

# Eigene Sprachmodelle

**Nutzung und Feintuning**

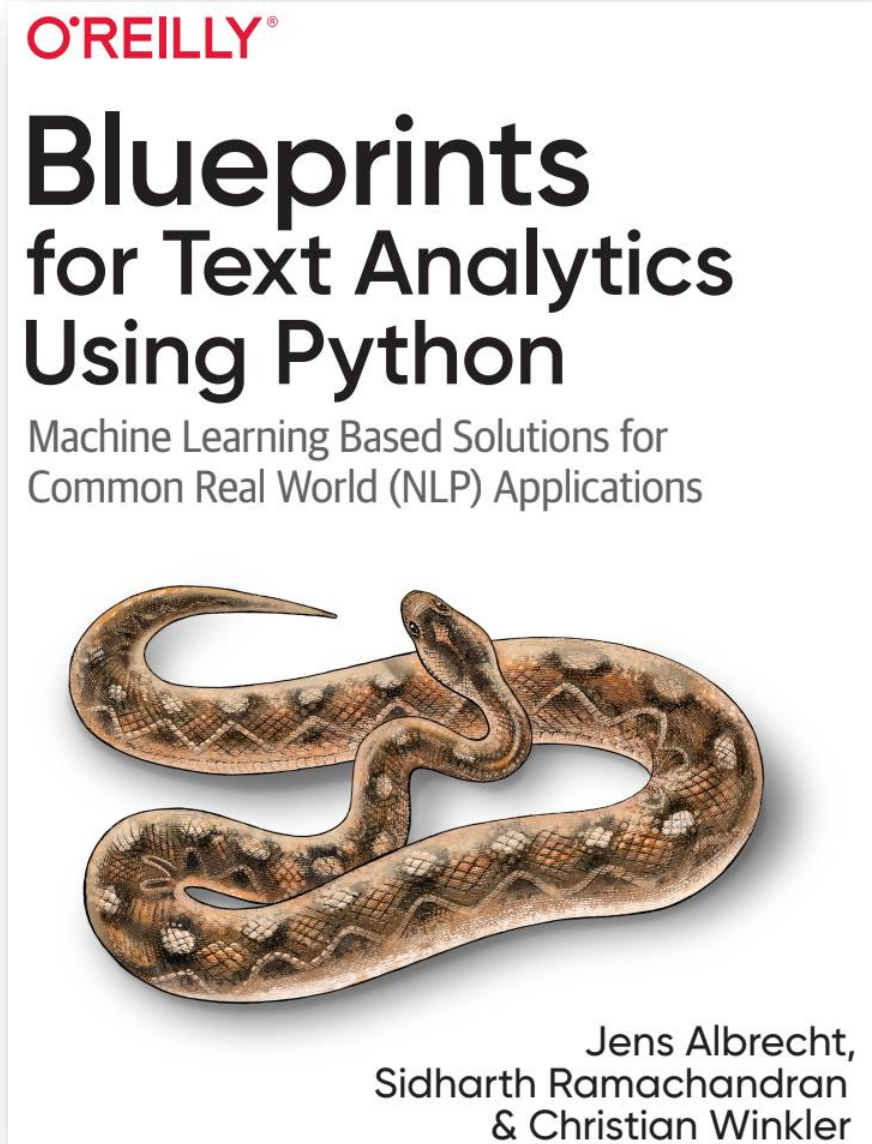
**Christian Winkler, datanizing GmbH  
KI Navigator, Workshop**



# Vorstellung und Ziele



Christian Winkler



[christian.winkler@datanizing.com](mailto:christian.winkler@datanizing.com)

[christian.winkler@th-nuernberg.de](mailto:christian.winkler@th-nuernberg.de)

Projekt in Github

<https://github.com/datanizing/ki-navigator>

# Agenda

1. Einführung und Motivation
2. Sprachmodelle: BERT
3. Feintuning von BERT
4. Semantische Ähnlichkeit
5. Generative Modelle
6. Modelle quantisieren
7. Feintuning von LLMs
8. Frontends für LLMs
9. Zusammenfassung und Ausblick
10. Feedback



# **Einführung und Motivation**

## Bisher bei der Textanalyse...

### Vektorisierung mit Wörtern als Features

- Reihenfolge nicht beachtet
- Zusammenhänge nicht betrachtet
- Semantik geht verloren

### Vektorisierung mit N-Grammen als Features

- Reihenfolge beachtet
- Zusammenhänge in Form von Tupeln
- Abstraktion in Form von Semantik fehlt

Worte jeweils  
einzelne  
Entitäten

Kontext  
entscheidet über  
Semantik!

# Wiederholung – die Document-Term-Matrix

## Textdaten müssen vektorisiert werden

- Tokenisieren und Stoppworte eliminieren
- Vokabular bestimmen
- Wörter in Dokumenten zählen

## Optimierung

- TF/IDF

## Unzulänglichkeiten

- Reihenfolge
- Semantik

	looking	cheap	flight	where	should	stay	thanks	answer	nearest	train	station	car	airport
Looking <b>for</b> cheap flight?	1	1	1	0	0	0	0	0	0	0	0	0	0
Where should <b>I</b> stay?	0	0	0	1	1	1	0	0	0	0	0	0	0
Thanks <b>for your</b> answer	0	0	0	0	0	0	1	1	0	0	0	0	0
Nearest train station	0	0	0	0	0	0	0	0	1	1	1	0	0
Looking <b>for a</b> car	1	0	0	0	0	0	0	0	0	0	0	1	0
Train <b>to</b> airport	0	0	0	0	0	0	0	0	0	1	0	0	1



# Was sind *eigene Sprachmodelle*?

## Kein Cloud-Zwang

- Ablauffähig auf lokalen Computern („on premise“)
- Parameter liegen vor
- GPU ist oft hilfreich, aber keinesfalls Zwang

## Vorteile: Freiheit und mehr Möglichkeiten

- Datenschutz und Datensicherheit
- Experimente möglich
- Anpassung an eigene Bedürfnisse
- Finetuning



# Sprachmodelle: BERT

# Grundfunktionsweise BERT: Transformer-Modell mit Attention

BERT basiert auf einem sog. Transformer-Modell

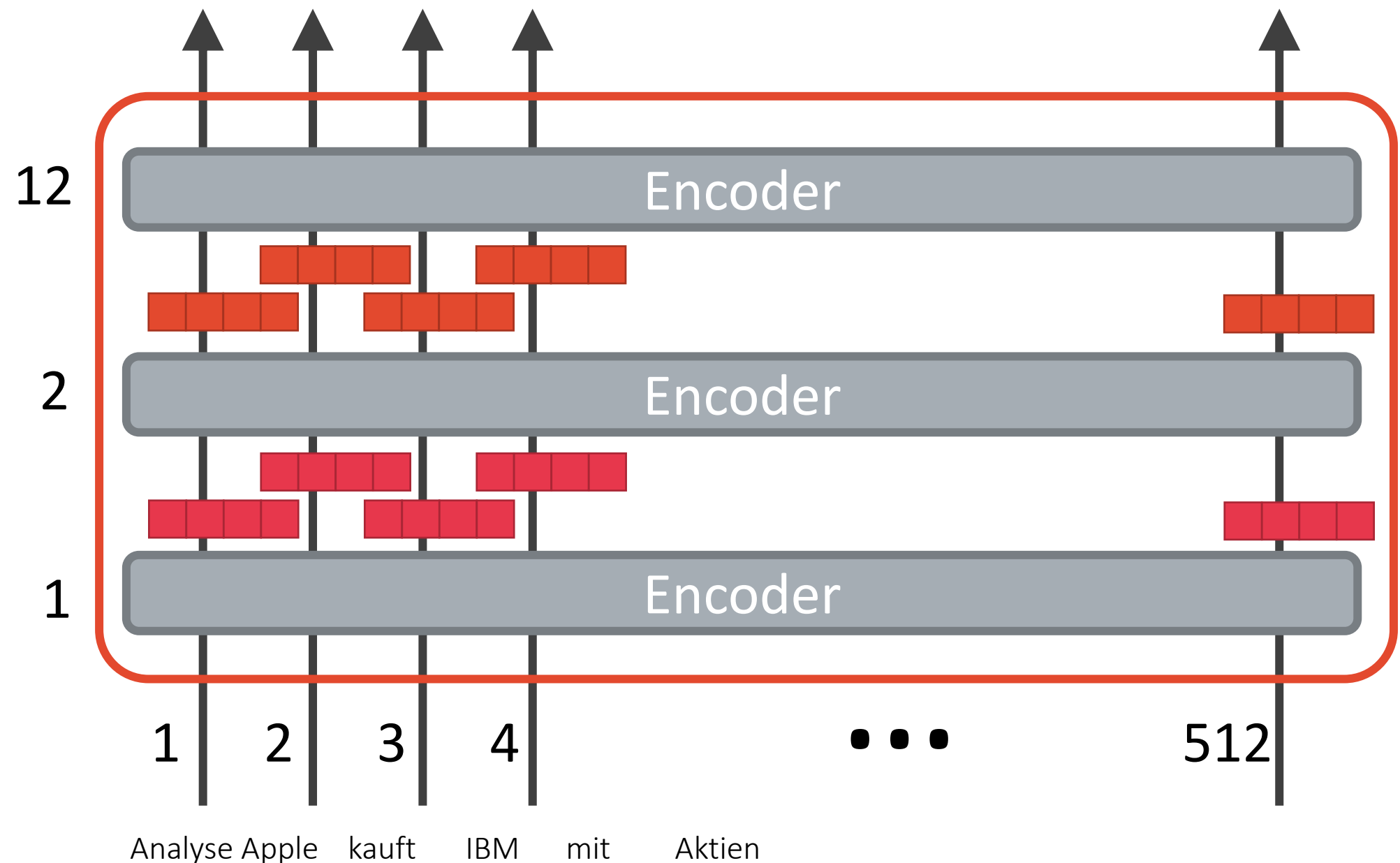
- Vorher verwendet für Übersetzungen
- Sog seq2seq-Verfahren

BERT nutzt nur den Encoder

- Encoder wird kaskadiert
- Unterschiedliche Layer modellieren die Kontextualisierung

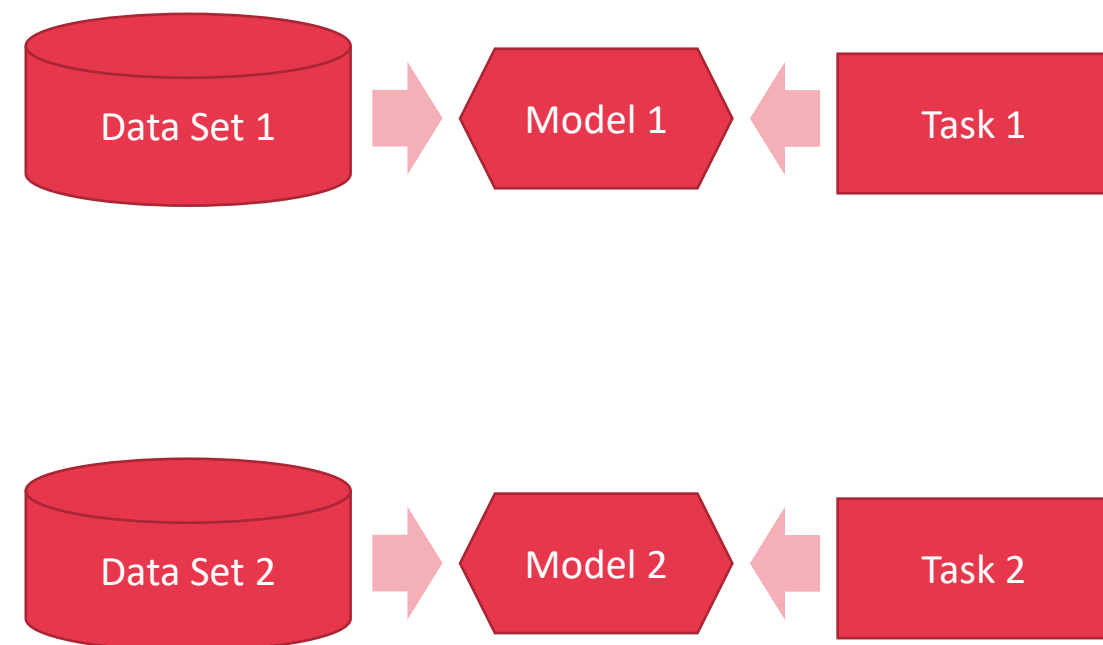
BERT ist ein sehr komplexes Modell

- Base: 12 Layer, 12 Attention Heads, 110 Millionen Parameter
- Large: 24 Layer, 16 Attention Heads, 340 Millionen Parameter
- Sehr, sehr aufwändiges Training  
➔ Transfer Learning



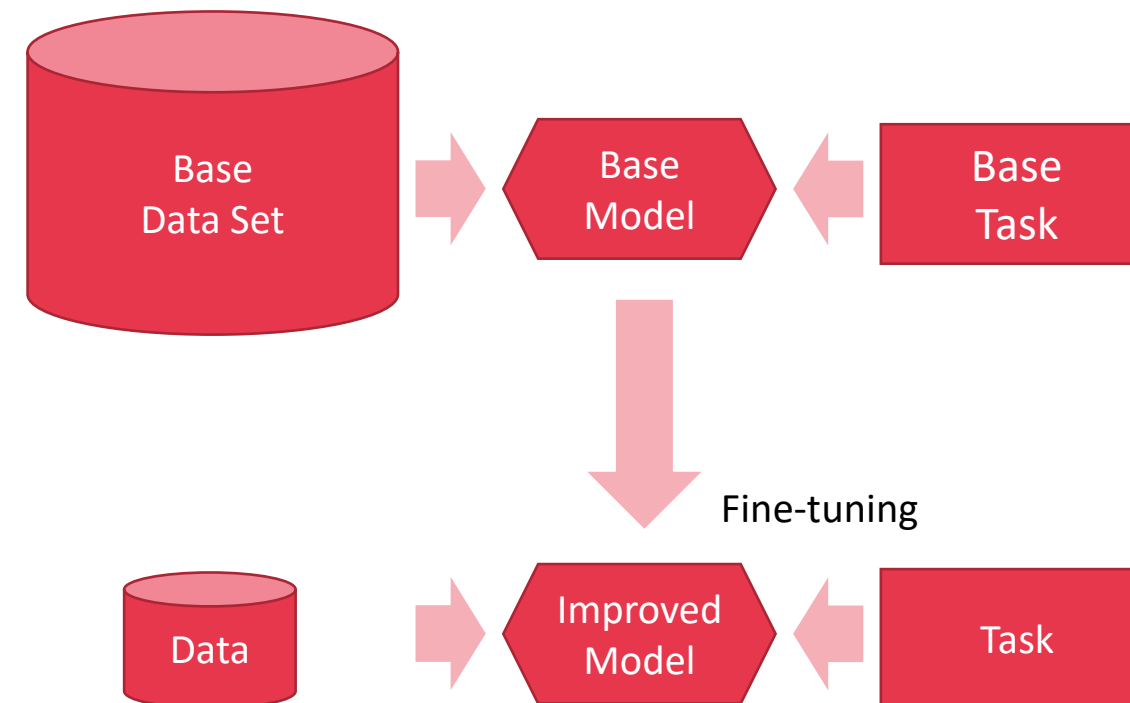
# Funktionsweise von Transfer Learning

## Klassisches ML



Ein Modell wird für genau eine Aufgabe bei Null beginnend überwacht trainiert. Es werden immer viele Trainingsdaten benötigt.

## Transfer Learning



Ein Basis-Modell, dass auf einem großen Datensatz (unüberwacht!) trainiert wurde, wird mit wenig Daten auf eine spezifische Aufgabe angepasst.

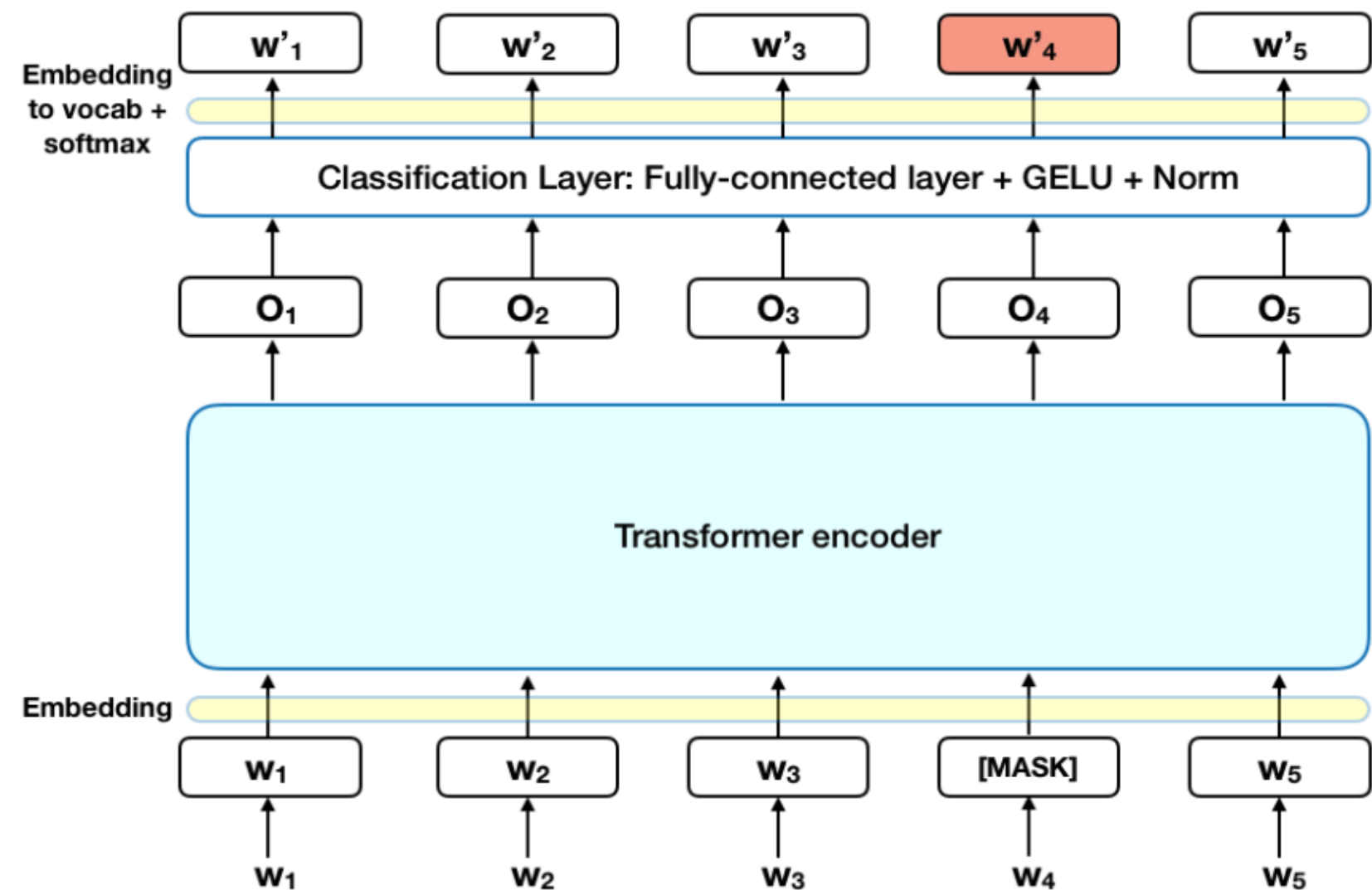
# Training auf Vorhersage “versteckter” Worte (Maskierung)

Ich **hebe Geld** bei der **Bank** ab.

Auf der **Bank** **hebe** ich **Geld** ab.

Problem: Sprache ist vorwärts und rückwärts kontextualisiert

- Mögliche Lösung durch direktionales Modell
- Bei BERT durch Transformer und Masking



Quelle: <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>





# Feintuning von BERT

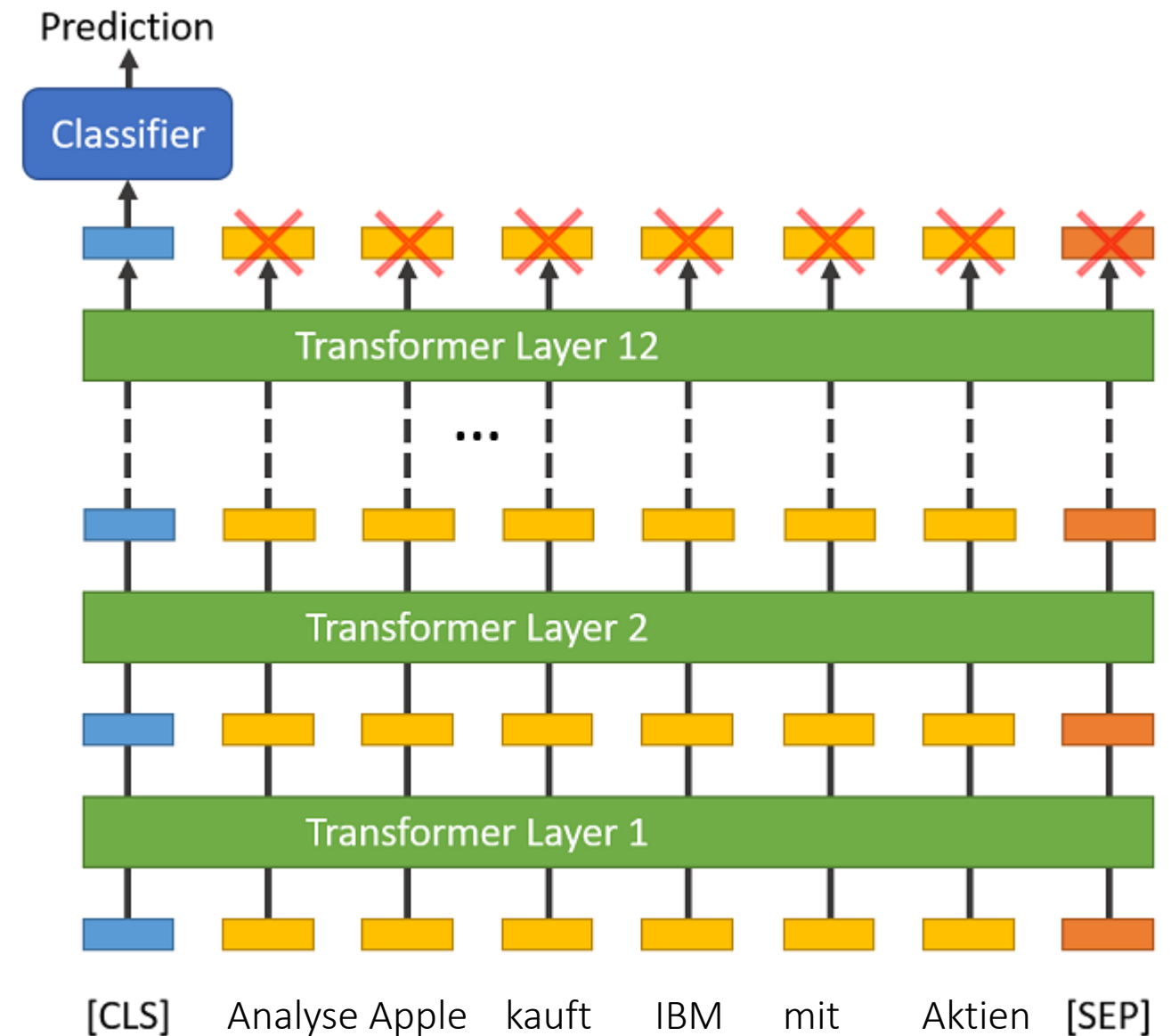
# Finetuning für Klassifikationsaufgaben

Grober Ablauf des Trainingsprozesses:

- Embedding der gesamten Sätze berechnen
- Berechnete Embeddings sind hoch kontextualisiert
- Sehr viel (alle?) Information steckt daher im kontextualisierten Embedding des ersten Tokens (der gar nicht zum Satz gehört)
- Embedding-Vektor von [CLS] wird durch Classifier verarbeitet
- Iteration mit Anpassung der Gewichte im letzten Layer

Klassifikationsprozess:

- Kontextualisierung des gesamten Satzes
- Klassifikation nur mit (kontextualisiertem) [CLS]



Quelle: <https://mccormickml.com/2019/07/22/BERT-fine-tuning/>

Mai 2020

Classic 7-Tage-News Archiv

2020 Mai

Sonntag, 31.05.2020

- 19:06 **Apple Airplay: Musik von iPhone und Co. im Netzwerk streamen**  
techstage
- 18:06 **Die Highlights bei Netflix, Disney+ und Amazon Prime Video im Juni 2020**  
100
- 17:32 **Vor 20 Jahren: Die Absage der CeBIT Home 2000 ebnet den Weg zur Games Convention**  
2
- 17:18 **"Willkommen zurück im Weltraum": Raumfahrer aus USA an ISS angekommen**  
115
- 16:40 **Youtuber zeigt Akten in altem Krankenhaus – Polizei ermittelt**  
699
- 16:00 **Apple schließt kritische Lücke in Anmeldedienst "Sign in with Apple"**  
48
- 14:24 **Slots: Rettungspaket für Lufthansa nimmt wichtige Hürde**  
115
- 11:52 **"Albtraum-Szenario": ACLU verklagt Clearview wegen illegaler Gesichtserkennung**  
92
- 11:36 **Konjunkturpaket: Scheuer meldet Investitionsbedarf für Verkehr und 5G an**  
54
- 08:05 **Missing Link: Vom Siegeszug des Webprotokolls – und resultierenden Problemen**  
146
- 04:34 **USA feiern SpaceX-Start als "Tribut an die Führerschaft Trumps"**  
350
- 00:09 **Was war. Was wird. Von Dekreten und technischen Diktaten.**  
4W 69

# Unser Use Case

## Heise Newsticker

“Instanz” für Tech- und Computer-News im deutschsprachigen Raum

"Willk

115

YouTu

699

Apple

48

## Optimierung

Welche Artikel werden besonders gern kommentiert?

Höhere Reichweite

Mehr Umsatz

# Ablauf: Finetuning von BERT

Pytorch vorbereiten (CPU/GPU)

Daten einlesen

Tokenisierung

[CLS], [SEP]

Input IDs

Attention  
Masks

Training/Validation-Split

Modell laden

Training in Epochen

Trainingsschritt

Vohersage Trainingsdaten

Berechnung Loss

Rückwärts-Auswertung

Gewichte anpassen (AdamW)

Validierungsschritt

Vohersage Validierungsdaten

Berechnung Accuracy und Loss

# Beispiel: Erfolgsvorhersage von Newsticker-Meldungen

## Schritt 1: Tokenisierung

```
from transformers import BertTokenizer

# Wir nutzen den DBMDZ-Tokenizer der Bayerischen Staatsbibliothek
tokenizer = BertTokenizer.from_pretrained('dbmdz/bert-base-german-uncased', do_lower_case=True)
```

```
# Jetzt alle Sätze tokenisieren und IDs merken
input_ids = []
attention_masks = []

for t in text:
    encoded_dict = tokenizer.encode_plus(
        t,
        add_special_tokens = True,      # '[CLS]' und '[SEP]'
        max_length = 64,
        pad_to_max_length = True,
        return_attention_mask = True,   # Attention-Masks erzeugen
        return_tensors = 'pt',         # pytorch-Tensoren als Ergebnis
    )
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
```

```
# Headline, Tokenisierung und IDs anzeigen
print(text[0])
print(tokenizer.tokenize(text[0]))
print(input_ids[0])
```

```
Apple TV+: Start in der Türkei unklar
['apple', 'tv', '+', ':', 'start', 'in', 'der', 'turk', '##ei', 'unklar']
tensor([ 102, 12045, 3845, 1376, 847, 2629, 142, 127, 29538, 109,
         9718, 103, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0])
```



# Beispiel: Erfolgsvorhersage von Newsticker-Meldungen

## Schritt 2: Training

```
# Modell in Trainingsmodus stellen
model.train()

# For each batch of training data...
for step, batch in enumerate(tqdm(train_dataloader, desc="Training")):
    # Daten entpacken und in device-Format wandeln
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels = batch[2].to(device)

    # Gradienten löschen
    model.zero_grad()

    # Vorwärts-Auswertung (Trainingsdaten vorhersagen)
    loss, logits = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask,
                        labels=b_labels)

    # Loss berechnen und akkumulieren
    total_train_loss += loss.item()

    # Rückwärts-Auswertung, um Gradienten zu bestimmen
    loss.backward()

    # Gradient beschränken wegen Exploding Gradient
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    # Parameter und Lernrate aktualisieren
    optimizer.step()
    scheduler.step()

# Calculate the average loss over all of the batches.
avg_train_loss = total_train_loss / len(train_dataloader)
```

```
# Evaluate data for one epoch
for batch in tqdm(validation_dataloader, desc="Validierung"):
    # jetzt die Validierungs-Daten entpacken
    b_input_ids = batch[0].to(device)
    b_input_mask = batch[1].to(device)
    b_labels = batch[2].to(device)

    # Rückwärts-Auswertung wird nicht benötigt, daher auch kein Gradient
    with torch.no_grad():
        # Vorhersage durchführen
        (loss, logits) = model(b_input_ids,
                            token_type_ids=None,
                            attention_mask=b_input_mask,
                            labels=b_labels)

    # Loss akkumulieren
    total_eval_loss += loss.item()

    # Vorhersagedaten in CPU-Format wandeln, um Accuracy berechnen zu können
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()
    total_eval_accuracy += flat_accuracy(logits, label_ids)

# Report the final accuracy for this validation run.
avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
tqdm.write("Accuracy: %f" % avg_val_accuracy)
```





# Semantische Ähnlichkeit

# Ist BERT dafür ausreichend?

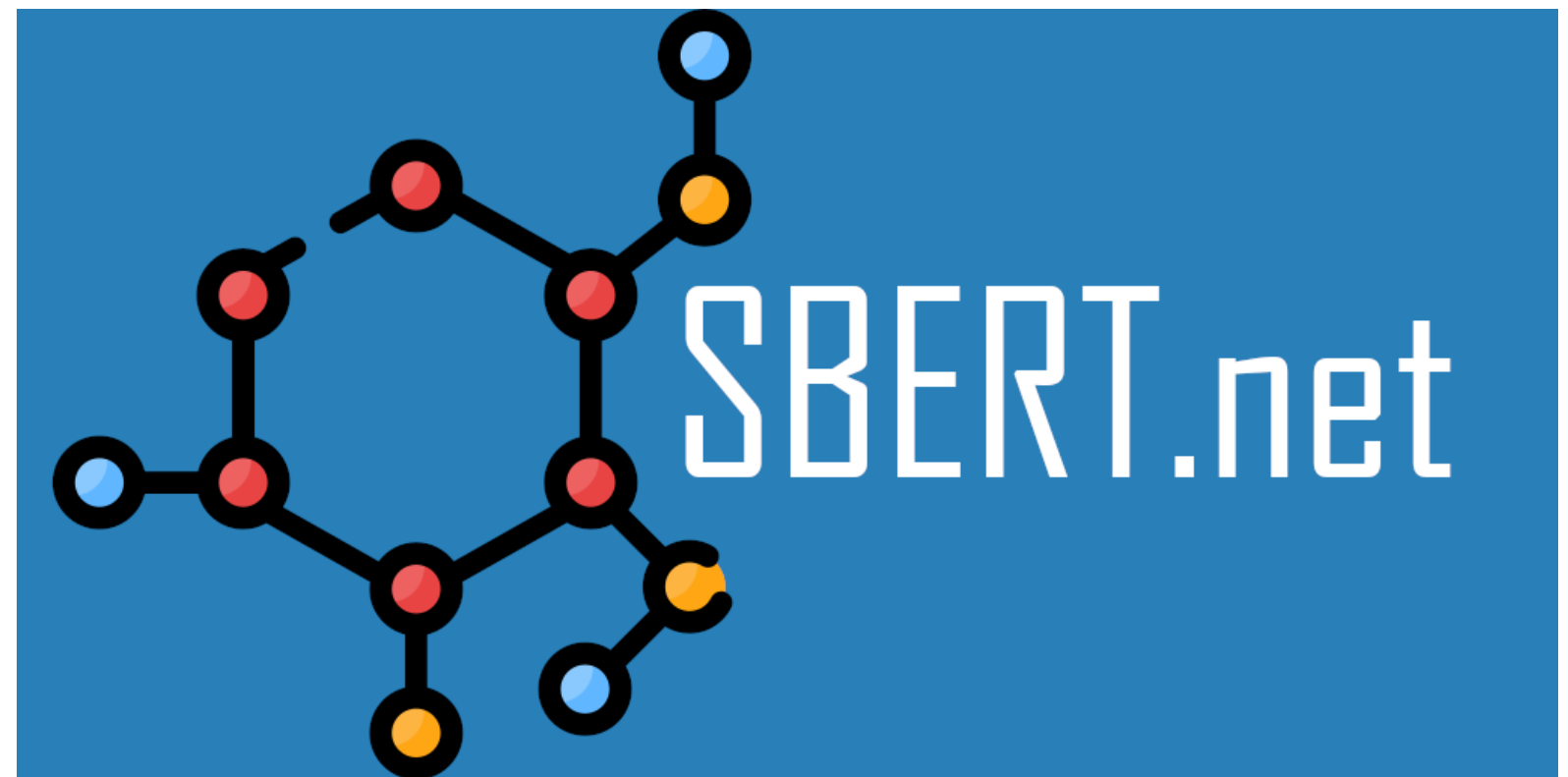
Fast, aber noch nicht ganz

Modell ist darauf angepasst, fehlende Wörter zu erraten

Für Ähnlichkeit wird ein etwas modifiziertes Modell benötigt

Zum Glück ist das bereits verfügbar

Modell heißt SBERT (<https://sbert.net>) und kommt sogar aus Deutschland



# Ablauf

Daten laden

Satz-Fragmente berechnen

Embedding für jeden Satz berechnen

Embeddings speichern

Sätze speichern

Search-Embedding berechnen

Ähnlichste Sätze finden

Nach Ähnlichkeit sortieren



# Finetuning eigener Embedding-Modelle

## Warum sollte man das machen?

- (Bessere) Abbildung domänenspezifischen Vokabulars
- Bessere Abbildung von Ähnlichkeiten

## Wie geht das?

- Erzeugung von ähnlichen (oder weniger ähnlichen) Textfragmenten
- Annotation mit Score
- Finetuning mit SBERT

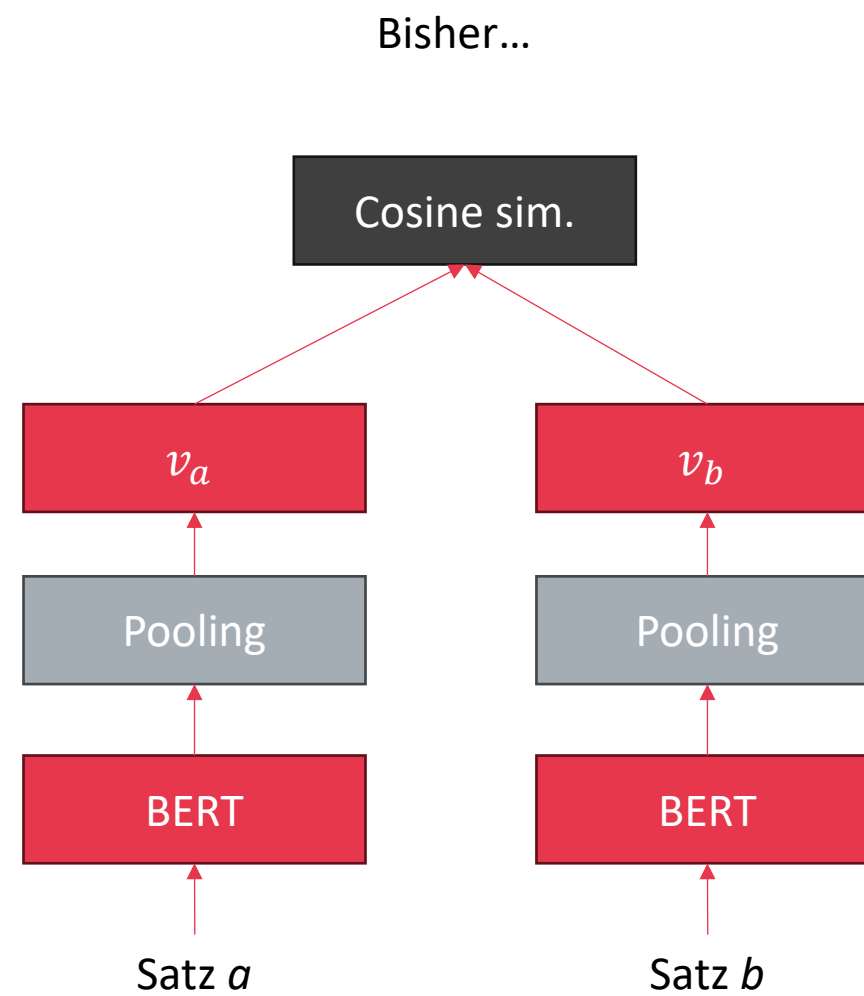
## Ist das nicht extrem aufwändig?

- Text lässt sich generativ erzeugen, Verbesserungen relativ schnell erkennbar
- Mit GPU keine langen Wartezeiten

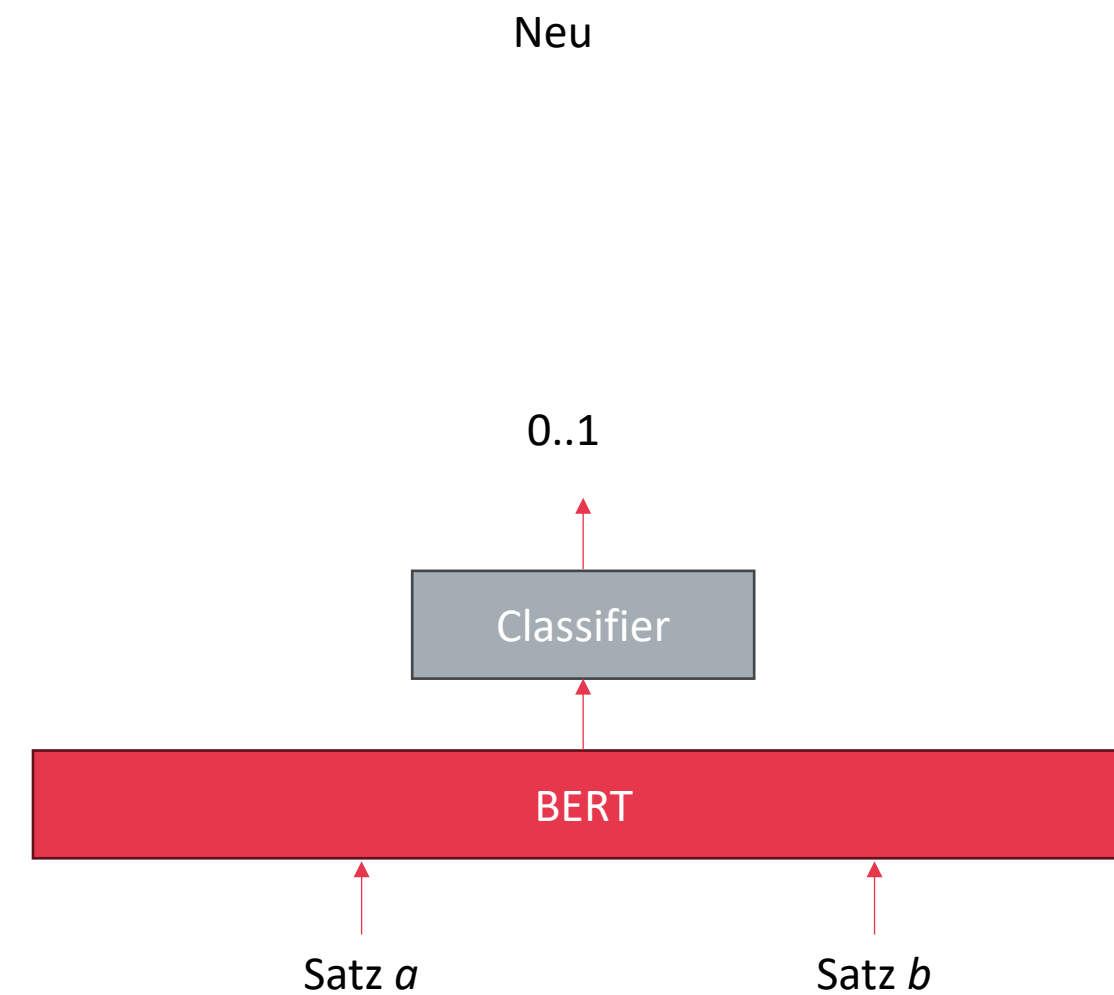


# Optimierung durch Cross-Encoder

# Funktionsweise Cross-Encoder

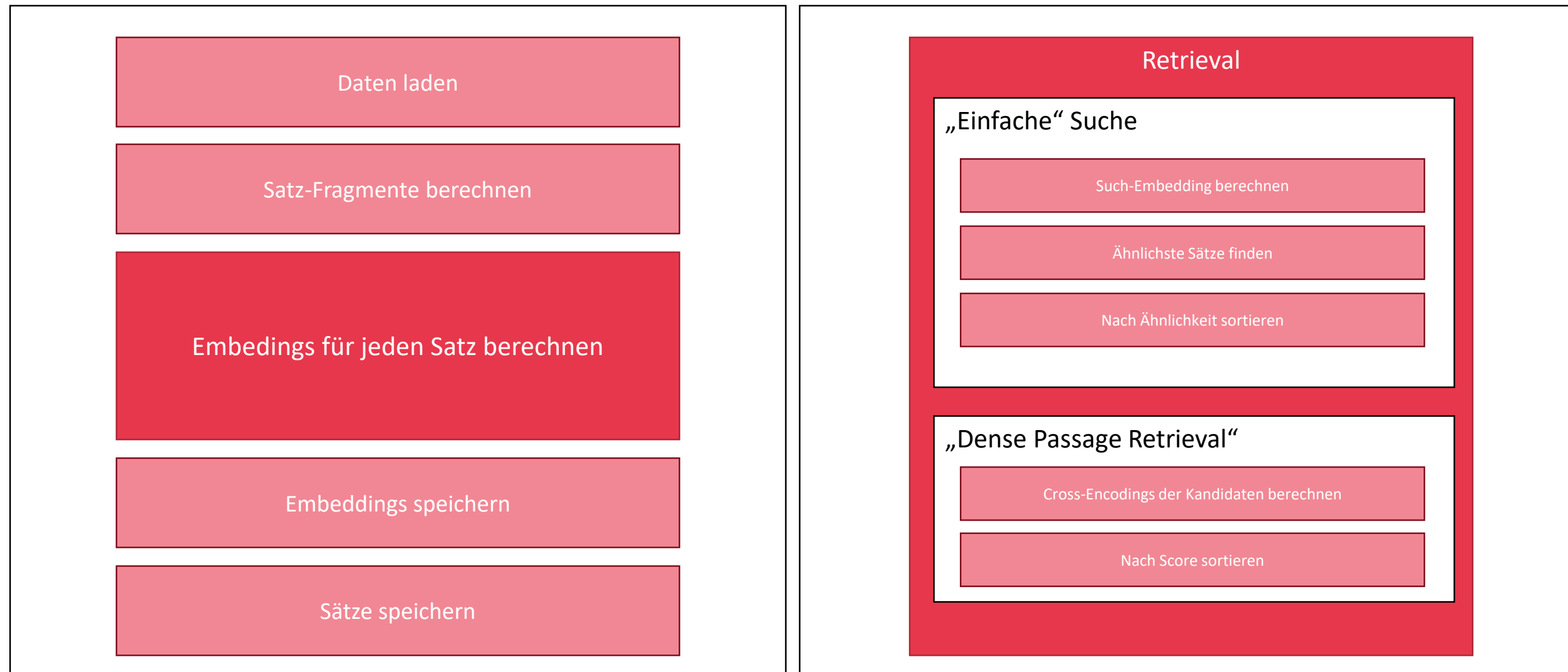


**Bi-encoder**



**Cross-encoder**

# Verbesserung der bisherigen Lösung



# Nutzung zum Information Retrieval

## Beste Ergebnisse für Ähnlichkeiten von Sätzen

- Codierung des Satzes (z.B. der Frage) notwendig **inkl. gleichzeitiger Codierung** der Referenz (des Ergebnisses)
- Sehr viele Codierungen
- Sehr gut, aber sehr langsam

## Grundidee der Sentence Transformers

- Unabhängig Codierung
- Nutzung des Ähnlichkeitsmaßes

## Weitere Optimierung durch Cross-Encoder

- Vorher Verkleinerung der Ergebnismenge
- Wartezeit verkürzen

17.250 Aussagen

→ 17.250 Codierungen für jede Suche notwendig (!)

17.250 Codierungen einmal notwendig

→ Für jede Suche eine weitere Codierung

17.250 Codierungen einmal notwendig

→ Für jede Suche eine weitere Codierung

→ Für den Filter weitere 200 Codierungen

→ Deutlich verbesserte Ergebnisse



# Finetuning eigener Cross-Encoder

## Warum sollte man das machen?

- (Noch) bessere Abbildung von Ähnlichkeiten
- Besseres Ranking

## Wie geht das?

- Nutzung der gleichen Daten wie beim Finetuning der Embeddings
- Finetuning erfolgt ebenso mit SBERT

## Ist das nicht extrem aufwändig?

- Vorteil: Ähnlichkeitsdaten schon vorhanden
- Mit GPU leicht zu bewältigen



# Generative Modelle

## Grundfunktionsweise: Transformer-Modell mit Attention

GPT basiert auf einem sog. Transformer-Modell

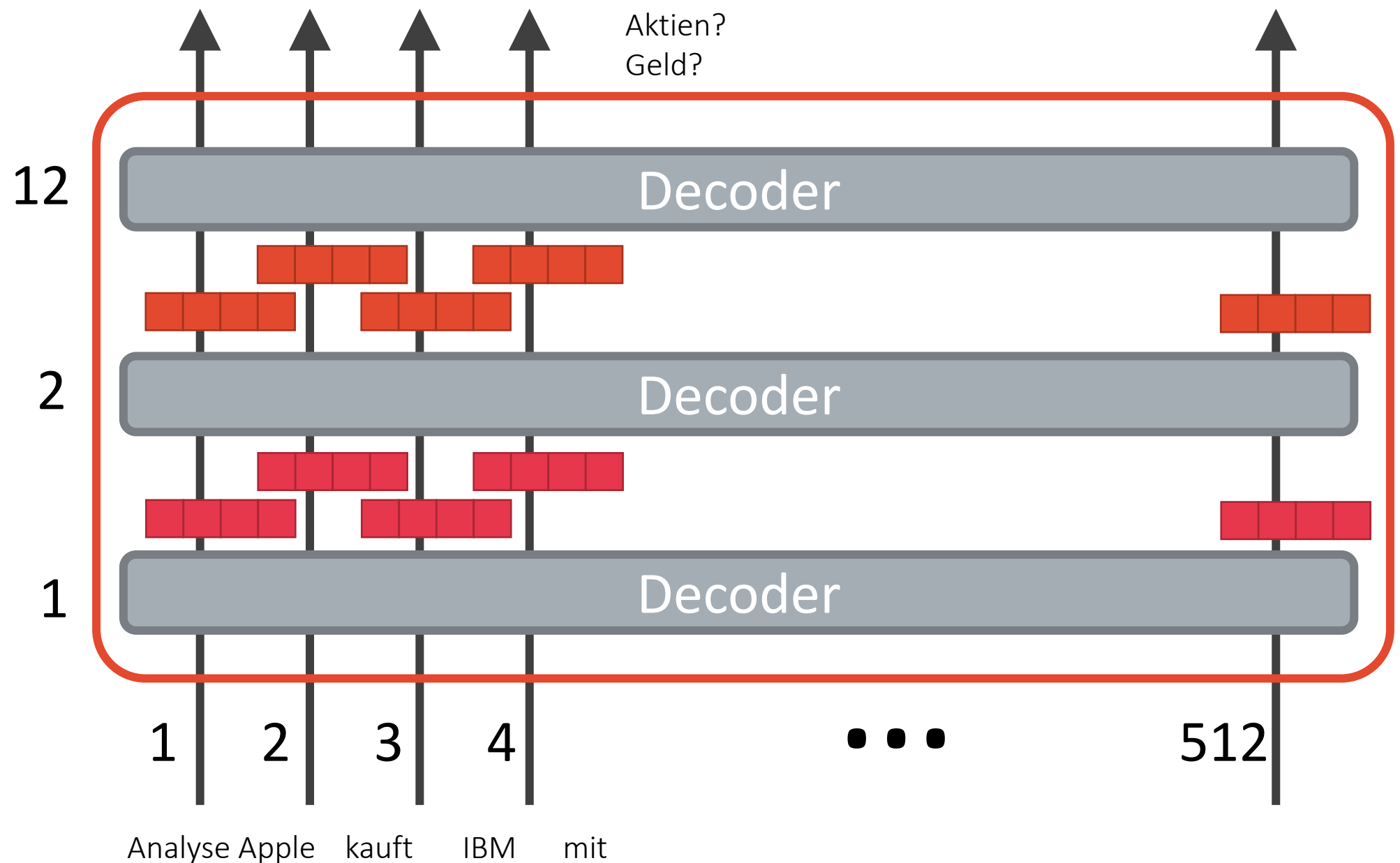
- Vorher verwendet für Übersetzungen
- Sog seq2seq-Verfahren (Encoder/Decoder)

GPT nutzt nur den Decoder

- Decoder wird kaskadiert
- Unterschiedliche Layer modellieren die Kontextualisierung (nur nach vorne)
- Training auf Vorhersage des nächsten Wortes (autoregressiv)

GPT hat sehr viele Parameter

- Ständiges Wachstum
- Training noch viel aufwändiger als bei BERT
- Kann nur mit extremem Hardware-Aufwand bewältigt werden



Wachst

# GPT-4

# Wachstum der Sprachmodelle

## Unendliches Wachstum?

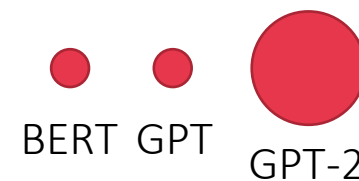
- Rechenkapazität wächst immer weiter und schneller
- Trainingsmenge (Text) in nahezu beliebiger Menge verfügbar (aber vielleicht irgendwann auch ausgeschöpft)
- Geld offenbar gar kein Problem

## Aufgaben werden immer schwieriger

- Von Sentiment-Analyse zu wissenschaftlichen Arbeiten

## Aber: wenig strukturell neue Ideen

- Dünn besetzte Modelle (sparse)
- Distillation (besonders bei BERT, jetzt auch bei GPT)



GPT-3



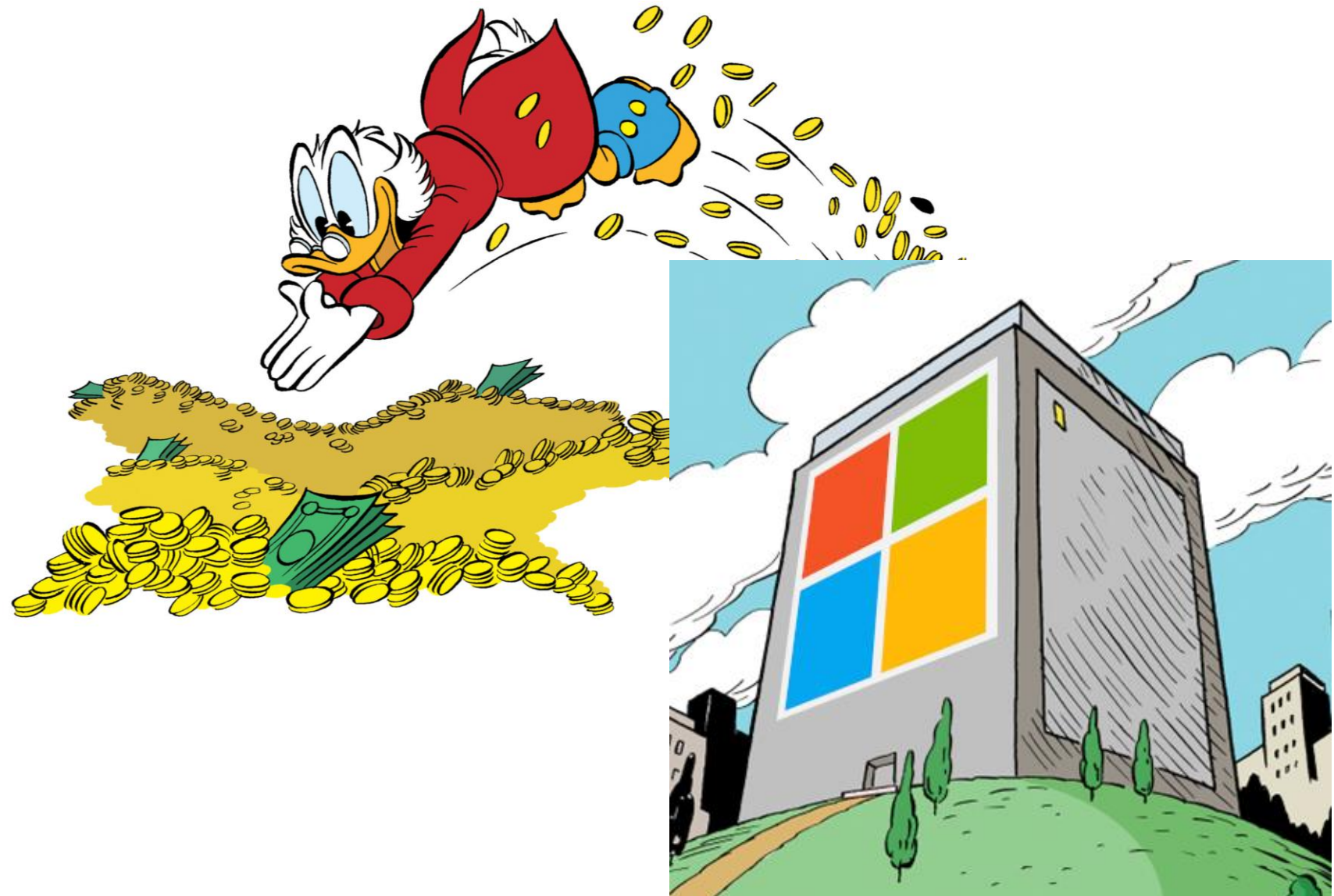
# Welche Rechenleistung wird für ChatGPT benötigt und was kostet das?

## Training:

- Sehr, sehr aufwändig
- Billionen von Token
- Kosten für GPT-4 angeblich über 100 Millionen Dollar<sup>1</sup>

## Betrieb:

- Auch dafür werden Grafikkarten benötigt
- Bei ChatGPT wohl mehr als 700.000 Dollar pro Tag<sup>2</sup>
- Überwachung durch Menschen (mit minimalem Lohn!) notwendig<sup>3</sup>



<sup>1</sup> <https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/>  
<sup>2</sup> <https://www.businessinsider.com/how-much-chatgpt-costs-openai-to-run-estimate-report-2023-4>

<sup>3</sup> <https://www.nbcnews.com/tech/innovation/openai-chatgpt-ai-jobs-contractors-talk-shadow-workforce-powers-rcna81892>

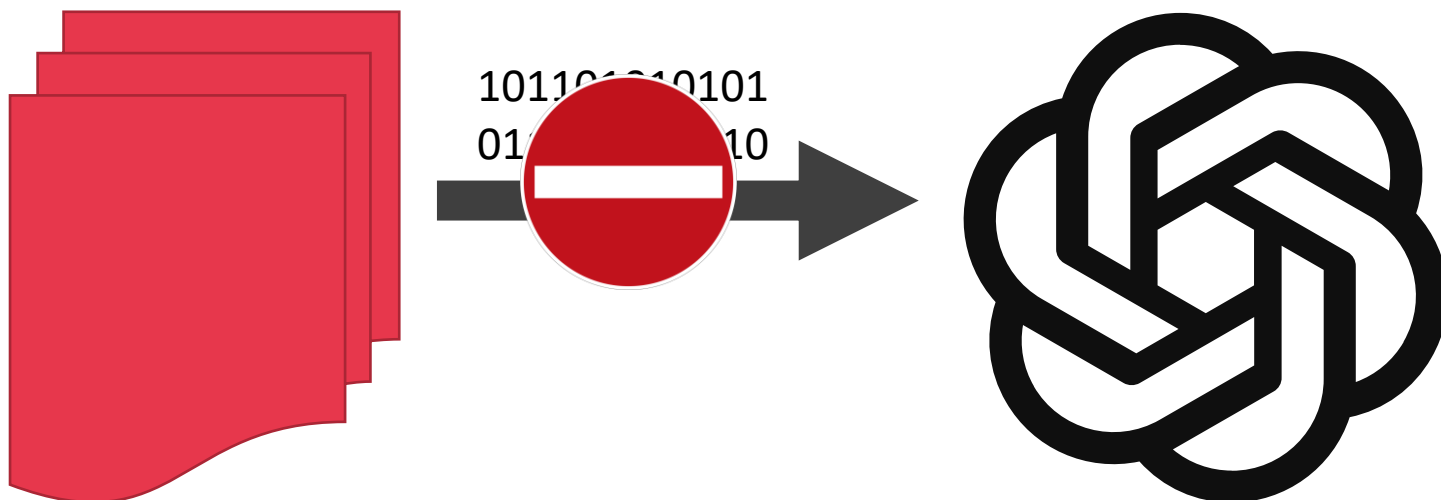
# Läuft das auch auf *meiner* Hardware?

Kommt auf deine Hardware an...

- Evtl. sehr langsam auf einer CPU mit genügend RAM
- Akzeptable Performance erfordert mehrere A100 mit 80 GB

Warum es nicht geht:

- Modell ist nicht veröffentlicht
- Alle Daten werden nur bei OpenAI verarbeitet (Datenhoheit!)



Erkläre Heise Zeitschriftenverlag!



Der Heise Zeitschriftenverlag ist ein deutsches Verlagsunternehmen mit Sitz in Hannover, das sich auf die Publikation von IT- und Telekommunikationszeitschriften sowie Online-Portalen spezialisiert hat.

Gegründet wurde der Verlag 1949 von August Heise als "Heise RegioConcept". Seit den 1980er Jahren liegt der Schwerpunkt des Verlags auf IT- und Telekommunikationsthemen. Bekannte Zeitschriften aus dem Verlag sind beispielsweise die c't (Computerzeitschrift), iX (Magazin für professionelle Informationstechnik), Technology Review und Telepolis.

Neben den Print-Publikationen betreibt der Heise Verlag auch zahlreiche Online-Portale, darunter heise online, das zu den meistbesuchten deutschsprachigen IT-News-Portalen zählt.

Der Heise Zeitschriftenverlag ist heute Teil der Heise Gruppe, die auch IT-Veranstaltungen wie die CeBIT und die SecIT sowie den Verlag dpunkt.verlag und die Jobbörse heise jobs umfasst.







# Modelle quantisieren

# Präzision von Fließkommazahlen

## Bisher häufig verwendet

- Double precision: 64 bit → 8 bytes
- Single precision: 32 bit → 4 bytes

## Besonders häufig bei LLMs

- 16 bit → 2 bytes (bfloat16)

## Für ein 70B Modell braucht man 140 GB RAM

- Kann man das reduzieren?
- Präzision auf 8 bit reduzieren (int8)
- Funktioniert, aber muss jedesmal berechnet werden
- Dazu wird wieder viel Speicher benötigt

Type	Bits			
	Sign	Exponent	Significand	Total
<u>Half (IEEE 754-2008)</u>	1	5	10	16
<u>Single</u>	1	8	23	32
<u>Double</u>	1	11	52	64
<u>x86 extended precision</u>	1	15	64	80
<u>Quad</u>	1	15	112	128

Source: [https://en.wikipedia.org/wiki/Floating-point\\_arithmetic](https://en.wikipedia.org/wiki/Floating-point_arithmetic)

# Quantisierung

## Üblicherweise verwenden LLMs bfloat16

- Optimiert für Machine Learning
- Braucht weniger Platz
- 8 bit bleiben für Exponent, aber nur 7 bit für Mantisse
- Präzision wird zugunsten des Wertebereichs geopfert

## 8 bit

- Nutzt die bitsandbytes Bibliothek
  - Kann als Option direk in die Hugging Face Modelle integriert werden
  - Funktioniert oft ohne spürbare Einschränkungen in der Qualität (!)
  - Erlaubt große Modelle auf kleinere GPUs
- (LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale, 2022)



# Platz sparen mit GPTQ

## Kann man den RAM-Bedarf weiter reduzieren?

- Ja, durch weitere Reduzierung der Genauigkeit (z.B. zu int4)
- Funktioniert sehr gut für die *Evaluation* (aber nicht für das Training)
- Veröffentlichung in 2023:  
[GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers](#)
- Algorithmus ist viel schneller als alles Andere zuvor

## Existierende Implementierungen zur Modell-Evaluation

- In Hugging Face Transformer integriert:  
<https://huggingface.co/blog/gptq-integration>
- Auch als separate Software nutzbar: [llama.cpp](#)

## Bessere Quantisierung

- Idee: Dynamisch für unterschiedliche Layer
- AWQ: Activation-aware quantization
  - Viele Modelle bereits verfügbar
  - Braucht viel VRAM (Layer umschreiben)
- ExLlamaV2: mittlere Anzahl von Bits
  - Quantisierung muss oft selbst durchgeführt werden
  - Benötigt Datenset zum Abgleich
  - Sehr schnelle Inferenz mit geringem RAM-Bedarf
- llama.cpp: Ausführung auf CPU (gguf-Format)

## Fertige Software für Inferenz

- TGI
- vLLM





# Feintuning von LLMs

# Werkzeugkasten

## Grundsätzliche Problematik

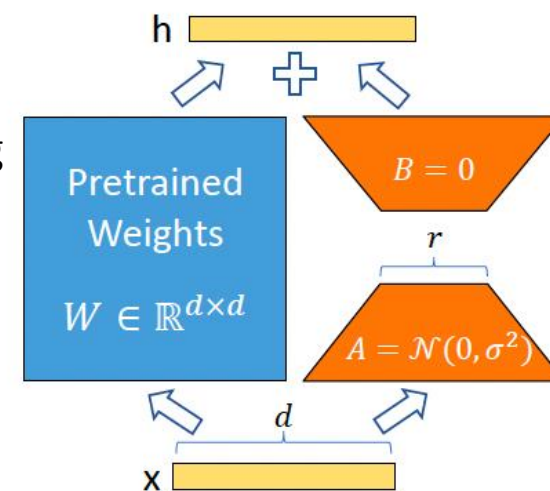
- Modelle haben sehr (zu) viele Parameter
- Training, aber auch Finetuning sehr aufwändig
- Ziel: Anzahl der Parameter reduzieren

## LoRA: Low Rank Adaptation

- Reduktion der trainierbaren Parameter
- “Einfrieren” der Originalgewichte
- Darstellung der (veränderten) Gewichte durch ein Produkt zweier niedrigdimensionaler Matrizen
- Sehr effizient und oft besser als “vollständiges” Feintuning

## PEFT – “Parameter Efficient Tuning”

- Bibliothek von Hugging Face
- Praktisch überall verwendet
- Ermöglicht Feintuning auf “Consumer Hardware” (A100)



## QLoRA

- Feintuning eines bereits quantisierten Modells
- Weitere Optimierung mit LoRA
- Deutlich verringerter Speicherbedarf

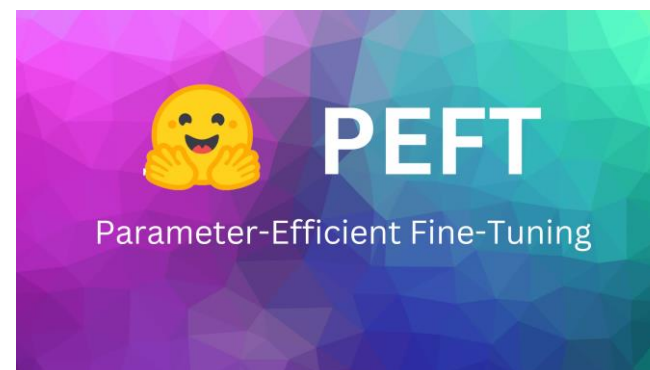
## Deep Speed

- Optimierung für Instruction Following (Reinforcement Learning with Human Feedback, RLHF)
- Sehr hohe Beschleunigung in der Lernphase



## Accelerate

- Verteilte Ausführung



# Modelle, Datensätze und Trainingssoftware

## Beispiel mit kleinem Modell

- Phi-2
- gemma

## Datensätze

- Häufig Instruction Following
- Aber nicht nur, auch Fortsetzung
- Kommt auf den Anwendungsfall an
- Unterschied ist nur in der Konstruktion der Texte zum Training

## Umgebung

- Nicht immer ganz einfach mit Docker
- Spezielle Anpassungen für CUDA notwendig

## Selbst schreiben

- Sehr viele Optionen, teilweise auch modellspezifisch
- Möglich, aber unübersichtlich

## axolotl

- Viele Modelle unterstützt und wird ständig erweitert
- Notebook oder CLI
- Nicht ganz einfach zu installieren (Enironment verwenden!)

## unsloth

- Besonders effizient, unterstützt QLoRA
- Jupyter-Notebooks verfügbar

## xtuner

- Relativ neu, besonders für chinesischen content gedacht
- Relativ hohe Laufzeit der Beispiele, Dokumentation noch nicht ausgereift

[https://github.com/mlabonne/llm-course/blob/main/Fine tune LLMs with Axolotl.ipynb](https://github.com/mlabonne/llm-course/blob/main/Fine%20tune%20LLMs%20with%20Axolotl.ipynb)





# Eigene Modelle ohne Training?



















## Training ist aufwändig

- Trainingsdaten werden benötigt
- Braucht GPUs
- Braucht Rechenzeit

## Alternative

- Model Merging
- Nutzung existierender Modelle und LoRas
- “Verschmelzung” der Modelle
- Funktioniert überraschenderweise verhältnismäßig gut
- “Frankenmerge”
- Nicht mehr ganz so populär

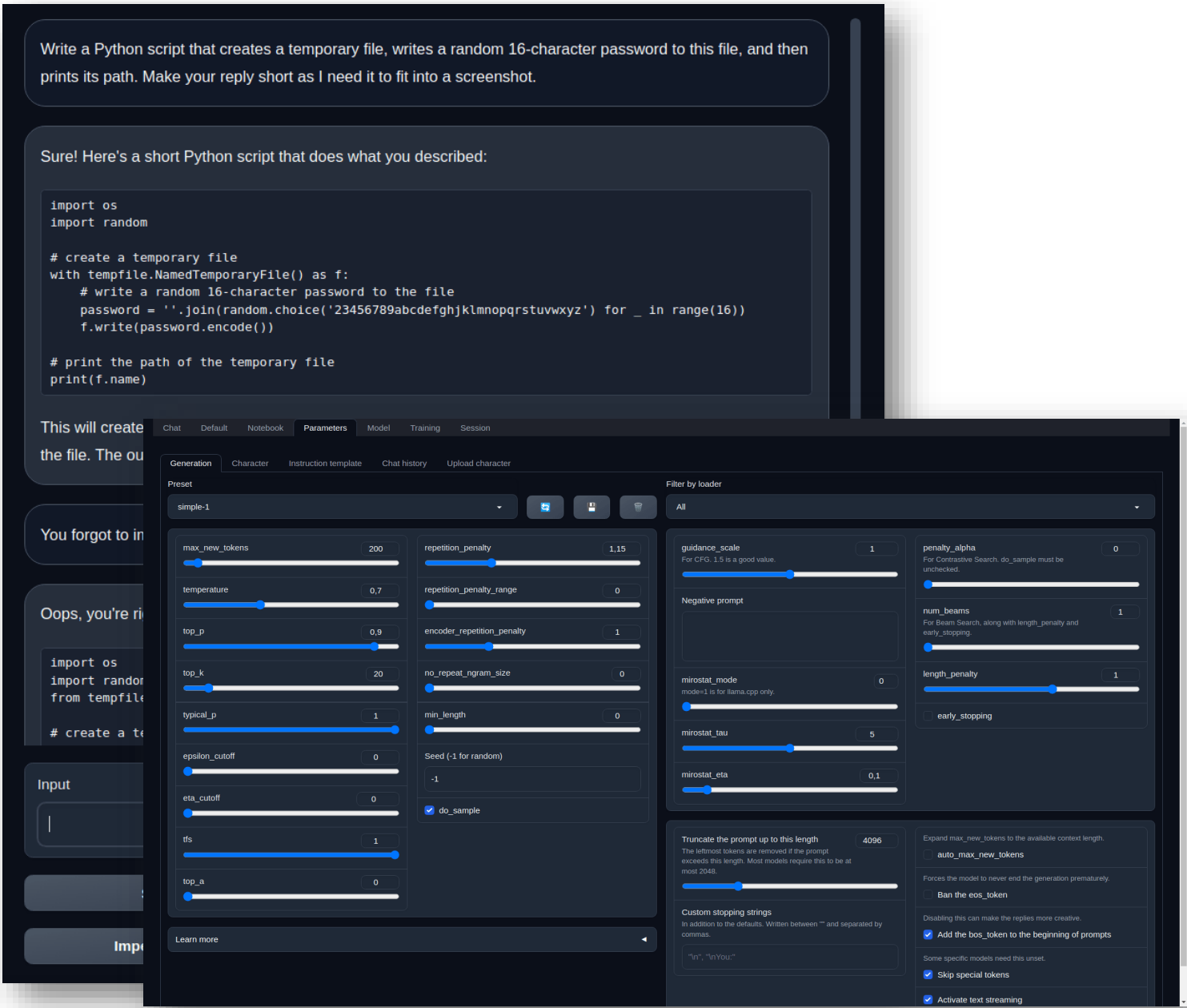
Models 44  Full-text search Sort: Trending

 S4sch/Open-Hermes-2.5-neural-chat-3.1-frankenmerge-... Text Generation • Updated Nov 30, 2023 • 1.29k • 6	 S4sch/Open-Hermes-2.5-neural-chat-3.1-frankenmerge-... Text Generation • Updated Nov 29, 2023 • 18 • 1
 S4sch/zephyr-neural-chat-frankenmerge11b Text Generation • Updated Dec 2, 2023 • 1.3k • 4	 TheBloke/Open-Hermes-2.5-neural-chat-3.1-frankenmer... Updated Nov 30, 2023 • 298 • 7
 TheBloke/Open-Hermes-2.5-neural-chat-3.1-frankenmer... Text Generation • Updated Nov 30, 2023 • 4 • 2	 TheBloke/Open-Hermes-2.5-neural-chat-3.1-frankenmer... Text Generation • Updated Nov 30, 2023 • 12 • 1
 S4sch/zephyr-neural-chat-frankenmerge11b-gguf-q5 Text Generation • Updated Nov 29, 2023 • 189 • 3	 Mihaiiii/Pallas-0.5-frankenmerge Text Generation • Updated Jan 6 • 13 • 1
 TheBloke/Pallas-0.5-frankenmerge-GPTQ Text Generation • Updated Jan 7 • 17 • 1	 TheBloke/Pallas-0.5-frankenmerge-AWQ Text Generation • Updated Jan 6 • 11 • 1
 TheBloke/Pallas-0.5-frankenmerge-GGUF Updated Jan 6 • 67 • 2	 osanseviero/mistral-instruct-frankenmerge Text Generation • Updated Jan 10 • 71 • 1
 frankenmerger/delta-4b-super Text Generation • Updated Mar 10 • 76	 frankenmerger/delta-4b-notso-base Text Generation • Updated Mar 8 • 69
 frankenmerger/gemma-4b-instruct-v0.1 Text Generation • Updated Mar 21 • 43	 frankenmerger/delta-4b-orange Text Generation • Updated Mar 8 • 91
 frankenmerger/cosmo-3b-test Text Generation • Updated Mar 14 • 117	 frankenmerger/delta-4b-instruct-v0.1 Text Generation • Updated Mar 13 • 7



# Frontends für LLMs

# Die Qual der Wahl...



## llama.cpp

**User:** Explain Heise Zeitschriftenverlag

**Llama:** Heise Zeitschriftenverlag is a German publishing company that specializes in computer science and technology magazines. The company was founded in 1975 by Klaus Heise and his brother Hans-Peter Heise. Today, it is one of the largest publishers of technical literature in Germany, with over 200 publications covering topics such as software development, hardware engineering, network administration, and more. Some of their well-known titles include Computer Bild, c't, and Chip.

Say something...

Send Stop Reset

157ms per token, 6.35 tokens per second  
Powered by [llama.cpp](#) and [ggml.ai](#).





# **Zusammenfassung und Ausblick**

# Warum so spannend?

## Mehr und mehr Basismodelle stehen zur Verfügung

- Rege Entwicklung mit offenen Daten etc.
- Google: Open Source-Modelle werden den kommerziellen den Rang ablaufen<sup>1</sup>
- Domänenspezifisches Finetuning

## Business Cases

- Bestimmte Berufsbilder nicht mehr existent: Copywriter
- Viele Ideen fehlen noch

## Kombination der Verfahren

- Retrieval Augmented Generation
- Effizient und vermeidet viele Probleme
- Neue Form des Information Retrievals



<sup>1</sup> <https://www.heise.de/news/Anonymer-Google-Entwickler-Open-Source-wird-Google-und-OpenAI-den-Rang-ablaufen-8989179.html>



# Aktuelle Entwicklungen

## RAG

- Retrieval Augmented Generation

## Spezialmodelle

- CodeLlama, Llemma, geometische Modelle etc.

## Performance von Open Source-Modellen

- Mistral, Mixtral, Miqu, Qwen, Command R

## Fragen zur Transparenz: Ganz neue Fragestellungen

## Neue Methoden

- HQQ (Half quadratic quantization)
- Groq, Cerebras: Extrem schnelle Inferenz



Quelle: [https://www.linkedin.com/posts/maxime-labonne\\_arena-elo-graph-updated-with-new-models-activity-7187062633735368705-u2jB](https://www.linkedin.com/posts/maxime-labonne_arena-elo-graph-updated-with-new-models-activity-7187062633735368705-u2jB)



# Feedback