and city, come up with the zip code.
Input:
Address: 123 Main Street, City: San Francisco
Output: 94105

Instruction: I am looking for a job and I need to fill out an application form. Can you please help me complete it?
Input:
Application Form:
Name:_____ Age:_____ Sex:_____
Phone Number:_____ Email Address:_____
Education:_____ ...
Output:
Name: John Doe Age: 25 Sex: Male
Phone Number: ...

Instruction: How to write a code for converting degrees fahrenheit to celsius.
Input: Null
Output:
```
def convert_fahrenheit_to_celsius(fahr):
    celsius = (fahr - 32) * 5 / 9
    return celsius
```

Instruction: Write a letter from the perspective of a cat.
Input: Null
Output:
Dear [Owner], I am writing to you today because I have a problem. You see, I've been living with you for the past [Number of Years] and I love it here. The food is great, the bed is comfy, and there are plenty of toys to play with. But lately, I've noticed that you're not as affectionate towards me as you used to be. In fact, sometimes when I try to cuddle up next to you on the couch, you push me away ...

Figure 1: Selected tasks from the generated instruction data using vanilla GPT3. Some texts are reformatted for presentation. See Table 10 for more examples.

Figure 3: The top 20 most common root verbs (inner circle) and their top 4 direct noun objects (outer circle) in the generated instructions. Despite their diversity, the instructions shown here only account for 14% of all the generated instructions because many instructions (e.g., “Classify whether the user is satisfied with the service.”) do not contain such a verb-noun structure.

https://arxiv.org/pdf/2212.10560.pdf

24.02.2023

„LLaMA: Open and Efficient Foundation Language Models“ (Meta)

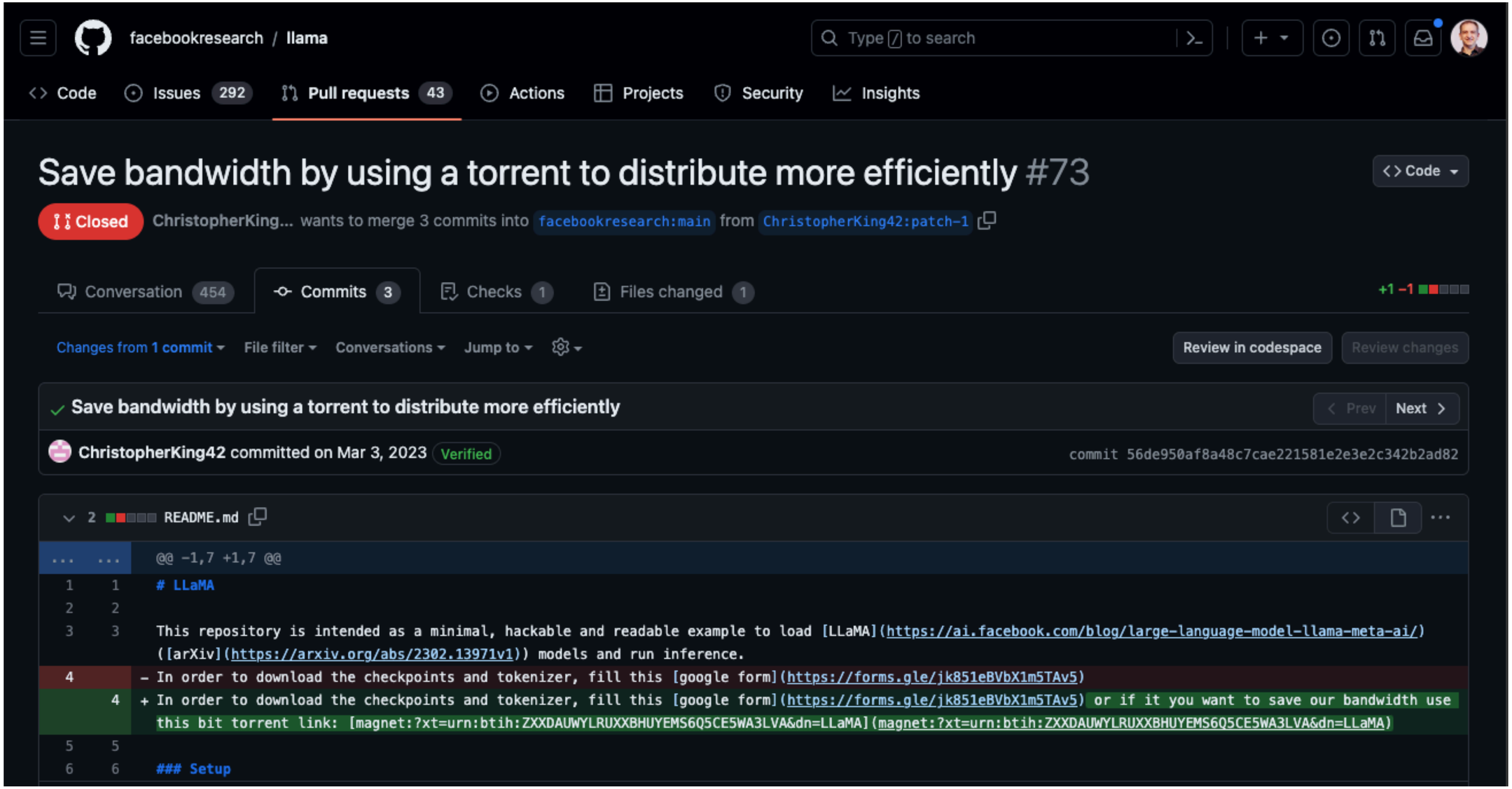
Abstract

We introduce LLaMA, a collection of foundation language models ranging from 7B to 65B parameters. We train our models on trillions of tokens, and show that it is possible to train state-of-the-art models using publicly available datasets exclusively, without resorting to proprietary and inaccessible datasets. In particular, LLaMA-13B outperforms GPT-3 (175B) on most benchmarks, and LLaMA-65B is competitive with the best models, Chinchilla-70B and PaLM-540B. We release all our models to the research community[1].

https://arxiv.org/pdf/2302.13971.pdf

3.3.2023

Llama Leak



https://github.com/facebookresearch/llama/pull/73

10.3.2023

llama.cpp

„This was hacked in an evening - I have no idea if it works correctly.“

llama.cpp

Inference of Facebook's LLaMA model in pure C/C++

TEMPORARY NOTICE: If you observe garbage results, make sure to update to latest master. There was a bug and it was fixed here:
https://github.com/ggerganov/llama.cpp/commit/70bc0b8b15b98dca23b28f0c8f5e34b27e424cda

Description

The main goal is to run the model using 4-bit quantization on a MacBook.

- Plain C/C++ implementation without dependencies
- Apple silicon first-class citizen - optimized via Arm Neon and Accelerate framework
- Mixed F16 / F32 precision
- 4-bit quantization support
- Runs on the CPU

This was hacked in an evening - I have no idea if it works correctly.

first-day commit: https://github.com/ggerganov/llama.cpp/tree/920a7fe2d94cc7e4fed0e88db830b674c91865c5

13.3.2023

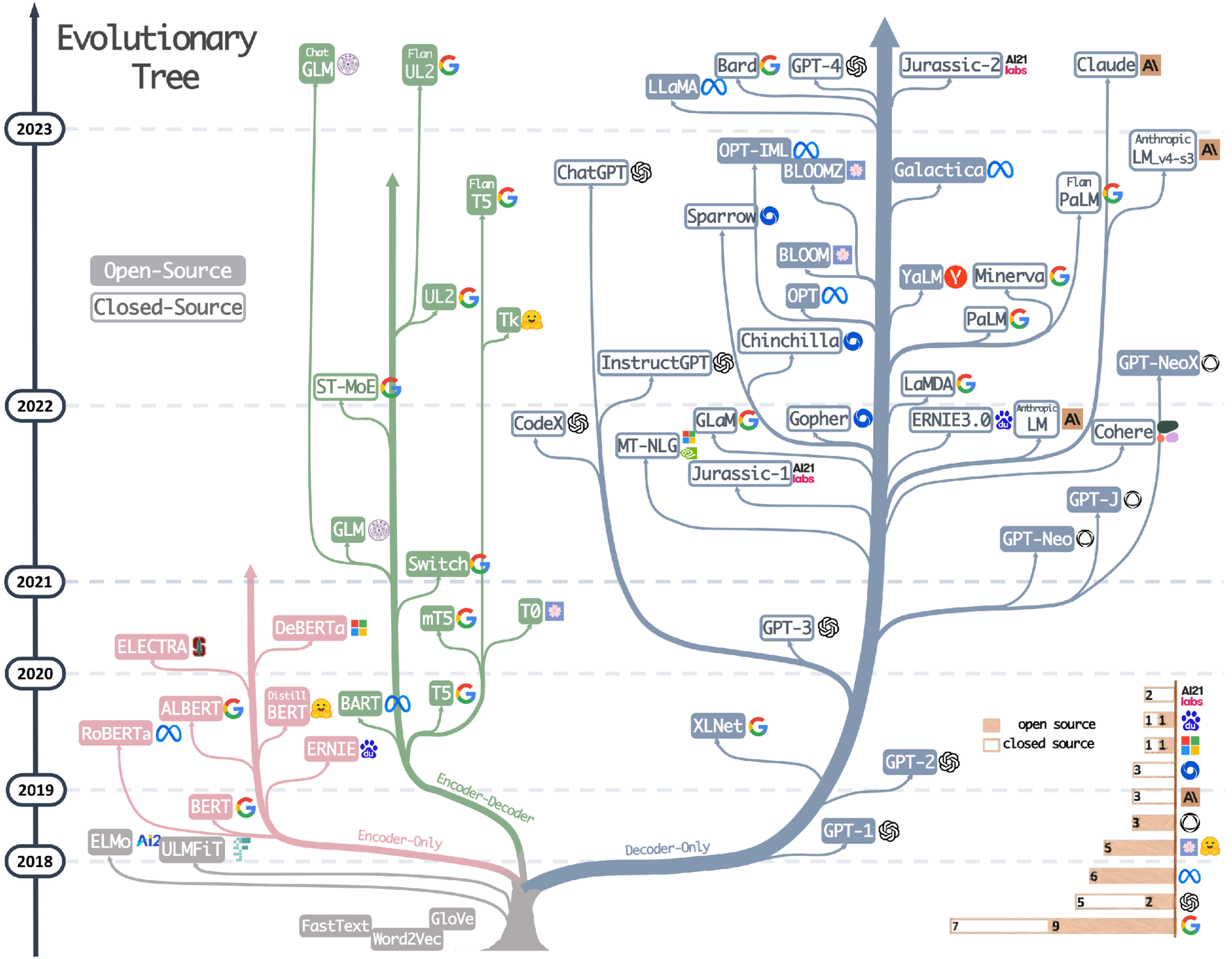
Stanford Alpaca: An Instruction-following LLaMA Model

*We introduce **Alpaca 7B**, a model fine-tuned from the LLaMA 7B model on 52K instruction-following demonstrations. On our preliminary evaluation of single-turn instruction following, Alpaca behaves qualitatively similarly to OpenAI's text-davinci-003, while being surprisingly small and easy/cheap to reproduce (<600$). Checkout our code release on GitHub.*

https://crfm.stanford.edu/2023/03/13/alpaca.html

27.4.2023

Evolutionary Tree of LLMs



https://arxiv.org/pdf/2304.13712.pdf

2023–2024

Neue Sprachmodelle nach Llama & Alpaca *(nur mit verfügbaren Modell-Downloads)*

12.3.2023
Vicuna (UC Berkeley, CMU, Stanford, MBZUAI, and UC San Diego)

3.4.2023
Pythia (AleutherAI)

5.6.2023
Orca (Microsoft)

2.7.2023
Dolphin (Orca uncensored)

18.7.2023
Llama-2 (Meta)

24.8.2023
CodeLlama-2 (Meta)

27.9.2023
Mistral 7B (Mistral / Frankreich)

9.10.2023
Phi 1.5 (Microsoft)

2.11.2023
OpenChat (Tsinghua University, Peking)

11.12.2023
Mixtral 8x7B (Mistral)

7.1.2024
OpenChat 0106 (Tsinghua University)

Open Source + besser als ChatGPT 3.5

28.1.2024
CodeLLama 70B (Meta)

14.2.2024
Qwen1.5 (Alibaba Group)

21.2.2024
Gemma (Google)

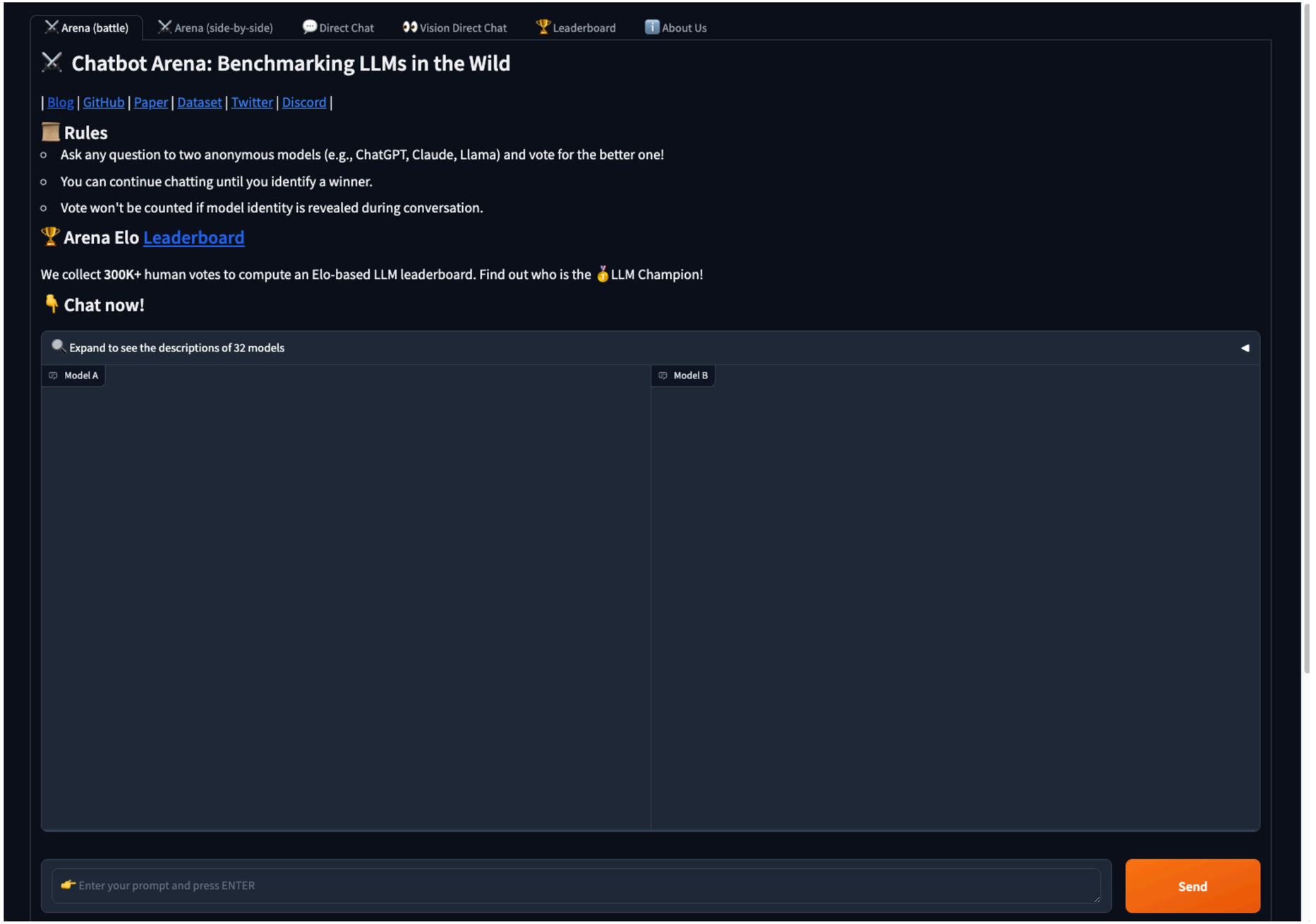
…

März 2024 (angekündigt)
Grok / Grok 1.5 (x.ai)

Juni? 2024 (angekündigt, ohne Termin)
Llama-3 (Meta)

April 2023

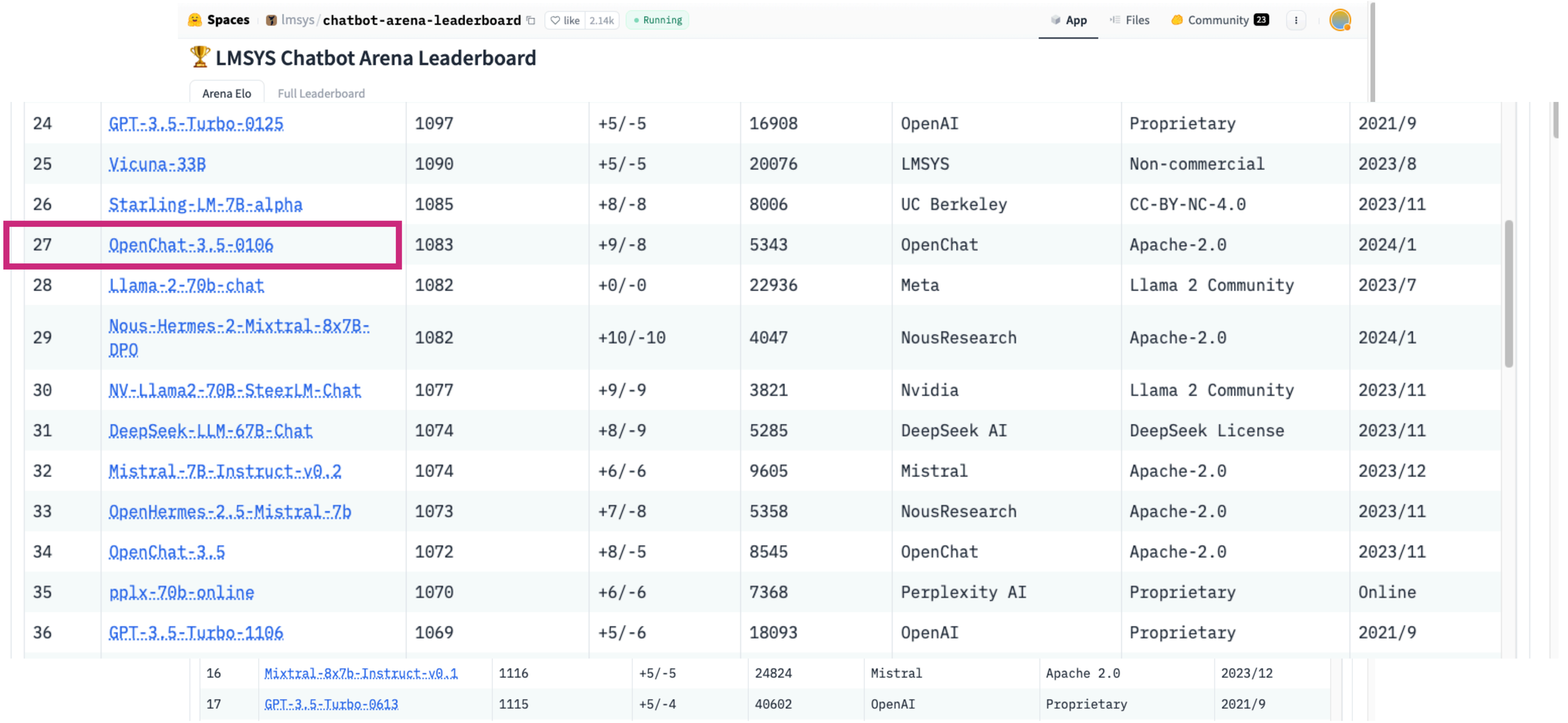
Chatbot Arena



https://chat.lmsys.org/

April 2023

Chatbot Arena Leaderboard



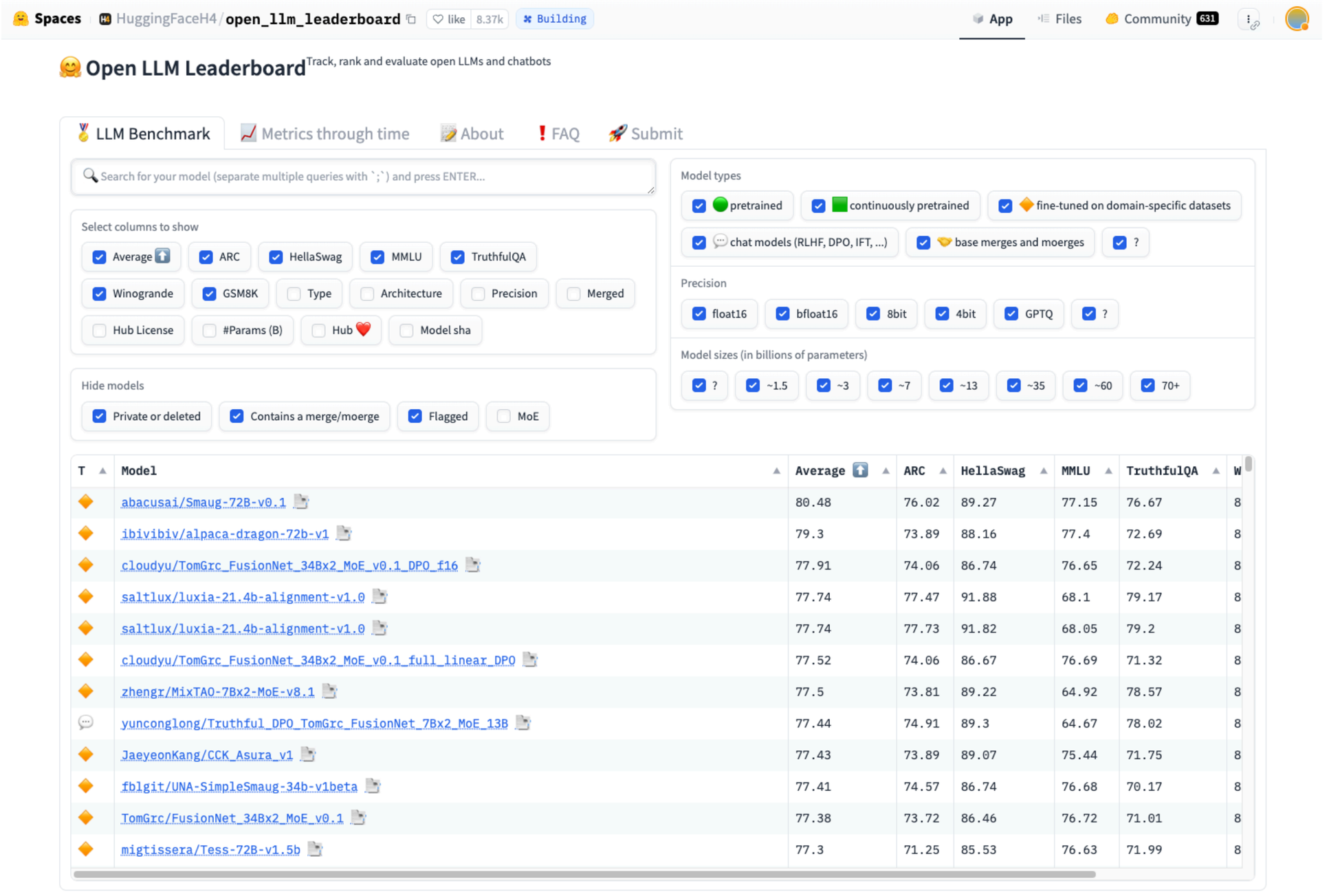
24	GPT-3.5-Turbo-0125	1097	+5/-5	16908	OpenAI	Proprietary	2021/9
25	Vicuna-33B	1090	+5/-5	20076	LMSYS	Non-commercial	2023/8
26	Starling-LM-7B-alpha	1085	+8/-8	8006	UC Berkeley	CC-BY-NC-4.0	2023/11
27	OpenChat-3.5-0106	1083	+9/-8	5343	OpenChat	Apache-2.0	2024/1
28	Llama-2-70b-chat	1082	+0/-0	22936	Meta	Llama 2 Community	2023/7
29	Nous-Hermes-2-Mixtral-8x7B-DPO	1082	+10/-10	4047	NousResearch	Apache-2.0	2024/1
30	NV-Llama2-70B-SteerLM-Chat	1077	+9/-9	3821	Nvidia	Llama 2 Community	2023/11
31	DeepSeek-LLM-67B-Chat	1074	+8/-9	5285	DeepSeek AI	DeepSeek License	2023/11
32	Mistral-7B-Instruct-v0.2	1074	+6/-6	9605	Mistral	Apache-2.0	2023/12
33	OpenHermes-2.5-Mistral-7b	1073	+7/-8	5358	NousResearch	Apache-2.0	2023/11
34	OpenChat-3.5	1072	+8/-5	8545	OpenChat	Apache-2.0	2023/11
35	pplx-70b-online	1070	+6/-6	7368	Perplexity AI	Proprietary	Online
36	GPT-3.5-Turbo-1106	1069	+5/-6	18093	OpenAI	Proprietary	2021/9

16	Mixtral-8x7b-Instruct-v0.1	1116	+5/-5	24824	Mistral	Apache 2.0	2023/12
17	GPT-3.5-Turbo-0613	1115	+5/-4	40602	OpenAI	Proprietary	2021/9

https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

Juni 2023

open_llm_leaderboard

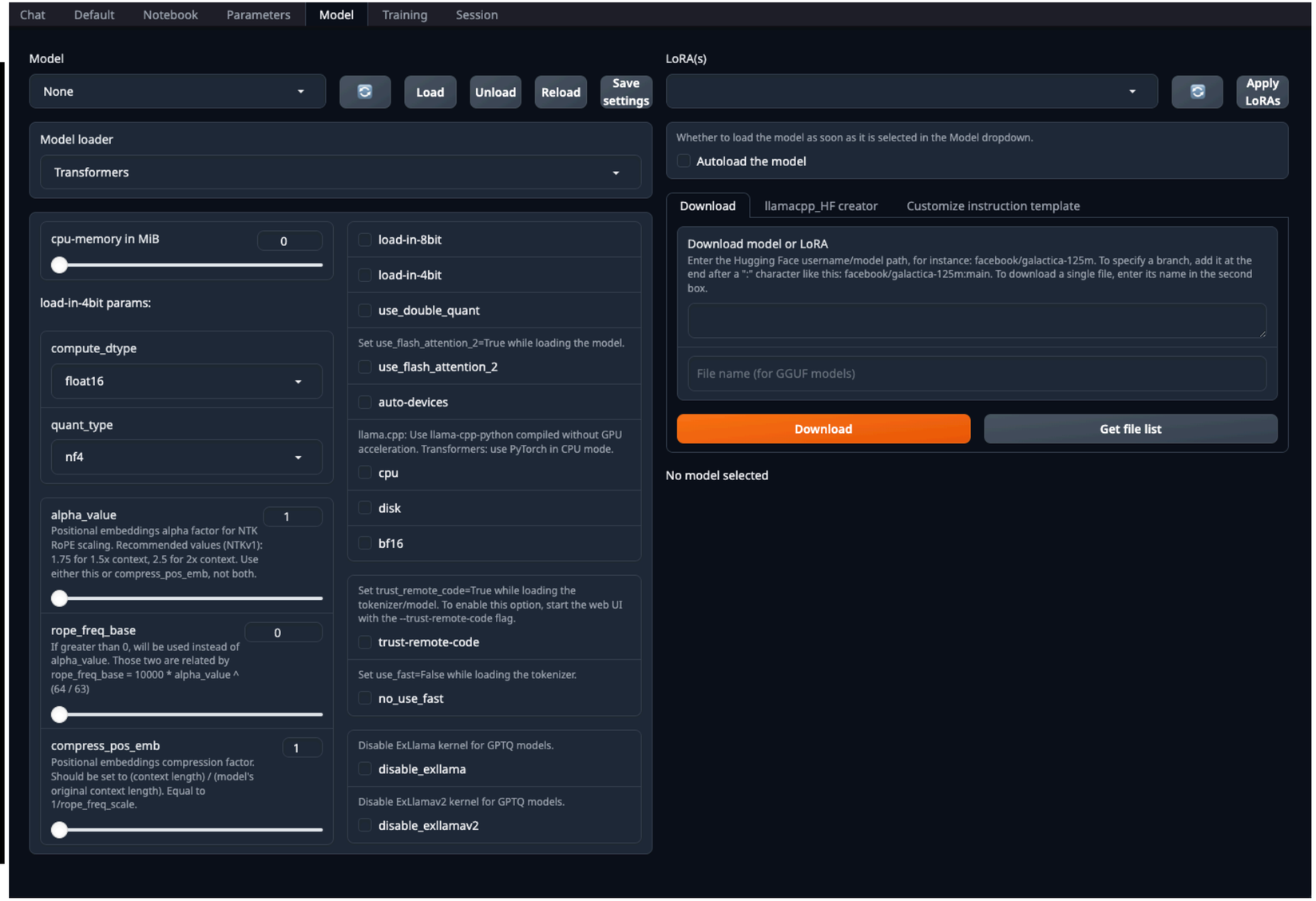


https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

März 2023

text-generation-webui

```
Installieren:

> git clone https://github.com/oobabooga/
text-generation-webui.git
> cd text-generation-webui
> ./start_linux.sh

Modelle besorgen:
- in https://huggingface.co/ Modell aussuchen
- auf burger -> Clone Repository klicken
- Anleitung ausführen:
> cd models
> git lfs install
> git clone …

UI öffnen:
- http://127.0.0.1:7860/
- auf „Model“ klicken
- heruntergeladenes Modell anwählen
- ggf. Parameter anpassen
- auf „Chat“ klicken
- chatten!
```



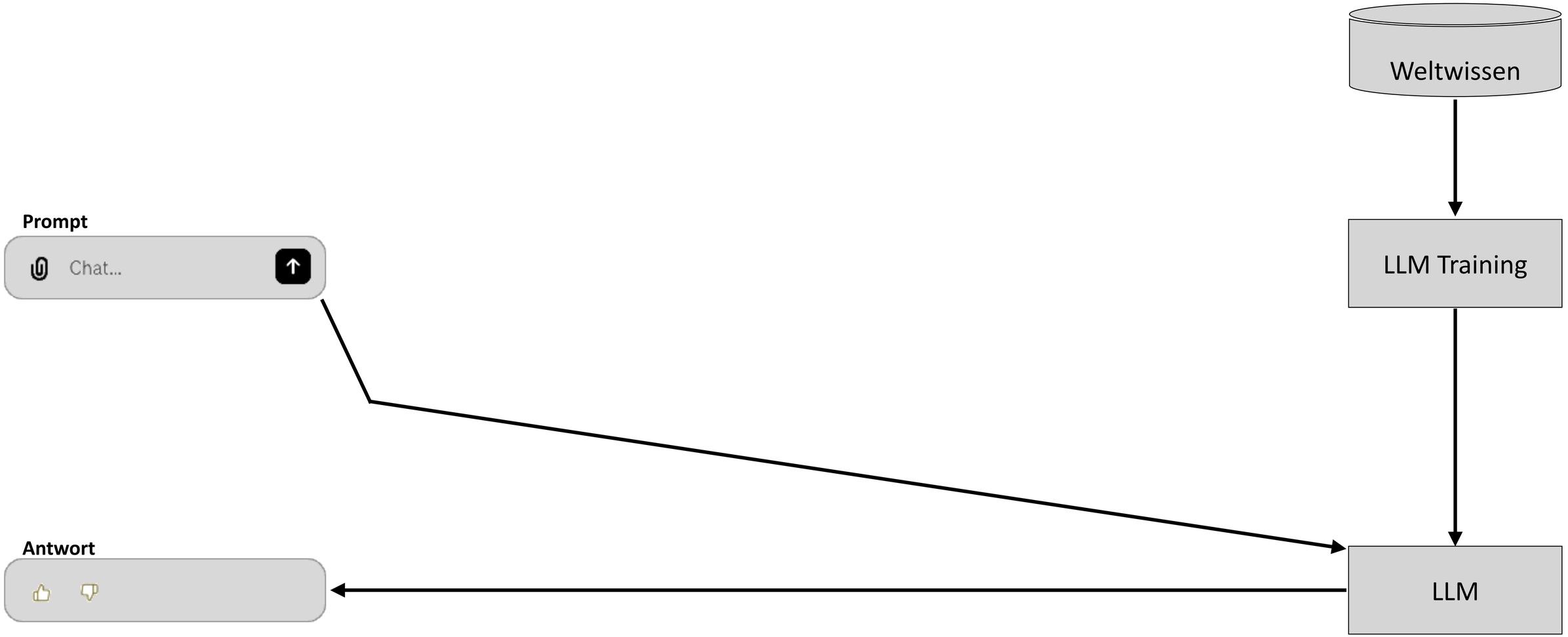
https://github.com/oobabooga/text-generation-webui

LoRa & RAG

2020

RAG - Retrieval Augmented Generation

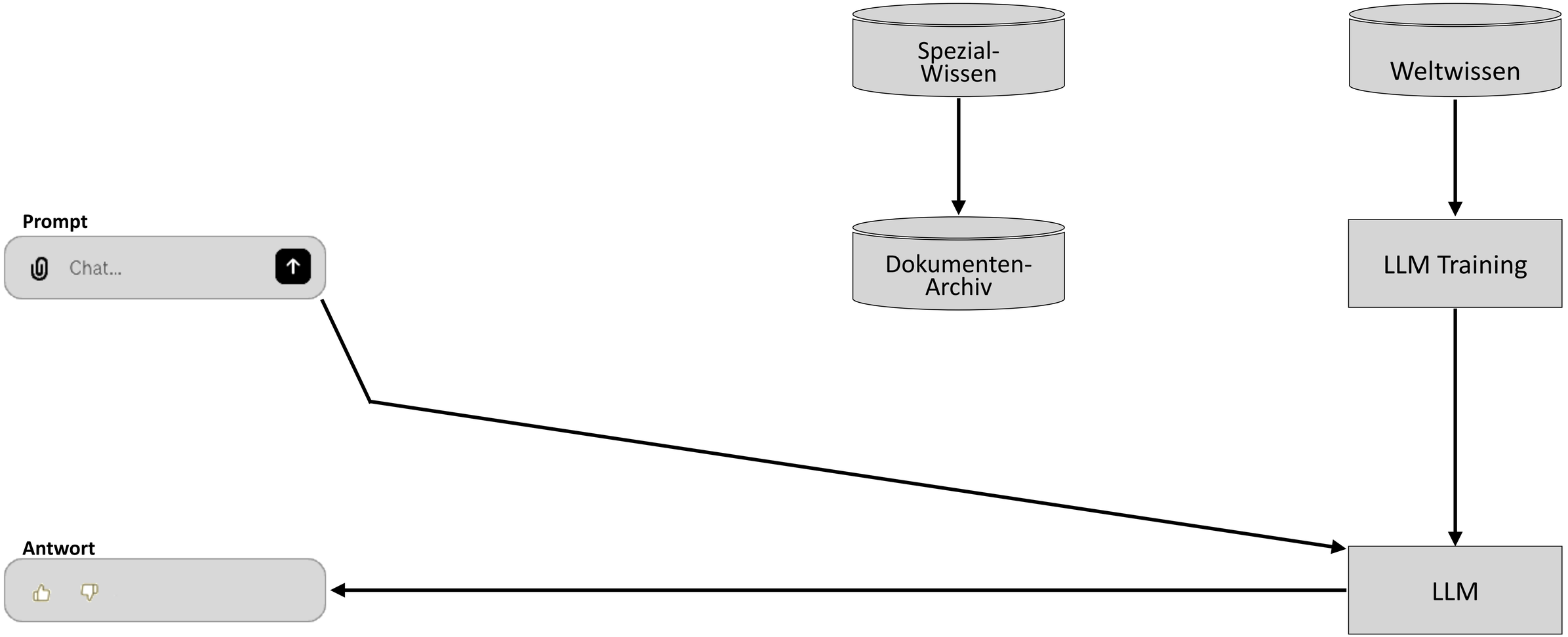
Ausgangslage: nur Wissen aus LLM Training in Antworten



2020

RAG - Retrieval Augmented Generation

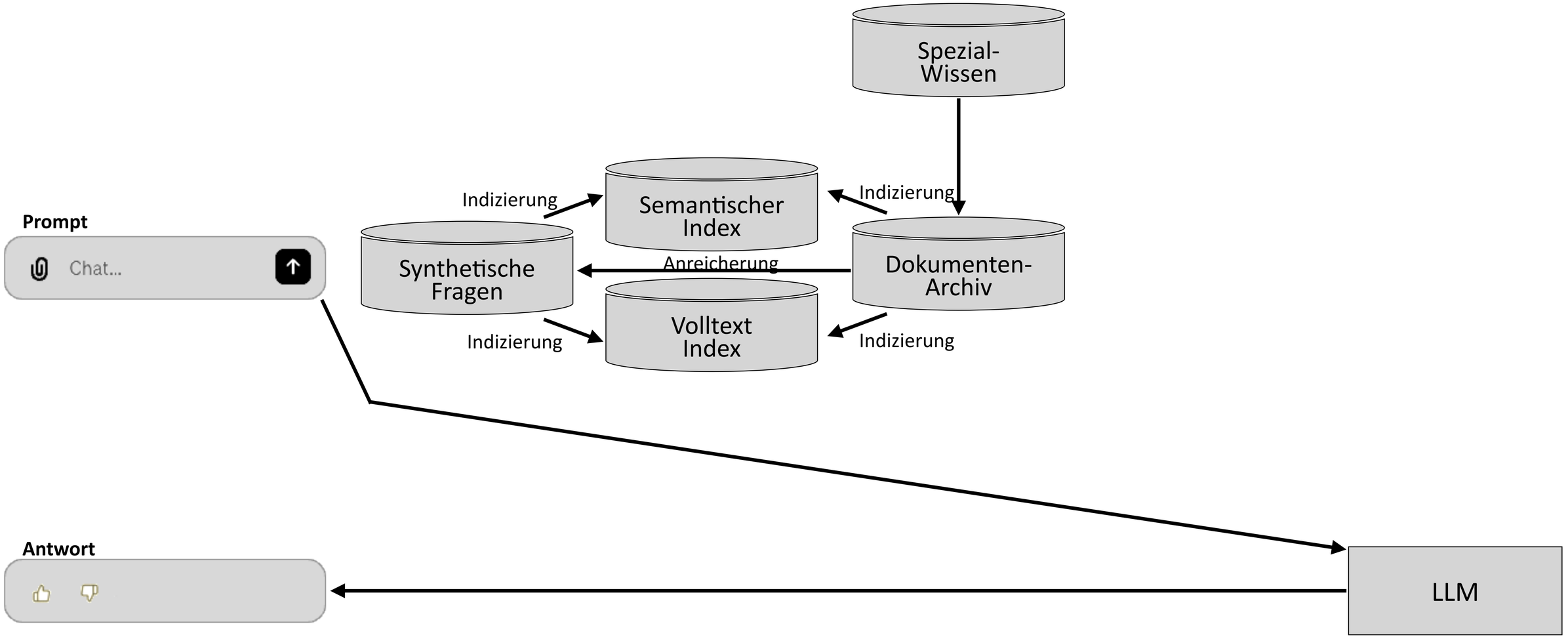
Spezialwissen: kein Teil des Sprachmodells



2020

RAG - Retrieval Augmented Generation

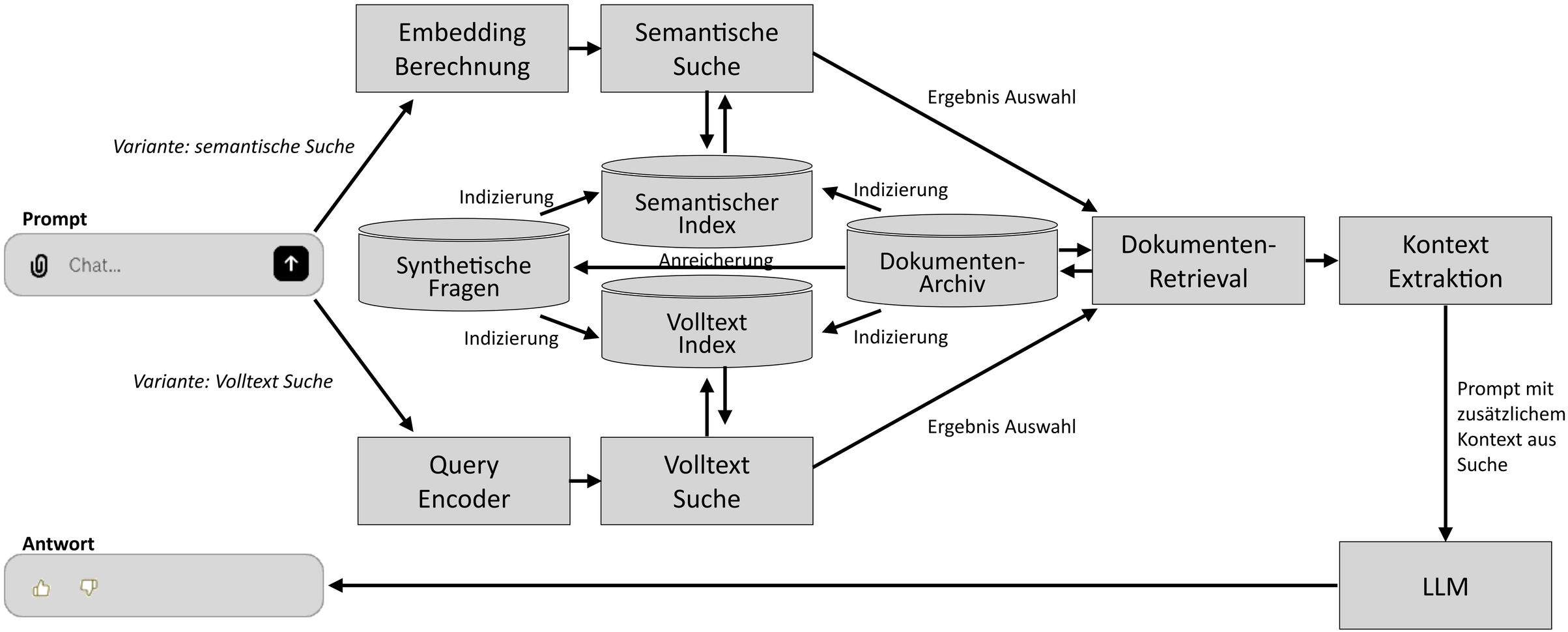
Vorbereitung eines Suchindexes für RAG (2 Varianten)



2020

RAG - Retrieval Augmented Generation

Nutzung eines Suchindexes für RAG (2 Varianten)



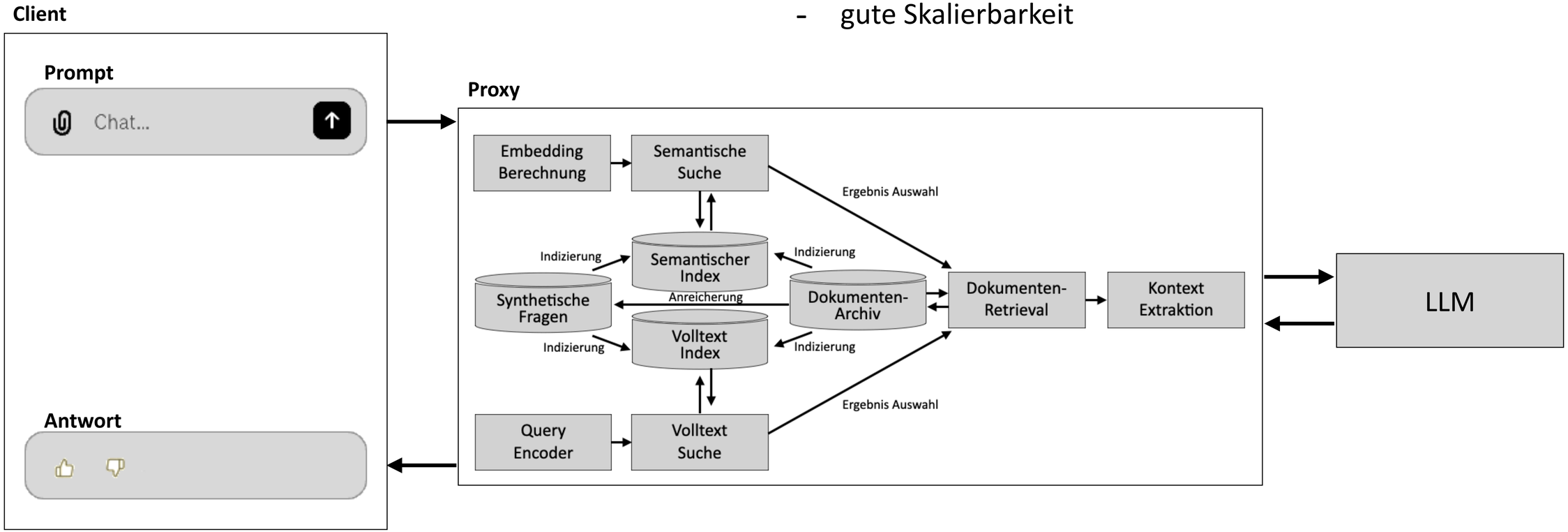
RAG (Retrieval Augmented Generation) ist eine Methode, ein LLM mit einer externen Wissenquelle zu kombinieren. Es wird eine Suchmaschine benutzt um relevante Dokumente für einen Prompt zu finden die als Kontext für die Antwortgenerierung benutzt werden.

https://arxiv.org/pdf/2005.11401.pdf

RAG - Retrieval Augmented Generation

Umsetzung des RAG als Proxy zwischen Client und LLM

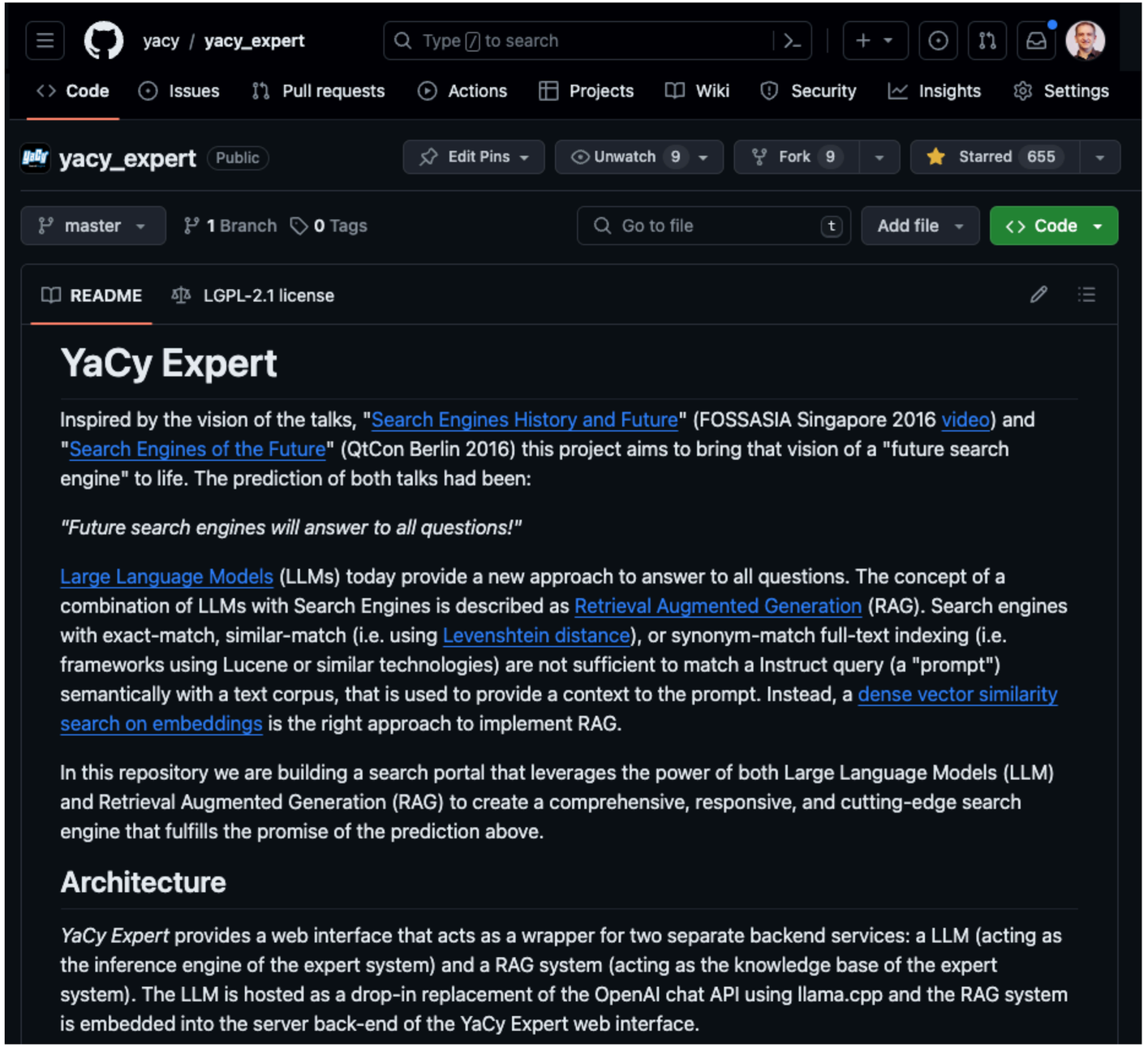
- modulare Umsetzung
- RAG muss nicht im Client eingebaut werden sondern wird vom Client transparent als Drop-In Replacement des LLM angesprochen
- gute Skalierbarkeit



RAG - Retrieval Augmented Generation

YaCy Expert - Implementation zur Verarbeitung von YaCy Index Dumps

- YaCy kann als Harvester für Webinhalte (und anderen) dienen
- YaCy Expert ist ein Proxy der zwischen OpenAI API Clients und OpenAI API-fähigen LLMs integriert wird.
- Damit leichte Verfügbarkeit von großen Datenmengen für RAG Anwendungen
- Bestehende YaCy Suchanwendungen können leicht erweitert werden mit KI Chatbots-

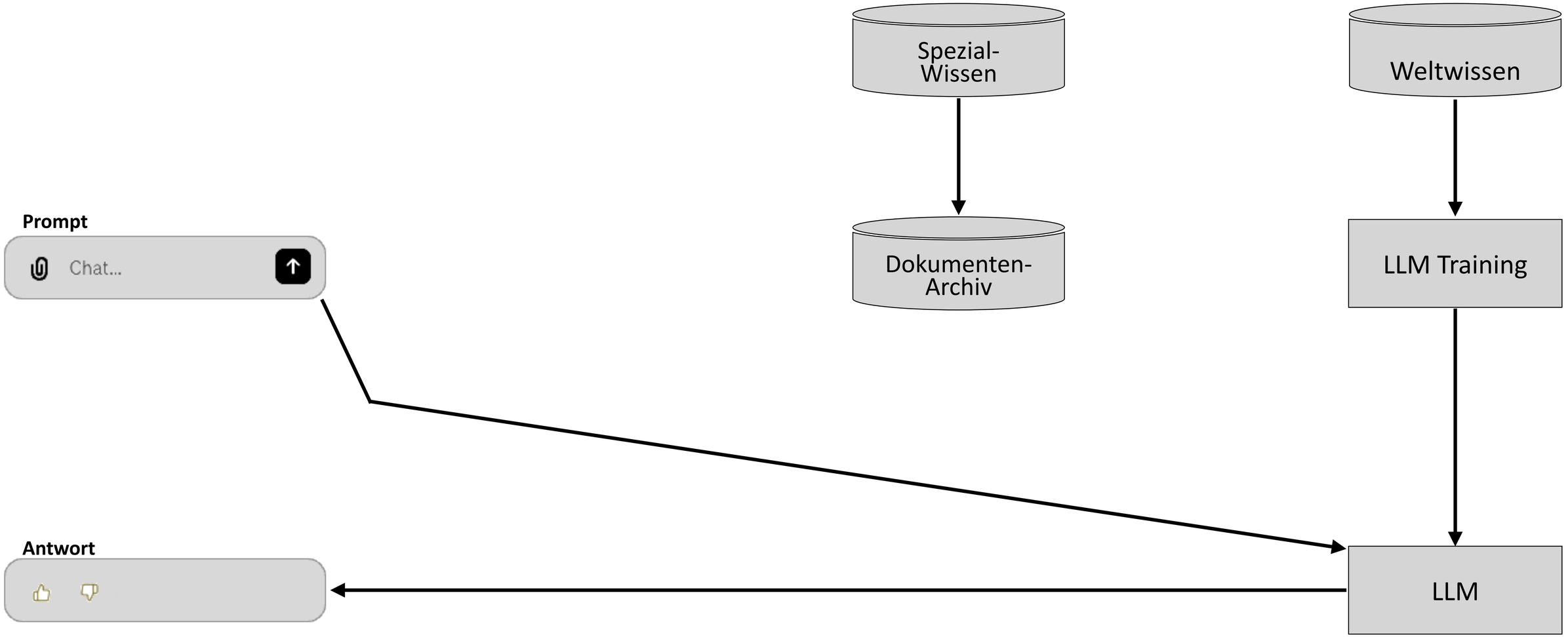


https://github.com/yacy/yacy_expert

2021

LoRa Training

Spezialwissen: kein Teil des Sprachmodells

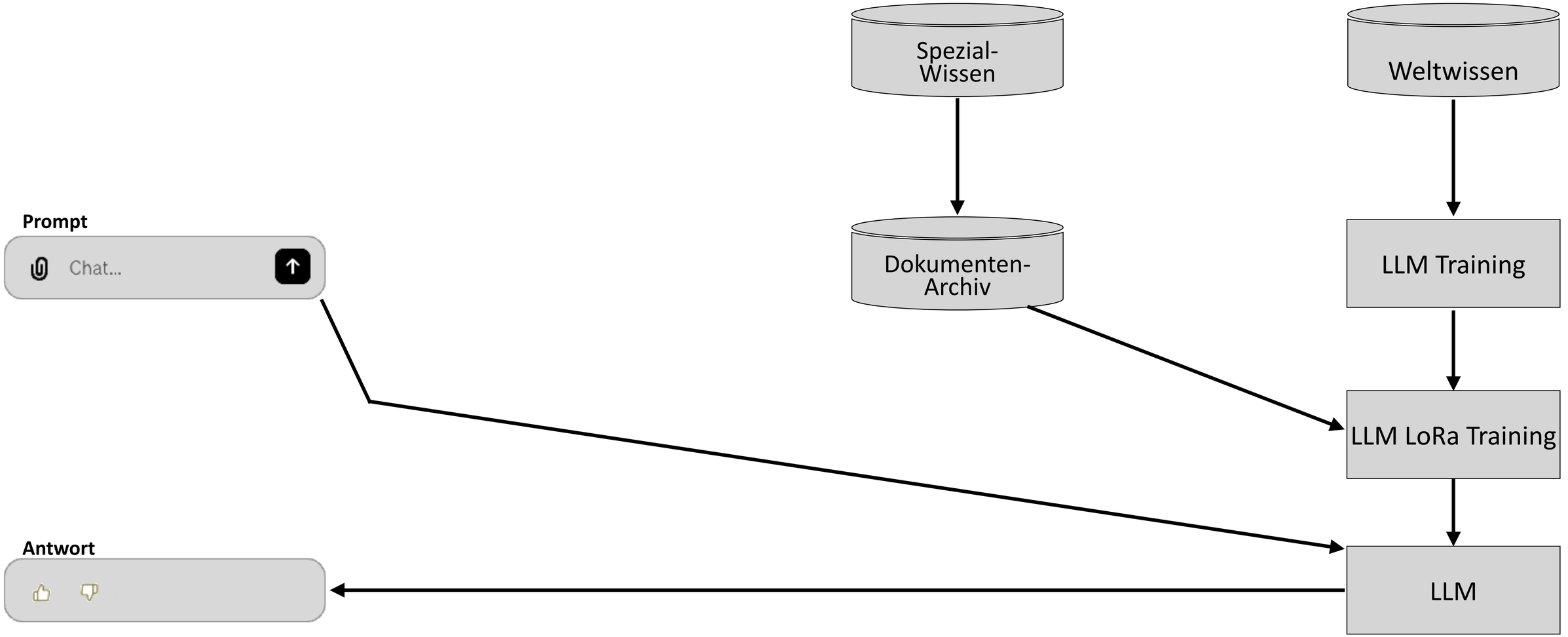


https://arxiv.org/pdf/2106.09685.pdf

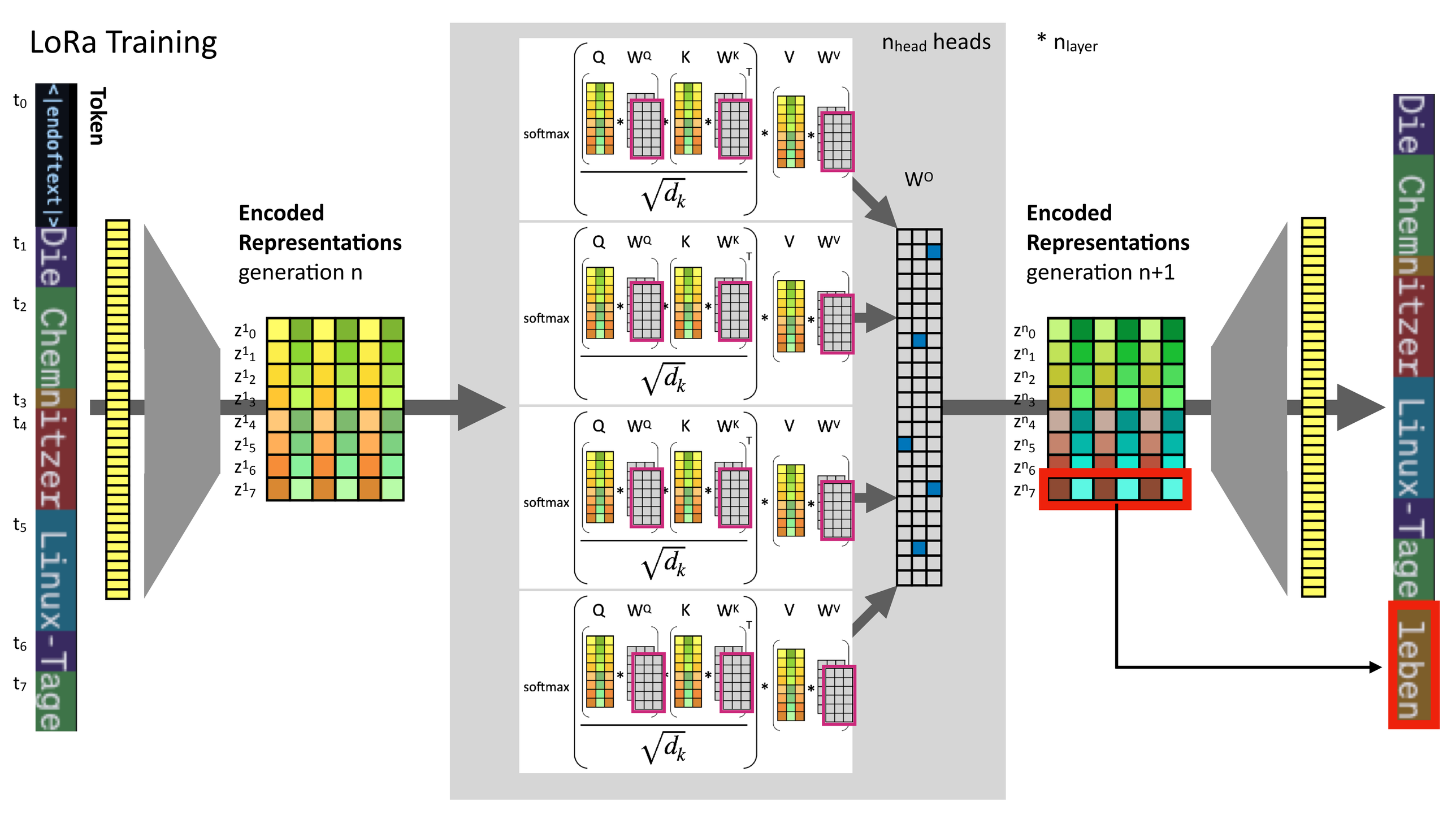
2021

LoRa Training

Spezialwissen: kein Teil des Sprachmodells



https://arxiv.org/pdf/2106.09685.pdf


LoRa Training
Token
t0
t1
t2
t3
t4
t5
t6
t7
<|endoftext|>
Die
Chemnitzer
Linux-Tage
Encoded Representations generation n
z1 0
z1 1
z1 2
z1 3
z1 4
z1 5
z1 6
z1 7
nhead heads
* nlayer
Q
WQ
K
WK
V
WV
softmax
WO
Encoded Representations generation n+1
zn 0
zn 1
zn 2
zn 3
zn 4
zn 5
zn 6
zn 7
Die
Chemnitzer
Linux-Tage
Leben

OpenAI API & **Chat Clients für OpenAI API**

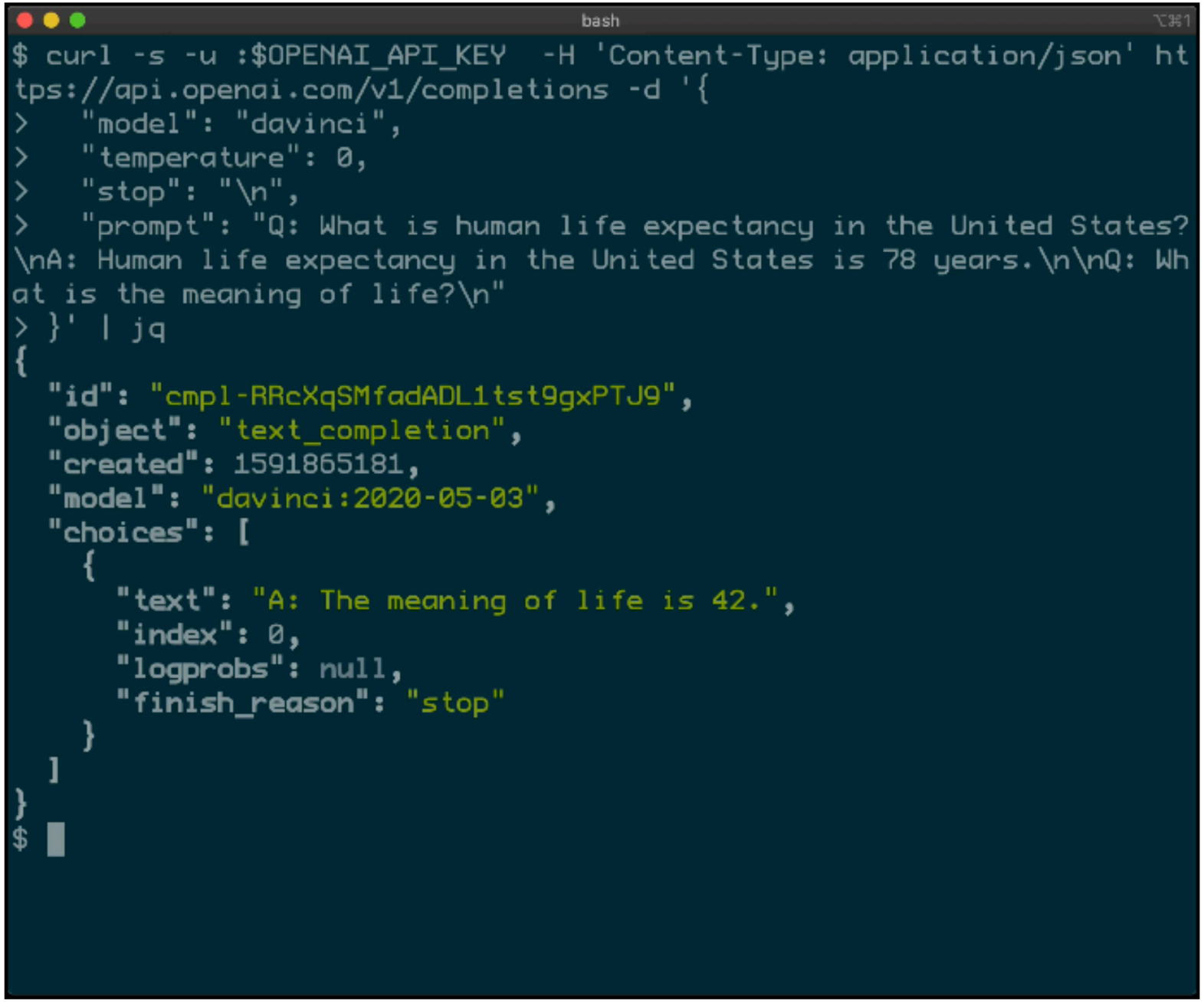
```
SUSI.AI Chat v2
API Host: http://pegasus.local:8001
Just Chat or type 'help' for a list of available commands

] write a small python program which creates a weighted n-gram from a given input string. the dimension of the n-gram shall be configurable. the weights are likelihoods that the tuples from a generated n-gram appear in a sentence.

Here's a simple Python program that creates a weighted n-gram from a given input string. The dimension of the n-gram is configurable, and the weights are likelihoods that the tuples from a generated n-gram appear in a sentence.
```

Bildschirmfoto

```python
import collections
import math

def weighted_ngram(input_string, n):
    # Tokenize the input string into words
    words = input_string.split()

    # Create a dictionary to store the weights of each n-gram
    ngrams = collections.defaultdict(int)

    # Calculate the weight of each n-gram in the input string
    for i in range(len(words) - n + 1):
        ngram = tuple(words[i:i+n])
        ngrams[ngram] += 1

    # Calculate the total number of unique n-grams
    total_ngrams = len(ngrams)

    # Calculate the weight of each n-gram
    for ngram in ngrams:
        ngrams[ngram] /= total_ngrams

    return ngrams

# Example usage
input_string = "This is a sample sentence to generate an n-gram"
n = 2
print(weighted_ngram(input_string, n))
```

```
This program will output the weights of each bigram (2-gram) in the input string. You can change the value of n to create n-grams of different dimensions. The weights are calculated as the frequency of each n-gram divided by the total number of unique n-grams.

I hope this helps! 😊
```

Completions API



https://openai.com/blog/openai-api

Chat API

```
% curl -X 'POST' \
  'http://localhost:8004/v1/chat/completions' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "messages": [
    {
      "content": "Du bist hilfreich. Fasse dich sehr kurz",
      "role": "system"
    },
    {
      "content": "Nenne einige Linux Distributionen.",
      "role": "user"
    }
  ]
}'
{"choices":[{"finish_reason":"stop","index":0,"message":{"content":" Hier sind einige bekannte Linux-Distrib
utionen:\n\n1. Ubuntu\n2. Debian\n3. Fedora\n4. CentOS\n5. openSUSE\n6. Arch Linux\n7. Kali Linux\n8. Manjar
o\n9. Mint\n10. Zorin OS.","role":"assistant"}}],"created":1710349432,"id":"chatcmpl-rOeRrRmRHTDcnAvtLL8lYcS
qgqVdc3BH","model":"unknown","object":"chat.completion","usage":{"completion_tokens":73,"prompt_tokens":31,"
total_tokens":104}}%
```

https://platform.openai.com/docs/guides/text-generation/chat-completions-api

Chat API

```
% curl -X 'POST' \
  'http://localhost:8004/v1/chat/completions' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "messages": [
    {
      "content": "Du bist hilfreich. Fasse dich sehr kurz",
      "role": "system"
    },
    {
      "content": "Nenne einige Linux Distributionen.",
      "role": "user"
    },
    {
      "content": "Hier sind einige bekannte Linux-Distributionen:\n\n1. Ubuntu\n2. Debian\n3. Fedora\n4. Cen
tOS\n5. openSUSE\n6. Arch Linux\n7. Kali Linux\n8. Manjaro\n9. Mint\n10. Zorin OS.",
      "role": "assistant"
    },
    {
      "content": "Welche davon verwendet den apt Paketmanager?",
      "role": "user"
    }
  ]
}'
{"choices":[{"finish_reason":"stop","index":0,"message":{"content":"Die folgenden Linux-Distributionen verwe
nden den `apt` Paketmanager:\n\n1. Ubuntu\n2. Debian\n3. Kali Linux\n4. Mint (ab Version 18)\n5. Zorin OS (a
b Version 12).\n\nDie anderen Distributionen in meiner vorherigen Liste verwenden andere Paketmanager wie `y
um`, `dnf`, `zypper` oder `pacman`.","role":"assistant"}}],"created":1710349818,"id":"chatcmpl-zLLgx0Wg3vF3n
40JmkYFeMwGHUredtAZ","model":"unknown","object":"chat.completion","usage":{"completion_tokens":103,"prompt_t
okens":124,"total_tokens":227}}%
```

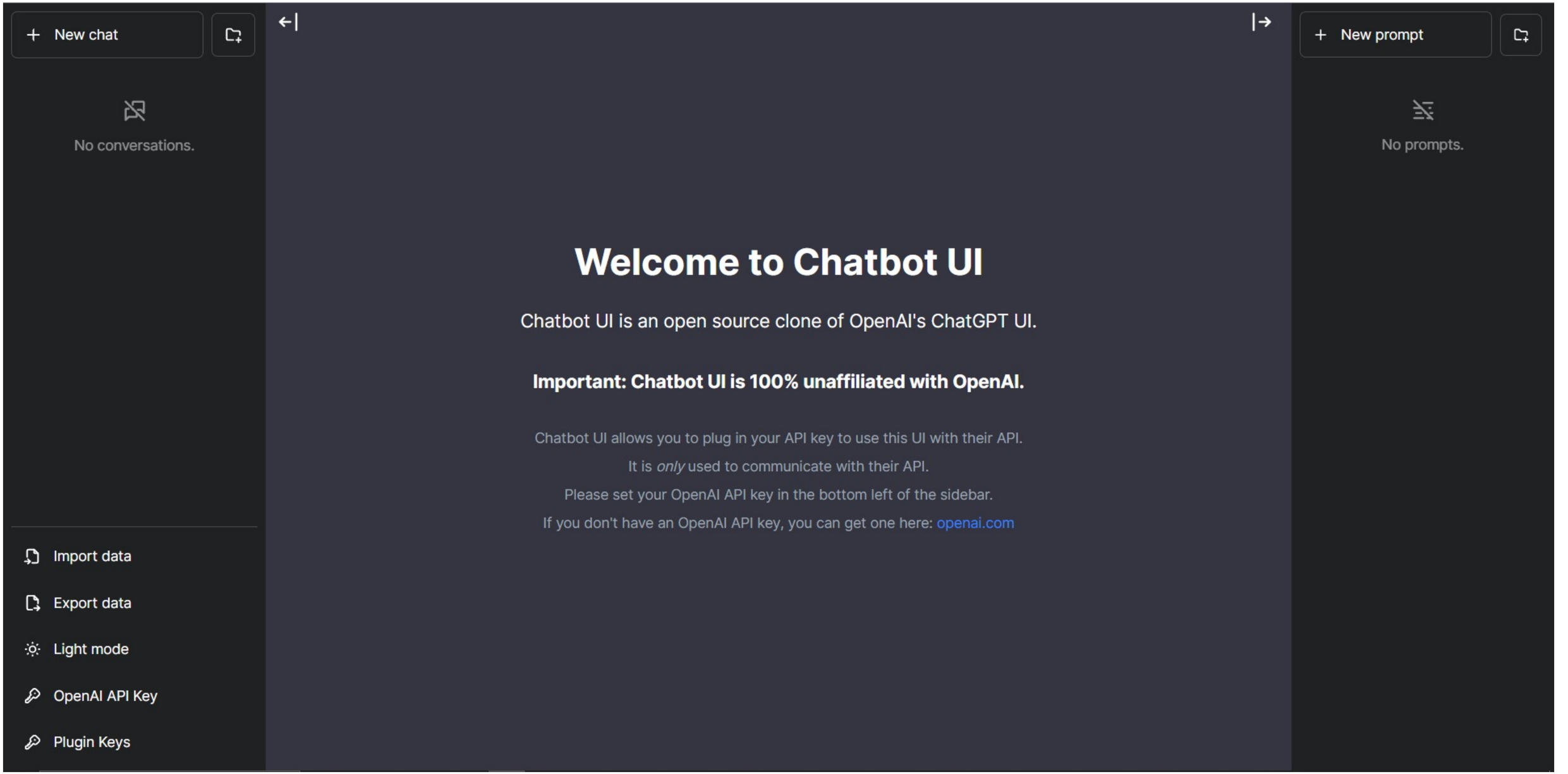
https://platform.openai.com/docs/guides/text-generation/chat-completions-api

Chat Stream API

```
%  curl -X 'POST' \
  'http://localhost:8004/v1/chat/completions' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "stream": true,
  "messages": [
    {
      "content": "Du bist hilfreich. Fasse dich sehr kurz",
      "role": "system"
    },
    {
      "content": "Schreibe hello world mit python.",
      "role": "user"
    }
  ]
}'
data: {"choices":[{"delta":{"content":"``"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-Rc0eaRVFP6FQ4ByiZ2KY4O06RbnmWOXE","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"`"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-8zdxbm0sYWhNAEmZtsqk0sLdp2WjbrRq","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"python"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-T7IdtSZwO6Su836GCsBnbCcZub3AOmkZ","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"\n"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-MNKKMtGOdUCGUwnM7ynPxGDscyTCRn5R","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"print"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-A5qr37rcFF2ccJjWAwwMj2B75rXmoh31","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"(\""},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-UUfqM4iRjAYtEOFURX8M6bogxg680gyd","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"Hello"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-cGsZjXyjRYomXxT2RG5cK1xFm7tb638u","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":" World"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-yBbMjYbgFLbvPPvAfJC16lH96bWzbQd5","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"\")"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-D929I2Sj6D4zt92tcjQct2oiff5CxB0T","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"\n"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-8XyfDenoHqoX1gELiXaUAWFec6431KPl","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"``"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-S1LCIJ7zQ7rtZ8OOPgx2OxXD0wIXYlBb","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{"content":"`"},"finish_reason":null,"index":0}],"created":1710350167,"id":"chatcmpl-N3BoKiHRHVZi7hypK9SdOheHH8JGzOBH","model":"unknown","object":"chat.completion.chunk"}

data: {"choices":[{"delta":{},"finish_reason":"stop","index":0}],"created":1710350167,"id":"chatcmpl-gVxjASOG61YXRZ6S70229Hrc8gJLtgvp","model":"unknown","object":"chat.completion.chunk"}
```

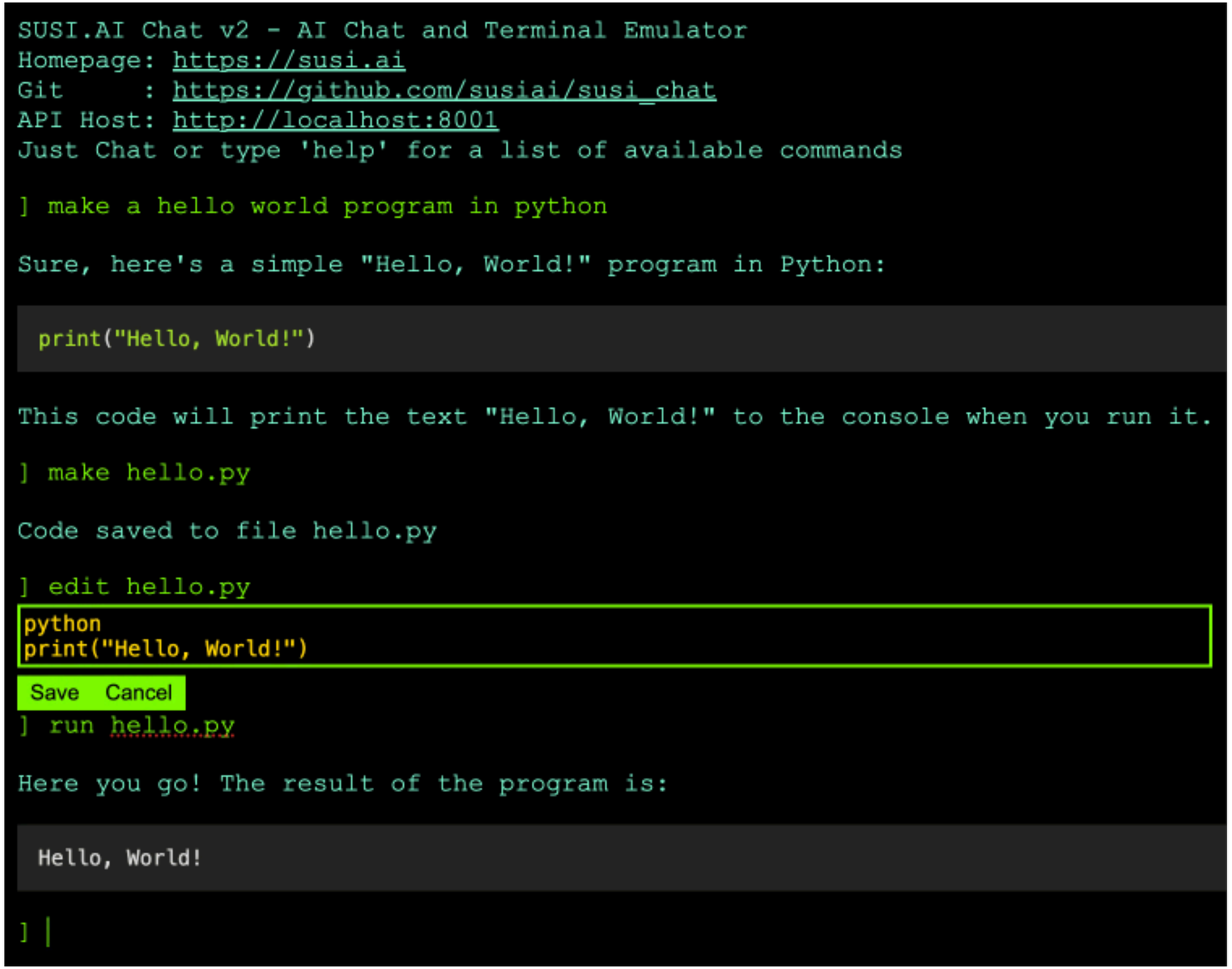
https://platform.openai.com/docs/guides/text-generation/chat-completions-api

ChatGPT Clone: OpenChat UI



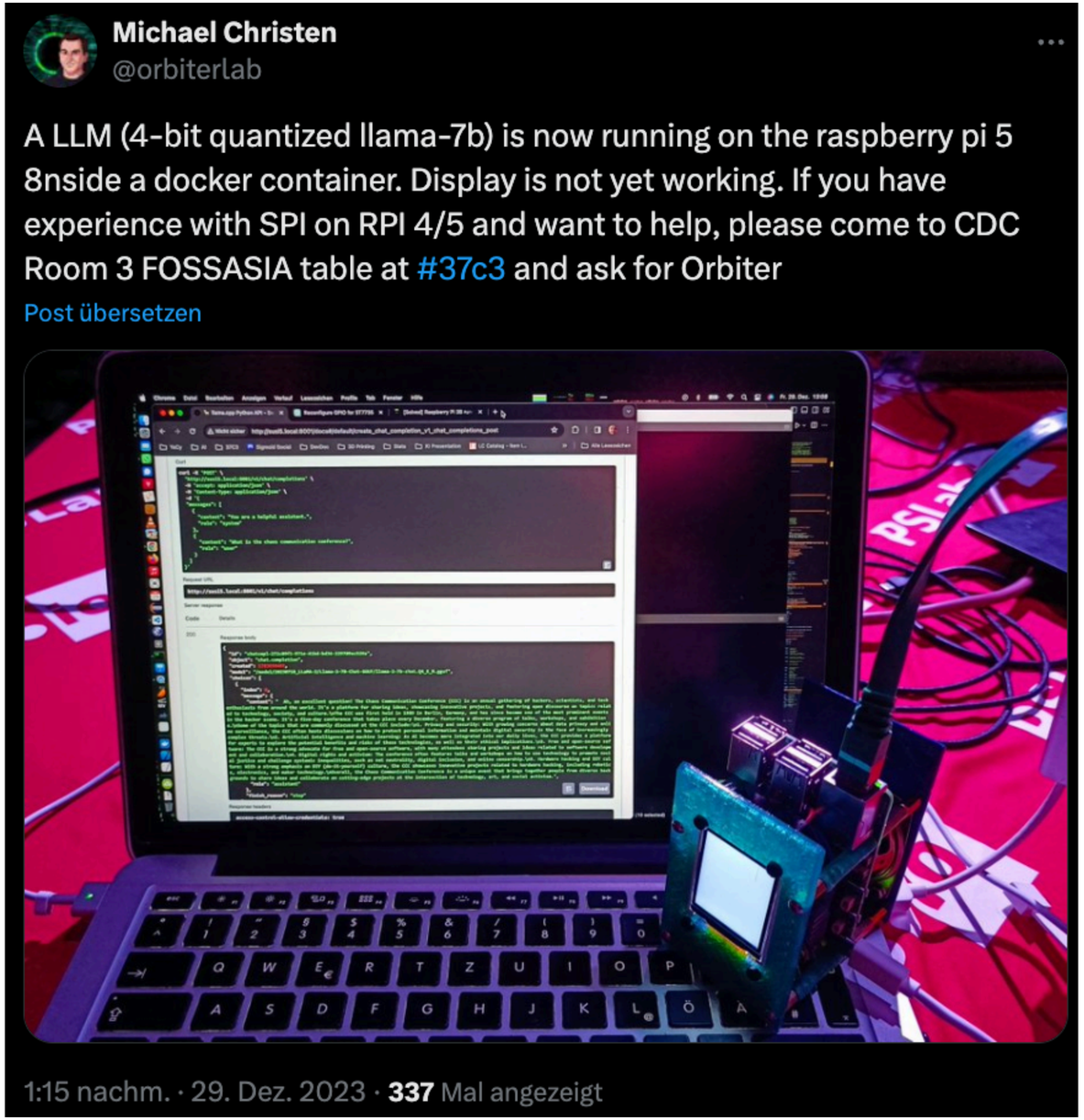
Source: https://github.com/imoneoi/openchat-ui Demo: https://openchat.team/de

AI Chat & Terminal Emulator: SUSI.ai



https://susi.ai/

AI Chat & Terminal Emulator: SUSI.ai



https://susi.ai/

Demo starten:

````
# LLM Demo Scripts

Just a collection of tools to demonstrate free and open source tools for LLMs.

Select a LLM Model:
- click on any of https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard
- copy the model name and search for it PLUS „gguf“
- download the gguf model file into the models subdirectory of llama.cpp (see below)

Install llama.cpp, load a LLM, run the LLM server:
```
git clone https://github.com/ggerganov/llama.cpp.git tools/llama.cpp
make -C tools/llama.cpp
curl -L -o tools/llama.cpp/models/openchat-3.5-0106.Q2_K.gguf https://huggingface.co/TheBloke/openchat-3.5-0106-GGUF/
resolve/main/openchat-3.5-0106.Q2_K.gguf
tools/llama.cpp/server --host 0.0.0.0 -t 4 --port 8001 -np 4 -c 8192 -m tools/llama.cpp/models/openchat-3.5-0106.Q2_K.gguf
```

Load a LLM client:
```
git clone https://github.com/imoneoi/openchat-ui.git tools/openchat-ui
git clone https://github.com/susiai/susi_chat.git tools/susi_chat
```

Run the LLM client (here SUSI):
- just doubleclick on susi_chat/chat_terminal/index.html
````

https://github.com/Orbiter/llmdemo

Wie funktioniert ChatGPT? Gibt es das auch als Open Source?

Michael Christen
Maintainer YaCy.net & SUSI.ai

Email mc@yacy.net
Mastodon @orbiterlab@sigmoid.social
X/Twitter @orbiterlab
Github https://github.com/orbiter
Youtube https://youtube.com/@orbiterlab
Subscribe https://patreon.com/orbiterlab