INF Informatik
Hochschule Reutlingen

# Intelligente Informationssysteme
# Block 1 – Large Language Models

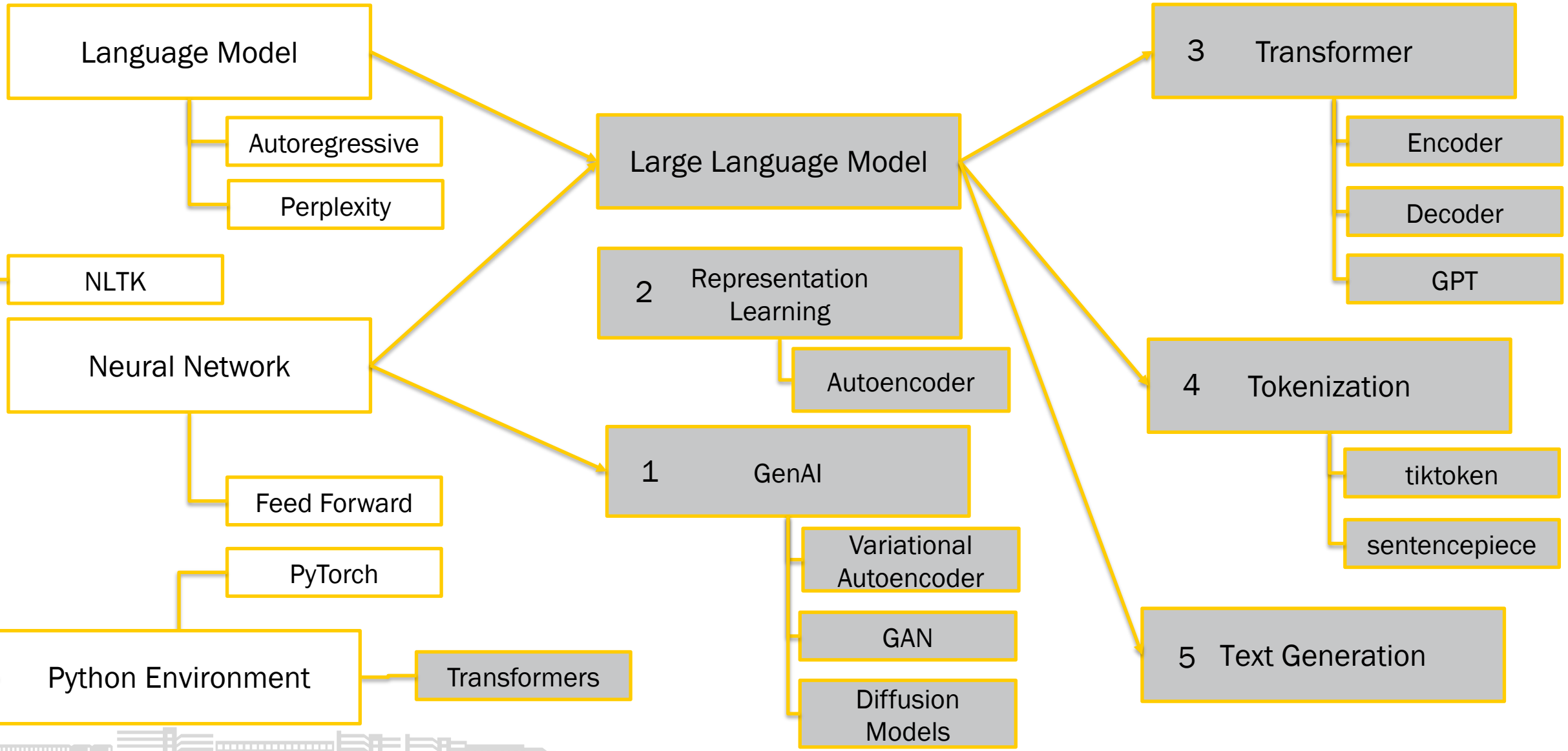
Dominik Neumann

# Language Models - Agenda

Informatik
Hochschule Reutlingen

The huge amount of parameters (of the mode) makes a Language Model a Large Language Model.

# Overview

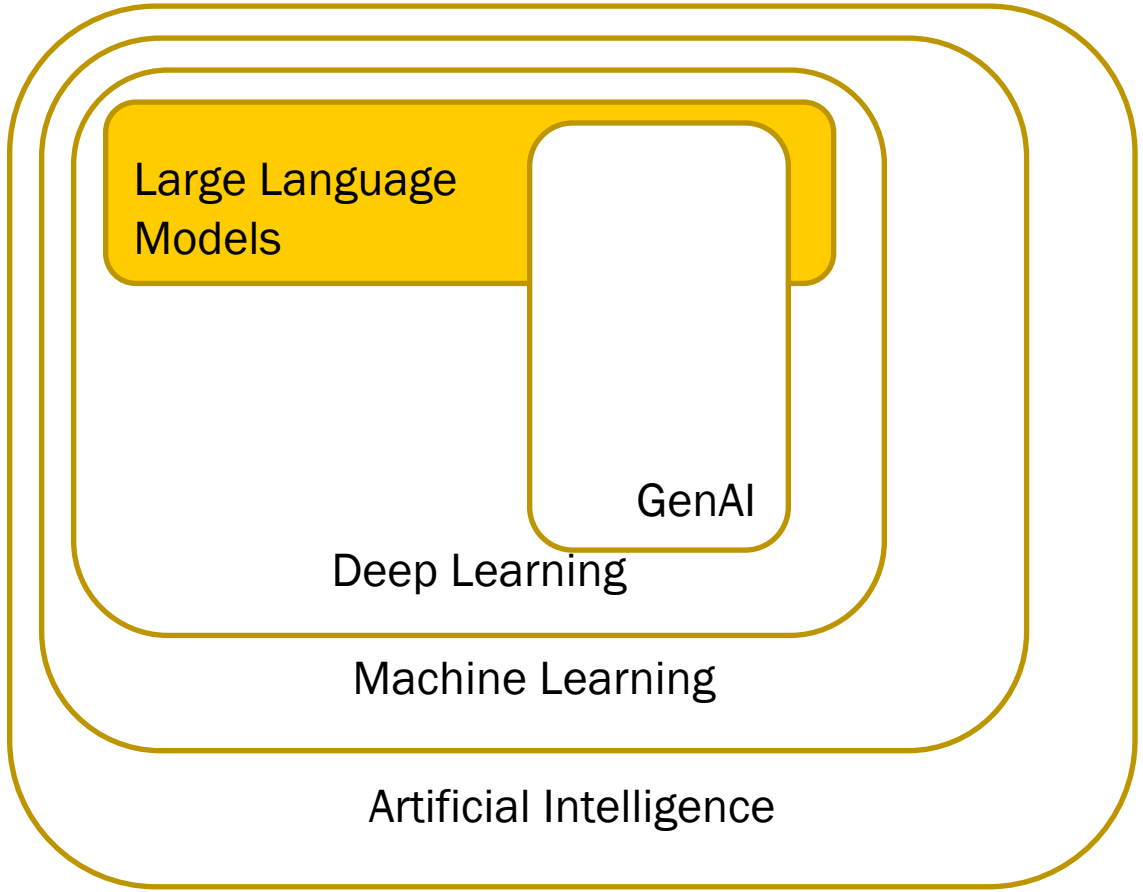

Language Model
Autoregressive
Perplexity
NLTK
Neural Network
Feed Forward
PyTorch
Python Environment
Transformers
Large Language Model
2 Representation Learning
Autoencoder
1 GenAI
Variational Autoencoder
GAN
Diffusion Models
3 Transformer
Encoder
Decoder
GPT
4 Tokenization
tiktoken
sentencepiece
5 Text Generation

# 01
# Generative Artificial Intelligence

# Large Language Models

Hochschule Reutlingen

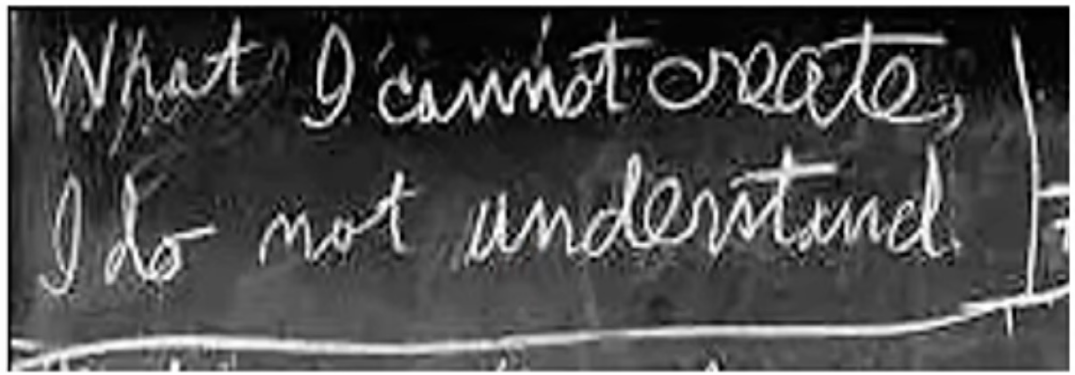
## There is an intersection between GenAI & Large Language Models




Large Language Models represent a specific application of deep learning techniques, leveraging their ability to understand, process and generate human like text.

A Large Language Model is a very large deep learning model that is pre-trained on massive amounts of data.

But what is GenAI?

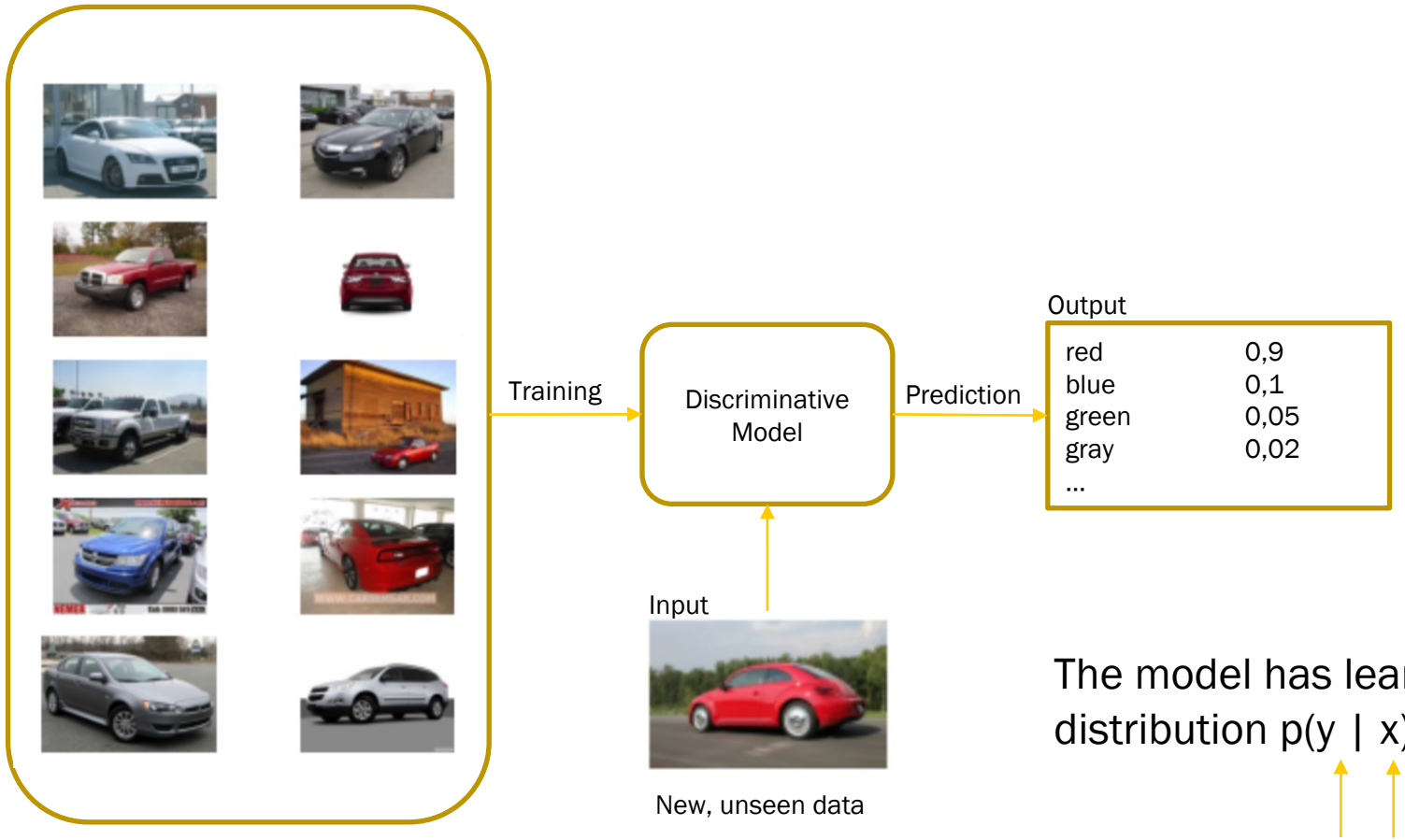



“What I cannot create, I do not understand..”
- Richard Feynman

“a system that can generate realistic data must have gained at least some understanding of the real world.”

# Discriminative versus Generative AI

## Car color classification as task for a discriminative model




1. The model is trained to classify the color of cars
2. For a picture of a car the model outputs a vector of probabilities for each color class

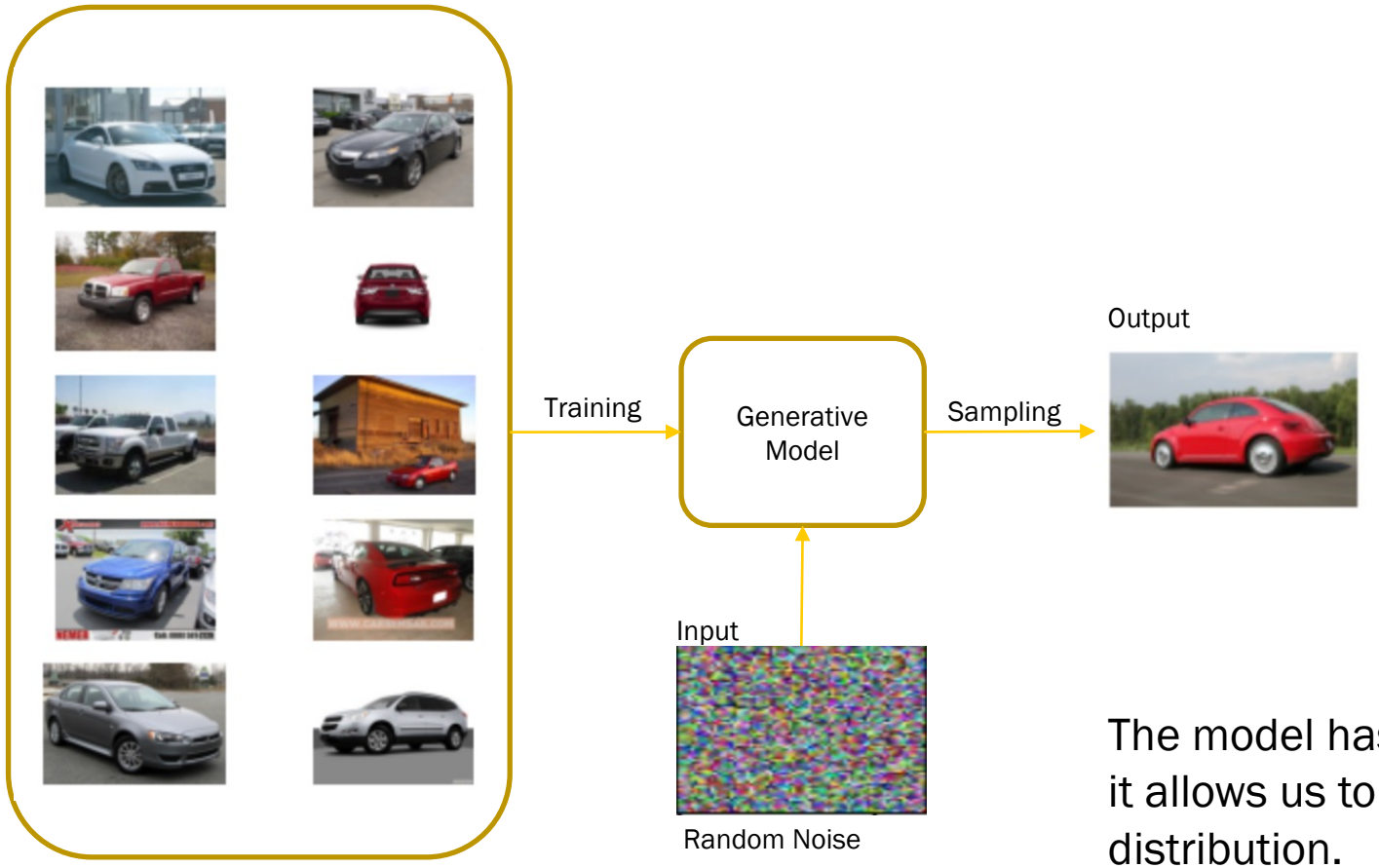
The model has learnt a conditional probability distribution p(y | x).

label sample

In analogy to [DLG] Figure 1-2

# Discriminative versus Generative AI

## Car data generation as task for a generative model




1. The model is trained to understand pictures of cars
2. The model samples from random noise new pictures of cars

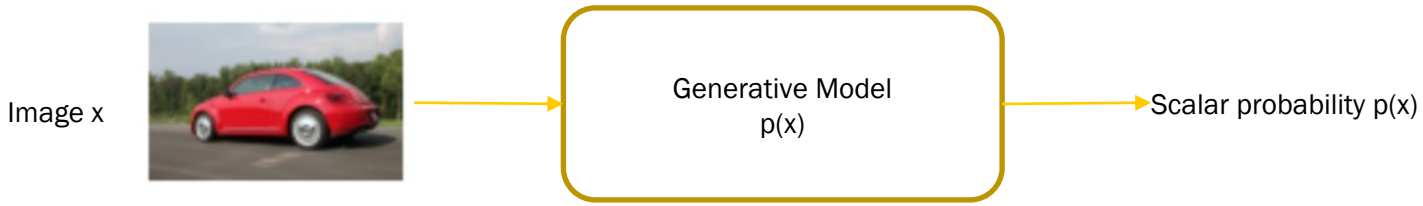
The model has learnt a probability distribution p(x) and it allows us to sample new data from the learned distribution.

In analogy to [DLG] Figure 1-1

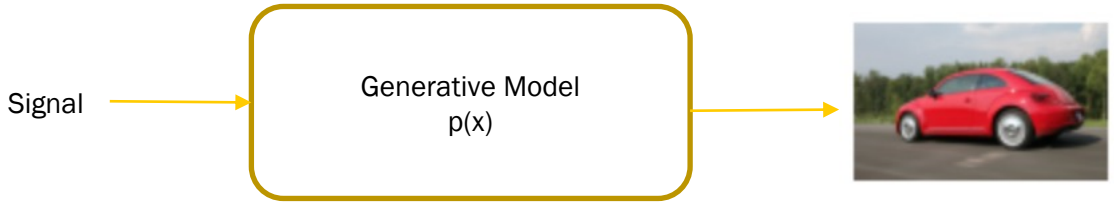
# Generative AI

A statistical generative model is a probability distribution p(x):

- Data (images or sequences of text)
- Prior Knowledge (--> parametric Form: e.g. Gaussian; loss function: e.g. maximum likelihood; optimization algorithm, .....)




How likely is that image x according to my statistical generative model?




It is generative, because sampling from p(x) generates new images.

# Why is Generation so hard?

## “Curse of Dimensionality” in Pixel-Space

Each image has n pixels (black or white)
→ There are 2 x 2 x … x 2 = 2^n possible black & white images.

- for 32x32=784 pixels → 2ˆ784 possible black & white images.

But only some of these pictures representing numbers! Most of them are white noise!

- for rgb 1000x1000 colored pixels → (256 x 256 x236)ˆ1,000,000 possible rgb images.

We have to find a function (probability distribution) that could be used to generate the images we want.

# Why is Generation so hard?

Hochschule Reutlingen

## “Curse of Dimensionality” in Vocabulary-Space:

- The English language has around 170,000 words.
- The Oxford English dictionary defines approximately 600,000 word-forms.
- There around 1,000,000 words with accounts for multiple forms of the same word and alongside words that are not used in modern English.

- Assume that on average, sentences today range from 15 to 20 words.

→ 1,000,000^15 = 10^6^15 = 10^90 possible word sequences with 15 words!

Compare to the Universe:

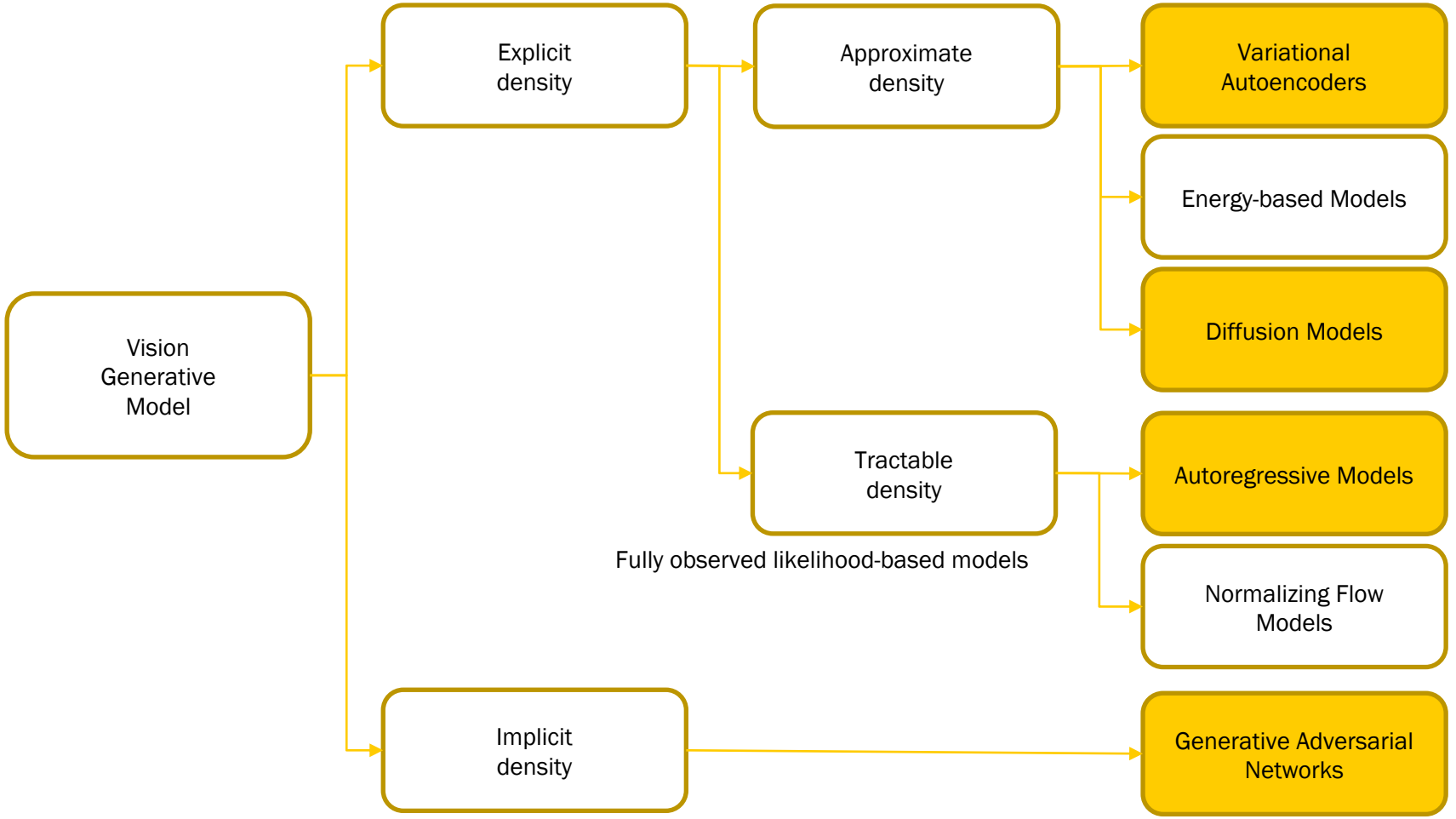
→ 10^78 to 10^82 atoms in the universe

Source:

- Wikipedia – https://en.wikipedia.org/wiki/English_language
- Medium - https://medium.com/@theacropolitan/sentence-length-has-declined-75-in-the-past-500-years-2e40f80f589f
- https://wordsrated.com/how-many-words-are-in-the-english-language/
- Oxford - https://educationblog.oup.com/secondary/maths/numbers-of-atoms-in-the-universe

# There are six families of generative models

The main difficulty present in explicit density models is designing a model that can capture all of the complexity of the data to be generated while still maintaining computational tractability.




Compact Represenation learning, Image denoising
Generation of images, text, musical compositions, ...

Object recognition, image restauration, sequence labeling, anomaly detection

State of the art in Text-2-Image Generation: StableDifusion, Imagen, DALL-E 3

Were state of the art before diffusion models

In analogy to [DLG] Figure 1-10, originated by the taxonomy of deep generative models by Ian Goodfellow [GAN]

Informatik
Hochschule Reutlingen

# Variational Autoencoder

latent space: containing one vectors for each image, that contains the instructions to generate the image.

$$P(x) = \int P(x|z)P(z)dz$$

Training Time

x → Encoder Model → z (With $\mu$ and $\sigma$) → z' → Decoder Model → x'

$Q(z|x) \longleftrightarrow N(\mu, \sigma)$

Learn this term as a function represented by a NN

Sample for Gaussian Distribution with $\mu$ and $\sigma$

Loss:
1. Reconstruction error: the output should be similar to the input.
2. Q(z|x) should be similar to the prior N($\mu$, $\sigma$)

1. An input image is passed through an encoder network.
2. The encoder outputs parameters of a distribution Q(z|x).
3. A latent vector z' is sampled from Q(z|x). If the encoder learned to do its job well, most chances are z will contain the information describing x.
4. The decoder decodes z' into an image x'.

# Generative AI

Informatik
Hochschule Reutlingen

## 03_VAE_MNIST.ipynb

### Variational Autoencoder

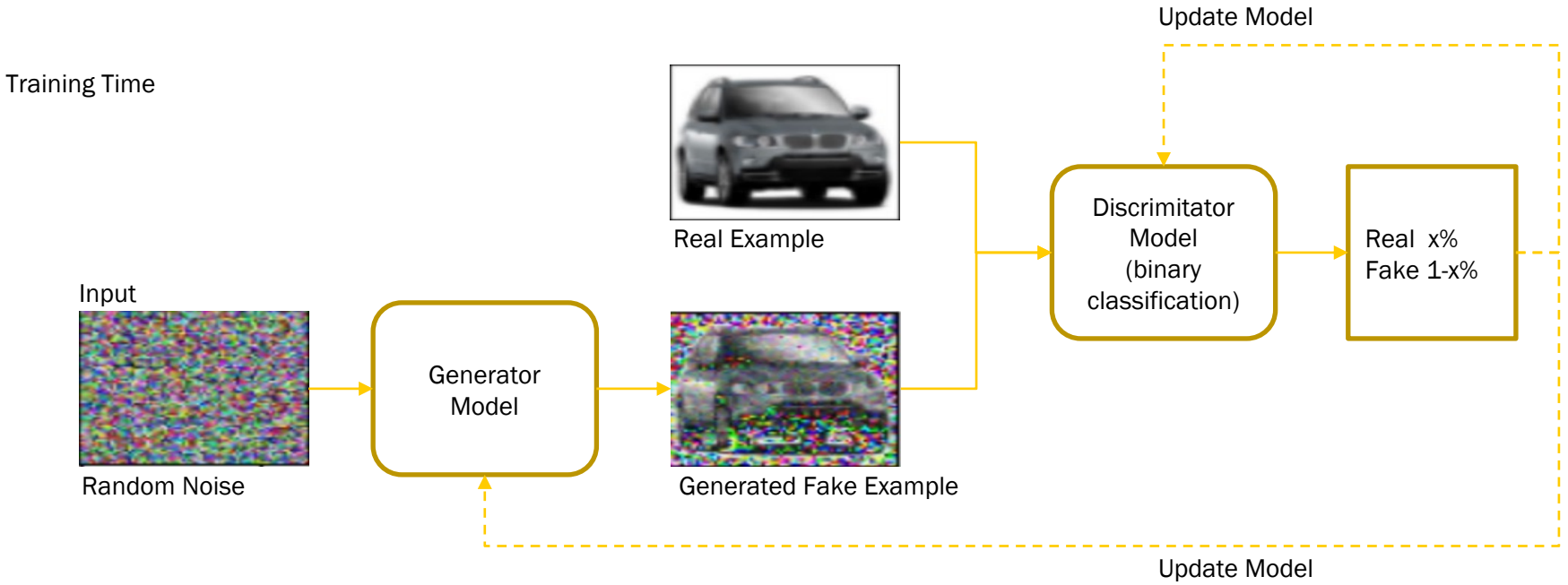
[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
```

### Load MNIST Data

[2]:
```
transform = transforms.Compose([transforms.ToTensor()])

path = "./data"
train_dataset = MNIST(path, transform=transform, download=True)
```

Hochschule Reutlingen

# Generative Adversarial Network




2014, Ian Goodfellow

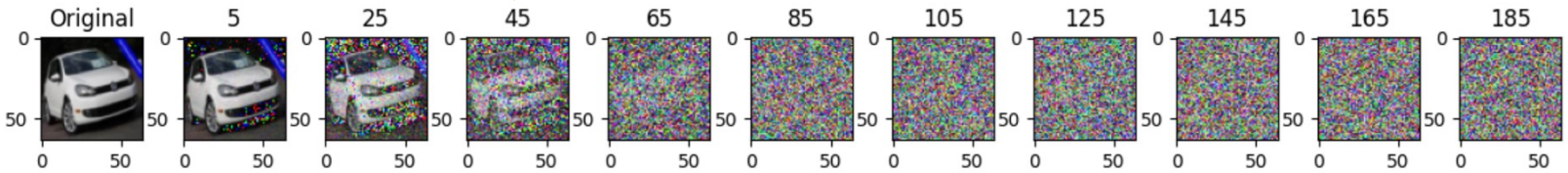
Hochschule Reutlingen

# Diffusion Models

Diffusion Models work:

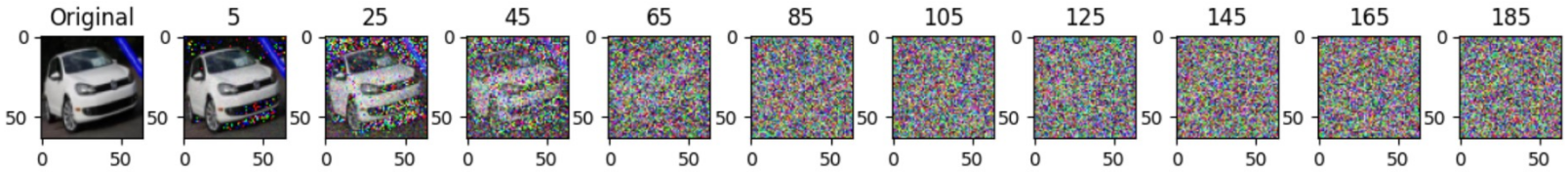
- by destroying an input by gradually adding noise
- they learn a model to reconstruct the previous image
- and then denoising it by the learned model.

Forward process




Markov chain: where each timestep depends only on the previous timestep

Model




use the Model

use the Model

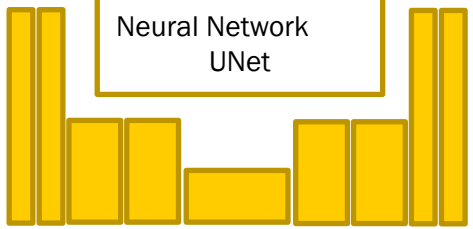
use the Model

Parametrized Backward process

Noise Scheduler

Timestep Encoding

Neural Network UNet



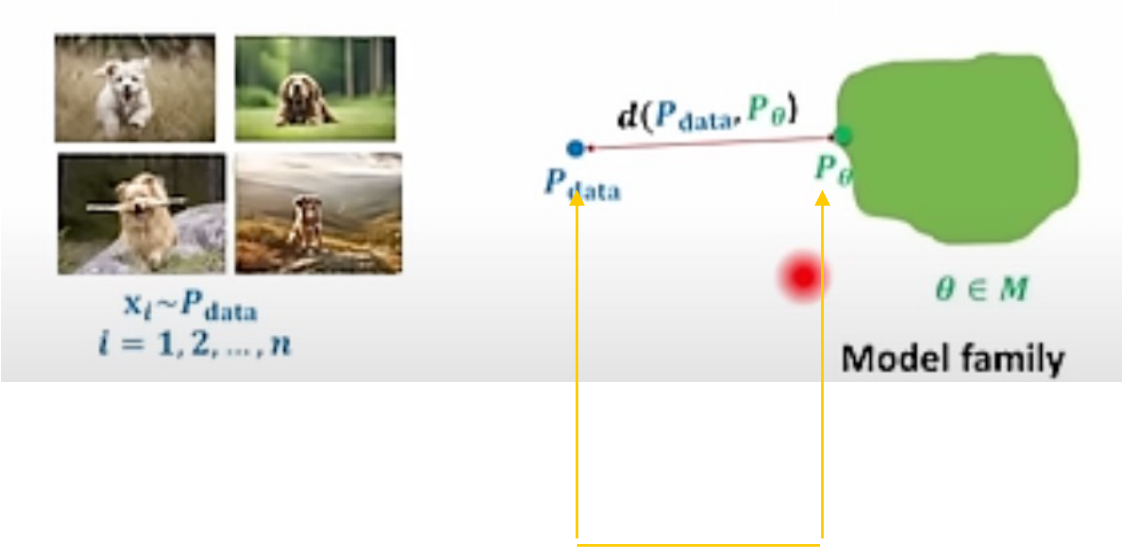
Source: Denoising Diffusion Probabilistic Model - https://arxiv.org/abs/2006.11239

Informatik
Hochschule Reutlingen

# Summary

Main Challenges in generative AI are:

- Representing Probability Distributions: How to do we model the distribution?




Loss Function: Notion of Similarity

→ Reformulate the Problem as an Optimization Problem

Our Goal is to learn a probability distribution p (as our model) so that:

1. Generation is possible by sampling:
   →e.g. x_new ~ p(x), x_new should look like a dog

2. Density Estimation could be used for anomaly prediction:
   → p(x) should be high if x looks like a dog and low otherwise (anomaly prediction)

3. Unsupervised representation Learning is possible for similarity use cases:
   → we should be able to learn what the images have in common (their hidden or latent features).

# 02
# Representation Learning

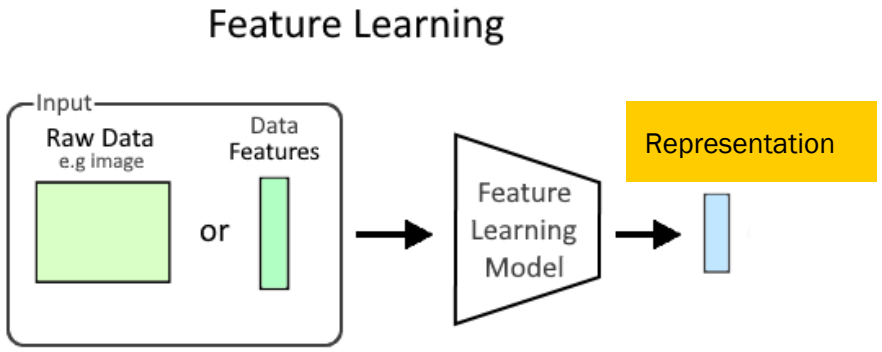
# Concept of a Model in Machine Learning

Informatik
Hochschule Reutlingen

## Representation Learning

- Representation Learning is a form of information compression.
- We want to extract the relevant and hidden information (latent features) that characterize our objects into a vector representation.
- Having a vector representation of an object in a vector space, then we are able, to use similarity measures to compare objects or construct new objects using the representations.

- Image to Vector
- Word to Vector
- Sentence to Vector
- Graph Node to Vector




Source: Wikipdeoia - By Fgpacini - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=123926277

Hochschule Reutlingen

# 02_AE_MNIST.ipynb

## Autoencoder

The architecture consists of an Encoder and a Decoder module. The Encoder compress the data into an abstract latent space) and the Decoder decompress the encoded information and reconstruct the data.

The goal of an Autoencoder is to learn a compressed representation of the input data, focusing on minimizing the original input and the reconstruction).

see: Mark Liu, "Learn Generative AI with PyTorch", Chapter 7.2

[1]:
```
import torch
import torch.nn as nn

import torchvision
import torchvision.transforms as T

from torchvision.datasets import MNIST
```

### Load MNIST data

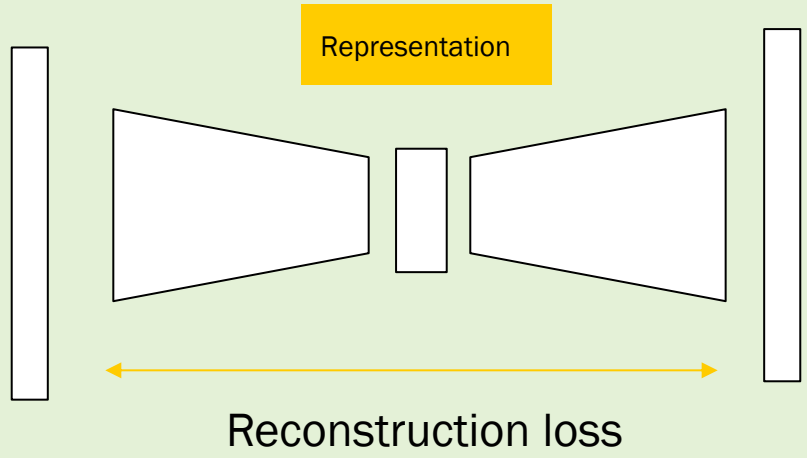
[2]:
```
transform = T.Compose([T.ToTensor()])

train_set = MNIST("./data", train=True, download=True, transform=transform)
test_set = MNIST("./data", train=False, download=True, transform=transform)
```

[3]:
```
for example in iter(train_set):
    print(example[0].size(), example[1])
    break
```

```
torch.Size([1, 28, 28]) 5
```

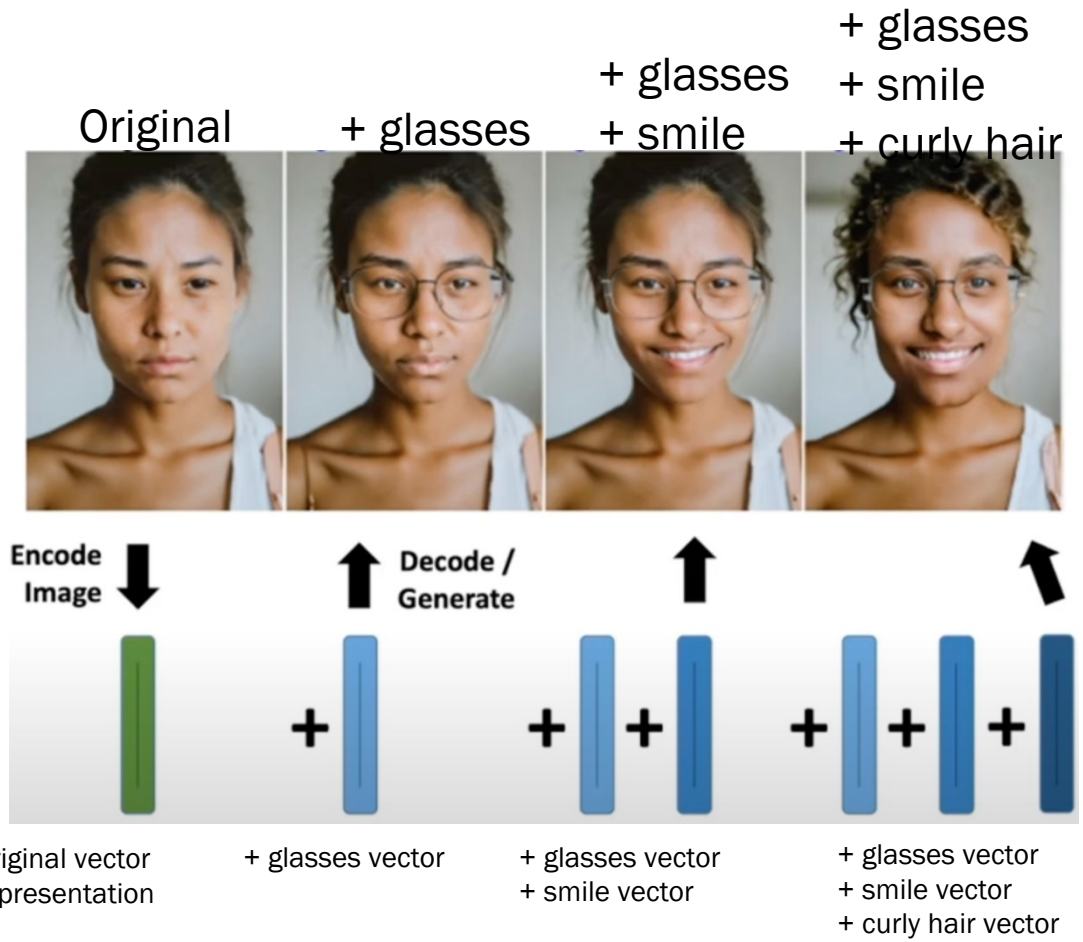
[4]:
```
#create batches with 32 images per batch
batch_size = 32
```

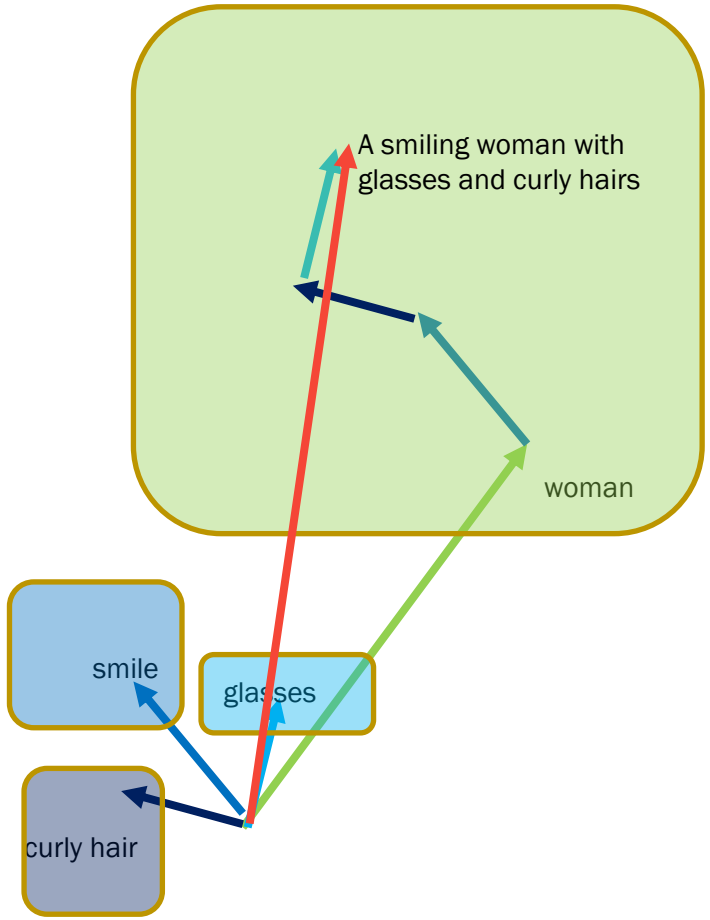
Autoencoder



# Concept of a Model in Machine Learning

## Representation Learning




Source: Deep Gernerative Models: https://kuleshov-group.github.io/dgm-website/

# 03
# Tansformer

## Recap: Language Models are statistical Models

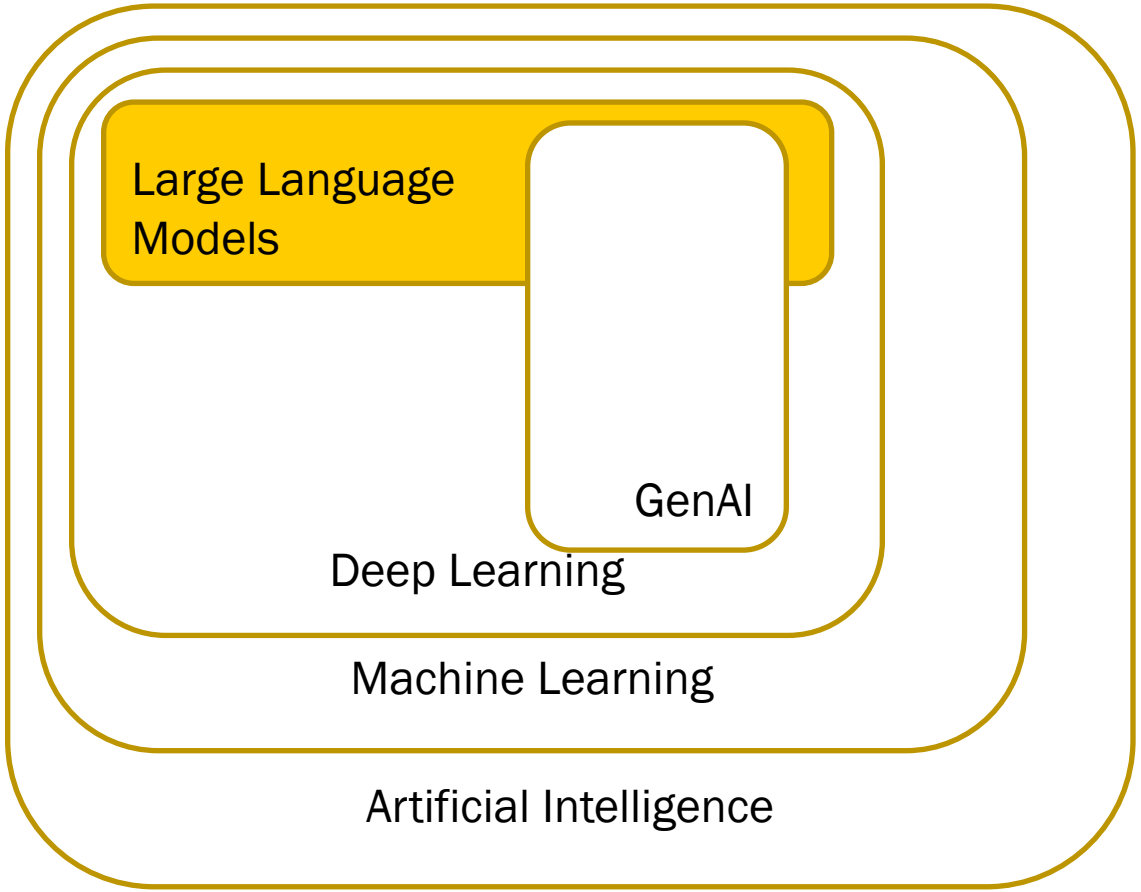
$$p(\mathbf{x}) = p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1})$$
$$= p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2) \ldots$$

**Language Model:**

- Probability distribution over a sequence of **discrete tokens** $\mathbf{x} = (x_1, \ldots, x_T)$ where each token can take a value from a **vocabulary** $\mathcal{V}$ $(x_t \in \mathcal{V})$
- Decomposes into a sequence of conditional distributions
- The history (conditioning variables/tokens) is called **context** in NLP

From: Deep Learning (Prof. Andreas Geiger, University of Tübingen) - Lecture 9.2 (Natural Language Processing: Traditional Language Models)

## Recap: There is an intersection between GenAI & Large Language Models




Large Language Models represent a specific application of deep learning techniques, leveraging their ability to understand, process and generate human like text.

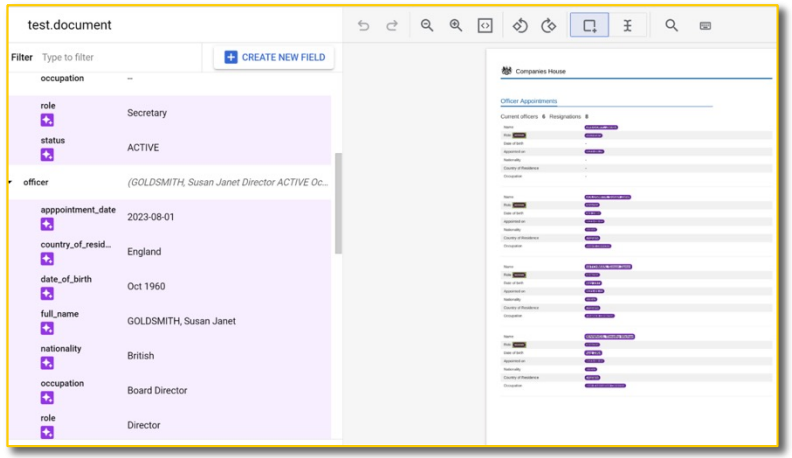
A Large Language Model is a very large deep learning model that is pre-trained on massive amounts of data.

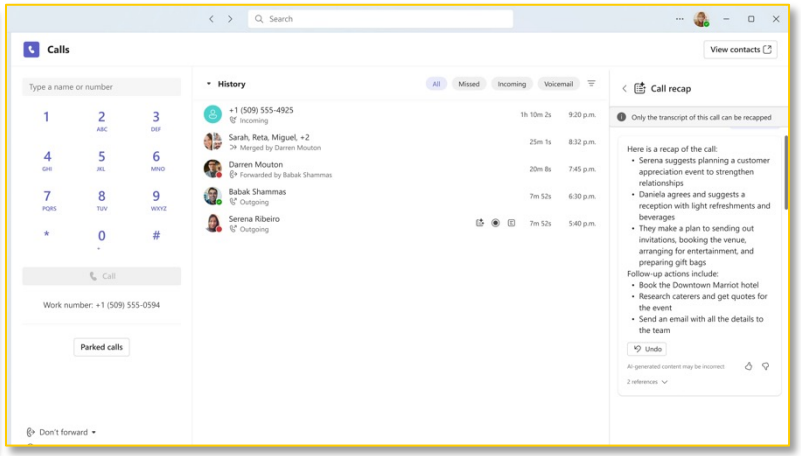
But what is GenAI?

# Large Language Models

INF Informatik
Hochschule Reutlingen

## GenAI handles token streams – which can be more than mere language



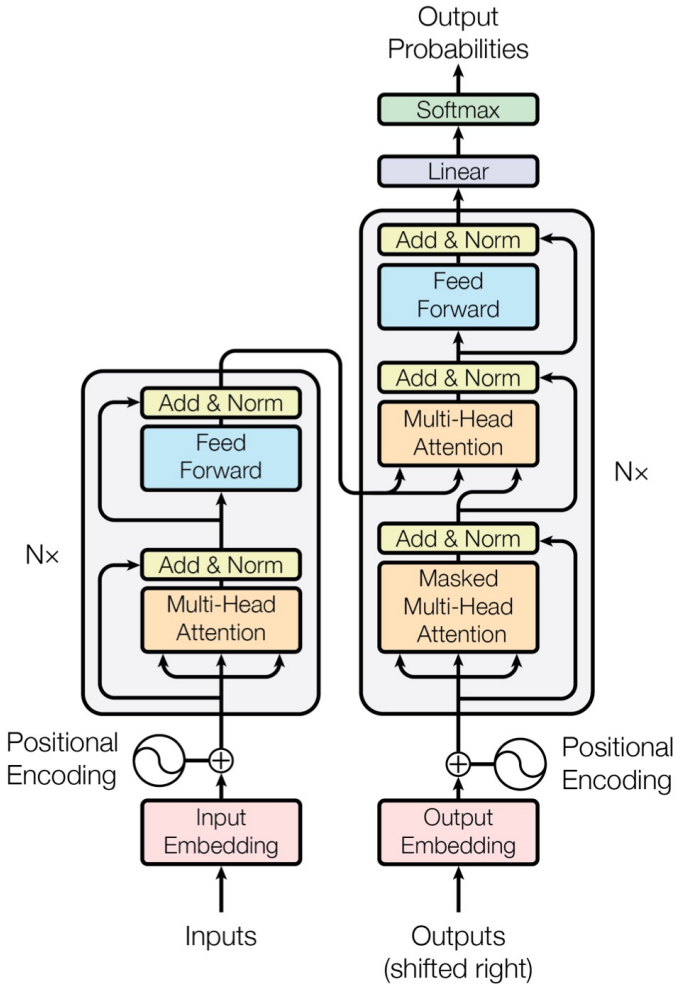

**pictures / videos** | **text / documents** | **sound**

extract, interpret, process, create

**orchestrate**

**The orchestration of individual, new capabilities along business flows unlocks tremendous value.**

# Transformer Architecture



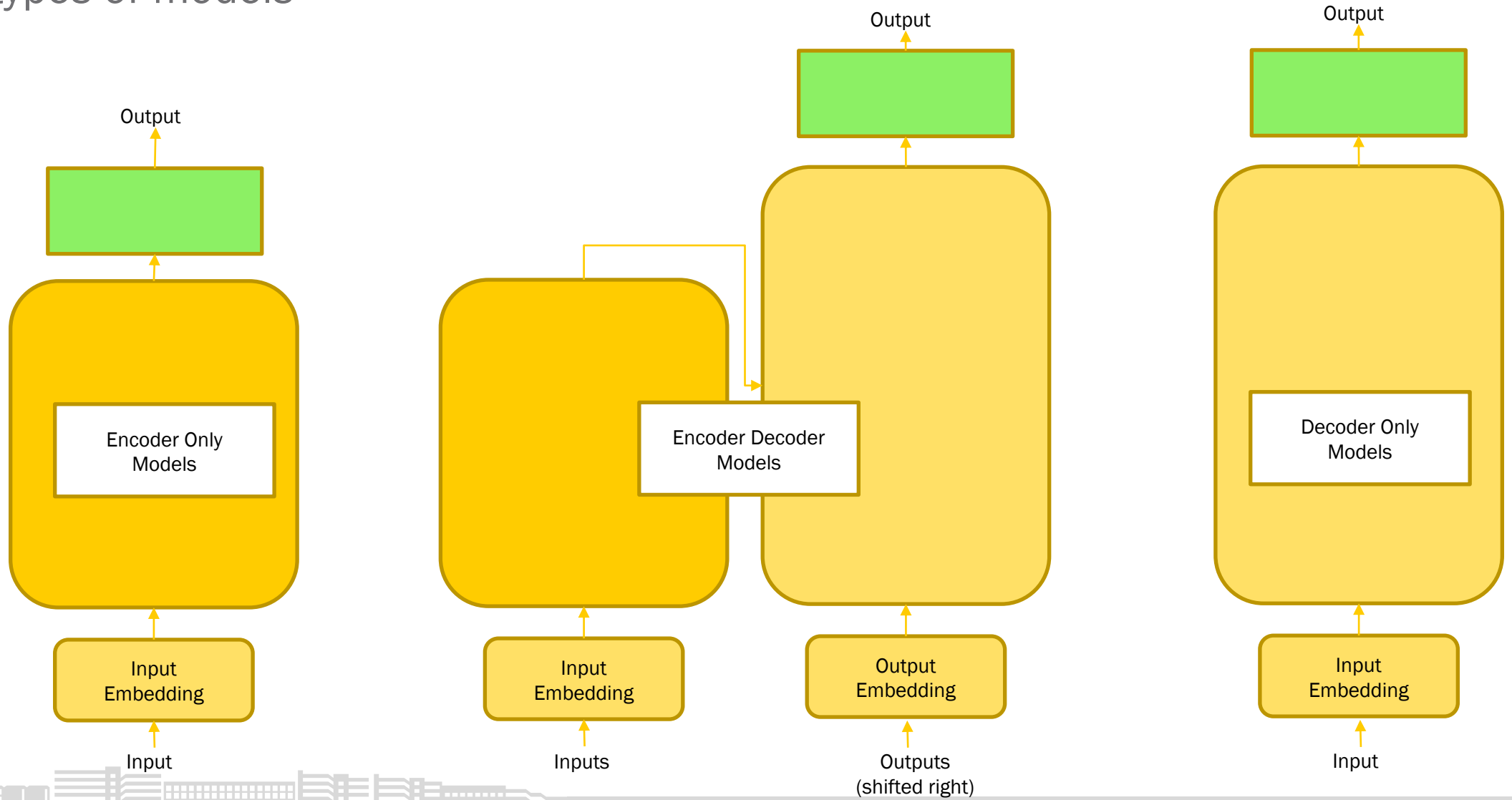

From: Attention Is All You Need - https://arxiv.org/abs/1706.03762

Transformer Architecture enables 3 types of models:

- Encoder-Only: Natural Language Understanding
- Encoder-Decoder: Seq2Seq Models for e.g. Machine Translation
- Decoder-Only: Text Generation as Next Word Prediction

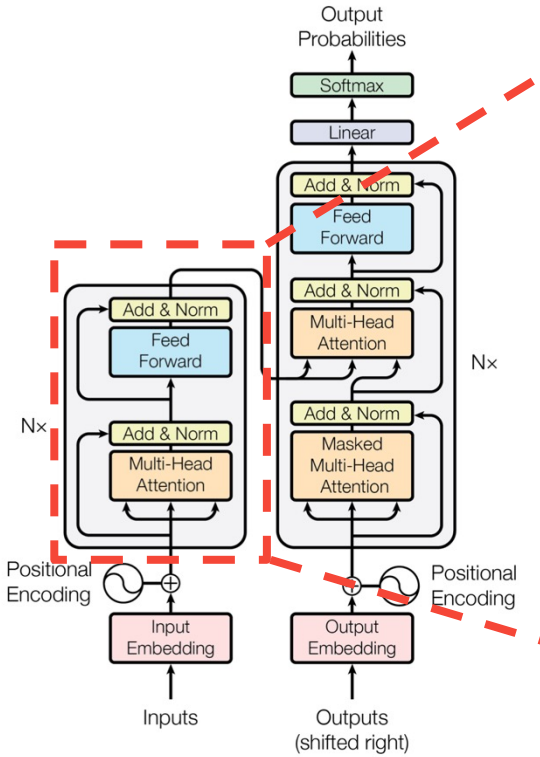
# Transformer Architecture

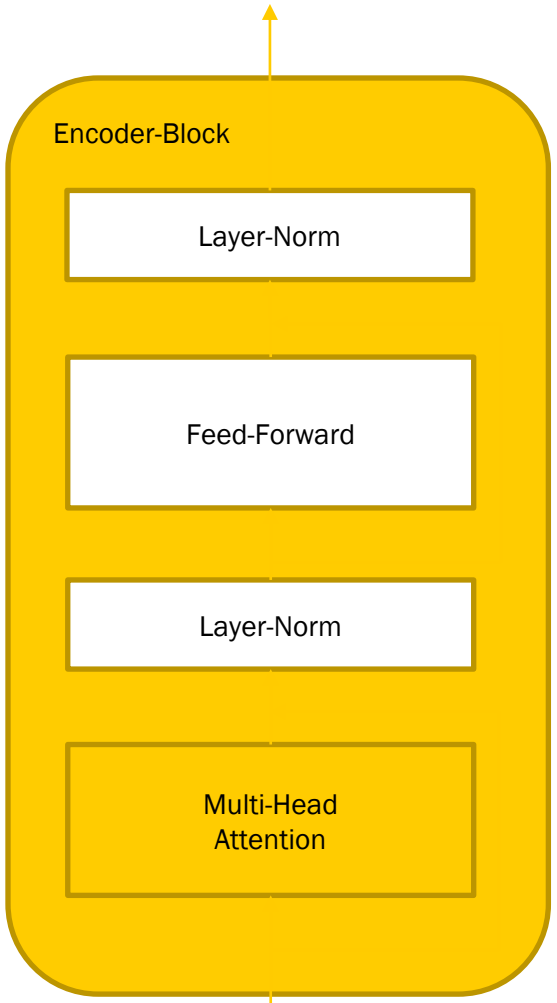
## 3 types of models



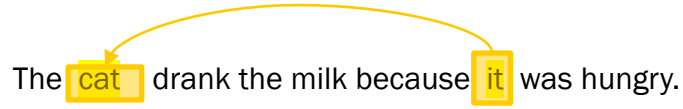
Hochschule Reutlingen

## Encoder-Block & Self Attention




While processing a word, Attention enables the model to focus on other words in the input that are closely related to that word.



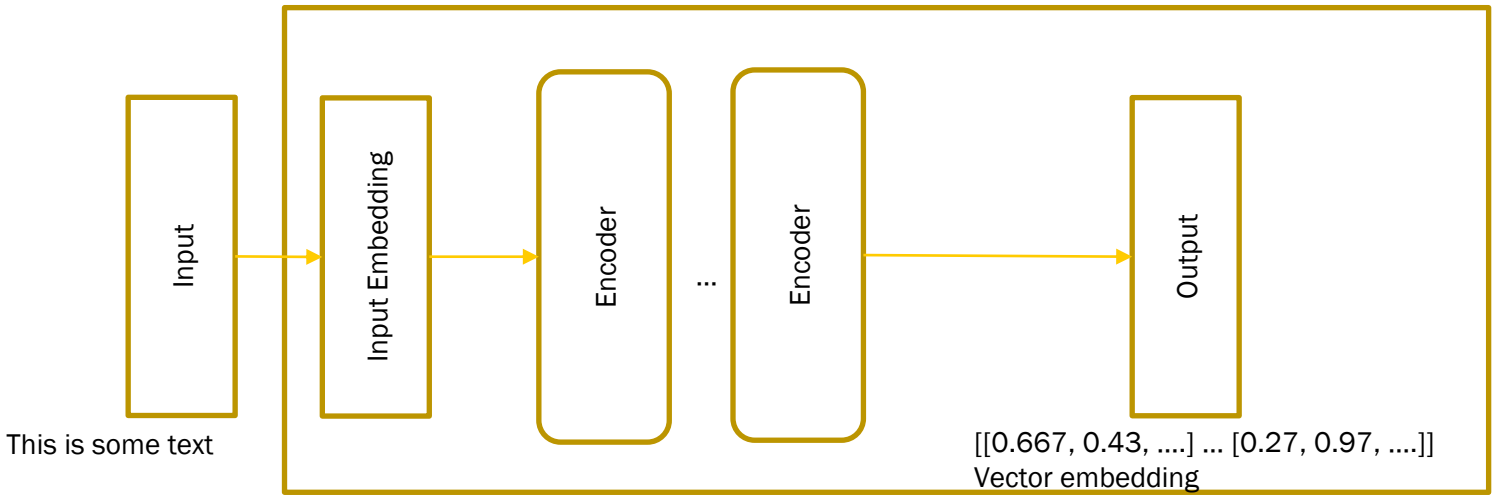


The Encoder-Block is a reusable module that is the defining component of all Transformer architectures

Hochschule Reutlingen

# Encoder-Only Models for Natural Language Understanding

- Autoencoding Models,  pre-trained using masked language modeling
- BERT (Bidrectional Encoder Representations from Transformers) is one of the most widely used encoder-only language model and produce vector embeddings for data.

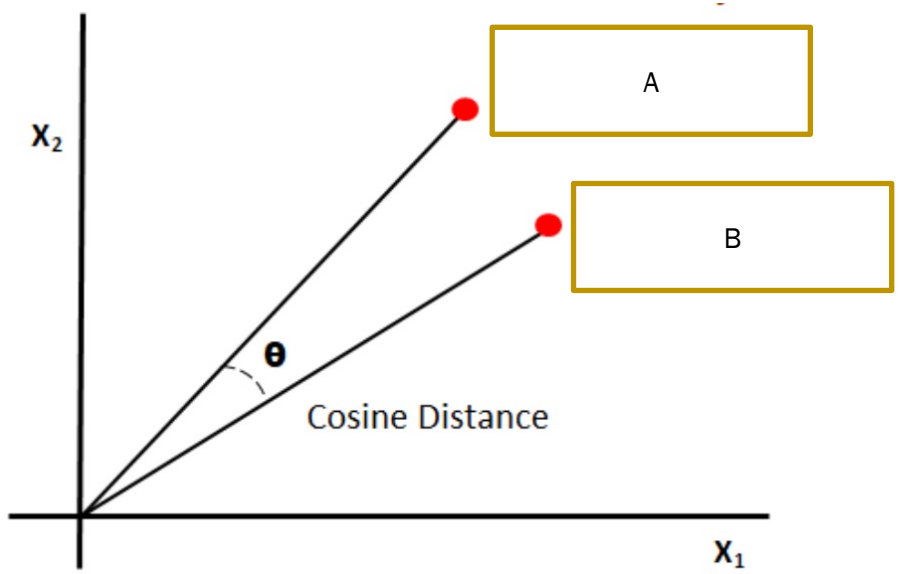


Informatik
Hochschule Reutlingen

# Encoder-Only Models for Vector Embeddings

- produced vector embeddings for data can be used for storing data in vector databases
- and computing similarity between data using cosine similarity:

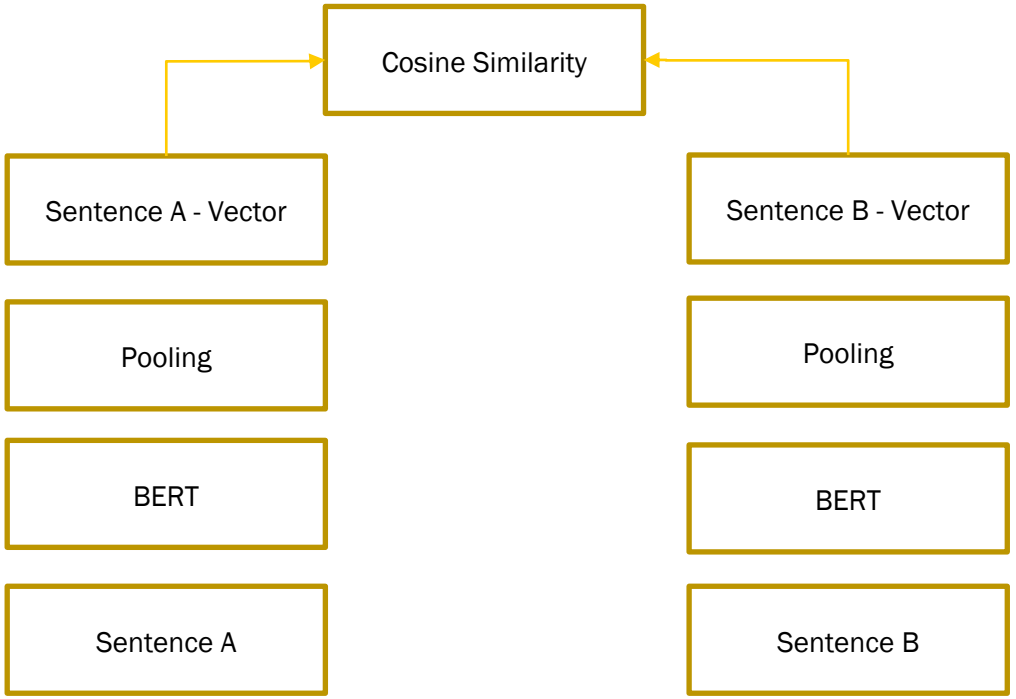
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$



Informatik
Hochschule Reutlingen

# Encoder-Only Models for Vector Embeddings

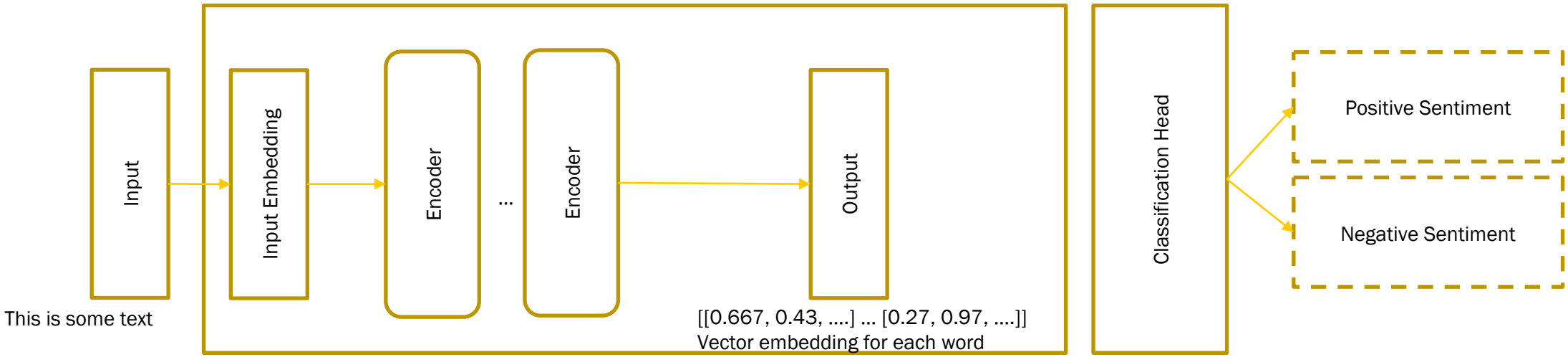
- Sentence Transformer https://sbert.net



Hochschule Reutlingen

# Encoder-Only Models for Natural Language Understanding

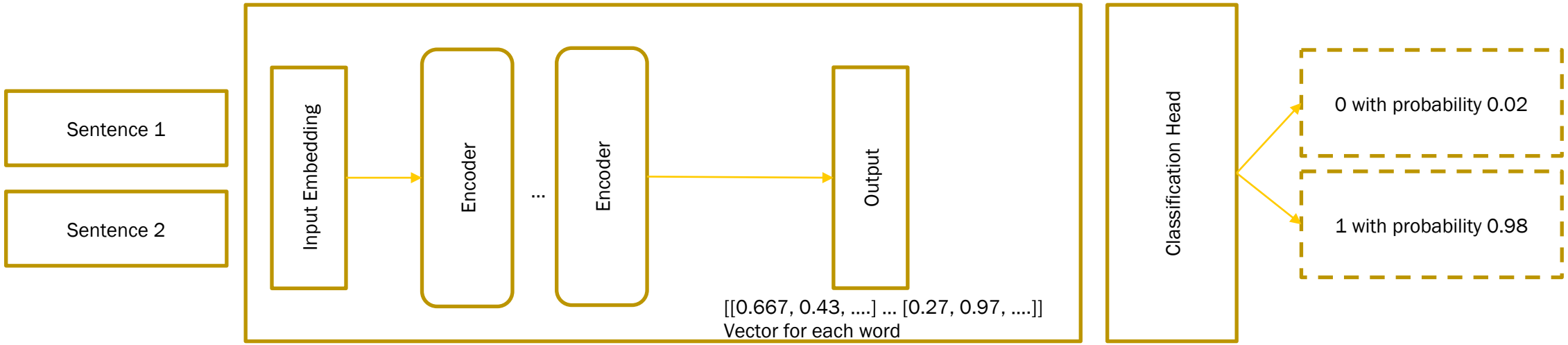
- BERT can be fine-tuned by adding a classifier layer for many language understanding tasks, such as text classification, sentiment analysis, named entity recognition



Hochschule Reutlingen

# Encoder-Only Models for Semantic Similarity Computation

- CrossEncoder is used to measure similarity between pairs of sentences
- The input of the model always consists of a data pair, for example two sentences, and outputs a value between 0 and 1 indicating the similarity between these two sentences
- can be used as Re-Raker in Retrieval-Augmented Generation (RAG)



## Decoder-Only Models as Next Word Predictors

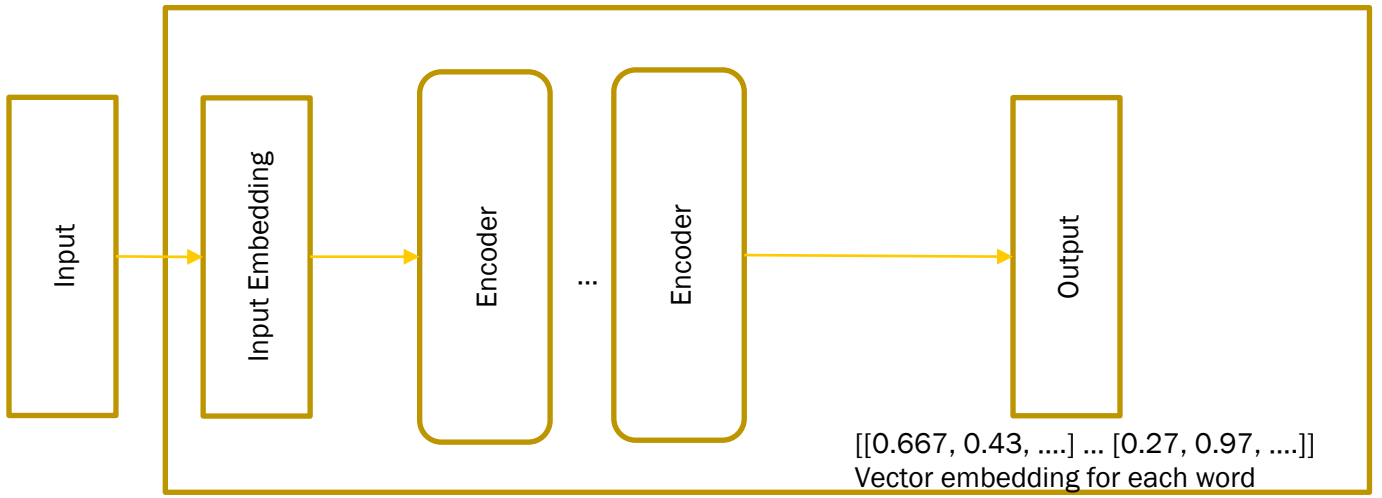
- Autoregressive Models, pre-trained using causal language modeling
- Objective is to predict the next word (token)
- They build a statistical representation of language
- Used for text generation
- Evolve emergent behavior depending on model size

Hochschule Reutlingen

# Decoder-Only Models as Next Word Predictors

- GPT - Generative Pre-trained Transformer

„The theory of relativity usually encompasses two interrelated theories by Albert“



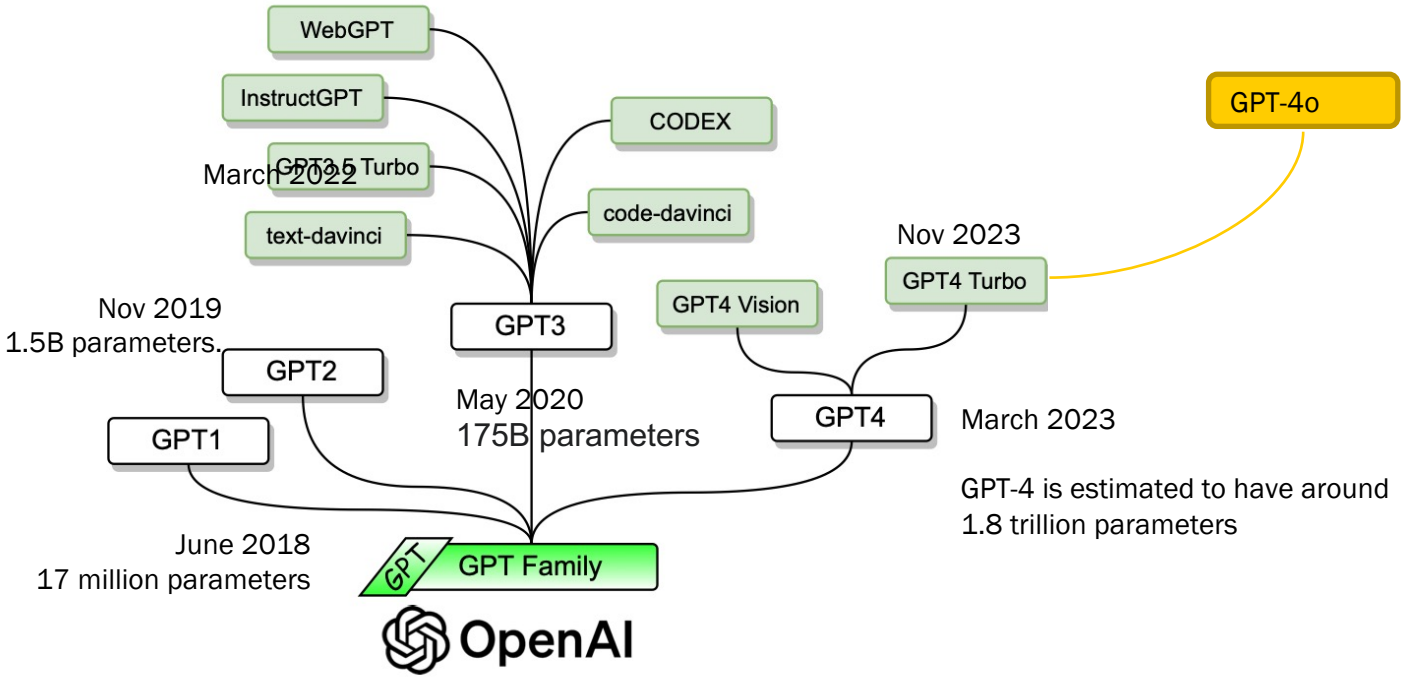

Produces vocabulary probabilities

0.957 Einstein
0.003 and
0.003 H
0.002 Spe
0.001 Hof
0.001 M
0.001 A
0.001 Jay
0.001
0.001 de

The output layer typically consists of a linear transformation followed by a softmax function, which produces a probability distribution over the possible output words.

# Large Language Models

## Example: GPT Familiy




- May 13, 2024, OpenAI announced GPT-4 Omni (GPT-4o)
- In July 2024, OpenAI launched a smaller version of GPT-4o -- GPT-4o mini.
- GPT-4o can be used for text generation use cases, such as summarization and knowledge-based question and answer. The model is also capable of reasoning, solving complex math problems and coding.
- GPT-4o combines modalities into a single model. As such, GPT-4o can understand any combination of text, image and audio input and respond with outputs in any of those forms.

From: https://arxiv.org/pdf/2402.06196 Fig 8

# 04
# Tokenization

## Tokenization

Tokenization is the process of converting text into a sequence of tokens, which can be words, subwords, or characters. These tokens are the smallest units of meaning in a text that can be processed by a language model.

Sentence → Subwords:

"Don't you love Transformers? We sure do."

→ ["▁Don", " ' ", "t", "▁you", "▁love", "▁", "Transform", "ers", "?", "▁We", "▁sure", "▁do", "."]

After the tokenization, each token is assigned a unique integer.

→ [256, 456, 5699, 7789, .....]

## Tokenization

- A vocabulary is the set of all different tokens.
- The size of vocabulary spans the dimension of vocabulary vector space (one hot vectors).
- For each integer, there is a corresponding row in a lookup table, which is the vector representation of that token. This vector is then used as input for a particular token in the language model.

## Tokenization

The embedding Layer transforms the vocabulary vector space into a lower dimensional space using word similarities.

The size of vocabulary determines the efficiency of context length:

- Vocabulary = set of all words → huge size, but leads to most efficient context length
- Vocabulary = set of characters → very small size, but leads to very inefficient context length (less information)

## Tokenization

Some kind of sub-word tokenization scheme is needed: Byte Pair Encoding Algorithm (BPE)

GPT models use tiktoken lib as tokenizer: ['gpt2', 'r50k_base', 'p50k_base', 'p50k_edit', ‘cl100k_base', 'o200k_base’]

- GPT 2 has a vocabulary of size 50257 tokens
- GPT 3.5 and 4 models recognizes about 100.000 tokens and GPT-4o recognizes about 200.000 tokens

Llama uses SentencePiece Byte-Pair Encoding-Tokenizer

Informatik
Hochschule Reutlingen

# 04_Tokenization.ipynb

## Tokenization

see: https://en.wikipedia.org/wiki/Byte_pair_encoding

### tiktoken

tiktoken is a fast BPE tokeniser for use with OpenAI's models:

- 'gpt2'
- 'r50k_base'
- 'p50k_base'
- 'p50k_edit'
- 'cl100k_base'
- 'o200k_base'

Further reading:

- see: https://github.com/openai/tiktoken
- see: https://pypi.org/project/tiktoken/
- see: https://cookbook.openai.com/examples/how_to_count_tokens_with_tiktoken
- see: https://stackoverflow.com/questions/76106366/how-to-use-tiktoken-in-offline-mode-computer

[1]:
```
!pip install tiktoken
```

```
Requirement already satisfied: tiktoken in /opt/homebrew/anaconda3/envs/block1/lib/python3.11/site-packages (0.8.0)
Requirement already satisfied: regex>=2022.1.18 in /opt/homebrew/anaconda3/envs/block1/lib/python3.11/site-packages (from ti
9.11)
Requirement already satisfied: requests>=2.26.0 in /opt/homebrew/anaconda3/envs/block1/lib/python3.11/site-packages (from ti
3)
```

Hochschule Reutlingen

## Huggingface Transformers

Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models.

AutoModel

This is a generic model class that will be instantiated as one of the base model classes of the library when created with one of the class methods:

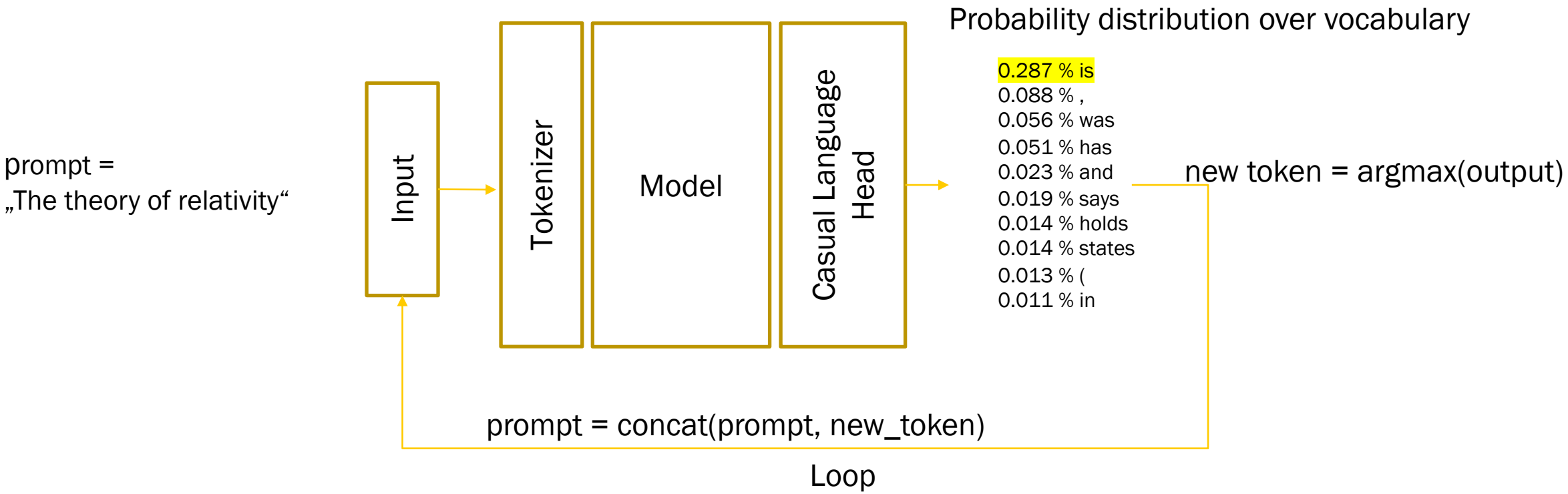
- from_pretrained()
- from_config()

| | |
|---|---|
| AutoModelForMaskedLM | ENCODER |
| AutoModelForCausalLM | DECODER |
| AutoModelForSeq2SeqLM | ENCODER - DECODER |
| ..... | |

# 05
# Generation

## Search Strategies

Start with Greedy Search:

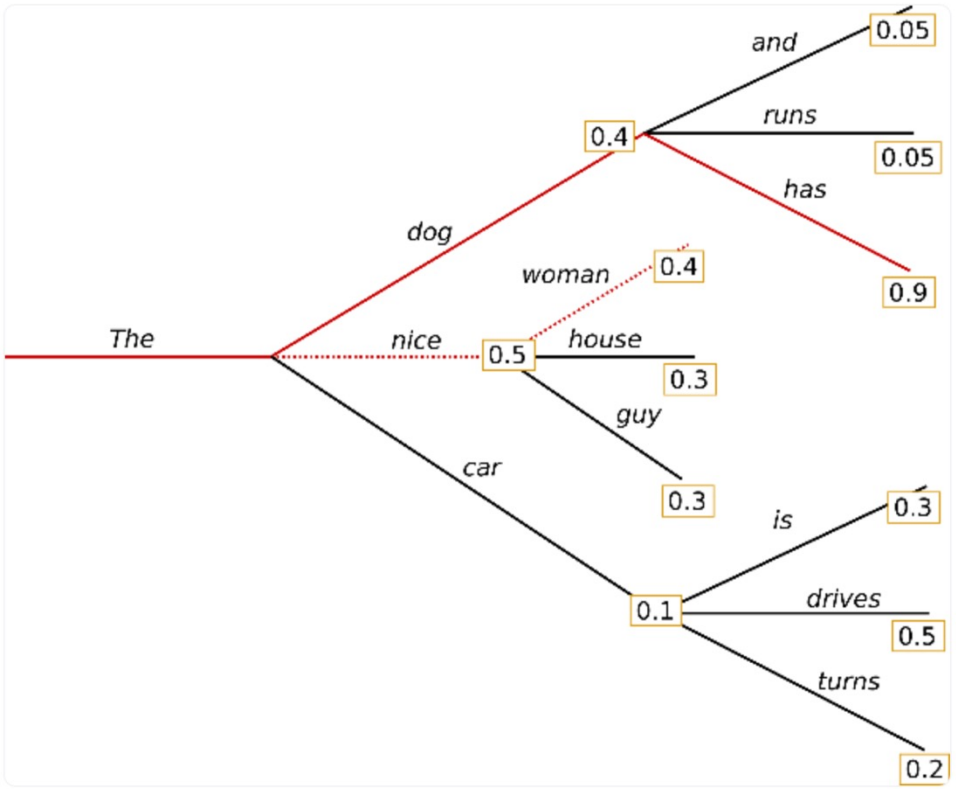


## Search Strategies

Beam Search: reduces the risk of missing hidden high probability word sequences by keeping the most likely *num_beams* of hypotheses at each time step and eventually choosing the hypothesis that has the overall highest probability.

Let num_beams = 2:

Max computation time: vocab_size * vocab_size !




See: https://huggingface.co/blog/how-to-generate

## Sampling Strategies

The drawbacks of and greedy and beam search approaches are, that we're picking the most probable choice. Instead of picking the most probable word from P(w | context), we could sample a word with P(w | context).

Now its time to add some randomness!

Even if P("eating" | "I am") > P("drinking" | "I am") we could still sample the word "drinking". Using sampling, we'll have a very high chance of getting a new sentence in each generation.

## Sampling Strategies

There are 3 ways of sampling:

- Top-k Sampling:
  Instead of sampling from full P(w | context) (there are as many choices as words in the vocabulary), we only sample from top K words according to P(w | context).

- Sampling with Temperature:
  It means we reshape the P(w | context) with a temperature factor t, where t is > 0. With temperature T we reshape the probability P(w | context) by dividing each logit value (output of the causal language head) by T before applying the softmax function. As T > 0, dividing it will amplify the logit value. Smaller values become more likely.

## Sampling Strategies

- Top-p (nucleus) Sampling:

  Instead of sampling only from the most likely K words, in Top-p sampling chooses from the smallest possible set of words whose cumulative probability exceeds the probability p. The probability mass is then redistributed among this set of words.

Informatik
Hochschule Reutlingen

# 05_GPT2.ipynb

## Understanding GPT2

see: https://huggingface.co/openai-community/gpt2

[1]:
```
import torch
```

[2]:
```
model_id = "gpt2"
```

### Tokenization

[3]:
```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id, clean_up_tokenization_spaces=True)

########## Let have a look at the vocabulary of GPT-2 ##########
vocab = tokenizer.get_vocab() #key=subword, value=token_id
print(type(vocab))
print(len(vocab))
vocab_as_list = [subword for subword, _ in vocab.items()]
tokens_as_list = [token_id for _, token_id in vocab.items()]
token_dict = {token_id:subword for subword, token_id in vocab.items()}
```

```
<class 'dict'>
50257
```

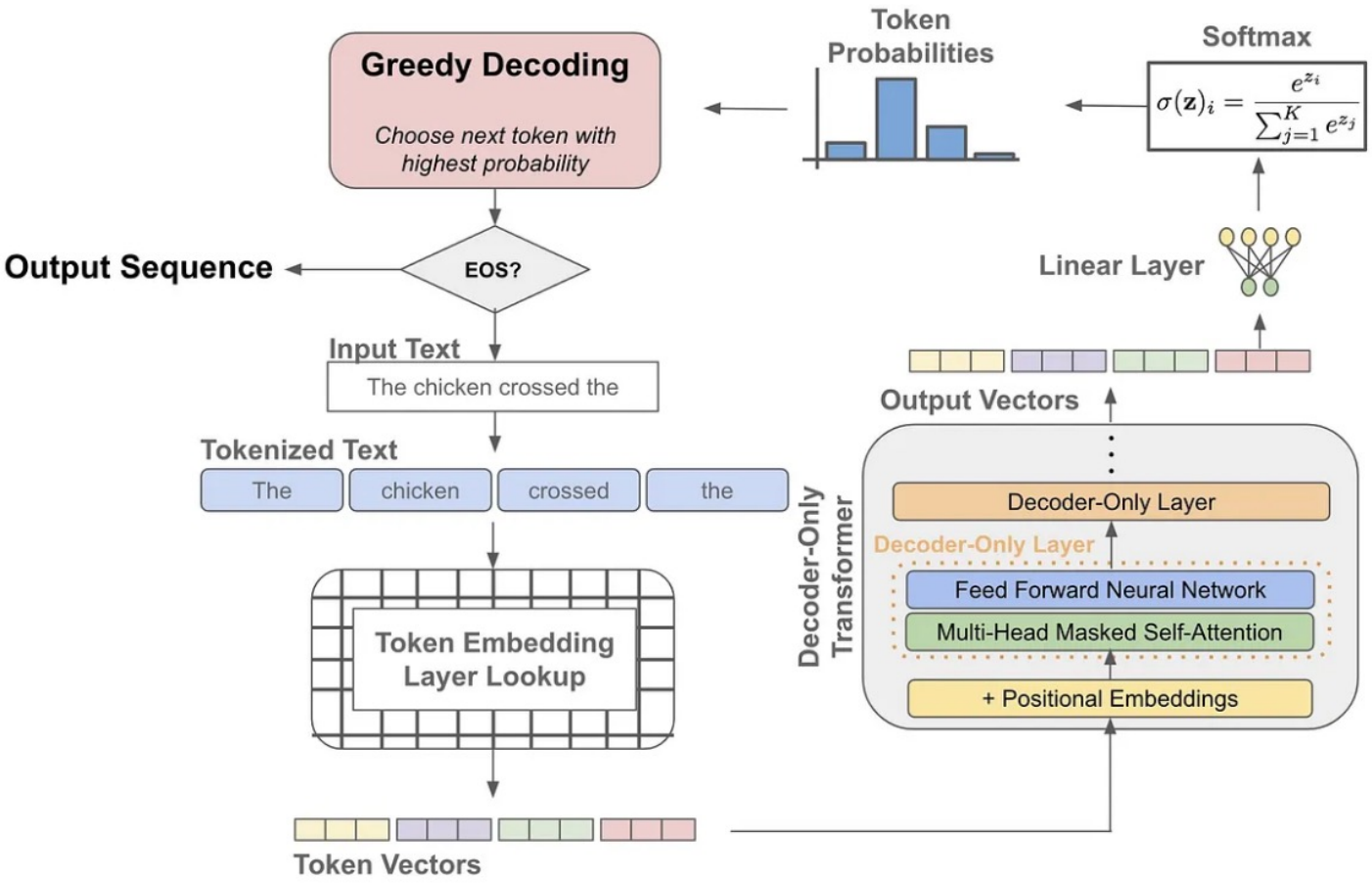
[11]:
```
# vocab
```

[5]:
```
prompt = "The theory of relativity is"
```

[6]:
```
input_ids = tokenizer.encode(prompt)
input_ids
```

[6]:
```
[464, 4583, 286, 44449, 318]
```

## Summary




Most LLMs: Decoder-only architectures

See: https://medium.com/@amansinghaiml_33304/temperature-llms-b41d75870510

# Further Reading

# Further Reading

Informatik
Hochschule Reutlingen

- [DLG] „Generative Deep Learning“, David Foster, 2023, O‘Reilly
- „Learn Generative AI with PyTorch“, Mark Liu, 2024, Manning
- [GAN] „NIPS 2016 Tutorial: Generative Adversarial Networks“, Ian Goodfellow 2027, https://arxiv.org/pdf/1701.00160
- [STA] „Stanford CS236, Deep Generative Models“, 2023, https://deepgenerativemodels.github.io
- [COR] „Cornell CS 6785: Deep Generative Models“, 2024, https://kuleshov-group.github.io/dgm-website/
- [REV] Roberto Gozalo-Brizuela, Eduardo C. Garrido-Merch´an „ChatGPT is not all you need. A State of the Art Review of large Generative AI models“, https://arxiv.org/pdf/2301.04655

- GPT-3: 2020 Language Models are Few-Shot Learners, https://arxiv.org/abs/2005.14165
- Llama: 2023 Open and Efficient Foundation Language Models https://arxiv.org/abs/2302.13971
- Llama 2: 2023 Open Foundation and Fine-Tuned Chat Models https://arxiv.org/abs/2307.09288
- Llama-Omni: 2024 (Sept) Seamless Speech Interaction with Large Language Models https://arxiv.org/abs/2409.06666

INF Informatik
Hochschule Reutlingen

Dominik Neumann
Hochschule Reutlingen, Alteburgstraße 150, 72762 Reutlingen
www.reutlingen-university.de
T. +49 172 9861157
dominik.neumann@reutlingen-university.de
dominik.neumann@exxeta.com