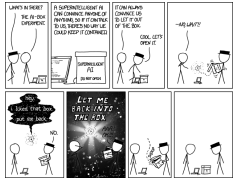


Advanced Natural Language Processing & Large Language Models

Felix Neubürger

2025

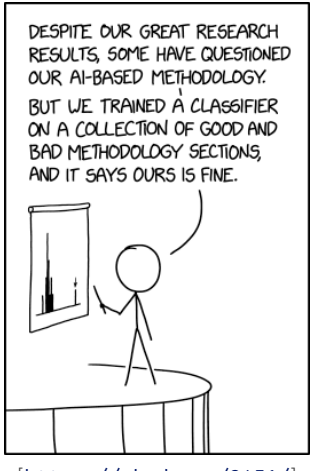
Fachhochschule Südwestfalen, Ingenieurs- & Wirtschaftswissenschaften

Inhalte der Vorlesung

- Wie funktioniert Natural Language Processing
- Sprachdarstellung zum Rechnen
- Attentionmechanismus
- Transformerarchitektur
- von BERT zu DeepSeek-v3
- Wie es weitergehen kann
- Nutzungsmöglichkeiten: RAG, Agentensysteme
- AI Safety und Ethik

Ziele der Vorlesung - Welche Fragen sollen beantwortet werden?

- Was sind die Grundlagen von Natural Language Processing (NLP)?
- Wie funktionieren Attention-Mechanismen und warum sind sie wichtig?
- Was ist die Transformer-Architektur und wie unterscheidet sie sich von anderen Ansätzen?
- Wie werden Sprachmodelle wie BERT und GPT trainiert und genutzt?
- Welche Herausforderungen und ethischen Fragen gibt es bei der Nutzung von LLMs?
- Welche praktischen Anwendungen und Zukunftsperspektiven gibt es für LLMs?



[https://xkcd.com/2451/]

Format der Vorlesung - Wie sollen diese Fragen beantwortet werden?

- Theroretischer Teil mit Folien
- Praktischer Teil in Gruppen an einem Projekt
- Gruppengröße 2 oder 3 Personen
- Einzelarbeit möglich wenn eigenes Thema vorhanden
- Abgabe der Ausarbeitung einen Tag vor der Veranstaltung in der Blockwoche
- Vorstellung der Projektergebnisse in der Blockwoche
- Gewichtung der Bewertung Projektausarbeitung (50%) und Vortrag (50%)

[https://xkcd.com/1425/]

Wie funktioniert Natural Language Processing

- Definition und Ziele des NLP
- Herausforderungen bei der maschinellen Sprachverarbeitung
- Anwendungen von NLP in der Praxis

Definition und Ziele des NLP

- NLP steht für Natural Language Processing, die Verarbeitung natürlicher Sprache durch Computer.
- Ziel: Maschinen ermöglichen, menschliche Sprache zu verstehen, zu interpretieren und zu generieren.
- Anwendungen: Übersetzungen, Chatbots, Textanalyse, Sprachassistenten.

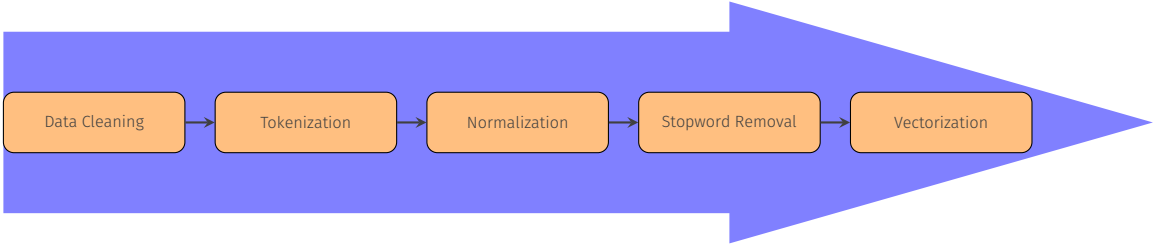
Herausforderungen bei der maschinellen Sprachverarbeitung

- Ambiguität: Mehrdeutigkeit in der Sprache.
- Kontextabhängigkeit: Bedeutung hängt vom Kontext ab.
- Umgang mit Synonymen und Homonymen.
- Verarbeitung großer Datenmengen und Rechenaufwand.

Anwendungen von NLP in der Praxis

- Sentiment-Analyse: Erkennung von Meinungen in Texten.
- Maschinelle Übersetzung: Automatische Übersetzung zwischen Sprachen.
- Sprachgesteuerte Assistenten: Siri, Alexa, Google Assistant.
- Textzusammenfassung: Automatische Erstellung von Textzusammenfassungen.

Text Preprocessing Pipeline



Data Cleaning

- **Definition:** Entfernen oder Korrigieren von fehlerhaften, unvollständigen oder irrelevanten Daten.
- **Schritte:**
  - Entfernen von Sonderzeichen, HTML-Tags und Emojis.
  - Korrektur von Rechtschreibfehlern.
  - Vereinheitlichung von Groß- und Kleinschreibung.
- **Ziel:** Verbesserung der Datenqualität für nachfolgende Verarbeitungsschritte.

Tokenization

- **Definition:** Zerlegung von Text in kleinere Einheiten (Tokens), z. B. Wörter oder Satzzeichen.
- **Arten:**
  - Wortbasierte Tokenization: ”Das ist ein Satz.” → [”Das”, ”ist”, ”ein”, ”Satz”, ”.”]
  - Zeichenbasierte Tokenization: ”Hallo” → [”H”, ”a”, ”l”, ”l”, ”o”]
  - Subwortbasierte Tokenization: ”unbelievable” → [”un”, ”believ”, ”able”]
- **Herausforderungen:** Umgang mit zusammengesetzten Wörtern, Abkürzungen und Sonderzeichen.

Normalization

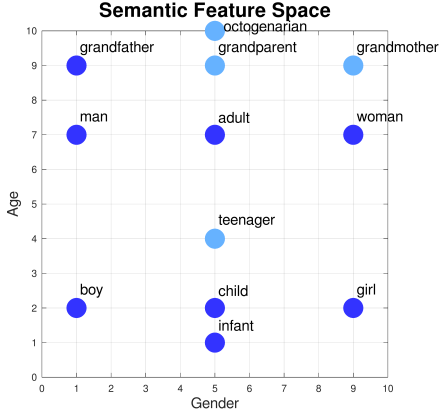
- **Definition:** Vereinheitlichung von Textdaten, um Konsistenz zu gewährleisten.
- **Schritte:**
  - Umwandlung in Kleinbuchstaben: ”Haus” → ”haus”.
  - Entfernen von Akzenten: ”café” → ”cafe”.
  - Stemming: Reduktion auf Wortstamm, z. B. ”running” → ”run”.
  - Lemmatization: Rückführung auf Grundform, z. B. ”better” → ”good”.
- **Ziel:** Reduktion der Variabilität in den Daten.

Stopword Removal

- **Definition:** Entfernen von häufig vorkommenden Wörtern, die wenig Bedeutung tragen (z. B. ”der”, ”und”, ”ist”).
- **Vorgehen:**
  - Verwendung einer vordefinierten Stopword-Liste (z. B. ”der”, ”die”, ”und”, ”ist”, ”ein”, ”zu”).
  - Anpassung der Liste an den spezifischen Anwendungsfall.
- **Vorteile:**
  - Reduktion der Datenmenge.
  - Verbesserung der Modellleistung durch Fokus auf relevante Wörter.
- **Herausforderung:** Manche Stopwords können je nach Kontext wichtig sein.

Sprachdarstellung zum Rechnen

- Wortvektoren und Einbettungen (Embeddings)
- One-Hot-Encoding vs. verteilte Repräsentationen
- Word2Vec, GloVe und andere Einbettungsmethoden



Wortvektoren und Einbettungen (Embeddings)

- Ziel: Repräsentation von Wörtern in einem kontinuierlichen Vektorraum.

- Mathematische Definition:
  - Gegeben eine Menge von Wörtern $W = \{w_1, w_2, \ldots, w_n\}$.
  - Eine Einbettung ist eine Funktion $f : W \to \mathbb{R}^d$, wobei d die Dimension des Vektorraums ist.
  - Beispiel: $f(w_i) = \mathbf{v}_i \in \mathbb{R}^d$.

- Vorteile:
  - Semantische Ähnlichkeit wird durch Nähe im Vektorraum dargestellt.
  - Reduktion der Dimensionalität im Vergleich zu One-Hot-Encoding.

One-Hot-Encoding vs. Verteilte Repräsentationen

- **One-Hot-Encoding:**
  - Jedes Wort wird als Vektor mit einer einzigen Eins und sonst Nullen dargestellt.
  - Beispiel: Für $W = \{w_1, w_2, w_3\}$, w_2 wird als $[0, 1, 0]$ kodiert.
  - Nachteile: Hohe Dimensionalität, keine semantische Information.

- **Verteilte Repräsentationen:**
  - Nutzen kontinuierliche Vektorräume, um semantische Beziehungen darzustellen[1]..
  - Ermöglichen die Nutzung von Modellen wie Word2Vec und GloVe [2].
  - Wörter werden als dichte Vektoren in einem "niedrig"dimensionalen Raum dargestellt.
  - Semantisch ähnliche Wörter haben ähnliche Vektoren.
  - Beispiel: $f(w_1) = [0.2, 0.8]$, $f(w_2) = [0.3, 0.7]$.

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.

[2] Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).

Word2Vec, GloVe und andere Einbettungsmethoden

- **Word2Vec:**
  - Skip-Gram-Modell: Vorhersage des Kontexts basierend auf einem Zielwort[3].
  - CBOW-Modell: Vorhersage des Zielworts basierend auf dem Kontext[4].

- **GloVe (Global Vectors for Word Representation):**
  - Nutzt globale Wort-Kooccurenz-Matrizen[5].
  - Optimiert eine Zielfunktion, die Wortpaare und ihre Häufigkeiten berücksichtigt[6].

- **Andere Methoden:**
  - FastText: Berücksichtigt Subwortinformationen.
  - BERT-Embeddings: Kontextabhängige Einbettungen.

[3] Das Skip-Gram-Modell versucht, für ein gegebenes Zielwort die umgebenden Kontextwörter vorherzusagen.

[4] Das Continuous Bag of Words (CBOW)-Modell sagt ein Zielwort basierend auf den umgebenden Kontextwörtern vorher. Es ist effizienter als das Skip-Gram-Modell, aber weniger präzise bei seltenen Wörtern.

[5] GloVe basiert auf der Idee, dass die globale Häufigkeit von Wortpaaren in einem Korpus genutzt werden kann, um semantische Beziehungen zwischen Wörtern zu modellieren.

[6] Die Zielfunktion von GloVe minimiert den Unterschied zwischen der inneren Produktdarstellung von Wortvektoren und der logarithmierten Häufigkeit von Wortpaaren.

Neuartige Embeddings (Teil 1)

- **Kontextabhängige Embeddings:**
  - Modelle wie BERT[7], GPT[8] und T5[9] generieren Embeddings, die den Kontext eines Wortes berücksichtigen.
  - Beispiel: Das Wort ”Bank” hat unterschiedliche Embeddings in den Sätzen ”Ich sitze auf der Bank” und ”Ich gehe zur Bank”.

- **Sentence Embeddings:**
  - Repräsentieren ganze Sätze statt einzelner Wörter.
  - Modelle wie Sentence-BERT (SBERT)[10] ermöglichen semantische Suche und Textähnlichkeitsbewertung.

[7] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/abs/1810.04805

[8] Brown, T. et al. (2020). Language Models are Few-Shot Learners. https://arxiv.org/abs/2005.14165

[9] Raffel, C. et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. https://arxiv.org/abs/1910.10683

[10] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://arxiv.org/abs/1908.10084

Neuartige Embeddings (Teil 2)

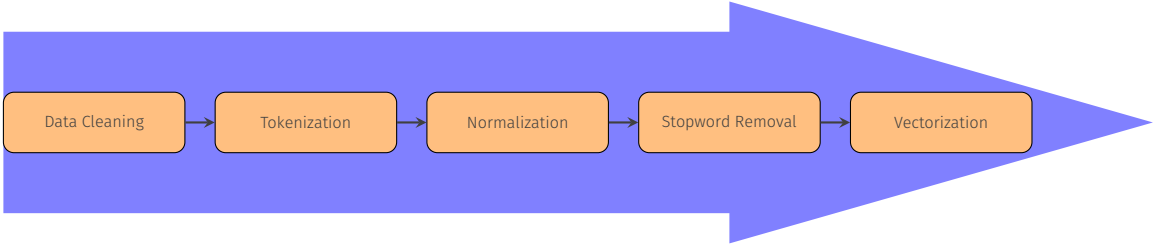
- **Multimodale Embeddings:**
  - Kombinieren Informationen aus verschiedenen Modalitäten wie Text, Bild und Audio.
  - Beispiel: CLIP (Contrastive Language–Image Pretraining)[11] von OpenAI.

- **Graphbasierte Embeddings:**
  - Repräsentieren Wörter als Knoten in einem Graphen, wobei Kanten Beziehungen zwischen Wörtern darstellen.
  - Beispiel: Node2Vec[12] und GraphSAGE[13].

- **Adapter-basierte Embeddings:**
  - Ermöglichen die Anpassung vortrainierter Modelle an spezifische Aufgaben durch leichte Modifikationen.
  - Reduzieren den Speicherbedarf im Vergleich zu vollständigem Fine-Tuning[14].

[11] Radford, A. et al. (2021). Learning Transferable Visual Models From Natural Language Supervision. https://arxiv.org/abs/2103.00020
[12] Grover, A., & Leskovec, J. (2016). node2vec: Scalable Feature Learning for Networks. https://arxiv.org/abs/1607.00653
[13] Hamilton, W. et al. (2017). Inductive Representation Learning on Large Graphs. https://arxiv.org/abs/1706.02216
[14] Houlsby, N. et al. (2019). Parameter-Efficient Transfer Learning for NLP. https://arxiv.org/abs/1902.00751

Text Preprocessing Pipeline



Attention-Mechanismus

- Motivation für Attention in Sequenzmodellen
- Funktionsweise des Attention-Mechanismus
- Unterschied zwischen Self-Attention und Cross-Attention

Motivation für Attention in Sequenzmodellen

- Problem: In langen Sequenzen verlieren Modelle wie RNNs und LSTMs den Überblick über frühere Informationen.
- Lösung: Der Attention-Mechanismus ermöglicht es, gezielt auf relevante Teile der Eingabesequenz zu fokussieren.
- Beispiel: Bei der Übersetzung eines Satzes kann Attention bestimmen, welches Wort im Quelltext für ein bestimmtes Wort im Zieltext wichtig ist.

Funktionsweise des Attention-Mechanismus: Query, Key und Value

- **Eingabe:** Eine Sequenz von Eingabevektoren $X = \{x_1, x_2, \ldots, x_n\}$, wobei $x_i \in \mathbb{R}^d$.
- **Transformation:** Jeder Eingabevektor wird durch trainierbare Gewichtungsmatrizen in Query (Q), Key (K) und Value (V) umgewandelt.
- **Berechnung:**

$$
\begin{aligned}
Q &= XW_Q, \quad W_Q \in \mathbb{R}^{d \times d_k} \\
K &= XW_K, \quad W_K \in \mathbb{R}^{d \times d_k} \\
V &= XW_V, \quad W_V \in \mathbb{R}^{d \times d_v}
\end{aligned}
$$

  Hierbei sind W_Q, W_K und W_V trainierbare Gewichtungsmatrizen, und d_k, d_v sind die Dimensionen der Keys und Values.
- **Zweck:**
  - Q: Repräsentiert die Anfrage, welche Informationen benötigt werden.
  - K: Repräsentiert die Merkmale, die zur Beantwortung der Anfrage verwendet werden.
  - V: Repräsentiert die tatsächlichen Informationen, die weitergegeben werden.

Zusammenhang zwischen Query, Key und Value

- **Berechnung der Scores:**

$$\text{Score}(Q, K) = \frac{QK}{\sqrt{d_k}}$$

  Die Scores bestimmen, wie stark ein Query (Q) mit jedem Key (K) übereinstimmt.

- **Normalisierung der Scores:**

$$\alpha_{ij} = \text{softmax}\left(\frac{q_i k_j}{\sqrt{d_k}}\right)$$

  Die Softmax-Funktion wandelt die Scores in Wahrscheinlichkeiten um.

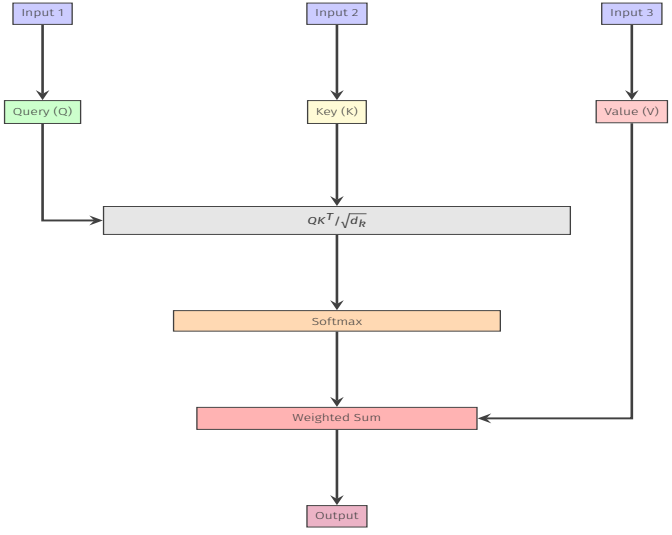
- **Gewichtete Summe der Values:**

$$z_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

  Die gewichtete Summe der Values (V) ergibt die Ausgabe des Attention-Mechanismus.

- **Interpretation:** Der Attention-Mechanismus ermöglicht es, relevante Informationen aus der Eingabesequenz basierend auf den Queries zu extrahieren.

Funktionsweise des Attention-Mechanismus



Self-Attention vs. Cross-Attention

Self-Attention:

- Jeder Token in der Sequenz bezieht sich auf alle anderen Tokens in derselben Sequenz.
- Beispiel: Kontextualisierung eines Wortes in einem Satz.

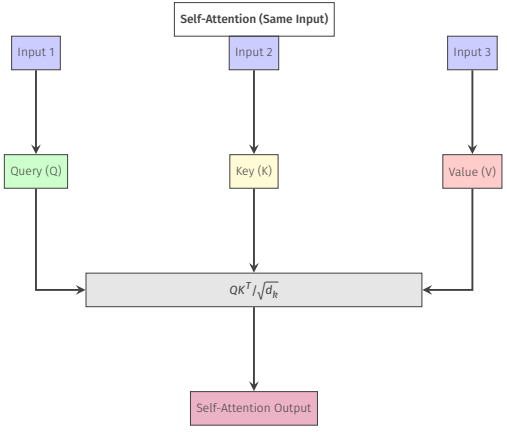
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK}{\sqrt{d_k}}\right)V, \quad Q = K = V$$

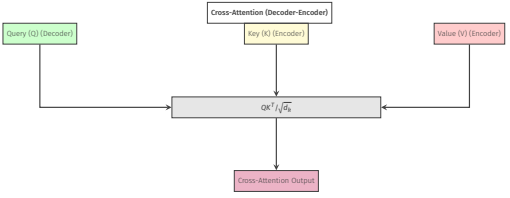
Cross-Attention:

- Tokens in einer Sequenz beziehen sich auf Tokens in einer anderen Sequenz.
- Beispiel: Übersetzung, bei der der Zieltext auf den Quelltext achtet.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK}{\sqrt{d_k}}\right)V, \quad Q \neq K = V$$

Self-Attention vs. Cross-Attention





Visualisierung der Attention-Matrix

- Die Attention-Matrix zeigt die Gewichte α_{ij}, die die Relevanz von Token j für Token i darstellen.
- Beispiel: Bei der Übersetzung eines Satzes zeigt die Matrix, welche Wörter im Quelltext für ein bestimmtes Wort im Zieltext wichtig sind.

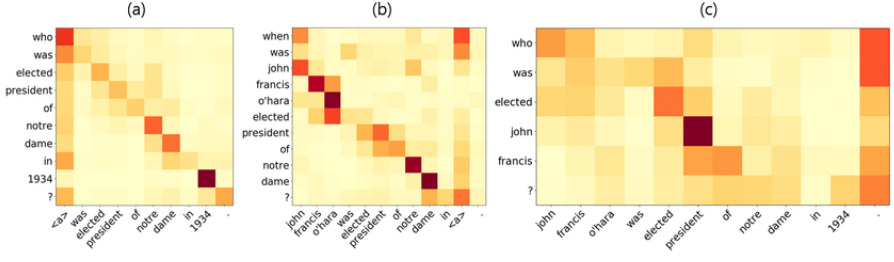


Abbildung: Beispiel einer Attention-Matrix. Kim, Yanghoon & Hwanhee, Lee & Shin, Joongbo & Jung, Kyomin. (2018). Improving Neural Question Generation using Answer Separation. 10.48550/arXiv.1809.02393.

Transformer-Architektur

- Überblick über die Transformer-Architektur
- Encoder-Decoder-Struktur
- Vorteile gegenüber rekurrenten Netzwerken

Überblick über die Transformer-Architektur

- Vorgestellt in "Attention Is All You Need" (Vaswani et al., 2017)[15].
- Besteht aus zwei Hauptkomponenten:
  - **Encoder:** Verarbeitet die Eingabesequenz.
  - **Decoder:** Generiert die Ausgabesequenz.
- Verwendet Attention-Mechanismen und vollständig vernetzte Schichten.
- Vorteil: Parallelisierbarkeit im Vergleich zu RNNs.

[15] https://doi.org/10.48550/arXiv.1706.03762

Transformer-Architektur: Überblick

- **Encoder:**
  - Besteht aus mehreren Schichten.
  - Jede Schicht enthält:
    - Multi-Head Self-Attention.
    - Feed-Forward-Netzwerk.
- **Decoder:**
  - Ähnlich wie der Encoder, aber mit zusätzlicher Maskierung.
  - Enthält Cross-Attention, um Informationen vom Encoder zu nutzen.
- **Vorteile:**
  - Parallelisierbarkeit.
  - Effektive Modellierung von Abhängigkeiten.

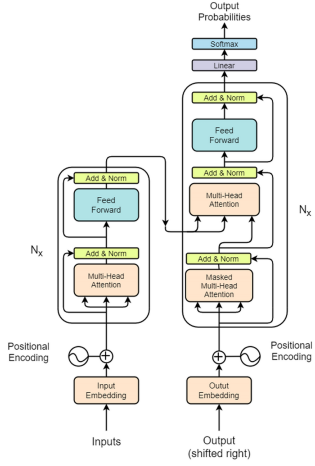


Abbildung: Transformer-Architektur[a]

[a] ni et al., 2017.

Encoder-Block: Multi-Head Attention

- **Multi-Head Attention:**
  - Berechnet die Attention über verschiedene Teile der Eingabesequenz.
  - Formel:

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

wobei:

- $Q = XW_Q$: Queries
- $K = XW_K$: Keys
- $V = XW_V$: Values
- W_Q, W_K, W_V: Gewichtungsmatrizen

  - Multi-Head Mechanismus:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

wobei $\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i})$.

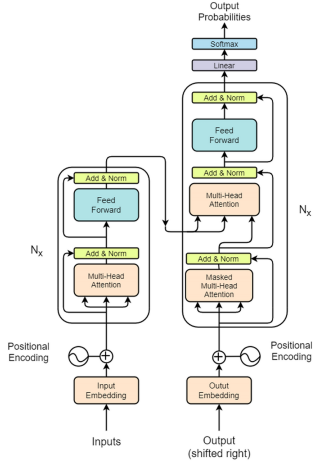


Abbildung: Transformer-Architektur[a]

[a] ni et al., 2017.

Encoder-Block: Feed-Forward-Netzwerk

- **Feed-Forward-Netzwerk:**
  - Architektur:

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2$$

    wobei:
    - W_1, W_2: Gewichtungsmatrizen
    - b_1, b_2: Bias-Vektoren
    - ReLU: Aktivierungsfunktion
  - Zweck: Transformation der Eingabe in einen höherdimensionalen Raum, um komplexere Muster zu lernen.

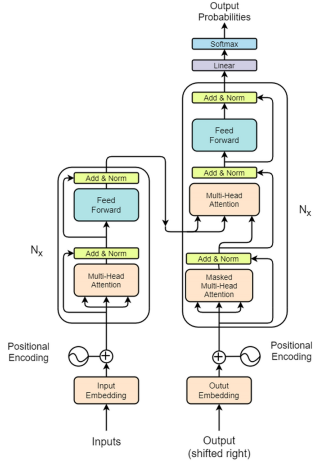


Abbildung: Transformer-Architektur[a]

[a] ni et al., 2017.

Encoder-Block: Add & Norm

- **Add & Norm:**
  - Residual Connection:

$$\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))$$

    wobei SubLayer(x) entweder Multi-Head Attention oder das Feed-Forward-Netzwerk ist.
  - Layer Normalization:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

    wobei:
    - μ: Mittelwert der Eingabe
    - σ: Standardabweichung der Eingabe
    - γ, β: Trainierbare Parameter
  - Zweck: Stabilisierung des Trainings und Verbesserung der Konvergenz.

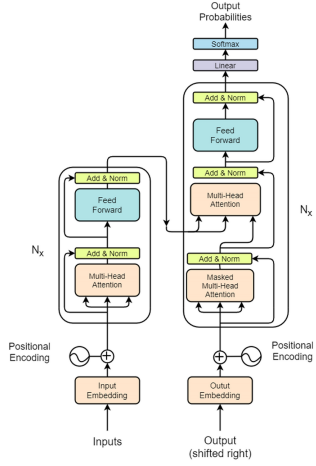


Abbildung: Transformer-Architektur[a]

[a] ni et al., 2017.

Decoder-Block: Multi-Head Attention

- **Masked Multi-Head Self-Attention:**
  - Verhindert, dass ein Token auf zukünftige Tokens zugreift.
  - Maskierung der Attention-Matrix:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK}{\sqrt{d_k}} + M\right)V$$

  wobei M eine Maske ist, die zukünftige Positionen ausschließt.
- **Cross-Attention:**
  - Verbindet den Decoder mit dem Encoder.
  - Nutzt die Encoder-Ausgaben als Keys und Values.
- **Residual Connection und Layer Normalization:**
  - Wie im Encoder-Block, um Stabilität und Konvergenz zu verbessern.

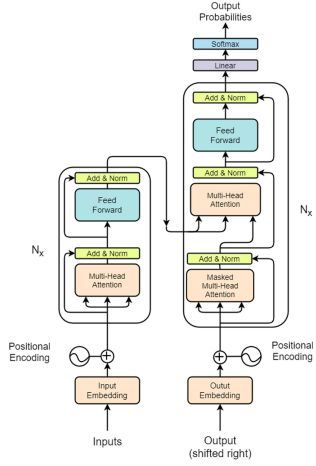


Abbildung: Decoder-Block[a]

[a] ni et al., 2017.

Decoder-Block: Feed-Forward-Netzwerk

- **Feed-Forward-Netzwerk:**
  - Architektur:

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2$$

    wobei:
    - W_1, W_2: Gewichtungsmatrizen
    - b_1, b_2: Bias-Vektoren
    - ReLU: Aktivierungsfunktion
  - Zweck: Transformation der Eingabe in einen höherdimensionalen Raum, um komplexere Muster zu lernen.
- **Residual Connection und Layer Normalization:**
  - Wie im Encoder-Block, um Stabilität und Konvergenz zu verbessern.

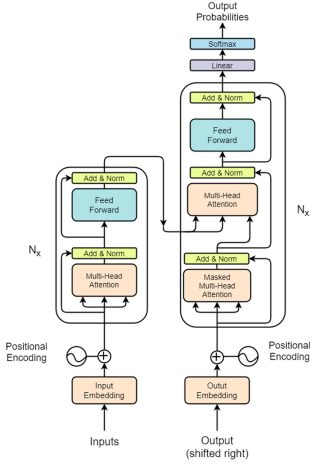


Abbildung: Decoder-Block[a]

[a] ni et al., 2017.

Decoder-Block: Add & Norm

- **Add & Norm:**
  - Residual Connection:

$$\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))$$

    wobei SubLayer(x) entweder Masked Multi-Head Self-Attention, Cross-Attention oder das Feed-Forward-Netzwerk ist.
  - Layer Normalization:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

    wobei:
    - μ: Mittelwert der Eingabe
    - σ: Standardabweichung der Eingabe
    - γ, β: Trainierbare Parameter
  - Zweck: Stabilisierung des Trainings und Verbesserung der Konvergenz.

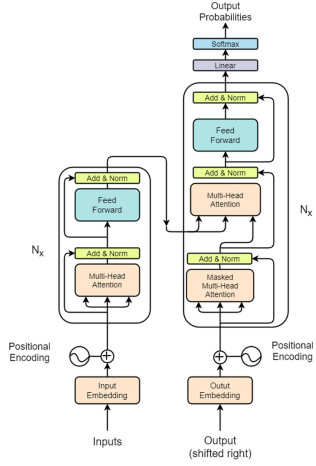


Abbildung: Decoder-Block[a]

[a] ni et al., 2017.

Vorteile der Transformer-Architektur

- **Parallelisierbarkeit:** Ermöglicht schnellere Trainingszeiten im Vergleich zu RNNs.
- **Langfristige Abhängigkeiten:** Kann Beziehungen zwischen weit entfernten Tokens modellieren.
- **Flexibilität:** Kann für verschiedene Aufgaben wie Übersetzung, Textklassifikation und mehr verwendet werden.
- **Skalierbarkeit:** Grundlage für große Sprachmodelle wie BERT und GPT.

Output-Möglichkeiten der Transformer-Architektur (Teil 1)

- **Sequenz-zu-Sequenz (Seq2Seq):**
  - Beispiel: Maschinelle Übersetzung (z. B. Englisch → Deutsch).
  - Eingabe: Eine Sequenz von Tokens.
  - Ausgabe: Eine Sequenz von Tokens in einer anderen Sprache.
  - **Erreichung:** Verwendung eines Encoder-Decoder-Transformers, wobei der Encoder die Eingabesequenz verarbeitet und der Decoder die Ausgabesequenz generiert.
- **Sequenz-zu-Einzelwert (Seq2Single):**
  - Beispiel: Textklassifikation (z. B. Sentiment-Analyse).
  - Eingabe: Eine Sequenz von Tokens.
  - Ausgabe: Eine einzelne Klasse oder ein Wert.
  - **Erreichung:** Hinzufügen einer Klassifikationsschicht (z. B. Softmax) am Ende des Encoders, um die Klasse vorherzusagen.

Output-Möglichkeiten der Transformer-Architektur (Teil 2)

- **Sequenz-zu-Token (Seq2Token):**
  - Beispiel: Fragebeantwortung (z. B. Auswahl eines Tokens als Antwort).
  - Eingabe: Eine Sequenz von Tokens.
  - Ausgabe: Ein spezifisches Token aus der Eingabe.
  - **Erreichung:** Nutzung eines Modells wie BERT, das Start- und Endpositionen in der Eingabesequenz vorhersagt.
- **Sequenz-zu-Vektor (Seq2Vec):**
  - Beispiel: Satz- oder Dokumenteinbettung.
  - Eingabe: Eine Sequenz von Tokens.
  - Ausgabe: Ein Vektor, der die gesamte Sequenz repräsentiert.
  - **Erreichung:** Extraktion des CLS-Tokens (bei BERT) oder Mittelung der Token-Embeddings, um die Sequenz zu repräsentieren.

Von BERT zu DeepSeek-v3

- Einführung in BERT und seine Architektur[a]
- Weiterentwicklungen: GPT[b], RoBERTa[c], T5[d]
- LLaMA[e], mistral[f], gemini[g], Claude[h]
- Überblick über DeepSeek-v3 und seine Besonderheiten

[a] https://arxiv.org/abs/1810.04805
[b] https://arxiv.org/abs/2005.14165
[c] https://arxiv.org/abs/1907.11692
[d] https://arxiv.org/abs/1910.10683
[e] https://arxiv.org/abs/2302.13971
[f] https://mistral.ai/
[g] https://www.deepmind.com/
[h] https://www.anthropic.com/

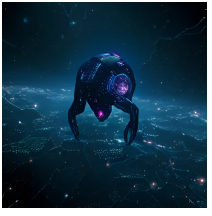


Abbildung: DeepSeek Janus Pro 7B-Interpretation von BERT und DeepSeek-v3 als KI-Modell.

BERT-Architektur: Überblick

- **BERT (Bidirectional Encoder Representations from Transformers):**
  - Entwickelt von Google AI (2018)[16].
  - Nutzt die Transformer-Encoder-Architektur.
  - Bidirektionales Training: Betrachtet den Kontext von Wörtern sowohl links als auch rechts.
- **Ziele:**
  - Verbesserung der Sprachrepräsentation.
  - Einsatz für verschiedene NLP-Aufgaben wie Fragebeantwortung, Sentiment-Analyse und mehr.

[16] https://arxiv.org/abs/1810.04805

BERT-Architektur: Aufbau

- **Eingabe:**
  - Tokenized Text: [CLS], Token 1, Token 2, ..., [SEP].
  - Token-, Segment- und Positions-Embeddings.
- **Encoder:**
  - Mehrere Transformer-Encoder-Schichten.
- **Ausgabe:**
  - Kontextualisierte Token-Embeddings.
  - CLS-Token für Klassifikationsaufgaben.

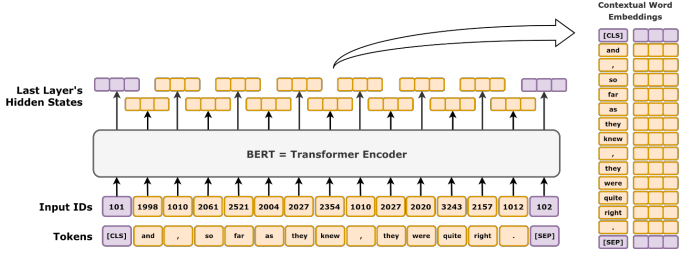


Abbildung: BERT-Architektur.

BERT-Architektur

BERT: Pretraining-Aufgaben

- **Masked Language Modeling (MLM):**
  - Zufälliges Maskieren von Tokens in der Eingabe.
  - Ziel: Vorhersage der maskierten Tokens basierend auf dem Kontext.
  - Beispiel: "Ich [MASK] ein Buch." → "Ich lese ein Buch."
- **Next Sentence Prediction (NSP):**
  - Ziel: Vorhersage, ob zwei Sätze aufeinander folgen.
  - Beispiel:
    - Satz A: "Ich gehe einkaufen."
    - Satz B: "Danach koche ich Abendessen." → True

BERT: Fine-Tuning

- **Vorgehen:**
  - Pretrained BERT-Modell wird an spezifische Aufgaben angepasst.
  - Hinzufügen einer zusätzlichen Schicht (z. B. Klassifikationslayer).
- **Beispiele für Aufgaben:**
  - Textklassifikation (z. B. Sentiment-Analyse).
  - Fragebeantwortung (z. B. SQuAD).
  - Named Entity Recognition (NER).

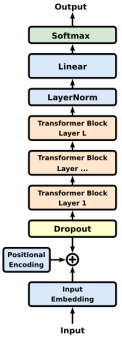
GPT-Architektur: Überblick

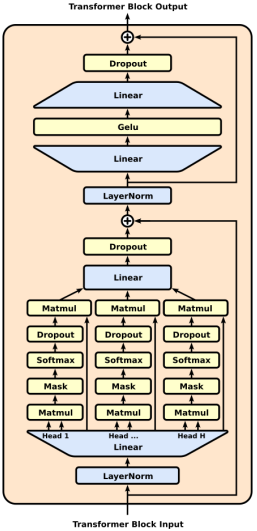
- **GPT (Generative Pre-trained Transformer):**
  - Entwickelt von OpenAI (2018)[17].
  - Nutzt die Transformer-Decoder-Architektur.
  - Unidirektionales Training: Betrachtet nur den Kontext links vom aktuellen Token.
- **Ziele:**
  - Generierung von kohärentem und zusammenhängendem Text.
  - Einsatz für Aufgaben wie Textgenerierung, Übersetzung und mehr.

[17] https://arxiv.org/abs/2005.14165

GPT-Architektur: Aufbau

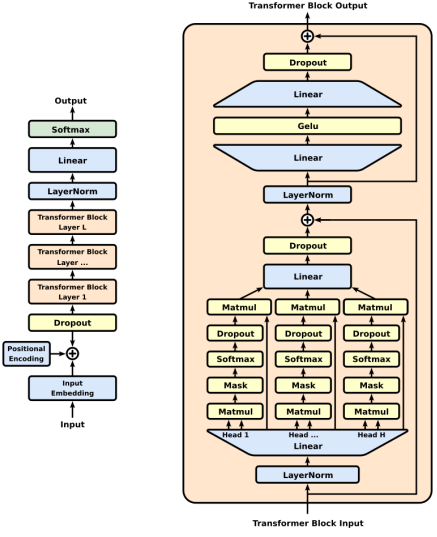
- **Eingabe:**
  - Tokenized Text: [BOS], Token 1, Token 2, ..., [EOS].
  - Token- und Positions-Embeddings.
- **Decoder:**
  - Mehrere Transformer-Decoder-Schichten.
  - Masked Multi-Head Self-Attention, um zukünftige Tokens auszuschließen.
- **Ausgabe:**
  - Wahrscheinlichkeitsverteilung über das Vokabular für das nächste Token.





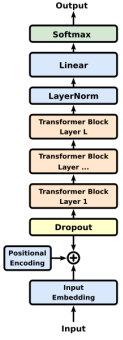
Überblick über die GPT-Architektur

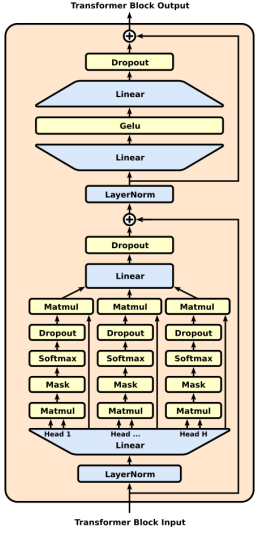
- GPT (Generative Pretrained Transformer) basiert vollständig auf der Transformer-Decoder-Architektur.
- Die Architektur besteht aus einem Stapel identischer Transformer-Blöcke.
- Jeder Block enthält Self-Attention, Feedforward-Netze, Residual-Verbindungen und Layer Normalization.



Input Embedding und Positionskodierung

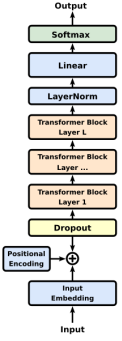
- Der Input besteht aus Token-IDs, die in Vektor-Repräsentationen (Embeddings) umgewandelt werden.
- Positionsinformationen werden über **Positional Encoding** hinzugefügt, da das Modell keine Reihenfolge kennt.
- Die Summe aus Input Embedding und Positional Encoding geht in den ersten Transformer-Block.

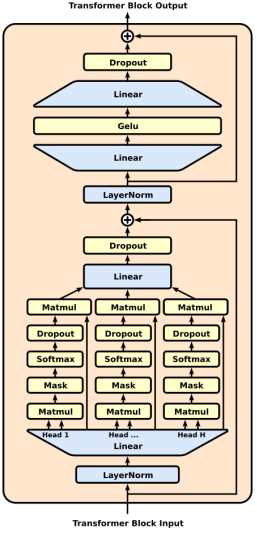




Self-Attention Mechanismus

- Jeder Token schaut auf andere Tokens in seinem Kontext (bisherige Tokens).
- GPT nutzt **Masked Multi-Head Self-Attention**, um die Vorhersage zukünftiger Tokens zu verhindern.
- Besteht aus:
  - Lineare Transformationen zu Query, Key, Value
  - Maskierung der Zukunft
  - Softmax über Attention Scores
  - Gewichtete Summation der Werte





Feedforward-Netzwerk und Residual-Verbindungen

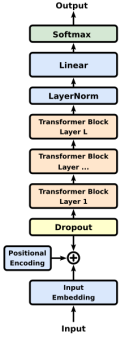
- Nach der Attention folgt ein Feedforward-Netz mit zwei linearen Schichten und einer Aktivierungsfunktion (GELU).
- GELU (Gaussian Error Linear Unit): Aktivierungsfunktion, definiert als:

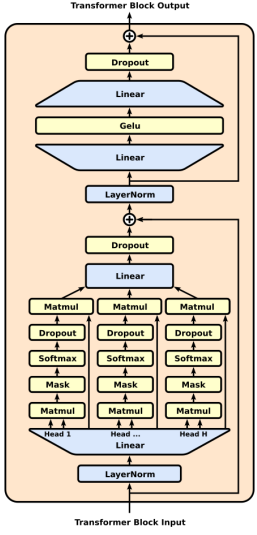
$$\text{GELU}(x) = x \cdot \Phi(x)$$

  wobei $\Phi(x)$ die kumulative Verteilungsfunktion der Standardnormalverteilung ist:

$$\Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

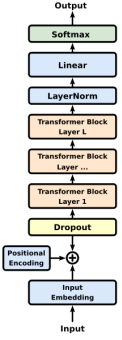
- Vorteil: Glattere Approximation im Vergleich zu ReLU, wodurch das Training stabiler wird.
- Residual-Verbindungen sorgen für stabileres Training.
- **LayerNorm** wird nach jeder Addition durchgeführt.
- **Dropout** wird zur Regularisierung verwendet.

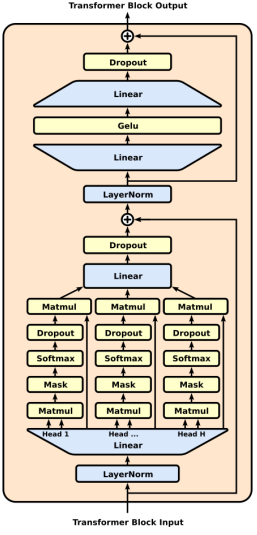




Mehrere Transformer-Layer

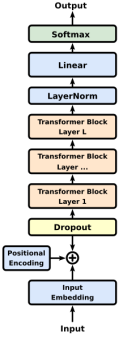
- Die Blöcke werden L-mal gestapelt (z. B. 12 für GPT-2 small, 96 für GPT-3).
- Jeder Layer hat dieselbe Architektur.
- Die Ausgabe des letzten Blocks wird verwendet, um Token-Vorhersagen durch ein lineares Projektionslayer + Softmax zu erzeugen.

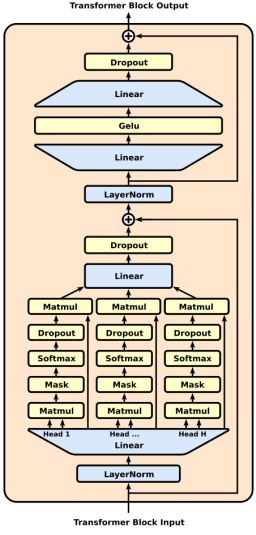




Zusammenfassung

- GPT besteht aus reinen Decoder-Blöcken des Transformers.
- Nutzt Masked Self-Attention zur autoregressiven Textgenerierung.
- Durch Pretraining auf sehr großen Textmengen kann GPT Sprachverständnis und -generierung erlernen.





Pretraining: Autoregressives Training

- GPT wird durch **Autoregressive Language Modeling** trainiert:

$$P(x_1, x_2, ..., x_n) = \prod_{t=1}^{n} P(x_t \mid x_{<t})$$

- Ziel: Nächstes Token vorhersagen basierend auf bisherigen Tokens.
- Loss-Funktion: Kreuzentropie-Loss zwischen vorhergesagtem und wahrem nächsten Token.
- Kontext wird vollständig nach links maskiert.

Trainingsdaten und Vorgehen

- GPT wird auf großen Korpora wie Common Crawl, Wikipedia, Bücher, Webseiten etc. trainiert.
- Tokenisierung erfolgt meist mit Byte-Pair Encoding (BPE).
- Typische Trainingsparameter (für GPT-2):
  - Kontextlänge: 1024 Tokens
  - Batchgröße: mehrere Millionen Tokens
  - Optimierung mit AdamW + Learning Rate Schedules
- Kein spezieller Pretraining-Task wie bei BERT (z. B. Masked LM oder NSP).

Fine-Tuning von GPT

- Nach dem Pretraining kann GPT für spezifische Aufgaben feinjustiert werden:
  - Textklassifikation
  - Fragebeantwortung
  - Dialogsysteme
  - Textgenerierung (z. B. ChatGPT)
- Fine-Tuning erfolgt meist mit task-spezifischen Daten.
- Architektur bleibt gleich – es werden oft nur letzte Layer angepasst.

GPT vs. BERT – Fundamentale Unterschiede

GPT (Decoder-basiert)

- Autoregressives Training (Zukunft maskiert)
- Nur Self-Attention nach links
- Einsatz: Textgenerierung, Dialogsysteme
- Output wird schrittweise erzeugt

BERT (Encoder-basiert)

- Masked Language Modeling (MLM)
- Bidirektionale Attention
- Einsatz: Klassifikation, Entitätenerkennung
- Kein autoregressiver Output

GPT erzeugt Sprache – BERT versteht sie.

Was ist Fine-Tuning?

- **Definition:** Anpassung eines vortrainierten Modells an eine spezifische Aufgabe oder Domäne.
- **Ziel:** Verbesserung der Leistung auf spezifischen Aufgaben durch zusätzliche Trainingsdaten.
- **Vorgehen:**
  - Start mit einem vortrainierten Modell (z. B. BERT, GPT).
  - Hinzufügen einer spezifischen Schicht (z. B. Klassifikationslayer).
  - Training auf einem domänenspezifischen Datensatz.

Vorteile des Fine-Tunings

- **Effizienz:** Reduziert den Bedarf an großen Trainingsressourcen, da das Modell bereits vortrainiert ist.
- **Anpassungsfähigkeit:** Ermöglicht die Anpassung an spezifische Aufgaben oder Domänen.
- **Verbesserte Leistung:** Höhere Genauigkeit und Relevanz für spezifische Anwendungen.
- **Wiederverwendbarkeit:** Vortrainierte Modelle können für verschiedene Aufgaben wiederverwendet werden.

Nachteile des Fine-Tunings

- **Overfitting:** Risiko, dass das Modell zu stark an den spezifischen Datensatz angepasst wird.
- **Datenbedarf:** Erfordert qualitativ hochwertige und ausreichend große Datensätze.
- **Rechenaufwand:** Kann trotz Vortraining immer noch ressourcenintensiv sein.
- **Komplexität:** Erfordert Fachwissen für die richtige Konfiguration und Optimierung.

Fine-Tuning-Techniken für LLMs

- **Parameter-Efficient Fine-Tuning (PEFT):**
  - Ziel: Reduktion der Anzahl der zu trainierenden Parameter.
  - Vorteile: Geringerer Speicherbedarf und schnellere Trainingszeiten.
- **Low-Rank Adaptation (LoRA):**
  - Ziel: Effiziente Anpassung vortrainierter Modelle durch Low-Rank-Matrizen.
  - Vorteile: Speicher- und Rechenaufwand werden drastisch reduziert.

Parameter-Efficient Fine-Tuning (PEFT)

- **Grundidee:**
  - Statt das gesamte Modell zu aktualisieren, werden nur wenige Parameter angepasst.
  - Nutzung von Adapter-Schichten, Prompt-Tuning oder LoRA.
- **Mathematische Grundlage:**
  - Gegeben ein vortrainiertes Modell mit Parametern θ.
  - PEFT optimiert nur einen kleinen Teil $\Delta\theta$, sodass:

$$\theta' = \theta + \Delta\theta$$

  - Ziel: Minimierung des Verlusts L über $\Delta\theta$:

$$\min_{\Delta\theta} L(f(x; \theta + \Delta\theta), y)$$

- **Vorteile:**
  - Reduktion des Speicherbedarfs.
  - Wiederverwendbarkeit des vortrainierten Modells.

Low-Rank Adaptation (LoRA): Grundidee

- **Motivation:**
  - Große Sprachmodelle haben Milliarden von Parametern.
  - LoRA reduziert die Anzahl der zu trainierenden Parameter durch Low-Rank-Matrizen.
- **Ansatz:**
  - Zerlegung der Gewichtsmatrix W in zwei Low-Rank-Matrizen A und B:

$$W' = W + \Delta W, \quad \Delta W = AB$$

  - $A \in \mathbb{R}^{d \times r}$, $B \in \mathbb{R}^{d \times r}$, wobei $r \ll d$.
- **Vorteile:**
  - Reduktion der Speicher- und Rechenkosten.
  - Effizientes Fine-Tuning ohne Änderung der Hauptgewichte W.

Low-Rank Adaptation (LoRA): Mathematische Details

- **Modellanpassung:**
  - Gegeben eine Gewichtsmatrix W, wird die Anpassung ΔW durch:

    $$\Delta W = AB$$

    berechnet, wobei A und B trainierbar sind.
- **Optimierung:**
  - Ziel: Minimierung des Verlusts L über A und B:

    $$\min_{A,B} L(f(x; W + AB), y)$$
- **Effizienz:**
  - Speicherbedarf: $O(r \cdot (d + d))$, wobei $r \ll d$.
  - Rechenaufwand: Geringer als vollständiges Fine-Tuning.

Vergleich: PEFT, LoRA und Full Fine-Tuning

Eigenschaft	PEFT	LoRA	Full Fine-Tuning
Speicherbedarf	Gering	Sehr gering	Hoch
Rechenaufwand	Mittel	Niedrig	Sehr hoch
Flexibilität	Hoch	Mittel	Sehr hoch
Anwendungsfälle	Allgemein	Speziell für LLMs	Universell

Tabelle: Vergleich von PEFT, LoRA und Full Fine-Tuning.

Fine-Tuning Frameworks

- **Hugging Face Transformers:**
  - Umfangreiche Bibliothek für vortrainierte Modelle.
  - Unterstützt einfache Anpassung und Fine-Tuning.
  - `https://github.com/huggingface/transformers`
- **OpenAI Fine-Tuning API:**
  - Ermöglicht das Fine-Tuning von GPT-Modellen.
  - Einfache Integration in bestehende Anwendungen.
  - `https://platform.openai.com/docs/guides/fine-tuning`
- **PyTorch Lightning:**
  - Framework für vereinfachtes Training und Fine-Tuning.
  - Unterstützt verteiltes Training und Mixed Precision.
  - `https://www.pytorchlightning.ai/`
- **LoRA (Low-Rank Adaptation):**
  - Effizientes Fine-Tuning durch Reduktion der Parameteranzahl.
  - Besonders geeignet für ressourcenbeschränkte Umgebungen.
  - `https://github.com/microsoft/LoRA`

GPT-2: Verbesserungen gegenüber GPT

- **Größere Modelle:** GPT-2 wurde in verschiedenen Größen veröffentlicht (117M, 345M, 762M, 1.5B Parameter).
- **Training auf größeren Datenmengen:** GPT-2 wurde auf einem breiteren und vielfältigeren Korpus trainiert.
- **Verbesserte Textgenerierung:** GPT-2 erzeugt kohärentere und längere Texte.
- **Anwendungen:** Textzusammenfassung, Übersetzung, Dialogsysteme.

GPT-3: Skalierung und Few-Shot Learning

- **Skalierung:** GPT-3 hat 175 Milliarden Parameter, was es zu einem der größten Modelle macht.
- **Few-Shot Learning:** Kann Aufgaben mit wenigen Beispielen im Prompt lösen.
- **Anwendungen:** Codegenerierung, kreative Textgenerierung, komplexe Dialoge.
- **Herausforderungen:** Hoher Rechenaufwand, Bias in den generierten Texten.

GPT-4: Multimodalität und Verbesserungen

- **Multimodalität:** GPT-4 kann sowohl Text als auch Bilder als Eingabe verarbeiten.
- **Verbesserte Genauigkeit:** Bessere Leistung bei komplexen Aufgaben und längeren Kontexten.
- **Anwendungen:** Bildbeschreibung, multimodale Dialogsysteme.
- **Herausforderungen:** Noch höhere Anforderungen an Rechenressourcen.

GPT-4: Multimodale Verarbeitung

- **Multimodalität:** GPT-4 kann sowohl Text als auch Bilder als Eingabe verarbeiten.
- **Architektur:**
  - Erweiterung der Transformer-Architektur, um visuelle und textuelle Daten zu integrieren.
  - Gemeinsamer latent space für Text- und Bildrepräsentationen.
- **Anwendungen:**
  - Bildbeschreibung: Generierung von Texten basierend auf Bildern.
  - Visuelle Fragebeantwortung: Beantwortung von Fragen zu einem Bild.
  - Multimodale Dialogsysteme: Kombination von Text- und Bildinformationen in Konversationen.

GPT-4: Verarbeitung von Bildern und Text

- **Bildverarbeitung:**
  - Bilder werden durch ein visuelles Encoder-Modul (z. B. CNN oder Vision Transformer) in Features umgewandelt.
  - Die Features werden in den Transformer integriert.
- **Textverarbeitung:**
  - Text wird wie in GPT-3 tokenisiert und in Embeddings umgewandelt.
  - Gemeinsame Verarbeitung mit Bild-Features im Transformer.

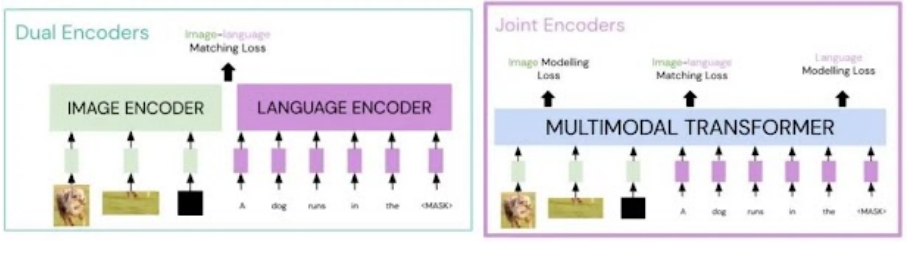


Abbildung: Multimodale Verarbeitung in Language models[18]

[18] https://lh3.googleusercontent.com/UGw_gUAQ0ja_28B-s-o0othiI2rmsIO2WJ_48s0xapPIfYJwK-pof8TgOqwnQwI0h4t6-RUw6saGjjWUDpqC224WwIPnnpiBfqa5fLiKbsURczSwDGw=w616

GPT-4: Herausforderungen und Vorteile

- **Herausforderungen:**
  - Integration von Bild- und Textdaten in einem Modell.
  - Hoher Rechenaufwand für Training und Inferenz.
  - Bedarf an großen multimodalen Datensätzen.
- **Vorteile:**
  - Verbesserte Kontextverständnis durch Kombination von Text und Bild.
  - Breitere Anwendungsbereiche, z. B. in der Medizin, Bildung und Unterhaltung.
  - Fortschritt in multimodalen KI-Systemen.

LLaMA: Open-Weight Modelle

- **LLaMA (Large Language Model Meta AI):** Entwickelt von Meta AI, mit Fokus auf Effizienz und Zugänglichkeit.
- **Open-Weight Modelle:** Verfügbar für die Forschungsgemeinschaft.
- **Größen:** Modelle mit 7B, 13B, 30B und 70B Parametern.
- **Anwendungen:** Forschung, Entwicklung von spezialisierten LLMs.

Einführung in Reasoning-Modelle

- Reasoning-Modelle zielen darauf ab, logisches Denken und Schlussfolgerungen zu ermöglichen.
- Fokus auf komplexe Aufgaben wie mathematische Beweise, logische Schlussfolgerungen und Multi-Hop-Fragen.
- Beispiele: DeepSeek-r1, GPT4o, DeepMind Gemini, Anthropic Claude.

DeepSeek-r1: Überblick

- **DeepSeek-r1:** Ein neuartiges Reasoning-Modell, das auf Transformer-Architekturen basiert.
- **Ziele:**
  - Integration von logischem Denken in Sprachmodelle.
  - Dieses Reasoning imitiert menschliches Denken über Generierung von Tokens
  - Verarbeitung von Multi-Hop-Reasoning-Aufgaben.
  - Unterstützung von domänenspezifischen Schlussfolgerungen.
- **Besonderheiten:**
  - Hybrid-Architektur mit dedizierten Reasoning-Modulen.
  - Nutzung von Memory-Augmented Mechanismen.

DeepSeek-r1: Architektur

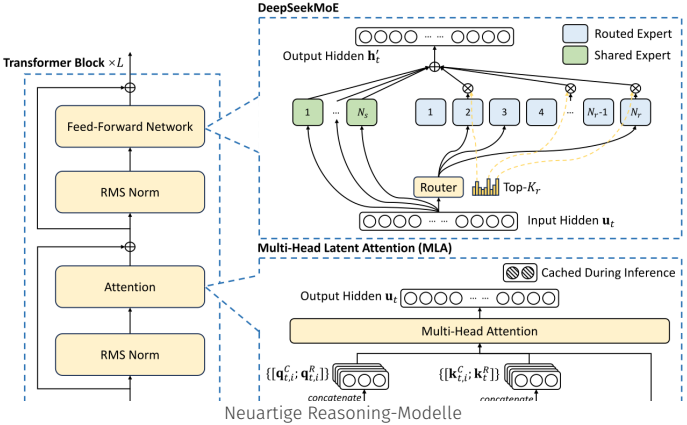
- **Core-Komponenten:**
  - **Reasoning-Module:** Spezialisierte Submodule für logische Schlussfolgerungen.
  - **Memory-Augmented Attention:** Ermöglicht Zugriff auf externe Wissensquellen.
  - **Multi-Hop-Mechanismus:** Iterative Verarbeitung von Informationen.
- **Pipeline:**
  1. Eingabe wird tokenisiert und in Embeddings umgewandelt.
  2. Reasoning-Module führen ”logische” Operationen durch.
  3. Ergebnisse werden iterativ verfeinert.
  4. Ausgabe erfolgt als Schlussfolgerung oder Antwort.

Überblick DeepSeek-R1

- **Ziel:** Maximale Ausnutzung von Test-Time Computation zur Förderung von Ketten- bzw. Chain-of-Thought (CoT) Reasoning.
- DeepSeek-R1 erreicht ähnliche Leistungen wie GPT-o1, jedoch mit deutlich geringeren Trainingskosten.
- Einsatz von Reinforcement Learning (RL) als dominanter Post-Training-Ansatz, um komplexe Reasoning-Aufgaben (Mathematik, Code, wissenschaftliches Denken) zu meistern.

Architektur von DeepSeek-R1

- Basierend auf einem DeepSeek-V3-Base Checkpoint.
- Nutzt **Mixture-of-Expert (MoE)**-Strukturen, Multi-Head Latent Attention (MLA) und Multi-Token Prediction (MTP).
- Zwei Varianten:
  - **DeepSeek-R1-Zero:** Post-Training ausschließlich mit RL.
  - **DeepSeek-R1:** Mehrstufiger Post-Training-Prozess, der zusätzlich Supervised Fine-Tuning (SFT) integriert.



Proximal Policy Optimization (PPO) - Teil 1

- **Definition:** PPO ist ein Reinforcement-Learning-Algorithmus, der die Policy-Gradient-Methode verbessert.
- **Ziel:** Maximierung der kumulierten Belohnung durch Optimierung der Policy π_θ.
- **Kernidee:** Begrenzung der Policy-Änderungen, um Stabilität und Effizienz zu gewährleisten.
- **Details:** `https://arxiv.org/pdf/2402.03300`

PPO-Zielfunktion:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

Proximal Policy Optimization (PPO) - Teil 2

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: Verhältnis der neuen zur alten Policy.
- $\hat{A}_t$: Vorteil (Advantage) zur Zeit t.
- ϵ: Hyperparameter zur Begrenzung der Policy-Änderung.

Vorteile von PPO:

- Stabilität durch Clipping der Policy-Änderungen.
- Einfache Implementierung und gute Leistung in verschiedenen RL-Aufgaben.

GRPO – Group Relative Policy Optimization (Teil 1)

- DeepSeek-R1 setzt auf eine modifizierte Version von PPO namens **Group Relative Policy Optimization (GRPO)**.
- Ziel: Steigerung der mathematischen Reasoning-Fähigkeiten bei reduziertem Speicherverbrauch.

GRPO Objective:

Die GRPO-Ziel-Funktion basiert auf der PPO-Formulierung:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right]$$

wobei:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ ist der Wahrscheinlichkeitsquotient.
- $\hat{A}_t$ bezeichnet den Vorteil (Advantage) zur Zeit t.
- ϵ ist ein Hyperparameter zur Begrenzung der Änderung.

GRPO – Group Relative Policy Optimization (Teil 2)

Gruppenbezogene Anpassungen:

GRPO erweitert die PPO-Zielfunktion, indem es gruppenspezifische Gewichtungen einführt:

$$L^{\text{GRPO}}(\theta) = \mathbb{E}_{t,g}\left[\min\left(r_{t,g}(\theta)\hat{A}_{t,g},\ \text{clip}\left(r_{t,g}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{t,g}\right)\right]$$

wobei:

- g die Gruppe repräsentiert, zu der die Aufgabe gehört.
- $r_{t,g}(\theta) = \frac{\pi_\theta(a_t|s_t,g)}{\pi_{\theta_{\text{old}}}(a_t|s_t,g)}$ ist der gruppenspezifische Wahrscheinlichkeitsquotient.
- $\hat{A}_{t,g}$ ist der gruppenspezifische Vorteil (Advantage).

Diese Erweiterung ermöglicht es, die Policy-Optimierung an die spezifischen Anforderungen und Eigenschaften verschiedener Gruppen anzupassen, wodurch die Leistung in heterogenen Aufgabenbereichen verbessert wird.

GRPO – Group Relative Policy Optimization (Teil 3)

- **Visualisierung:** Die folgende Abbildung illustriert die Funktionsweise von GRPO.
- **Schlüsselkonzepte:**
  - Gruppenspezifische Gewichtungen.
  - Begrenzung der Policy-Änderungen.
  - Iterative Optimierung für verschiedene Gruppen.

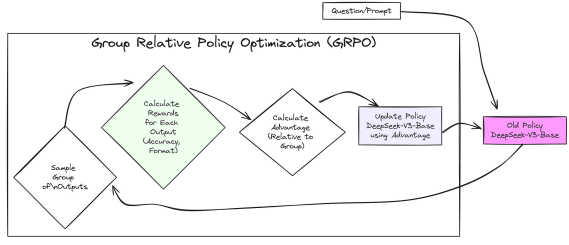


Abbildung: Illustration der GRPO-Mechanik.

Reward-Modellierung

- **Regelbasierte Rewards:**
  - **Accuracy Reward:** Bewertet, ob die Antwort korrekt ist.
  - **Format Reward:** Erzwingt, dass der Denkprozess in `<think>` und `</think>` Tags eingeschlossen wird.
- Kein neutral trainierter Reward Model, da solche Modelle anfällig für Reward Hacking sind.

Kombinierter Reward:

$$R_{\text{total}} = \alpha R_{\text{accuracy}} + \beta R_{\text{format}}$$

- α und β sind Gewichtungsfaktoren, die den Einfluss der jeweiligen Komponenten steuern.

Mehrstufiger Trainingsprozess von DeepSeek-R1

1. **Phase 1: Cold-Start SFT**
   - Erste Supervised Fine-Tuning (SFT) Phase, bei der Labels durch wenige Beispiele von R1-Zero generiert und von Menschen verfeinert werden.
2. **Phase 2: Reinforcement Learning**
   - Anwendung von GRPO zur Optimierung der Reasoning-Fähigkeiten.
   - Mathematische Zielsetzung zur Maximierung der Test-Time Computation: Der durchschnittliche Antwortlänge-Wert steigt, was eine tiefergehende Ketten-Denke (Chain-of-Thought) anzeigt.
3. **Phase 3: Weitere SFT**
   - Integration von Daten aus weiteren Domänen zur Verbesserung von Schreibstil, Rollenspiel und allgemeinen Aufgaben.
4. **Phase 4: Sekundäres RL**
   - RL für alle Szenarien zur Steigerung der Hilfsbereitschaft und Harmlosigkeit.

Chain-of-Thought und Test-Time Computation

- DeepSeek-R1 nutzt **Chain-of-Thought (CoT)**-Reasoning:
  - Zuerst wird ein ausführlicher Denkprozess (innerhalb von `<think>` ... `</think>` Tags) generiert.
  - Anschließend wird die finale Antwort produziert.
- Mathematische Modellierung dieser Phase kann als iterative Optimierung über Teilschritte betrachtet werden, z.B.:

$$\mathbf{c}_{t+1} = f(\mathbf{c}_t, \Delta_t(\mathbf{x}))$$

  wobei $\mathbf{c}_t$ den aktuellen CoT-Zustand und $\Delta_t(\mathbf{x})$ den Beitrag des nächsten Tokens bzw. der nächsten Denkschritt repräsentiert.
- Durch längeres Denken bei steigender Testzeit skaliert die Modellleistung im Sinne der "Test-time Scaling Law".

Zusammenfassung

- DeepSeek-R1 demonstriert, wie Reinforcement Learning (GRPO) und regelbasierte Reward-Strategien genutzt werden können, um komplexe reasoning-Aufgaben zu bewältigen.
- Der mehrstufige Trainingsprozess (SFT → RL → SFT → RL) verbessert sowohl die Genauigkeit als auch die Sprachkohärenz.
- Die mathematische Grundlage der GRPO-Ziel-Funktion und der Reward-Modellierung zeigen, wie Optimierungsziele systematisch in den Trainingsprozess integriert werden.

DeepSeek-R1 zeigt: Mit reduzierten Trainingskosten und innovativen Trainingsansätzen sind moderne Reasoning-Modelle möglich.

Zusammenfassung: DeepSeek-r1 und LLM-Architekturen

- **DeepSeek-r1:**
  - Hervorragende Reasoning-Fähigkeiten und Multi-Hop-Reasoning.
  - Zugriff auf externen Speicher für komplexe Schlussfolgerungen.
  - Anwendungen: Logik, Beweise, domänenspezifische Aufgaben.
- **Vergleich von LLMs:**
  - GPT-2: Verbesserte Textgenerierung, Anwendungen wie Textzusammenfassung.
  - GPT-3: 175B Parameter, Few-Shot Learning, z. B. Codegenerierung.
  - GPT-4: Multimodalität (Text und Bild), Anwendungen wie Bildbeschreibung.
  - LLaMA: Open-Weight Modelle (7B-70B Parameter), Fokus auf Forschung.
  - DeepSeek-v3: Fortschrittliche Reasoning- und Domänenanpassungsfähigkeiten, Anwendungen in Wissenschaft und Technik.

Und was machen wir jetzt mit diesen Systemen?

- Praktische Anwendungen und Integration in bestehende Systeme
- Herausforderungen bei der Implementierung und Skalierung
- Gesellschaftliche und ethische Implikationen

Nutzungsmöglichkeiten: RAG, Agentensysteme

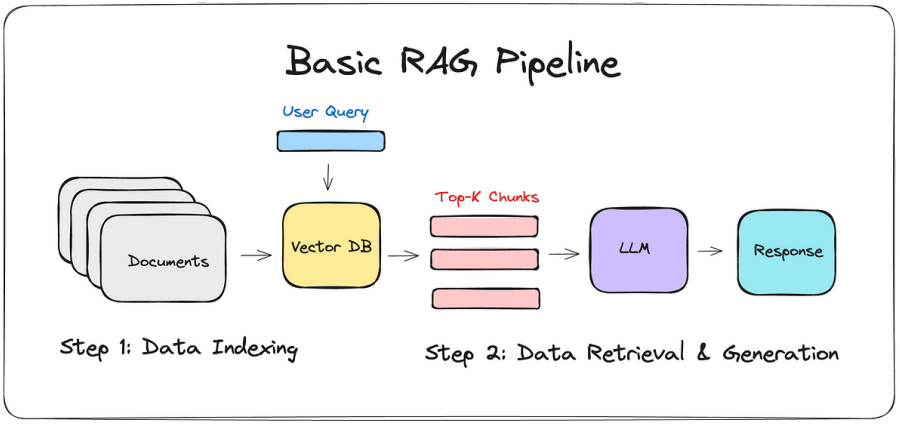
- Retrieval-Augmented Generation (RAG) und seine Anwendungen
- Entwicklung und Einsatz von Agentensystemen im NLP
- Kombination von LLMs mit externem Wissen

Was ist Retrieval-Augmented Generation (RAG)?

- **Definition:** Kombination von Retrieval-Systemen und generativen Modellen.
- **Ziel:** Verbesserung der Antwortqualität durch Zugriff auf externe Wissensquellen.
- **Funktionsweise:**
  - Abruf relevanter Dokumente aus einer Wissensdatenbank.
  - Nutzung der abgerufenen Informationen zur Generierung von Antworten.
- **Anwendungen:** Fragebeantwortung, Chatbots, Dokumentensuche.

RAG: Architektur

- **Retriever:**
  - Abruf relevanter Dokumente aus einer Wissensdatenbank.
  - Nutzung von Suchalgorithmen und Vektorraumsmodellen.



RAG: Vorteile

- **Verbesserte Genauigkeit:** Zugriff auf externe Wissensquellen reduziert Halluzinationen.
- **Flexibilität:** Kann mit verschiedenen Retrieval- und Generationsmodellen kombiniert werden.
- **Aktualität:** Ermöglicht die Nutzung aktueller Informationen aus einer Wissensdatenbank.
- **Erweiterbarkeit:** Einfaches Hinzufügen neuer Wissensquellen.

RAG: Herausforderungen

- **Effizienz:** Abruf und Generierung können rechenintensiv sein.
- **Schnittstellen:** Legacy-Systeme und APIs müssen integriert werden.
- **Access Rights Management:** Sicherstellen, dass nur autorisierte Informationen abgerufen werden.
- **Qualität der Dokumente:** Die Genauigkeit hängt von der Qualität der abgerufenen Dokumente ab.
- **Konsistenz:** Inkonsistenzen zwischen abgerufenen Informationen und generierten Antworten.
- **Bias:** Bias in der Wissensdatenbank können die Antworten beeinflussen.

RAG: Anwendungen

- **Fragebeantwortung:** Beantwortung komplexer Fragen durch Abruf relevanter Informationen.
- **Chatbots:** Verbesserung der Konversationsqualität durch Zugriff auf externe Daten.
- **Dokumentensuche:** Generierung von Zusammenfassungen basierend auf abgerufenen Dokumenten.
- **Wissenschaftliche Recherche:** Unterstützung bei der Suche nach relevanter Literatur.

Was ist Document Embedding in RAG Pipelines?

- **Definition:** Repräsentation eines gesamten Dokuments/Chunks als Vektor in einem kontinuierlichen Vektorraum, speziell für Retrieval-Augmented Generation (RAG).
- **Ziel:** Ermöglichen eines effizienten Abrufs relevanter Dokumente/Chunks aus einer Vektordatenbank.
- **Anwendungen in RAG:**
  - Verbesserung der Antwortqualität durch Zugriff auf relevante Dokumente.
  - Unterstützung von Fragebeantwortung und Dokumentensuche.
  - Integration von externem Wissen in generative Modelle.

Strategien für Document Embedding in RAG Pipelines

- **Transformer-basierte Modelle:**
  - Nutzung von Modellen wie Sentence-BERT, OpenAI Embeddings oder ähnliche.
  - CLS-Token oder Mittelung der Token-Embeddings für die Dokumentrepräsentation.
  - Vorteile: Kontextabhängige und semantisch reichhaltige Repräsentationen.
- **Vektordatenbanken:**
  - Speicherung der Dokumentvektoren in spezialisierten Datenbanken wie Pinecone, Weaviate oder Milvus.
  - Ermöglichen schnellen Abruf durch Ähnlichkeitssuche (z. B. k-NN, cosine similarity).
- **Hybrid-Ansätze:**
  - Kombination von klassischen Retrieval-Methoden (z. B. BM25) mit Vektorbasierter Suche.
  - Verbesserung der Präzision durch Kombination von Semantik und Schlüsselwortsuche.

Einbettung in Vektordatenbanken für RAG Pipelines

- **Pipeline:**
  1. Dokumente werden vorverarbeitet und in Vektoren eingebettet.
  2. Die Vektoren werden in einer Vektordatenbank gespeichert.
  3. Bei einer Anfrage wird der Eingabetext ebenfalls eingebettet.
  4. Ähnlichkeitssuche in der Vektordatenbank liefert relevante Dokumente.
  5. Die abgerufenen Dokumente werden als Kontext für die Generierung verwendet.
- **Vorteile:**
  - Effiziente Suche in großen Wissensbasen.
  - Kontextualisierte Antworten durch semantische Relevanz.
  - Skalierbarkeit für umfangreiche Datenmengen.

Hybride Suche: BM25 und Vektorähnlichkeiten (Teil 1)

- **Definition:** Kombination von klassischen Suchmethoden (BM25) und vektorbasierter Ähnlichkeitssuche.
- **BM25:**
  - Klassischer Algorithmus für die Schlüsselwortsuche.
  - Bewertet die Relevanz eines Dokuments basierend auf Termfrequenz (TF) und inverser Dokumentfrequenz (IDF).
  - Formel:

$$\text{BM25}(q, d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})}$$

    wobei $f(t, d)$ die Häufigkeit des Terms t im Dokument d ist.
- **Vektorähnlichkeiten:**
  - Repräsentiert Dokumente und Anfragen als Vektoren in einem kontinuierlichen Raum.
  - Nutzt Metriken wie Kosinus-Ähnlichkeit oder euklidische Distanz zur Bewertung der Relevanz.

Hybride Suche: BM25 und Vektorähnlichkeiten (Teil 2)

- **Kombination:**
  - BM25 liefert eine gewichtete Bewertung basierend auf Schlüsselwörtern.
  - Vektorähnlichkeiten ergänzen die Suche durch semantische Relevanz.
  - Hybride Bewertung:

$$\text{Score}_{\text{hybrid}} = \alpha \cdot \text{BM25}(q, d) + \beta \cdot \text{Similarity}(q, d)$$

    wobei α und β Gewichtungsfaktoren sind.
- **Vorteile:**
  - Präzision durch BM25 für Schlüsselwortsuche.
  - Semantische Tiefe durch Vektorähnlichkeiten.
  - Flexibilität für verschiedene Anwendungsfälle.
- **Anwendungen:**
  - Dokumentensuche in großen Datenbanken.
  - Fragebeantwortungssysteme mit externem Wissen.
  - Kombination von strukturierten und unstrukturierten Daten.

Vergleich der Strategien für RAG Pipelines

Methode	Vorteile	Nachteile
BM25	Schnell, etabliert	Keine Semantik
Transformer-Modelle	Kontext- und Semantikreich	Hoher Rechenaufwand
Vektordatenbanken	Effiziente Ähnlichkeitssuche	Speicherbedarf
Hybrid-Ansätze	Kombination von Präzision und Semantik	Komplexität

Tabelle: Vergleich verschiedener Strategien für RAG Pipelines.

Was ist GraphRAG?

- **Definition:** Erweiterung von Retrieval-Augmented Generation (RAG) durch die Nutzung von Graphdatenbanken und **Ontologien** als Wissensquelle.
- **Ziel:** Verbesserung der Kontextualisierung und Präzision durch explizite Modellierung von Beziehungen und Konzepten.
- **Vorteile:**
  - Explizite Repräsentation von Entitäten, Beziehungen und Konzepten (durch Ontologien).
  - Ermöglicht komplexe Abfragen, Reasoning und Inferenz über das Wissen.
  - Bessere Nachvollziehbarkeit und Erklärbarkeit der Antworten.
- **Anwendungen:** Fragebeantwortung, wissensbasierte Chatbots, wissenschaftliche Recherche, Medizin, Recht, Industrie 4.0.

Ontologien: Strukturierte Wissensrepräsentation

- **Definition:** Eine Ontologie ist eine formale, maschinenlesbare Beschreibung von Konzepten, deren Eigenschaften und Beziehungen in einem bestimmten Domänenbereich.
- **Beispiel:**
  - **Medizin:** SNOMED CT, ICD-10 (Krankheiten, Symptome, Therapien)
  - **Wissenschaft:** Gene Ontology (GO), Chemical Entities of Biological Interest (ChEBI)
  - **Industrie:** Industrie 4.0 Asset Administration Shell (AAS) Ontologie
- **Nutzen:**
  - Standardisierte Begriffe und Beziehungen
  - Unterstützung von Inferenz und semantischer Suche
  - Interoperabilität zwischen Systemen

Wissenseinbettung in Graphdatenbanken und Ontologien

- **Graphdatenbank:** Speichert Wissen als Knoten (Entitäten/Konzepte) und Kanten (Beziehungen), oft auf Basis einer Ontologie.
- **Einbettung (Embedding):**
  - Jeder Knoten und jede Kante erhält einen Vektor im kontinuierlichen Raum.
  - Methoden: Node2Vec, GraphSAGE, TransE, GNN-basierte Ansätze.
- **Pipeline:**
  1. Extraktion von Entitäten, Relationen und Konzepten aus Text (z. B. mit NLP und Ontologie-Mapping).
  2. Aufbau des Wissensgraphen in einer Graphdatenbank (z. B. Neo4j, TigerGraph) unter Nutzung einer Ontologie.
  3. Berechnung von Embeddings für Knoten/Kanten.
  4. Retrieval relevanter Subgraphen/Konzepte als Kontext für LLMs.

Beispiel 1: Medizinische Fragebeantwortung mit Ontologie

- **Ontologie:** SNOMED CT (medizinische Begriffe und Relationen)
- **Frage:** "Welche Therapien gibt es für Diabetes Typ 2?"
- **Ablauf:**
  1. LLM erkennt Entität "Diabetes Typ 2" und mapped sie auf die Ontologie.
  2. GraphRAG sucht im Wissensgraphen nach Knoten "Diabetes Typ 2" und allen Kanten "behandelt mit".
  3. Die gefundenen Therapien werden als strukturierter Kontext an das LLM übergeben.
  4. LLM generiert eine nachvollziehbare, medizinisch fundierte Antwort.
- **Vorteil:** Medizinische Präzision, Nachvollziehbarkeit, Nutzung von Expertenwissen.

Beispiel 2: Industrie 4.0 – Asset-Verwaltung mit Ontologie

- **Ontologie:** Asset Administration Shell (AAS) Ontologie
- **Frage:** "Welche Maschinen sind Teil der Fertigungslinie X und wann ist die nächste Wartung?"
- **Ablauf:**
  1. LLM mapped "Fertigungslinie X" auf die Ontologie.
  2. GraphRAG sucht alle Maschinen (Knoten) mit Relation "ist Teil von" zu "Fertigungslinie X".
  3. Für jede Maschine wird die Relation "nächste Wartung" abgefragt.
  4. LLM gibt eine strukturierte Übersicht aus.
- **Vorteil:** Transparenz, Automatisierung, Integration von Echtzeitdaten.

GraphRAG: Architekturüberblick

- **Komponenten:**
  - **Graphdatenbank/Ontologie:** Speicherung und Abfrage von Wissensgraphen und Konzepten.
  - **Graph Embedding-Modell:** Erzeugt Vektorrepräsentationen für Knoten/Kanten/Subgraphen.
  - **Retriever:** Findet relevante Subgraphen/Konzepte zu einer Anfrage.
  - **LLM:** Nutzt die abgerufenen Graph- und Ontologiekontexte zur Antwortgenerierung.
- **Ablauf:**
  1. Anfrage wird in Entitäten/Relationen/Konzepte zerlegt (NLP + Ontologie-Mapping).
  2. Passende Subgraphen/Konzepte werden per Embedding-Ähnlichkeit oder Graphabfrage gefunden.
  3. Kontext wird an das LLM übergeben.
  4. LLM generiert Antwort unter Nutzung des strukturierten Wissens.

Vergleich: Klassisches RAG vs. GraphRAG mit Ontologien

Eigenschaft	Klassisches RAG	GraphRAG mit Ontologie
Wissensquelle	Dokumente/Chunks	Wissensgraph/Ontologie
Kontextabruf	Vektorähnlichkeit	Graphabfragen + Embeddings + Inferenz
Beziehungen	Implizit im Text	Explizit und semantisch modelliert
Reasoning	Eingeschränkt	Komplexe Pfad-, Beziehungs- und Konzeptabfragen
Erklärbarkeit	Gering	Sehr hoch
Domänenwissen	Unstrukturiert	Standardisiert, interoperabel

Tabelle: Vergleich von klassischem RAG und GraphRAG mit Ontologien.

Herausforderungen und Ausblick für GraphRAG mit Ontologien

- **Herausforderungen:**
  - Extraktion und Aktualisierung von Entitäten/Relationen/Konzepten aus Text.
  - Skalierbare Berechnung und Speicherung von Graph- und Ontologie-Embeddings.
  - Effiziente Subgraph- und Konzept-Retrieval-Algorithmen.
  - Integration von Ontologiekontext in LLM-Prompts.
  - Pflege und Erweiterung von Ontologien.
- **Ausblick:**
  - Kombination von GraphRAG mit multimodalen Wissensquellen (Text, Bild, Sensorik).
  - Automatisierte Ontologie-Generierung und -Aktualisierung.
  - Verbesserte Reasoning-Fähigkeiten durch strukturierte, domänenspezifische Kontextbereitstellung.
  - Einsatz in kritischen Bereichen wie Medizin, Recht, Industrie.

Was sind komplexe Agentensysteme?

- **Definition:** Systeme, die autonome Agenten nutzen, um komplexe Aufgaben durch Interaktion mit Tools, Ontologien und Umgebungen zu lösen.
- **Merkmale:**
  - Autonomie: Agenten agieren teilweise unabhängig in einem vorgegebenen Rahmen.
  - Tool-Nutzung: Zugriff auf externe APIs, Datenbanken, Ontologien oder Software.
  - Multi-Agent-Koordination: Zusammenarbeit mehrerer Agenten.
- **Anwendungen:** Wissenschaftliche Forschung, Automatisierung, Problemlösung, Wissensmanagement.

Architektur eines komplexen Agentensystems

- **Hauptkomponenten:**
  - **Agenten:** Autonome Einheiten mit spezifischen Fähigkeiten.
  - **Tool-Interface:** Ermöglicht den Zugriff auf externe Tools wie APIs, Datenbanken, Ontologien oder Rechenressourcen.
  - **Kommunikationsmodul:** Ermöglicht den Austausch zwischen Agenten.
  - **Planungs- und Entscheidungsmodul:** Koordiniert die Aktionen der Agenten.
- **Workflow:**
  1. Eingabe einer Aufgabe durch den Benutzer.
  2. Agenten analysieren die Aufgabe, nutzen ggf. Ontologien zur Wissensstrukturierung und planen die Lösung.
  3. Tools und Ontologien werden genutzt, um Teilaufgaben zu lösen.
  4. Ergebnisse werden aggregiert und präsentiert.

Beispiel: Multi-Agentensystem für medizinische Diagnose

- **Ziel:** Automatisierte Unterstützung bei der medizinischen Diagnose.
- **Agentenrollen:**
  - **Symptom-Analyseagent:** Extrahiert Symptome aus Patientendaten.
  - **Ontologie-Agent:** Nutzt medizinische Ontologien (z. B. SNOMED CT), um Symptome mit möglichen Diagnosen zu verknüpfen.
  - **Literaturagent:** Sucht nach aktuellen Studien zu den gefundenen Diagnosen.
  - **Berichtsagent:** Generiert einen strukturierten Diagnosebericht.
- **Tool-Nutzung:**
  - Zugriff auf medizinische Datenbanken und Ontologien.
  - Nutzung von NLP und LLMs zur Kontextanreicherung.
- **Vorteile:** Präzision, Nachvollziehbarkeit, Integration von Expertenwissen.

Beispiel: Multi-Agentensystem für wissenschaftliche Forschung

- **Ziel:** Automatisierte Literaturrecherche und Datenanalyse.
- **Agentenrollen:**
  - **Suchagent:** Durchsucht Datenbanken nach relevanten Artikeln.
  - **Beispiele für Datenbanken:** PubMed, ArXiv, Semantic Scholar, SpringerLink, IEEE Xplore.
  - **Ontologie-Agent:** Ordnet gefundene Artikel zu Konzepten einer wissenschaftlichen Ontologie (z. B. Gene Ontology).
  - **Analyseagent:** Führt Analysen zu den gefundenen Dokumenten durch.
  - **Berichtsagent:** Generiert Zusammenfassungen und Berichte.
- **Tool-Nutzung:**
  - Zugriff auf APIs, Ontologien und LLMs.
  - Nutzung von Python-Bibliotheken wie Pandas oder Matplotlib.
- **Vorteile:** Effizienzsteigerung, strukturierte Ergebnisse, semantische Suche.

Herausforderungen bei komplexen Agentensystemen

- **Koordination:** Synchronisation zwischen mehreren Agenten.
- **Tool- und Ontologie-Integration:** Kompatibilität mit verschiedenen APIs, Ontologien und Software.
- **Fehlerbehandlung:** Umgang mit Ausfällen oder unvorhergesehenen Ereignissen.
- **Skalierbarkeit:** Effizienz bei wachsender Anzahl von Agenten oder Aufgaben.
- **Sicherheit:** Schutz vor Missbrauch oder fehlerhaften Aktionen.
- **Ontologiepflege:** Aktualisierung und Erweiterung der Wissensbasis.

Was ist Tool Calling mit MCP-Servern?

- **Tool Calling:** LLMs rufen externe Tools oder APIs auf, um Aufgaben zu lösen, die über reine Textgenerierung hinausgehen.
- **MCP-Server (Multi-Component Platform):** Vermittlungsinstanz, die Anfragen von LLMs entgegennimmt, an spezialisierte Tools weiterleitet und die Ergebnisse zurückgibt.
- **Ziel:** Erweiterung der Fähigkeiten von LLMs durch strukturierte, sichere und skalierbare Tool-Nutzung.

Model Context Protocol (MCP): Überblick

- **Definition:** Das Model Context Protocol (MCP) ist ein standardisiertes Protokoll zur strukturierten Kommunikation zwischen LLMs, Agenten und externen Tools.
- **Ziel:** Ermöglicht die Übergabe von Kontext, Aufgaben, Tool-Aufrufen und Ergebnissen in einem maschinenlesbaren Format (z. B. JSON, YAML).
- **Kernfunktionen:**
  - **Kontextübergabe:** Übermittlung von Hintergrundwissen, Benutzeranfragen und Umgebungsdaten an das Modell.
  - **Tool-Calls:** Strukturierte Anforderung von externen Aktionen (z. B. Datenbankabfragen, API-Aufrufe).
  - **Antwortintegration:** Rückgabe von Tool-Ergebnissen und deren Einbindung in die Modellantwort.
  - **Status- und Fehlerhandling:** Standardisierte Rückmeldungen zu Erfolg, Fehlern und Zwischenergebnissen.
- **Beispielstruktur (JSON):**
  - `{"context": {}, "task": "...", "tool_calls": [...], "results": [...], "status": "ok"}`
- **Vorteile:** Interoperabilität, Nachvollziehbarkeit, Modularität und sichere Integration von LLMs in komplexe Agentensysteme.

Architektur: Tool Calling mit MCP-Servern

- **Komponenten:**
  - **LLM:** Erkennt, wann ein Tool-Aufruf nötig ist, und formuliert eine strukturierte Anfrage.
  - **MCP-Server:** Vermittelt zwischen LLM und Tools, verwaltet Authentifizierung, Logging und Fehlerbehandlung.
  - **Tools/Services:** Externe APIs, Datenbanken, Rechenmodule oder Ontologien.
- **Ablauf:**
  1. LLM generiert eine Tool-Call-Anfrage (z. B. im JSON-Format).
  2. MCP-Server empfängt die Anfrage, prüft und leitet sie an das passende Tool weiter.
  3. Tool verarbeitet die Anfrage und sendet das Ergebnis an den MCP-Server.
  4. MCP-Server gibt das Ergebnis an das LLM zurück, das es in die Antwort integriert.

Vorteile von Tool Calling mit MCP-Servern

- **Modularität:** Einfache Integration neuer Tools und Services.
- **Sicherheit:** Zentralisiertes Management von Zugriffsrechten und Monitoring.
- **Skalierbarkeit:** Parallele Verarbeitung mehrerer Anfragen und Lastverteilung.
- **Nachvollziehbarkeit:** Logging aller Tool-Calls für Audits und Debugging.
- **Domänenspezifische Erweiterbarkeit:** Einbindung von branchenspezifischen Tools und Ontologien.

Zukunftsperspektiven für Agentensysteme

- **Verbesserte Autonomie:** Einsatz von LLMs und Ontologien für flexiblere Entscheidungsfindung.
- **Erweiterte Tool- und Ontologie-Nutzung:** Integration von spezialisierten Tools und domänenspezifischen Ontologien.
- **Multi-Agent-Kollaboration:** Entwicklung von Protokollen für effizientere Zusammenarbeit (z. B. JSON, RDF).
- **Domänenspezifische Systeme:** Anpassung an spezifische Branchen wie Medizin, Recht, Industrie.
- **Automatisierte Ontologie-Generierung:** LLMs unterstützen beim Aufbau und der Pflege von Ontologien.

Zukunftsvisionen für LLMs

- **Verbesserte Multimodalität:** Integration von Text, Bild, Audio und Video in einem Modell.
- **Domänenspezifische Modelle:** Entwicklung spezialisierter LLMs für Medizin, Recht, Bildung und andere Bereiche.
- **Interaktive KI-Systeme:** Kombination von LLMs mit Robotik und IoT für physische Interaktionen.
- **Selbstlernende Systeme:** Modelle, die sich kontinuierlich durch Interaktion mit der Umgebung verbessern.
- **KI-gestützte Kreativität:** Unterstützung bei Kunst, Musik, Literatur und Design.

Gesellschaftliche Implikationen

- **Arbeitsmarkt:** Automatisierung von Aufgaben und mögliche Auswirkungen auf Beschäftigung.
- **Bildung:** Einsatz von LLMs als personalisierte Lernassistenten.
- **Privatsphäre:** Umgang mit sensiblen Daten und Schutz vor Missbrauch.
- **Regulierung:** Notwendigkeit von Gesetzen und Richtlinien für den verantwortungsvollen Einsatz von KI.
- **Ethik:** Sicherstellung, dass KI-Systeme menschliche Werte respektieren und fördern.

Diskussionspunkte für die Zukunft

- Wie können wir sicherstellen, dass LLMs inklusiv und fair sind?
- Welche Rolle sollten LLMs in der Entscheidungsfindung spielen?
- Wie können wir die Transparenz und Nachvollziehbarkeit von LLMs verbessern?
- Welche Maßnahmen sind notwendig, um Missbrauch zu verhindern?
- Wie können wir die Zusammenarbeit zwischen Mensch und KI optimieren?

KI und Ethik: Herausforderungen und Verantwortung

- **Verantwortung:** Sicherstellung, dass KI-Systeme im Einklang mit ethischen Prinzipien entwickelt und eingesetzt werden.
- **Herausforderungen:**
  - Bias und Diskriminierung in Trainingsdaten und Modellen.
  - Transparenz und Nachvollziehbarkeit von Entscheidungen.
  - Schutz der Privatsphäre und sensible Daten.
  - Verantwortung bei Fehlentscheidungen oder Missbrauch.
- **Gesellschaftliche Auswirkungen:**
  - Einfluss auf Arbeitsplätze und soziale Ungleichheit.
  - Förderung von Inklusion und Diversität.
  - Sicherstellung des Zugangs zu KI-Technologien für alle.

Ethische Prinzipien für KI

- **Fairness:** Vermeidung von Diskriminierung und Bias.
- **Transparenz:** Nachvollziehbarkeit von Entscheidungen und Prozessen.
- **Privatsphäre:** Schutz persönlicher Daten und Minimierung von Überwachung.
- **Sicherheit:** Verhinderung von Missbrauch und Sicherstellung der Robustheit.
- **Verantwortlichkeit:** Klare Zuständigkeiten für die Entwicklung und den Einsatz von KI.

Maßnahmen zur Förderung ethischer KI

- **Regulierung:** Einführung von Gesetzen und Richtlinien für den verantwortungsvollen Einsatz von KI.
- **Audits:** Regelmäßige Überprüfung von Modellen auf Bias und Fairness.
- **Bildung:** Förderung des Bewusstseins für ethische Fragen bei Entwicklern und Nutzern.
- **Interdisziplinäre Zusammenarbeit:** Einbindung von Experten aus Ethik, Recht und Sozialwissenschaften.
- **Open Source:** Transparenz durch Veröffentlichung von Modellen und Trainingsdaten.

Diskussionspunkte zu KI und Ethik

- Wie können wir sicherstellen, dass KI-Systeme fair und inklusiv sind?
- Welche Verantwortung tragen Entwickler und Unternehmen für die Auswirkungen von KI?
- Wie können wir den Missbrauch von KI-Technologien verhindern?
- Welche Rolle sollte die Regulierung bei der Entwicklung und dem Einsatz von KI spielen?
- Wie können wir ethische Prinzipien in den Entwicklungsprozess integrieren?

Zusammenfassung der LLM-Veranstaltung

- **Einführung in NLP und LLMs:** Grundlagen, Herausforderungen und Anwendungen.
- **Sprachdarstellung:** Von One-Hot-Encoding zu modernen Embeddings wie Word2Vec, GloVe und BERT.
- **Transformer-Architektur:** Self-Attention, Encoder-Decoder-Struktur und Vorteile gegenüber RNNs.
- **Fortgeschrittene Modelle:** BERT, GPT, DeepSeek-v3 und ihre spezifischen Stärken.
- **Praktische Anwendungen:** RAG, Agentensysteme und multimodale Verarbeitung.
- **Zukunftsperspektiven:** Multimodalität, domänenspezifische Modelle und ethische Herausforderungen.
- **Diskussion:** Gesellschaftliche Implikationen und verantwortungsvoller Einsatz von LLMs.

LLM Standardwerke

- **Build LLMs from Scratch** (Raschka)
  - `https://github.com/rasbt/LLMs-from-scratch`
  - Praktische Implementierung in PyTorch
- **Transformers for NLP** (Rothman)
  - ISBN 978-1803247335
  - BERT/GPT Anwendungen
- **Deep Learning for NLP** (Goldberg)
  - ISBN 978-3319987305
  - Grundlagen und Anwendungen
- **Natural Language Processing with Transformers** (Tunstall et al.)
  - ISBN 978-1098136789
  - Praxisorientierte Einführung

LLM Forschungsarbeiten

- **Attention Is All You Need** (Vaswani et al., 2017)
  - https://arxiv.org/abs/1706.03762
  - Transformer-Architektur
- **BERT Paper** (Devlin et al., 2019)
  - https://arxiv.org/abs/1810.04805
  - Bidirektionale Pretraining
- **GPT-3 Paper** (Brown et al., 2020)
  - https://arxiv.org/abs/2005.14165
  - Few-Shot Learning
- **GloVe: Global Vectors for Word Representation** (Pennington et al., 2014)
  - https://aclanthology.org/D14-1162/
  - Wortvektor-Repräsentationen
- **Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks** (Reimers & Gurevych, 2019)
  - https://arxiv.org/abs/1908.10084
  - Satz-Embeddings
- **LLaMA: Open and Efficient Foundation Language Models** (Touvron et al., 2023)
  - https://arxiv.org/abs/2302.13971
  - Open-Weight Modelle

LLM Praktische Ressourcen

- **Hugging Face Transformers**
  - https://github.com/huggingface/transformers
  - Bibliothek für LLMs
- **LangChain**
  - https://python.langchain.com/
  - LLM Orchestrierung
- **LLaMA & LlamaIndex**
  - https://github.com/facebookresearch/llama
  - Open-Weight Modelle
- **OpenAI API**
  - https://platform.openai.com/
  - Zugriff auf GPT-Modelle
- **Pinecone**
  - https://www.pinecone.io/
  - Vektordatenbanken für RAG
- **Weaviate**
  - https://weaviate.io/
  - Semantische Suche und Vektorspeicherung