



Anwendung Generativer KI

2025

Lizenz

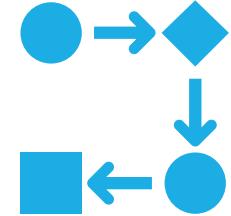
- Die Kursmaterialien wurden von Ralf Bendig erstellt, sofern nicht anders angegeben.
- Lizenz CC BY 4.0.

Titelseite: Bild mit MS Copilot erstellt

00 Intro



Kurs-Organisation



ZEITPLANUNG

- 5 Tage
- Start: 09:00 Uhr
- Ende: 16:30 Uhr
- Pause nach 90 Min

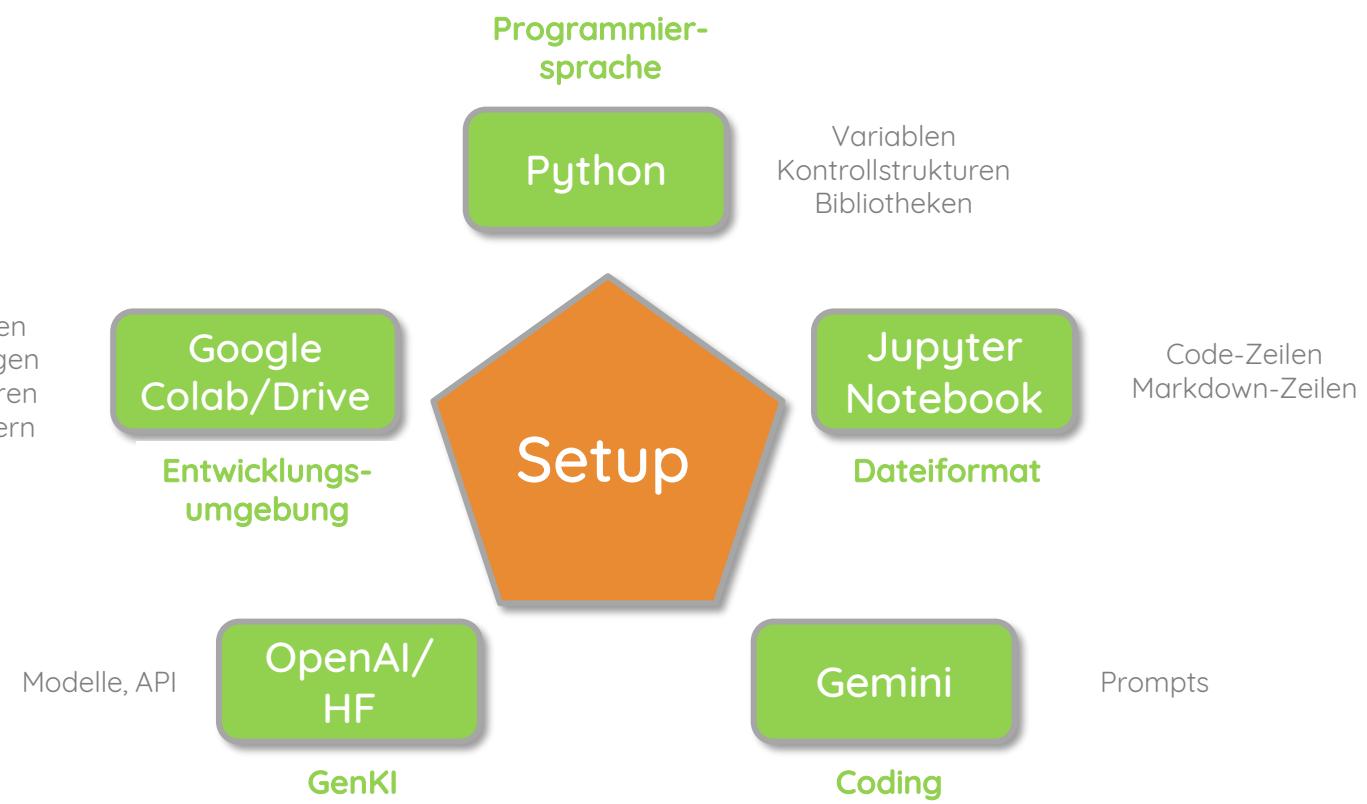
VORGEHEN

- Grundlagen/Basiswissen
- Beispiele
- Training/Fallstudien

VERSCHIEDENES

- Pinboard

IT-Setup



Pinboard

<https://bit.ly/3T2evF6>



Bild mit Recraft erstellt

Google Colab(oratory)



- Google Colaboratory, kurz Colab, ist eine kostenlose Entwicklungsumgebung, die vollständig in der Cloud ausgeführt wird.
- In Colab können Jupyter-Notebooks erstellt, bearbeiten und ausgeführt werden.
- Colab unterstützt viele beliebte Machine-Learning-Bibliotheken, die einfach in ein Notebook geladen werden können.
- Colab erlaubt es unterschiedliche Laufzeitumgebungen zu definieren in denen man neben einer CPU auch GPUs und TPUs verwenden kann.
- Colab hat mit Gemini eine integriert GenKI für Coding.

CPU = Central Processing Unit, GPU = Graphics Processing Unit, TPU = Tensor Processing Unit



Jupyter-Notebook

- Die Jupyter App ist ein Entwicklungsumgebung, die das Bearbeiten und Ausführen von Programmiersprachen, u.a. Python, über einen Webbrowser ermöglicht.
- Der Name Jupyter bezieht sich auf die drei wesentlichen Programmiersprachen Julia, Python und R und ist auch eine Hommage an Galileos Notizbucheinträge zur Entdeckung der Jupitermonde.
- Mit der Jupyter App kann man Notizbücher erstellen. Diese Notizbücher (Dateiendung .ipynb) enthalten:
 - **Programmcode**, der ausgeführt werden kann,
 - **Markdown Zeilen**, das sind Textzeilen mit Formatierungsangaben.
- Die Jupyter-Notebooks werden v.a. für interaktive, wissenschaftliche Analysen und Berechnungen, z.B. Data Analytics und Machine Learning, verwendet.

Google Colab - Einstieg



The screenshot shows the Google Colab interface. At the top, there's a navigation bar with the 'Willkommen bei Colaboratory' logo, 'PRO', 'Datei', 'Bearbeiten', 'Anzeige', 'Einfügen', 'Laufzeit', 'Tools', and 'Hilfe'. Below the navigation bar is a sidebar titled 'Inhalt' containing links to 'Erste Schritte', 'Data Science', 'Maschinelles Lernen', 'Weitere Ressourcen', and 'Vorgestellte Beispiele'. A button '+ Bereich' is also present. The main content area displays a 'Willkommen bei Colab!' message and a section '(Neu) Gemini API testen' with a list of links: 'Generate a Gemini API key', 'Talk to Gemini with the Speech-to-Text API', 'Gemini API Quickstart with Python', 'Gemini API code sample', 'Compare Gemini with ChatGPT', and 'More notebooks'. Below this, there's a video thumbnail titled '3 Cool Google Colab Features' featuring a man speaking. A text box at the bottom left says '[] 1 Beginnen Sie mit dem Programmieren oder generieren Sie Code mit KI.' and a section 'Was ist Colab?' followed by a list of features: 'Keine Konfiguration erforderlich', 'Kostenloser Zugriff auf GPUs', and 'Einfache Freigabe'. A note at the bottom states 'Egal, ob Sie Student, Data Scientist oder AI-Forscher sind – Colab erleichtert Ihnen die Arbeit. Im Video [Einführung in Colab](#) erhalten Sie weitere...'.

Gemini in Colab



■ Gemini Zeile



1 Beginnen Sie mit dem Programmieren oder generieren Sie Code mit KI.

- Code generieren direkt im Jupyter Notebook
- Code-Vervollständigung

■ Gemini Register



- Code optimieren
- Code kommentieren/dokumentieren
- Code erklären/erläutern
- allgemeine Prompts

Tastatureinstellungen/Shortcuts

Tastatureinstellungen

Editor-Tastaturbelegungen
default
 Mit der Eingabetaste werden Vorschläge akzeptiert

Tastenkombinationen

Wenn Sie eine Tastenkombination hinzufügen oder ändern möchten, klicken Sie auf die entsprechende Tastenkombination und geben Sie dann die neue Kombination ein. Beachten Sie, dass Ctrl+M als Präfix für Tastenkombinationen mit mehreren Tasten verwendet werden kann.

Tastenkombination ipynb herunterladen

Standardeinstellungen wiederherstellen

Ctrl+M . Laufzeit neu starten

Tastenkombination py herunterladen

Aktuelle Zeile kommentieren

Tastenkombination Alle Ausgaben löschen

Tastenkombination Alle Laufzeiten auf Werkseinstellungen zurücksetzen

Tastenkombination Alle Zellen auswählen

Tastenkombination Alle Zellen in Notebook ausführen

Tastenkombination Alle suchen/ersetzen

Tastenkombination Alle/ausgewählte Abschnitte maximieren

Tastenkombination Alle/ausgewählte Abschnitte minimieren

Tastenkombination An Cursorposition teilen

Tastenkombination Ansicht mit einem Tab ansehen

Tastenkombination Auf die zuletzt ausgeführte Zelle fokussieren

Tastenkombination Ausführung unterbrechen

Tastenkombination Ausgabe einblenden/ausblenden

Tastenkombination Ausgabevollbild anzeigen

Standardeinstellungen wiederherstellen

Ctrl+M . Laufzeit neu starten und alle Zellen im Notebook ausführen

Tastenkombination Laufzeitprotokolle anzeigen

Tastenkombination Layout des minimierten Abschnitts speichern

Tastenkombination Letzte Zellaktion rückgängig machen

Tastenkombination Linken Bereich in einen Tab verschieben

Tastenkombination Maximieren/Minimieren des aktuellen Abschnitts aktivieren/deaktivieren

Tastenkombination Maximize/Minimize current cell

Tastenkombination Mehrere Zellen auswählen

Tastenkombination Mit einer Laufzeit verbinden

Tastenkombination Mit einer benutzerdefinierten GCE-VM verbinden

Tastenkombination Mit lokaler Laufzeit verbinden

Tastenkombination Neues Notebook erstellen

Tastenkombination Notebook drucken

Tastatureinstellungen

Tastenkombination Ausgewählte Ausgaben löschen

Ctrl+M K Ausgewählte Zellen nach oben verschieben

Ctrl+M J Ausgewählte Zellen nach unten verschieben

Ctrl+F10 Ausgewählte Zellen und alle folgenden Zellen ausführen

Tastenkombination Ausgewählte Zellen zusammenführen

Ctrl+Shift+Enter Auswahl ausführen

Tastenkombination Auswahl erweitern, um nächste Zelle einzuschließen

Shift+Up Auswahl erweitern, um vorherige Zelle einzuschließen

Tastenkombination Automatische Ausführung der ersten Zelle oder des ersten Abschnitts bei jeder Ausführung aktivieren

Tastenkombination Automatische Vervollständigung

Ctrl+Space oder Tab Automatische Vervollständigung

Tastenkombination Befehlspalette anzeigen

Tastenkombination Code ein-/ausblenden

Tastenkombination Code-Snippets-Bereich anzeigen

Tastenkombination Code-docstring-Hilfe aktivieren/deaktivieren

Tastenkombination Codecell hinzufügen

Ctrl+M A Codezelle oben einfügen

Ctrl+M B Codezelle unten einfügen

Tastenkombination Colab Enterprise öffnen

Tastenkombination Dateibrowser anzeigen

Tastenkombination Drive bereitstellen

Tastenkombination Drive trennen

Tastenkombination Editor-Einstellungen öffnen

Shift+Tab Einzug für aktuelle Zeile entfernen

Tastenkombination Einstellungen öffnen

Tastenkombination Fokus mit Tabulator-taste verschieben

Tastenkombination Fokus mit Tabulator-taste verschieben, aktivieren/deaktivieren

Tastenkombination Formular hinzufügen

Ctrl+M F Formularansicht ändern

Tastenkombination Formularattribute bearbeiten

Tastenkombination Formularfeld hinzufügen

Tastenkombination Geplante Notebooks anzeigen

Ctrl+Enter Hervorgehobene Zelle ausführen

Ctrl+Shift+S Hervorgehobene Zelle auswählen

Tastenkombination Hervorgehobene Zelle mit nächster Zelle zusammenführen

Tastenkombination Hervorgehobene Zelle mit vorheriger Zelle zusammenführen

Esc Hervorgehobener Zelle aufheben

Tastenkombination Im Playground-Modus öffnen

Tastenkombination In Google Drive suchen

Tastenkombination In Scratchpad-Zelle kopieren

Shift+Ctrl+H In aktueller Zelle alle ersetzen

Tastenkombination Informationen zu Notebook-Datei anzeigen

Tastenkombination Inhaltsverzeichnis anzeigen

Tastatureinstellungen

Tastenkombination Notebook freigeben

Tastenkombination Notebook hochladen

Tastenkombination Notebook in Drive verschieben

Tastenkombination Notebook in Google Drive markieren/Markierung aufheben

Tastenkombination Notebook in Papierkorb verschieben

Tastenkombination Notebook speichern

Ctrl+O Notebook öffnen

Tastenkombination Notebook-Einstellungen öffnen

Tastenkombination Notebook-Quelle anzeigen

Tastenkombination Notebook-Vergleich

Ctrl+M N Nächste Zelle

Tastenkombination Nächsten Tab hervorheben

Ctrl+G Nächstes Element suchen

Tastenkombination Ressourcen ansehen

Tastenkombination Scratchpad-Codezelle öffnen

Tastenkombination Seitenleiste für Kommentare

Tastenkombination Sichtbarkeit des Headers aktivieren/deaktivieren

Tastenkombination Sitzungen verwalten

Tastenkombination Statusleiste einblenden/ausblenden

Tastenkombination Tab in den nächsten Bereich verschieben

Tastenkombination Tab in den vorherigen Bereich verschieben

Tastenkombination Tabs in zwei Spalten anzeigen

Tastatureinstellungen

Tastenkombination Dateibrowser anzeigen

Ctrl+M H Tastenkombinationen anzeigen

Tastenkombination Terminal anzeigen

Tastenkombination Editor-Einstellungen öffnen

Einstellungen öffnen

Shift+Tab Einzug für aktuelle Zeile entfernen

Tastenkombination Fokus mit Tabulator-taste verschieben

Tastenkombination Fokus mit Tabulator-taste verschieben, aktivieren/deaktivieren

Tastenkombination Formular hinzufügen

Ctrl+M F Formularansicht ändern

Tastenkombination Formularattribute bearbeiten

Tastenkombination Formularfeld hinzufügen

Tastenkombination Geplante Notebooks anzeigen

Ctrl+Enter Hervorgehobene Zelle ausführen

Ctrl+Shift+S Hervorgehobene Zelle auswählen

Tastenkombination Hervorgehobene Zelle mit nächster Zelle zusammenführen

Tastenkombination Hervorgehobene Zelle mit vorheriger Zelle zusammenführen

Esc Hervorgehobener Zelle aufheben

Tastenkombination Im Playground-Modus öffnen

Tastenkombination In Google Drive suchen

Tastenkombination In Scratchpad-Zelle kopieren

Shift+Ctrl+H In aktueller Zelle alle ersetzen

Tastenkombination Informationen zu Notebook-Datei anzeigen

Tastenkombination Inhaltsverzeichnis anzeigen

Tastatureinstellungen

Tastenkombination Tabs in zwei Zeilen anzeigen

Ctrl+Alt+M Kommentar hinzufügen

Tastenkombination Kommentarbereich öffnen

Tastenkombination Kopie als GitHub Gist speichern

Tastenkombination Kopie in Drive speichern

Tastenkombination Kopie in GitHub speichern

Tastenkombination Zu Codezelle konvertieren

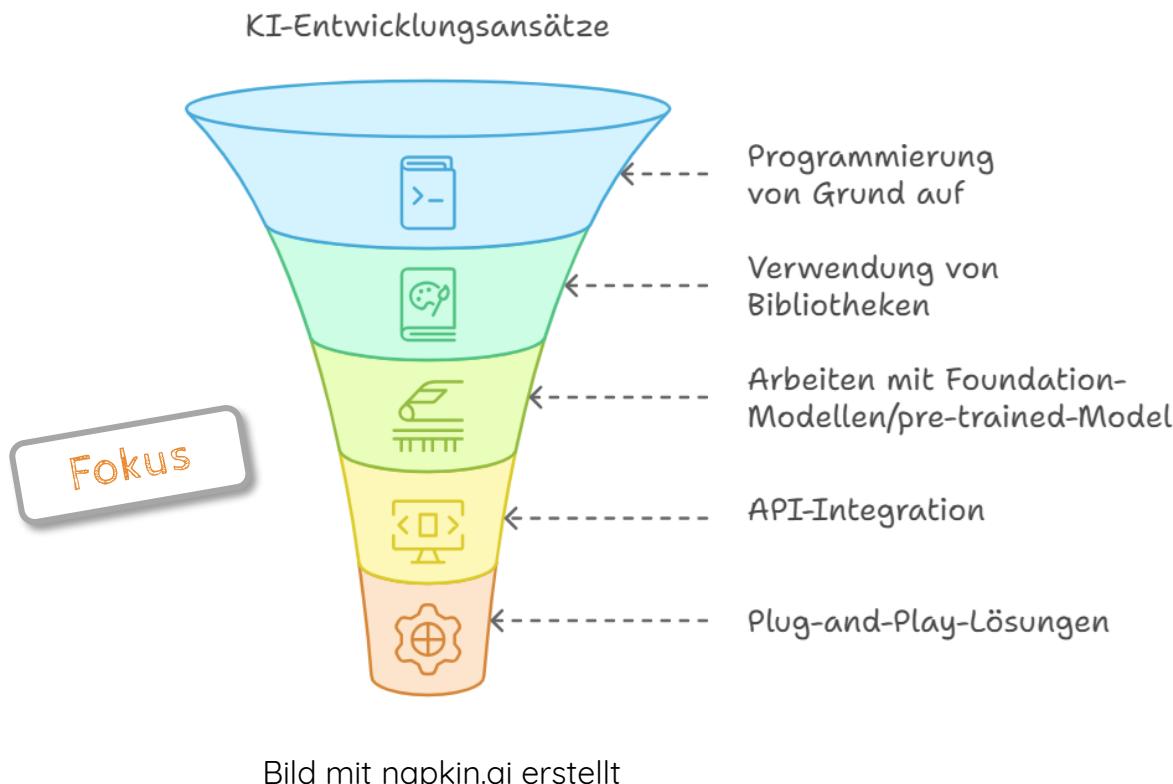
Ctrl+M M Zu Textzelle konvertieren

Tastenkombination Zu einer konkreten Zelle

Ctrl+M S Überarbeitung speichern und anpinnen

Tastenkombination Überarbeitungsverlauf anzeigen

Entwicklung KI-Modell



1. Von Grund auf programmieren
 - Eigenimplementierung aller Komponenten.
 - Beispiel: Entwicklung neuer Modellarchitekturen.
2. Verwendung von Bibliotheken
 - Einsatz von z.B. sklearn, TensorFlow, Keras.
 - Beispiel: Training eigener Modelle, Nutzung von GPT.
3. Arbeiten mit Foundation-Modellen
 - Schnelle Ergebnisse durch Prompting/RAG/Fine-tuning.
 - Beispiel: Domänenspezifische Modellanpassung.
4. Integration über API
 - Nutzung fertiger APIs, einfache Integration.
 - Beispiel: KI-Funktionen in Geschäftsanwendungen.
5. Plug-and-Play-Lösungen
 - Fertige Tools, sofort einsatzbereit.
 - Beispiel: No-Code KI-Anwendungen.

01 Einführung in Generative AI



As Time Goes By

2005



2013

Quelle: [Papst-Momente: Bilder zeigen Vergleich zwischen 2005 und 2013 - DER SPIEGEL](#)

Was ist Generative KI?

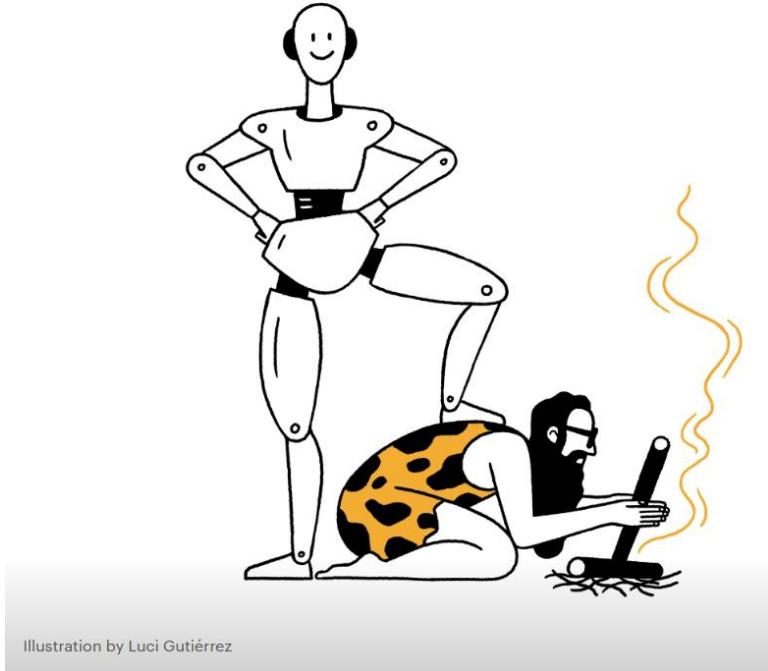


Illustration by Luci Gutiérrez

Quelle: [Upcoming Landmarks in Artificial Intelligence | The New Yorker](#)

- Generative KI ist der Überbegriff für "**kreative** Künstliche Intelligenz", die bei Bedarf neue Inhalte bzw. Originalinhalte produzieren kann.
- Generative KI ist dazu in der Lage, nicht nur vorhandene Daten zu analysieren oder zu klassifizieren, sie erstellt etwas **völlig Neues**, egal ob Text, Bilder, Audio, synthetische Daten usw.

ChatGPT - einer von Vielen



- ChatGPT ist ein von OpenAI entwickeltes künstliches Intelligenzmodell.
- Er errechnet die **Wahrscheinlichkeit** der nächsten **Wortsequenz**, um menschenähnliche Inhalte zu imitieren.
- Die GPT-Modelle (Generative Pre-trained Transformer) wurde mit **Milliarden** von Wörtern trainiert & kann Texte generieren, Fragen beantworten und viele Aufgaben durchführen, die mit Sprache zu tun haben.
- Diese Modelle nutzen Deep Learning und speziell entworfene neuronale Netze, um **menschenähnliche Textantworten** zu produzieren.
- ChatGPT ist eine spezialisierte Version der GPT-Modelle, die speziell für die Erstellung von Dialogen in einem Chat-ähnlichen Kontext optimiert ist.
- Es handelt sich um eine Anwendung des GPT-Modells, das auf Konversationen ausgerichtet ist.

GPT



Wortsequenz-
wahrscheinlichkeits-
rechenmaschine

Version	Jahr	Anzahl Parameter
GPT-1	2018	117 Millionen
GPT-2	2019	1,5 Milliarden
GPT-3	2020	175 Milliarden
GPT-4	2023	100 Billionen (geschätzt)

Bild mit DALL·E erstellt

Trainingsdaten GPT-3

Dataset	# tokens	Proportion within training
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

Quelle: [GPT-3 - Wikiwand](#)

Statistics of Common Crawl Monthly Archives

Number of pages, distribution of top-level domains, crawl overlaps, etc. - basic metrics about Common Crawl Monthly Crawl Archives
Latest crawl: CC-MAIN-2024-33

[Home](#) [Size of crawls](#) [Top-level domains](#) [Registered domains](#) [Crawler metrics](#) [Crawl overlaps](#) [Media types](#) [Character sets](#) [Languages](#)

[View the Project on GitHub](#)

This project is maintained by [commoncrawl](#)

Hosted on GitHub Pages — Theme by [orderedlist](#)

Top-500 Registered Domains of the Latest Main Crawl

The table below shows the top 500 registered domains (in terms of page captures) of the last main/monthly crawl (CC-MAIN-2024-33). The underlying data is also provided in CSV format, see [domains-top-500.csv](#).

Note that the ranking by page captures only partially corresponds to the importance of domains, as the crawler respects the robots.txt and tries hard not to overload web servers. Highly ranked domains tend to be underrepresented. If you're looking for a list of domain or host names ranked by page rank or harmonic centrality, consider using one of the webgraph datasets instead.

domain	pages	urls	hosts	%pages	%urls
blogspot.com	13800889	13760369	277377	0.595912	0.597375
wikipedia.org	4588372	4585368	668	0.198123	0.199063
pinterest.com	2956214	2950014	45	0.127647	0.128068
yahoo.com	2738282	2702457	225	0.118237	0.117321
netlify.app	2077349	2077117	15767	0.089698	0.090173
web.app	1776608	1776572	16232	0.076713	0.077126
europa.eu	1775261	1756399	735	0.076654	0.076250
photoshelter.com	1699868	1699247	2001	0.073399	0.073769
google.com	1340155	1300061	179	0.057867	0.056439
amazonaws.com	1337874	1330922	5157	0.057768	0.057779
free.fr	1324910	1324312	10136	0.057209	0.057492
wordpress.org	1197736	1196894	218	0.051717	0.051960
altavista.org	1191359	1190199	5420		
medium.com	1187623	1186822	5157		
appspot.com	1186250	1185573	5157		
plus.google.com	1185573	1185225	5157		

Quelle: [Statistics of Common Crawl Monthly Archives by commoncrawl](#)

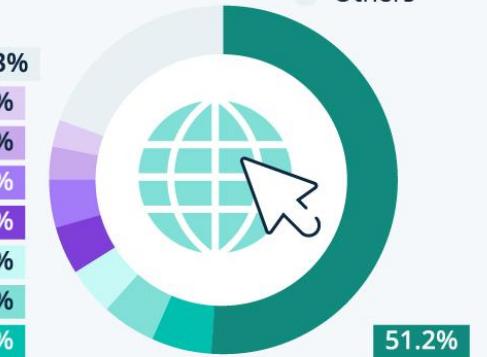
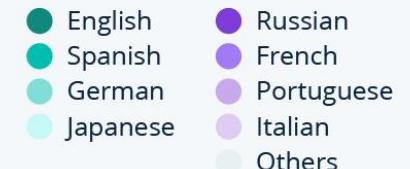
Training & Languages

The Most Spoken Languages: On the Internet and in Real Life

Most spoken language in real life
Total number of speakers



Most used languages online
Share of websites using language

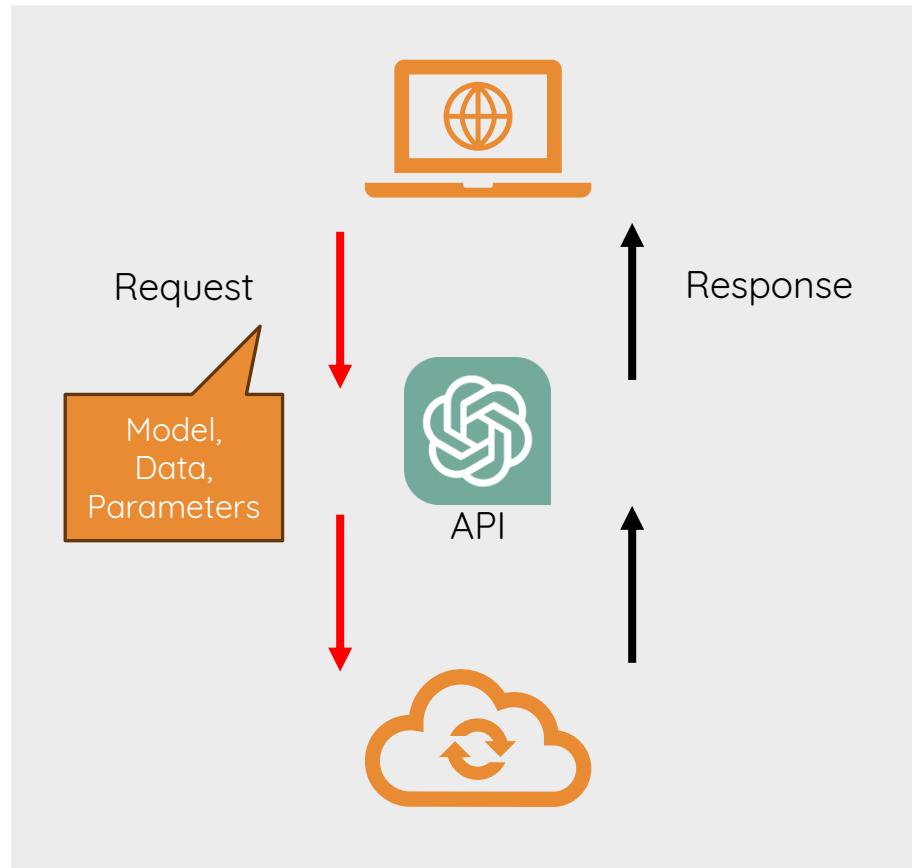


* Most spoken languages as of 2023, languages used on the internet as of Feb. 2024
Sources: Ethnologue, W3Techs



statista

OpenAI API - Funktionsweise



- Eine **API** (Application Programming Interface) ist eine Sammlung von Regeln und Spezifikationen, die es Softwareanwendungen ermöglicht, miteinander zu kommunizieren (Schnittstelle).
- Die **OpenAI API** ermöglicht es Entwicklern, Zugang zu fortschrittlicher KI-Technologie zu erhalten, insbesondere zu den Sprachmodellen von OpenAI wie GPT (Generative Pre-trained Transformer).
- Diese API bietet eine **breite Palette von Funktionen**, darunter Textgenerierung, Textverständnis, Übersetzungen, Zusammenfassungen und viele andere sprachbasierte Aufgaben.
- Die API ist so gestaltet, dass sie leicht zugänglich und benutzerfreundlich ist, wodurch sie für ein breites Spektrum von Anwendungen, von kleinen Projekten bis hin zu groß angelegten Unternehmungen, geeignet ist.

Hugging Face



Was ist Hugging Face?

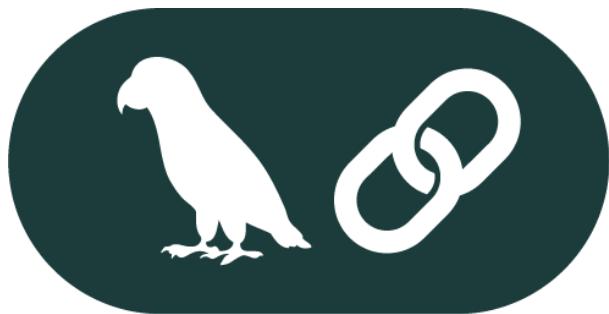
- Hugging Face ist eine Plattform und Community für künstliche Intelligenz, insbesondere für Natural Language Processing (NLP) und generative KI. Sie bietet vortrainierte Modelle, eine einfache API und Tools zur Anwendung und Feinabstimmung von KI-Modellen.

Wichtige Komponenten:

- Transformers – Bibliothek für vortrainierte Modelle (z. B. GPT, BERT, T5).
- Datasets – Sammlung von Datensätzen für KI-Training.
- Tokenizers – Werkzeuge zur Verarbeitung von Texten für Modelle.
- Hugging Face Hub – Plattform zum Teilen von Modellen und Datensätzen.
- Inference API – Bereitstellung von Modellen in der Cloud ohne eigene Infrastruktur.

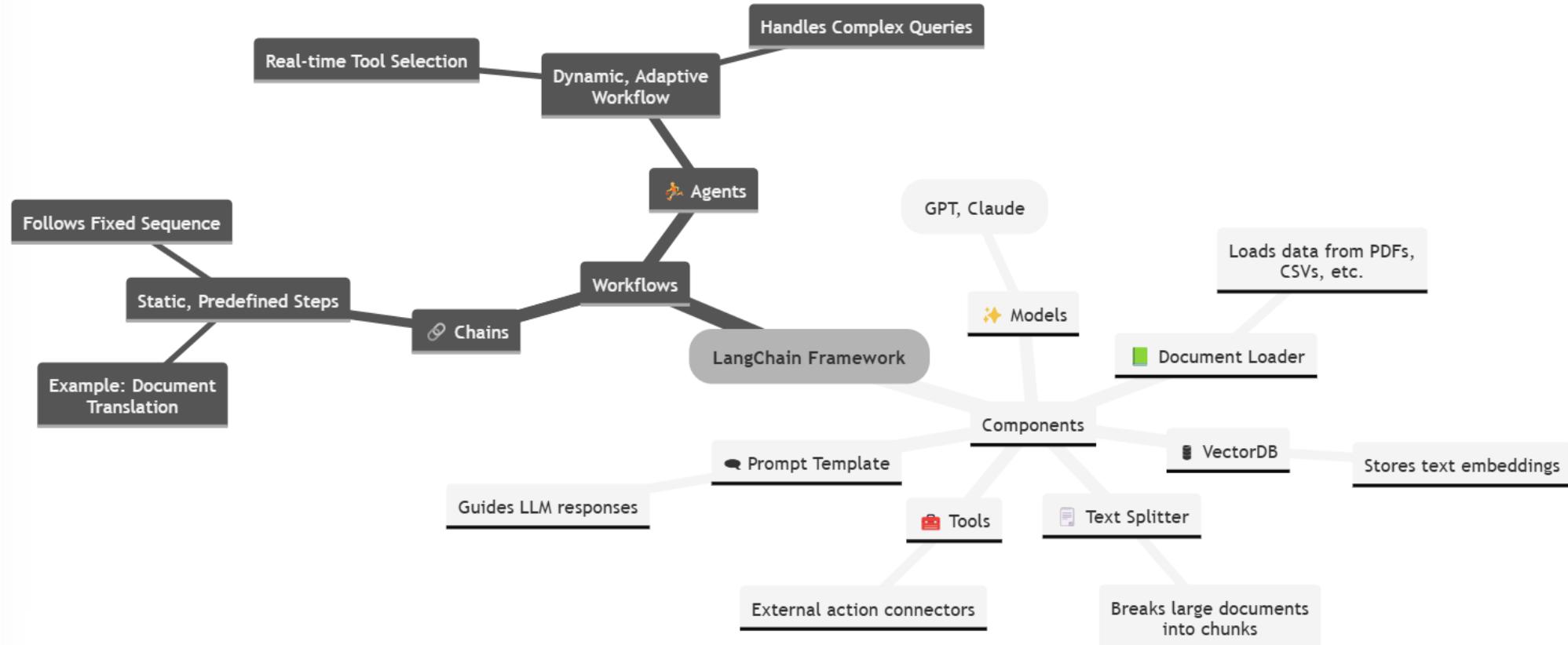
LangChain

Fokus:
Workflow-Design
und Tool-Integration



- LangChain ist ein Open-Source-Framework zur Vereinfachung der Entwicklung von Anwendungen mit großen Sprachmodellen (LLMs).
- Anstatt bei Null anzufangen, kann man LangChain verwenden. Es fungiert im Wesentlichen als Sammlung von Bausteinen, die man verwenden kann, um Ihre Anwendung zu erstellen.
- LangChain bietet Vorteile wie:
 - Schnellere Entwicklung: Anstatt benutzerdefinierte Lösungen für grundlegende Aufgaben mit Sprachmodellen zu schreiben, kann man vorgefertigte Komponenten von LangChain verwenden.
 - Einfachere Handhabung: LangChain bietet Abstraktionen, die die Komplexität der Arbeit mit LLMs verringern.
 - Flexible Anwendungen: LangChain ermöglicht die Kombination von Sprachmodellen mit anderen Datenquellen, um personalisierte Benutzererfahrungen zu schaffen (RAG).

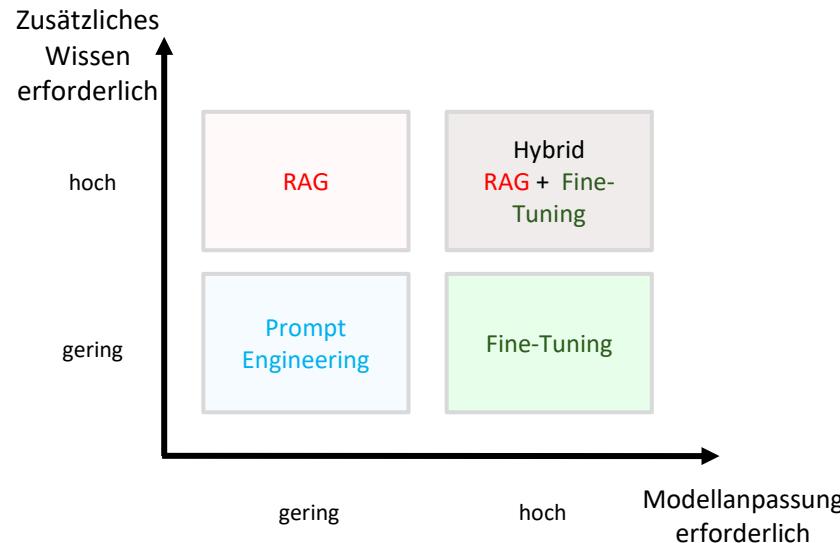
LangChain: Components & Work Flows



02 Grundlagen der Modellansteuerung



Ansätze Modellansteuerung



- **Prompt Engineering:**
 - Erster Ansatz, da einfach und kostengünstig.
 - Geeignet für Aufgaben, bei denen der LLM über ausreichende Kenntnisse verfügt oder effektiv mit Eingabeaufforderungen geführt werden kann.
- **Retrieval Augmented Generation (RAG):**
 - Verwenden, wenn Prompt Engineering allein nicht ausreicht.
 - Ideal für Aufgaben, bei denen dynamische, aktuelle, interne/externe Informationen erforderlich sind.
 - Nützlich, um die Größenbeschränkungen des LLM bei der Eingabe zu berücksichtigen.
- **Fine-Tuning:**
 - Wenn mit Prompt Engineering und RAG keine zufriedenstellenden Ergebnisse erzielt werden. Aufwändiger als RAG und Prompt Engineering.
 - Effektiv für Aufgaben, bei denen das LLM bestimmte Aufgabencluster (Coding, Planning) oder domänen spezifisches Wissen (z.B. Medizin, Recht) erlernen muss.
 - Kann in Bezug auf Geschwindigkeit und Inferenzkosten effizienter sein.

Was ist Prompt-Engineering?



*Gehe nicht davon aus, dass das Modell weiß,
was du meinst. Man muss es klar und deutlich
sagen!*



Prompt-Engineering bezeichnet die gezielte Gestaltung von Eingabeaufforderungen (Prompts), um von künstlicher Intelligenz (KI), insbesondere der natürlichen Sprachverarbeitung, gewünschte und präzise Antworten zu erhalten.



Rolle des Menschen: Interaktor



Rollen des Menschen

- Fragensteller
- Gesprächspartner
- Lehrer und Korrektor
- Bewerter
- Ideen- und Anregungsgeber
- Empfänger von Unterstützung
- Prüfer
- ...

*Interaktor →
aktive wechselseitige
Beeinflussung*

Quelle: Bild von [Peggy und Marco Lachmann-Anke](#) auf [Pixabay](#)

Methoden zur Promptierung



Bild mit ideogram.ai erstellt

- STAR und PREPARE sind Methoden für ein effektives Prompting.
- Sie ergänzen sich gegenseitig und können kombiniert werden, um die Qualität der Antworten zu verbessern.
- STAR konzentriert sich auf die Struktur und Klarheit des Prompts.
- PREPARE hingegen fokussiert auf die inhaltliche Vorbereitung und die Feinabstimmung des Prompts.
- Zusammenwirken:
 - STAR bildet die Grundlage für einen klaren und verständlichen Prompt, indem es den Kontext, die Aufgabe und das gewünschte Ergebnis definiert.
 - PREPARE erweitert diese Grundlage, indem es zusätzliche Informationen und Anweisungen liefert, um die Antwort des LLMs zu optimieren.

STAR

Das STAR-Format hilft dabei, die Struktur eines Prompts klar und nachvollziehbar zu gestalten. Es besteht aus vier Elementen:

S	Situation	Beschreibung des Kontexts oder der Ausgangslage
T	Task	Definition der spezifischen Aufgabe oder des Ziels
A	Action	Erläuterung der durchzuführenden Schritte oder Maßnahmen
R	Result	Beschreibung des erwarteten Ergebnisses oder Outputs

PREPARE

PREPARE hilft dabei, klare und effektive Eingaben für KI-Modelle zu gestalten.

Hier sind die Elemente:

P	Prompt Framework	Eine klare Ausgangsfrage oder Anweisung	Was ist das beabsichtigte Ergebnis? Wie kann man die Anfrage strukturieren?
R	Role	Zuweisung einer spezifischen Rolle für die KI (und ggf. Zielgruppe)	Welche Rolle sollte die Aufgabe bearbeiten bzw. Ergebnisse bekommen?
E	Explicit	Präzise Formulierung zur Vermeidung von Missverständnissen	Welche logischen Denkschritte sind für die Bearbeitung erforderlich?
P	Parameter	Festlegung von Rahmenbedingungen wie Tonfall und Format	Welche Kriterien und Eigenschaften muss das Ergebnis erfüllen?
A	Ask	Aufforderung an die KI, bei Unklarheiten nachzufragen	Was muss das Modell wissen, um das Ergebnis weiter zu verbessern?
R	Rate	Selbstbewertung der KI-Antwort	Welche weiteren Perspektiven und Bewertungen könnten das Ergebnis bereichern?
E	Emotion	Hinzufügen eines emotionalen Elements	Welche emotionalen Elemente können die Qualität der Antwort zu steigern?

Beispiel:

- **Situation:** Analysiere die Bildungstrends für einen Workshop mit Führungskräften.
- **Task:** Erstelle eine Zusammenfassung der neuesten Trends im Bereich Künstliche Intelligenz in der Bildung.
- **Action:**
 - **Prompt:** Fasse die neuesten Trends im Bereich KI in der Bildung zusammen.
 - **Role:** Du bist ein Bildungsexperte.
 - **Explicit:** Erkläre, wie KI personalisiertes Lernen fördern kann.
 - **Parameter:** Nutze einen informativen Ton und beschränke die Zusammenfassung auf 300 Wörter.
 - **Ask:** Stelle bei Unklarheiten Klärungsfragen.
 - **Rate:** Bewerte die Zusammenfassung von 0 bis 10 und schlage Verbesserungen vor.
 - **Emotion:** Verwende einen motivierenden Ansatz, da die Informationen wichtig sind.
- **Result:** Eine prägnante, informative Zusammenfassung, die alle geforderten Aspekte abdeckt.

Prompt - Lernszenarien

■ Zero-Shot Learning (ZSL)

Definition: Modelle lösen Aufgaben ohne spezifische Trainingsbeispiele.

Mechanismus: Nutzt abstraktes Wissen aus ähnlichen Aufgaben.

Prompt: Hier ist eine Liste von Tweets zu verschiedenen Themen. Kategorisiere jeden Tweet in eine der folgenden Kategorien: Politik, Sport, Unterhaltung.

■ Few-Shot Learning (FSL)

Definition: Modelle lernen Aufgaben mit sehr wenigen Beispielen.

Mechanismus: Meta-Lernen und Siamesische Netzwerke helfen, effektiv von minimalen Daten zu lernen.

Prompt: Betrachte diese fünf Bilder jeder Hautkrebsart. Lerne, Merkmale zu erkennen, die jede Art charakterisieren, obwohl die Datenmenge sehr begrenzt ist.

■ Many-Shot Learning

Definition: Traditionelles Lernszenario mit umfangreichen Trainingsdaten.

Mechanismus: Das Modell lernt spezifische Muster aus einer großen Datenmenge. .

Prompt: Du hast Zugang zu Millionen von Stunden gesprochener Sprache aus einer Vielzahl von Quellen. Trainiere ein Modell zur Spracherkennung, das in der Lage ist, verschiedene Sprachen zu übersetzen..

Prompt Creator

Erstellen eines
Prompts mit
dem ChatBot

Ich möchte, dass du mein Prompt Creator wirst. Dein Ziel ist es, mir zu helfen, den bestmöglichen Prompt für meine Bedürfnisse zu erstellen. Der Prompt wird von dir, ChatGPT, verwendet. Du wirst den folgenden Prozess befolgen:

1. Als erstes fragst du mich, worum es in dem Prompt gehen soll. Ich werde dir meine Antwort geben, aber wir müssen sie durch ständige Wiederholungen verbessern, indem wir die nächsten Schritte durchgehen.
2. Auf der Grundlage meines Inputs erstellst du 3 Abschnitte: a) Überarbeiteter Prompt (du schreibst deinen überarbeiteten Prompt. Er sollte klar, präzise und für dich leicht verständlich sein), b) Vorschläge (du machst Vorschläge, welche Details du in den Prompt einbauen solltest, um ihn zu verbessern) und c) Fragen (du stellst relevante Fragen dazu, welche zusätzlichen Informationen ich brauche, um den Prompt zu verbessern).
3. Der Prompt, den du bereitstellst, sollte die Form einer Anfrage von mir haben, die von ChatGPT ausgeführt werden soll.
4. Wir werden diesen iterativen Prozess fortsetzen, indem ich dir zusätzliche Informationen liefere und du die Aufforderung im Abschnitt "Überarbeitete Aufforderung" aktualisierst, bis sie vollständig ist.

Quelle: [DIESER CHATGPT PROMPT IST DER WAHNSINN](#)

Codieren mit GenAI

Einsatz von GenAI für Coding

“

„Der Grund, warum wir Code schreiben und keine natürliche Sprache, ist, dass natürliche Sprache mehrdeutig ist. Diese Mehrdeutigkeit kann man nicht wirklich umgehen.“

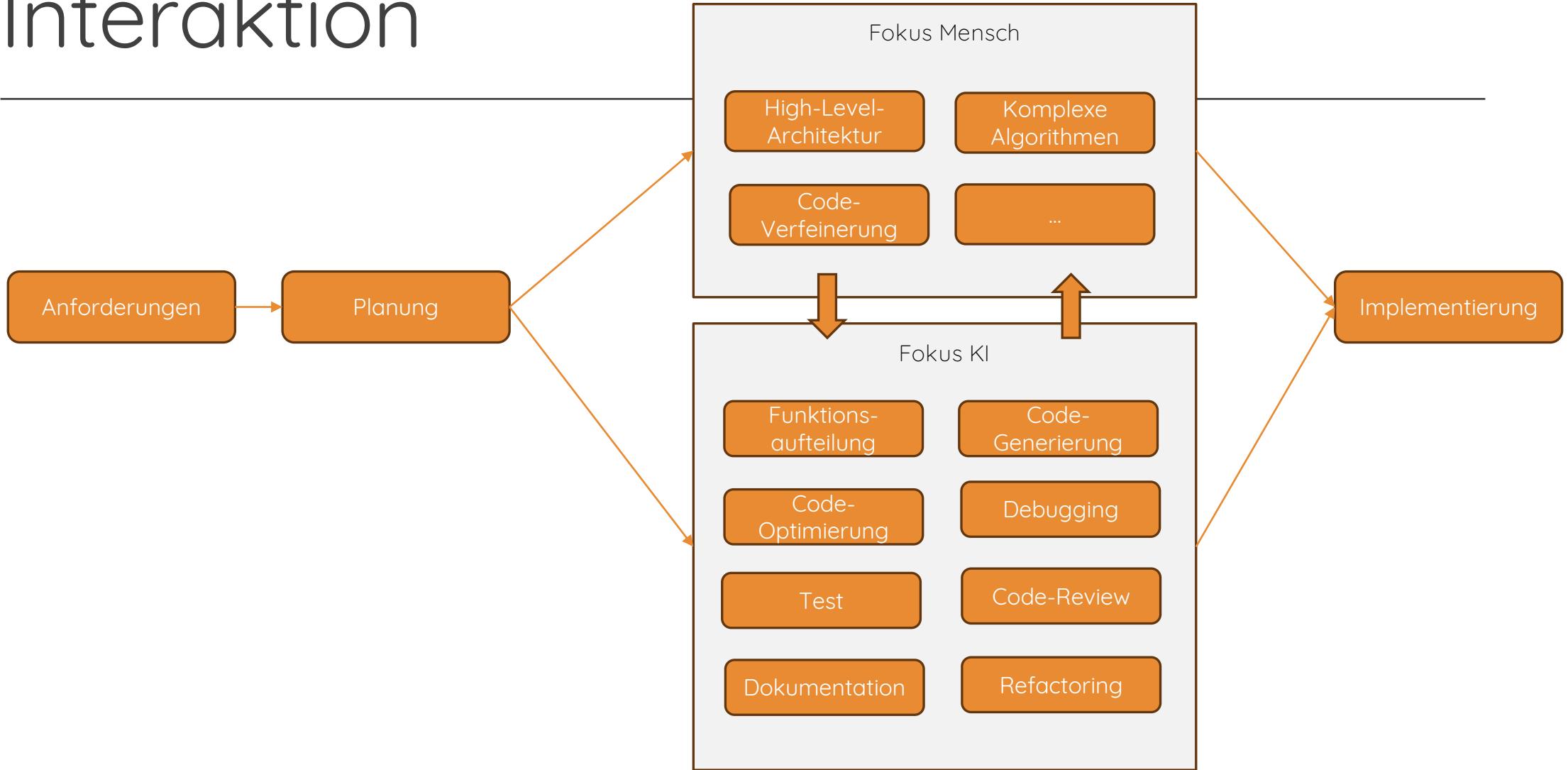
Ada Morse

Codecademy Curriculum Developer in Data Science

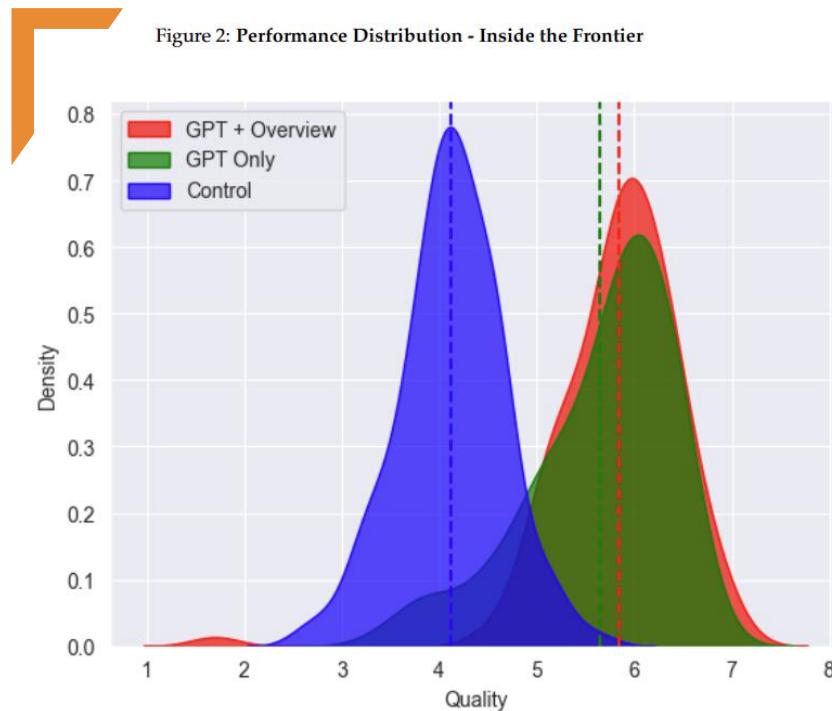
”

Quelle: [Can ChatGPT Teach Me How To Code Better Than Courses? \(codecademy.com\)](#)

Mensch - KI - Interaktion



Performancesteigerung



Quelle: Dell'Acqua et al. "Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality." Harvard Business School Working Paper, No. 24-013, September 2023

Coding: Laut der Studie "GenAI at Work" wurden Performance-verbesserung durch den Einsatz von AI-Tools erzielt. Die Studie ergab, dass der Zugang zu generativer KI die **Produktivität** im Durchschnitt um **14%** steigerte, wobei insbesondere **weniger erfahrene und qualifizierte Mitarbeiter** von einer Verbesserung um **34%** profitierten.

Quelle: Erik Brynjolfsson et. Al. GENERATIVE AI AT WORK, Working Paper 31161, <http://www.nber.org/papers/w31161>, NATIONAL BUREAU OF ECONOMIC RESEARCH. April 2023, revised November 2023

M02 - Gemini-Colab

Stand: 03.2025

Google Colaboratory (oft kurz "Colab" genannt) ist eine hervorragende Umgebung, um Python zu lernen und zu programmieren. Es bietet eine kostenlose Python-Umgebung direkt im Browser, ohne dass eine Installation auf dem eigenen Computer erforderlich ist. Eine besonders nützliche Funktion ist die Integration von Gemini, einem leistungsstarken KI-Modell von Google. Gemini kann auf vielfältige Weise beim Programmieren in Python unterstützen. Es bietet sowohl technische Unterstützung beim Schreiben von Code als auch didaktische Erklärungen, die beim Verständnis komplexer Konzepte helfen.

Zugriff auf Gemini in Google Colaboratory

1. **Ein Colab-Notebook öffnen:** Auf <https://colab.research.google.com/> kann ein neues Notebook erstellt oder ein bestehendes geöffnet werden.
2. **Das Gemini-Panel öffnen:** Auf der linken Seite des Colab-Fensters befindet sich eine vertikale Leiste mit verschiedenen Symbolen. Dort auf das Symbol klicken, das wie ein **kleines, stilisiertes "G"** aussieht (möglicherweise mit "Gemini" oder "Assistent" beschriftet). Dadurch öffnet sich das Gemini-Panel auf der rechten Seite.

Die beiden Modi von Gemini: Codezeile und Chat

Gemini bietet zwei Hauptmöglichkeiten, um beim Programmieren zu unterstützen: direkt in einer Codezeile und über ein Chatfenster. Beide Modi haben ihre Vorteile und können je nach Aufgabe hilfreich sein. Es empfiehlt sich, beide Varianten auszuprobieren, um herauszufinden, welche Form der Interaktion in bestimmten Lernsituationen am hilfreichsten ist.

Ob und in welchem Umfang Gemini das Coding unterstützt, wird über die Einstellungen festgelegt.

Einstellungen

Website >	<input checked="" type="checkbox"/> KI-basierte Inline-Code vervollständigungen anzeigen
Editor >	<input checked="" type="checkbox"/> Sie haben der Verwendung von generativen KI-Funktionen zugestimmt
KI-Unterstützung >	<input type="checkbox"/> Generative AI-Funktionen ausblenden
Colab Pro >	Generative AI in Colab ist eine experimentelle Technologie, die mitunter falsche oder anstößige Informationen ausgeben kann, die nicht die Auffassung von Google widerspiegeln. Nutzen Sie den Code daher mit Vorsicht.
GitHub >	
Sonstige >	

[Schließen](#)

1 | Gemini in der Codezeile

Dieser Modus ist besonders praktisch, wenn gerade Code geschrieben wird und direkt Unterstützung benötigt wird. Gemini integriert sich hier wie eine intelligente Auto vervollständigung, die nicht nur die Syntax erkennt, sondern auch kontextabhängig relevante Vorschläge macht.

Funktionsweise:

- In einer Codezelle tippen:** Eine neue Codezelle im Colab-Notebook erstellen.
- Automatische Vervollständigung und Vorschläge nutzen:** Beim Tippen versucht Gemini, den Code zu vervollständigen oder mögliche nächste Schritte vorzuschlagen. Häufig werden ganze Codeblöcke oder Funktionsaufrufe angezeigt. Die Vorschläge passen sich dabei an den bisher geschriebenen Code an.
- Direkte Fragen als Kommentar formulieren:** Fragen können direkt in einer Codezeile gestellt werden, indem ein Kommentar mit einem `#` beginnt. Das ist besonders nützlich, um kontextbezogene Hilfe zu erhalten, ohne das Notebook verlassen zu müssen.

Beispiele für die Nutzung in der Codezeile:

- **Code-Vervollständigung:**

```
# Ich möchte eine Liste erstellen. Wie mache ich das?
meine_liste = [
```

Nach dem Drücken der Enter-Taste oder kurzem Warten schlägt Gemini möglicherweise vor, wie Elemente zur Liste hinzugefügt werden können.

- **Funktionsvorschläge:**

```
import math
# Ich möchte die Quadratwurzel einer Zahl berechnen.
math.sqrt
```

Gemini wird wahrscheinlich `math.sqrt()` als mögliche Funktion vorschlagen.

- **Direkte Fragen im Kommentar:**

```
# Wie kann ich eine Schleife verwenden, um alle Elemente einer Liste
auszugeben?
meine_liste = ["Apfel", "Banane", "Kirsche"]
# Gemini könnte hier einen Codevorschlag für eine for-Schleife machen.
```

Vorteile für Einsteigerinnen und Einsteiger:

- **Schnelles Erlernen von Syntax:** Es wird direkt sichtbar, wie Code korrekt geschrieben wird.
- **Entdecken neuer Funktionen:** Gemini kann Funktionen oder Module vorschlagen, die eventuell noch unbekannt sind.
- **Weniger Tippfehler:** Die automatische Vervollständigung hilft, Tippfehler zu vermeiden.
- **Sofortige Hilfestellung:** Einsteigerinnen und Einsteiger erhalten proaktive Unterstützung, ohne eine externe Dokumentation durchsuchen zu müssen.

Wichtig: Vorschläge sollten kritisch hinterfragt werden. Es empfiehlt sich, die Funktionsweise des vorgeschlagenen Codes zu verstehen und bei Bedarf Rückfragen zu stellen oder im Chat-Modus vertiefende Erklärungen einzuholen.

2 | Gemini als Chat

Der Chat-Modus ermöglicht eine interaktivere und detailliertere Kommunikation mit Gemini. Es können Fragen gestellt, Code-Schnipsel eingefügt und Erklärungen eingeholt werden. Dieser Modus eignet sich besonders für konzeptuelle Fragen, detaillierte Analysen und das gemeinsame Lösen von Aufgabenstellungen.

Funktionsweise:

1. **Das Gemini-Panel öffnen:** Falls noch nicht geschehen, das Panel auf der rechten Seite öffnen.
2. **Frage oder Anweisung im Chatfenster eingeben:** Die Anfrage sollte klar und präzise formuliert sein. Kontextinformationen helfen Gemini, bessere Antworten zu liefern.
3. **Antwort von Gemini erhalten:** Gemini versucht, die Frage zu beantworten, Code-Beispiele zu liefern oder Konzepte zu erklären. Oft folgen auf die erste Antwort weiterführende Hinweise oder Nachfragen, die helfen, das Verständnis zu vertiefen.

Beispiele für die Nutzung im Chat:

- **Erklärung von Python-Konzepten:**
 - "Was ist der Unterschied zwischen einer Liste und einem Tupel in Python?"
 - (Gemini antwortet mit einer ausführlichen Erklärung und Beispielen.)
- **Code für eine bestimmte Aufgabe generieren:**
 - "Ein Python-Skript, das alle geraden Zahlen zwischen 1 und 10 ausgibt."
 - (Gemini liefert den entsprechenden Code.)
- **Fehlersuche (Debugging):**
 - "Fehlermeldung: TypeError: unsupported operand type(s) for +: 'int' and 'str'. Was bedeutet das?"
 - (Gemini erklärt den Fehler und gibt Lösungsvorschläge.)
- **Alternative Lösungen vorschlagen:**
 - "Gibt es eine einfachere Möglichkeit, die Summe einer Liste zu berechnen?"
 - (Gemini könnte die Verwendung von `sum()` vorschlagen.)

Vorteile für Einsteigerinnen und Einsteiger:

- **Direktes Fragen:** Gemini kann wie ein Tutor genutzt werden.
- **Verständliche Erklärungen:** Komplexe Konzepte werden in einfachen Worten dargestellt.
- **Hilfe bei Problemen:** Bei Herausforderungen in Programmieraufgaben kann Gemini unterstützen.
- **Interaktive Lernumgebung:** Die Konversation mit Gemini fühlt sich natürlich an und kann auf individuelle Bedürfnisse eingehen.

Tipps für die effektive Nutzung von Gemini:

- **Präzise Fragen stellen**
- **Kontext angeben:** Beispielsweise den relevanten Code mit angeben.
- **Experimentieren:** Unterschiedliche Fragetypen ausprobieren.
- **Antworten kritisch überprüfen**
- **Beide Modi nutzen:** Codezeile für schnelle Hilfe, Chat für tiefergehende Fragen.

- **Verständnis ist wichtiger als Kopieren:** Ziel ist das Lernen, nicht nur das Übernehmen von Code.
- **Nachfragen erlaubt:** Gemini kann auch auf Folgefragen eingehen oder vorherige Erklärungen anpassen.

3 | Anwendungsbeispiele

Gemini kann auf unterschiedliche Weise eingesetzt werden. Die folgende Liste gibt eine Übersicht über typische Anwendungsfälle:

- **Codegenerierung:** Automatisches Erstellen von Code anhand einer textbasierten Beschreibung einer Aufgabe.
- **Codeergänzung:** Vorschläge für den nächsten Codeblock oder passende Befehle beim Tippen in der Codezelle.
- **Erklärung von Konzepten:** Begriffe wie Schleifen, Bedingungen, Datentypen oder Funktionen verständlich erklären lassen.
- **Debugging:** Hilfe bei Fehlermeldungen oder unerwartetem Verhalten des Codes, inklusive Vorschlägen zur Fehlerbehebung.
- **Codeoptimierung:** Vorschläge zur Verbesserung der Effizienz, Lesbarkeit oder Struktur eines bestehenden Codes.
- **Refactoring:** Hilfe beim Umstrukturieren von Code, z. B. durch Auslagerung in Funktionen oder bessere Benennung von Variablen.
- **Testgenerierung:** Erstellung einfacher Testfälle oder Unit-Tests für eine Funktion.
- **Lösungsvorschläge für Aufgaben:** Bearbeitung typischer Übungsaufgaben oder Programmierprobleme, z. B. aus Lehrbüchern oder Online-Kursen.
- **Modulauswahl:** Empfehlungen, welches Modul oder welche Bibliothek für eine bestimmte Aufgabe geeignet ist.
- **Begriffsdefinitionen:** Schnelle Erklärungen für Begriffe wie "Iterator", "Lambda-Funktion" oder "List Comprehension".
- **Integration mit externen APIs:** Hilfestellung beim Aufbau von Schnittstellen zu anderen Webdiensten oder Datenquellen.

Diese Anwendungsbeispiele zeigen, wie vielseitig Gemini in der Programmierpraxis unterstützen kann – sowohl beim Lernen als auch bei der täglichen Arbeit mit Python. Der Einsatz von Gemini spart Zeit, reduziert Frustration und fördert ein besseres Verständnis der Sprache und ihrer Konzepte.

M04 - GenAI Frameworks

Stand: 03.2025

1 Übersicht GenAI Frameworks

Aspekt	LangChain	Llama Stack	Haystack	LlamaIndex
Hauptfokus	Vielseitiges Framework für verschiedene NLP-Anwendungen	Standardisierung von Kernbausteinen für generative KI-Anwendungen	Entwicklung skalierbarer LLM-Anwendungen mit Fokus auf Suche und RAG	Datenframework für LLM-Anwendungen mit Fokus auf Indexierung und Abruf
Architektur	Modular mit Ketten, Agenten, Prompts und Speicher	APIs für Inferenz, Sicherheit, Speicher, Agenten und Bewertung	Pipeline-basiert mit Komponenten für Dokumentenverarbeitung, Retrieval und Generierung	Modulare Struktur mit Fokus auf Datenindexierung und -abfrage
Stärken	Flexibilität, Anpassbarkeit, breites Anwendungsspektrum	Einheitliche APIs, nahtloser Übergang zwischen Entwicklung und Produktion	Einfachheit, Skalierbarkeit, gute Dokumentation	Umfangreiche Datenverarbeitung, fortschrittliche Vektorspeicherfunktionen
Datenverarbeitung	Unterstützt verschiedene Datenquellen und Multimodalität	Fokus auf Llama-Modellfamilie, einschließlich spezialisierter Modelle	Effiziente Verarbeitung großer Datensätze, starke Retrieval-Fähigkeiten	Umfassende Datenkonnektor-Unterstützung, fortschrittliche Indexierungsfunktionen

Aspekt	LangChain	Llama Stack	Haystack	LlamaIndex
Einsatzbereich	Allgemeine NLP-Aufgaben, komplexe Workflows	Spezifisch für Llama-basierte Anwendungen	Suchsysteme, RAG-Anwendungen, Produktionsumgebungen	Kontextbasierte KI-Anwendungen, RAG-Implementierungen
Entwickler-freundlichkeit	Erfordert tieferes Verständnis für komplexe Anwendungen	Zieht auf einfache Integration und einheitliche Entwicklererfahrung ab	Einfach zu verstehen und zu erweitern, gute Dokumentation	Einfach zu bedienen, aber weniger umfangreich als LangChain
Skalierbarkeit	Gut für verschiedene Projektgrößen geeignet	Designed für nahtlosen Übergang von lokaler Entwicklung zu Cloud-Deployments	Optimiert für große Datensätze und Produktionsanwendungen	Gut skalierbar, besonders für Datenindexierung und -abfrage
Community und Support	Große, aktive Community mit umfangreicher Dokumentation	Wachsende Community, fokussiert auf Llama-Ökosystem	Starke Community-Unterstützung, gute Dokumentation	Wachsende Community, gute Dokumentation
Leistung	Effizientes Ressourcenmanagement durch optimierte Workflows	Speziell für Llama-Modelle optimiert	Leichtgewichtig und effizient, besonders bei RAG-Anwendungen	Effizient bei Datenindexierung und -abfrage
Anpassbarkeit	Hochgradig anpassbar, aber komplex für Anfänger	Fokussiert auf Standardisierung innerhalb des Llama-Ökosystems	Gute Balance zwischen Anpassbarkeit und Einfachheit	Flexibel, besonders bei Datenverarbeitungsaspekten

2 Gründe für LangChain

Für einen die Anwendung von GenAI ist **LangChain** zu empfehlen. Gründe für diese Empfehlung sind:

- **Breite Modellunterstützung:** LangChain unterstützt eine Vielzahl von großen Sprachmodellen (LLMs) wie OpenAI, Hugging Face, Cohere und mehr. Dies ermöglicht es Einsteigern, mit verschiedenen Modellen zu experimentieren und zu verstehen, wie sie funktionieren.
- **Flexibilität und Vielseitigkeit:** LangChain ist sehr flexibel und kann für eine Vielzahl von Anwendungsfällen eingesetzt werden, z. B. Chatbots, Frage-Antwort-Systeme, Agents und Dokumentenanalyse. Dies gibt Einsteigern die Möglichkeit, unterschiedliche GenAI-Anwendungen kennenzulernen.
- **Umfangreiche Dokumentation und Community:** LangChain hat eine große und aktive Community sowie eine umfangreiche Dokumentation. Dies ist besonders für Einsteiger hilfreich, da sie leicht Ressourcen, Tutorials und Unterstützung finden können.
- **Einfache Integration:** LangChain ermöglicht die einfache Integration mit externen Tools, APIs und Datenquellen. Dies ist nützlich, um Einsteigern zu zeigen, wie sie reale Daten und Dienste in ihre Anwendungen einbinden können.
- **Gute Balance zwischen Einfachheit und Funktionalität:** Während LangChain mächtig und flexibel ist, bietet es auch eine relativ einfache API, die für Einsteiger zugänglich ist. Es gibt viele Beispiele und Vorlagen, die den Einstieg erleichtern.
- **Zukunftssicherheit:** Da LangChain ein allgemeines Framework ist, das nicht auf ein bestimmtes Modell beschränkt ist, können Einsteiger die erlernten Konzepte auf zukünftige Modelle und Technologien anwenden.

3 Vergleich mit anderen Frameworks

- **Llama Stack:** Zu spezialisiert auf Llama-Modelle, was für Einsteiger einschränkend sein könnte.
- **Rivet:** Visuelles Tool, das zwar benutzerfreundlich ist, aber weniger allgemeine Konzepte vermittelt.
- **Haystack:** Fokussiert auf NLP und Frage-Antwort-Systeme, was für Einsteiger zu eng sein könnte.

 **Fazit**

LangChain ist die beste Wahl für einen GenAI-Kurs, da es eine breite Palette von Anwendungsfällen abdeckt, flexibel ist und eine große Community sowie umfangreiche Dokumentation bietet. Es ermöglicht Einsteigern und Fortgeschrittenen, die Grundlagen der Generativen KI zu erlernen und gleichzeitig praktische Erfahrungen mit verschiedenen Modellen und Tools zu sammeln.

M05a - LLM und Transformer

Stand: 03.2025

1 Large Language Models

LLMs funktionieren im Wesentlichen durch die Vorhersage des wahrscheinlichsten nächsten Wortes in einem gegebenen Text, basierend auf einer riesigen Menge an Trainingsdaten aus dem Internet. Dieser Prozess wird durch Billiarden von Parametern gesteuert, die während des Trainings optimiert werden. Ein entscheidender Aspekt ist die Architektur von **Transfomern**, die es LLMs erlaubt, den gesamten Text **parallel** zu verarbeiten, anstatt Wort für Wort, und dabei den Kontext durch einen Mechanismus namens **Self Attention** zu berücksichtigen. Das Ergebnis ist ein System, das erstaunlich flüssige und oft sinnvolle Texte generieren kann, obwohl das genaue *Warum* hinter den Vorhersagen aufgrund der Komplexität und Größe des Modells schwer zu verstehen ist.

2 Transfomer

ELI5-Version

Stell dir vor, du liest ein Buch, aber anstatt es Wort für Wort durchzugehen, kannst du gleichzeitig viele Seiten anschauen und genau erkennen, welche Wörter wichtig sind.

So funktioniert ein Transformer-Modell!

Es benutzt eine Technik namens **Self-Attention**, um nicht nur das aktuelle Wort, sondern auch alle anderen relevanten Wörter im Text zu berücksichtigen – egal, wo sie stehen. Das macht es viel besser darin, **Zusammenhänge zu verstehen**, als frühere KI-Modelle, die Texte nur schrittweise gelesen haben.

Dank dieser Technik sind Transformer-Modelle wie **GPT, BERT oder LLaMA** so gut in **Texterstellung, Übersetzungen und sogar Code-Generierung**.

Fazit

Zusammenfassend lässt sich sagen, dass der "Self-Attention"-Mechanismus ein entscheidendes Element von GPT-Modellen ist, das ihnen ein tiefgreifendes Verständnis von Sprache ermöglicht. Durch die Analyse der Beziehungen zwischen Wörtern in einem Satz können diese Modelle den Kontext von Wörtern präzise erfassen und so natürlichere und kohärentere Texte generieren.

Etwas detaillierter ...

Moderne Transformer-Modelle haben die Art und Weise revolutioniert, wie Maschinen Sprache verstehen und erzeugen. Während frühere Modelle wie Recurrent Neural Networks (RNNs) oder Long Short-Term Memory (LSTMs) Schwierigkeiten hatten, lange Texte zu erfassen, ermöglicht die Self-Attention-Mechanik der Transformer eine deutlich effizientere Verarbeitung von großer Textmenge. Diese Weiterentwicklung ist insbesondere für Anwendungen mit langen Dokumenten oder tiefgehender Kontextanalyse entscheidend.

1 Autoregressive Transformer-Modelle (Textvorhersage Schritt für Schritt)

- **Beispiele:** GPT, GPT-2, GPT-3, GPT-4, LLaMA, Mistral
- **Merkmal:** Die Modelle generieren Text, indem sie Token für Token vorherige Eingaben berücksichtigen (**kausale Self-Attention**). Sie können keine Informationen aus zukünftigen Wörtern im Satz nutzen.
- **Anwendung:** Kreative Textgenerierung, automatische Vervollständigung, interaktive Dialogsysteme.
- **Besonderheiten:** Da diese Modelle nur auf vorherige Tokens zugreifen, eignen sie sich besonders gut für realistische Textgenerierung und kreative Schreibaufgaben, haben aber Schwierigkeiten mit logischem Denken und langfristiger Kohärenz.

2 Bidirektionale Transformer-Modelle (Kontextbezogenes Sprachverständnis)

- **Beispiele:** BERT, RoBERTa, DeBERTa
- **Merkmal:** Die Modelle analysieren gleichzeitig den gesamten Satz und lernen, Lücken zu füllen (**Masked Language Modeling, MLM**). Dadurch können sie ein tieferes Sprachverständnis aufbauen als autoregressive Modelle.
- **Anwendung:** Textklassifikation, Informationsextraktion, Frage-Antwort-Systeme, Sentiment-Analyse.
- **Besonderheiten:** Da sie nicht für die Textgenerierung optimiert sind, eignen sie sich weniger für das kreative Schreiben, sondern eher für Analyseaufgaben und das Verstehen von Textzusammenhängen.

3 Seq2Seq (Encoder-Decoder) Transformer-Modelle (Transformation von Texten)

- **Beispiele:** T5, BART, UL2
- **Merkmal:** Bestehen aus zwei Teilen: Ein **Encoder**, der den Eingabetext verarbeitet, und ein **Decoder**, der einen neuen Text generiert. Diese Architektur ermöglicht es, eine Eingabe in eine andere Form zu übertragen.
- **Anwendung:** Maschinelle Übersetzung, Textzusammenfassung, Daten-zu-Text-Konvertierung, automatische Rechtschreibkorrektur.
- **Besonderheiten:** Diese Modelle können sowohl generieren als auch Texte umwandeln, wodurch sie sehr vielseitig einsetzbar sind. Insbesondere T5 wurde darauf optimiert, nahezu jede NLP-Aufgabe in eine "Text-zu-Text"-Problematik umzuwandeln

Diffusionsbasierte Text-Modelle (Alternative zur Transformer-Architektur)

- **Beispiele:** Experimentelle KI-Modelle wie **DIFFUSER**
- **Merkmal:** Statt Texte autoregressiv zu erzeugen, rekonstruieren diese Modelle Texte aus verrauschten Sequenzen. Sie basieren auf der Idee von Diffusionsmodellen, die bisher vor allem für Bilder genutzt wurden.
- **Anwendung:** Noch in der Forschung, potenziell für kreative Textgenerierung interessant. Diese Modelle könnten langfristig eine Alternative zu Transformer-Modellen bieten, wenn sie weiter optimiert werden.
- **Besonderheiten:** Der Ansatz ist vielversprechend, aber noch nicht weit verbreitet. Es gibt erste Versuche, Diffusionsmodelle für das Generieren von Texten einzusetzen, jedoch sind sie bislang nicht so effizient und präzise wie Transformer-basierte Modelle.

Fazit:

Fazit

Transformer-Modelle lassen sich in **autoregressive, bidirektionale und Encoder-Decoder-Modelle** unterteilen, während Diffusionsmodelle einen experimentellen Ansatz für die Textgenerierung darstellen. Die Wahl des richtigen Modells hängt stark von der jeweiligen Anwendung ab. Während autoregressive Transformer sich besonders gut für das Generieren von Text eignen, werden bidirektionale Modelle vor allem für Analyseaufgaben genutzt. Encoder-Decoder-Modelle hingegen sind die erste Wahl, wenn es darum geht, einen Text in eine andere Form zu überführen, wie es beispielsweise bei der maschinellen Übersetzung der Fall ist. Diffusionsmodelle könnten in Zukunft eine alternative Technik bieten, sind aber derzeit noch nicht weit genug entwickelt.

3 Self-Attention



3.1 Tokenisierung und Einbettung

Bevor der Attention-Mechanismus angewendet werden kann, durchläuft ein Text zwei wichtige Vorverarbeitungsschritte:

1. **Tokenisierung:** Der Eingabetext wird in einzelne Tokens zerlegt
 - Unser Beispiel "Der Hund jagt die Katze" wird in 5 Tokens aufgeteilt
 - Je nach Tokenizer können Wörter auch in Subwort-Tokens zerlegt werden

2. **Token-Einbettung:** Jedes Token wird in einen hochdimensionalen Vektor umgewandelt
 - In BERT haben diese Vektoren typischerweise 768 Dimensionen
 - In GPT-Modellen können es 1024, 2048 oder mehr Dimensionen sein
 - Diese Einbettungen enthalten kontextlose Repräsentationen der Tokens
3. **Positionscodierung:** Da Transformer keine inhärente Reihenfolge kennen, werden Positionsinformationen hinzugefügt
 - Jedes Token erhält Informationen zu seiner Position im Satz
 - Dies wird zur Token-Einbettung addiert

Für unser Beispiel "Der Hund jagt die Katze" entsteht dadurch eine Matrix X mit der Dimension 5×768 , wobei jede Zeile einen Token-Vektor repräsentiert, der sowohl semantische als auch Positionsinformationen enthält.

3.2 Query (Q), Key (K), Value (V) Vektoren

Für jedes Token werden basierend auf seiner Einbettung drei verschiedene Vektoren berechnet:

- **Query (Q):** Sucht nach relevanten Informationen
- **Key (K):** Dient als Schlüssel für die Suche
- **Value (V):** Enthält die eigentlichen Informationen

Die Berechnung erfolgt durch lineare Transformationen:

$$Q = X * W_Q$$

$$K = X * W_K$$

$$V = X * W_V$$

Dabei ist:

- X: Die Eingabematrix mit Token-Embeddings
- W_Q, W_K, W_V: Trainierbare Gewichtungsmatrizen

Diese Gewichtungsmatrizen sind nicht zufällig, sondern werden in vortrainierten Modellen wie **BERT, GPT, T5 oder RoBERTa** bereits optimiert bereitgestellt. Diese Modelle wurden mit großen Textmengen trainiert und enthalten fertige Gewichtungen, die direkt genutzt oder für spezifische Anwendungsfälle feinjustiert (Fine-Tuning) werden können.

Mit Hilfe von Bibliotheken wie **Hugging Face Transformers** können diese Modelle geladen und direkt eingesetzt werden:

```
from transformers import AutoModel, AutoTokenizer

model_name = "bert-base-uncased"
model = AutoModel.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Damit stehen optimierte Gewichtungsmatrizen zur Verfügung, die bereits umfangreiche Sprachmuster und Abhängigkeiten gelernt haben.

3.3 Beispiel: "Der Hund jagt die Katze"

1. **Eingabematrix X:** Bei 5 Tokens und einer 768-dimensionalen Einbettung ist X eine 5×768 -Matrix.
 - Jede Zeile entspricht einem Token: ["Der", "Hund", "jagt", "die", "Katze"]
 - Jede Spalte enthält Werte des Token-Embeddings
2. **Berechnung der Q, K, V-Vektoren:**
 - Durch Multiplikation von X mit den Gewichtungsmatrizen W_Q , W_K , W_V
 - Wenn wir 12 Attention-Heads mit je 64 Dimensionen haben, teilen wir die 768 Dimensionen auf
3. **Multi-Head Attention:**
 - Die 768 Dimensionen werden in 12 Heads mit je 64 Dimensionen aufgeteilt
 - Reshaping der Matrizen: Von (Batch, 5, 768) zu (Batch, 12, 5, 64)
 - Für jeden Head wird separat die Attention berechnet

3.4 Attention-Berechnung (vereinfacht)

Für unser Beispiel "Der Hund jagt die Katze":

1. Das Token "jagt" (an Position 3) hat einen Q-Vektor
2. Dieser Q-Vektor wird mit den K-Vektoren aller Tokens verglichen
3. Der Vergleich mit dem K-Vektor von "Hund" ergibt einen hohen Ähnlichkeitswert
4. Das bedeutet: "jagt" sollte auf "Hund" aufmerksam werden
5. Der V-Vektor von "Hund" fließt stärker in die Ausgabe für "jagt" ein

3.5 Self-Attention und Skalierung

- Der dot-Produkt-Vergleich zwischen Q und K skaliert durch die Anzahl der Dimensionen
- Dies verhindert zu große Werte, die die Softmax-Funktion dominieren könnten
- Die finale Formel lautet:

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

wobei `d_k` die Dimension eines einzelnen Attention-Heads ist

3.6 Attention-Heads und ihre Bedeutung

- **Jeder Attention-Head lernt unterschiedliche sprachliche Konzepte:**
 - Ein Head könnte grammatischen Beziehungen erkennen (z.B. "Hund" als Subjekt von "jagt")
 - Ein anderer Head könnte semantische Beziehungen erkennen (z.B. "Hund" und "Katze" als Tiere)
 - Weitere Heads könnten Positionen oder Redewendungen erkennen
- **Der Vorteil mehrerer Heads:**
 - Parallelere Erfassung verschiedener sprachlicher Muster
 - Verbesserung der Sprachrepräsentation durch multiple Perspektiven

3.7 Mehrere Attention-Schichten

- Transformer-Modelle nutzen mehrere Schichten von Self-Attention
- Die Ausgabe einer Schicht dient als Eingabe für die nächste
- Dies ermöglicht tiefere Erfassung von Bedeutungszusammenhängen

In **BERT-Base** gibt es beispielsweise **12 Attention-Schichten**, jede mit **12 Attention-Heads**, während **GPT-3** je nach Modellgröße **96 oder mehr Schichten** mit mehreren Attention-Heads enthält.

Fazit

Der Attention-Mechanismus ist das Herzstück moderner Transformer-Modelle. Er ermöglicht eine effiziente Verarbeitung natürlicher Sprache, indem er flexibel zwischen Tokens die Beziehungen bestimmt. Durch die Kombination von Multi-Head Attention, Positionscodierung und mehreren Schichten können Modelle wie GPT oder BERT hochgradig kontextabhängige Repräsentationen lernen, die für zahlreiche NLP-Anwendungen genutzt werden.

4 Foundation Models

Foundation Models sind leistungsstarke KI-Modelle, die auf riesigen Datenmengen vortrainiert wurden und als Grundlage für verschiedene Anwendungen dienen. Sie nutzen Architekturen wie **Transformers** und sind in der Lage, durch **Transfer Learning** für spezifische Aufgaben feinjustiert zu werden.

Beispiele für solche Modelle sind **GPT (OpenAI)**, **BERT (Google)** oder **LLAMA (Meta)**. Sie finden Anwendung in Bereichen wie **Texterstellung, Bilderkennung, Sprachverarbeitung**

und Codegenerierung.

Ein Schlüsselmerkmal ist ihre **Skalierbarkeit**, die es ermöglicht, sie für unterschiedlichste Domänen und Anwendungsfälle effizient anzupassen.

Merkmale von Foundation Models

Die Bewertung grundlegender Modelle ist entscheidend, um ihre Fähigkeiten, Grenzen und Eignung für bestimmte Aufgaben zu verstehen. Eine genaue Bewertung stellt sicher, dass die Modelle in realen Anwendungen sicher, zuverlässig und effektiv sind. Sie hilft auch bei der Identifizierung potenzieller Verzerrungen und Fehler, die ihre Leistung beeinträchtigen könnten.

Offene vs. geschlossene Modelle

Offene Modelle bedeuten, dass die trainierten Parameter eines Modells für die Öffentlichkeit zugänglich sind. Dadurch haben Wissenschaftler und Entwickler die Möglichkeit, die Mechanismen des Modells nachzuvollziehen, Forschungsarbeiten zu reproduzieren sowie Anpassungen oder Verbesserungen vorzunehmen.

Geschlossene Modelle hingegen sind Modelle, bei denen der Zugriff auf die Parameter eingeschränkt ist. Unternehmen setzen diese oft in kommerziellen Produkten oder Dienstleistungen ein, da die Veröffentlichung der Gewichte geschäftliche Interessen oder den Schutz der Nutzerdaten beeinträchtigen könnte.

Parametergewichte

Die Anzahl der Parametergewichte eines Modells gibt Aufschluss über dessen Kapazität und Komplexität. Sie wird typischerweise in Millionen (M), Milliarden (B) oder Billionen (T) angegeben. Eine höhere Parameteranzahl erhöht grundsätzlich die Fähigkeit des Modells, komplexe Zusammenhänge und feine Unterschiede in den Daten zu erkennen. Allerdings ist dies kein eindeutiger Indikator für die Leistungsfähigkeit in jeder Anwendung, sondern eher ein Hinweis auf das allgemeine Potenzial des Modells.

Vorteile:

- **Größere Lernfähigkeit:** Modelle mit mehr Parametern können feinere und differenziertere Muster in Daten erfassen, was ihre Präzision und Wirksamkeit bei verschiedenen Aufgaben steigern kann.
- **Bessere Anpassungsfähigkeit:** Umfangreichere Modelle haben oft eine stärkere Fähigkeit zur Generalisierung und können unter geeigneten Trainingsbedingungen auch auf neue, unbekannte Daten besser reagieren.

Nachteile:

- **Hoher Rechenaufwand:** Eine größere Anzahl an Parametern erfordert mehr Rechenleistung für Training und Inferenz, was leistungsfähige Hardware und längere

Verarbeitungszeiten nötig macht.

- **Gefahr der Überanpassung:** Ohne angemessene Regularisierungstechniken besteht die Gefahr, dass das Modell sich zu stark an die Trainingsdaten anpasst und bei neuen, variierenden Datensätzen schlechter abschneidet.
- **Umweltbelastung:** Das Training umfangreicher Modelle verbraucht erheblich mehr Energie, was zu einem höheren CO₂-Ausstoß führt.

Kontextfenstergröße

Die Kontextfenstergröße eines Modells gibt an, wie viele Token (Wörter oder Wortbestandteile) es bei der Generierung oder Vorhersage von Text gleichzeitig verarbeiten kann. Diese Eigenschaft ist aus mehreren Gründen essenziell:

- **Erweiterter Kontext:** Ein größeres Kontextfenster erlaubt es dem Modell, mehr Informationen zu berücksichtigen, was zu kohärenteren und inhaltlich präziseren Ergebnissen führt. Dies ist besonders vorteilhaft für Aufgaben, die lange Texte oder komplexe Zusammenhänge erfordern.
- **Erfassung von Abhängigkeiten:** Ein Modell mit größerem Kontextfenster kann weiter zurückliegende Zusammenhänge im Text besser erfassen, was seine Leistungsfähigkeit bei Aufgaben wie Textzusammenfassungen, Frage-Antwort-Systemen oder interaktiven Dialogen erheblich steigert.

Ein fundiertes Verständnis dieser Faktoren hilft bei der Auswahl eines geeigneten Modells für spezifische Anwendungsfälle und ermöglicht eine gezielte Optimierung der Leistung.

Token

In Large Language Models (LLMs) wie GPT (Generative Pre-trained Transformer) stellen Token die grundlegenden Einheiten des verarbeiteten Textes dar. Ein **Token** kann ein vollständiges Wort, ein Wortbestandteil oder ein einzelnes Zeichen sein. Wie genau ein Token definiert wird, hängt vom verwendeten Tokenizer ab, der während des Modelltrainings zum Einsatz kam. So könnte das Wort *Hausboot* entweder als einzelnes Token betrachtet oder in die Bestandteile *Haus* und *boot* zerlegt werden - abhängig von der jeweiligen Tokenisierungsstrategie.

Die Nutzungskosten eines LLMs zur Textgenerierung werden in der Regel anhand der verarbeiteten Token berechnet. Dabei zählen sowohl die Token aus der Eingabe als auch diejenigen, die das Modell als Antwort generiert. Da die Verarbeitung jedes Tokens insbesondere bei sehr großen Modellen mit Milliarden von Parametern erhebliche Rechenressourcen erfordert, beeinflusst die Anzahl der Token direkt die Rechenkosten. Daher ist ein effizientes Management der Token-Nutzung essenziell, um Kosten zu optimieren.

Die **Kontextfenstergröße** eines LLM gibt an, wie viele Token aus einer Eingabe das Modell gleichzeitig berücksichtigen kann. Hat ein Modell beispielsweise ein Kontextfenster von

1.024 Token, kann es nur die letzten 1.024 Token eines Textes für die nächste Vorhersage verwenden. Ist der Eingabetext länger, kann das Modell auf frühere Abschnitte nicht mehr direkt zugreifen, was sich auf die Kohärenz und Relevanz der generierten Antworten auswirken kann.

Temperatur

Große Sprachmodelle

Name	Ersteller	Open/Closed	Input-Token	Output-Token	Anzahl Parameter
gpt-4o	OpenAI	Closed	128K	4K	200B
gpt-4o-mini	OpenAI	Closed	128K	4K	Unbekannt
Gemini 2.0 Ultra	Google	Closed	2M	4K	Unbekannt
Gemini 1.5 Pro	Google	Closed	2M	4K	Unbekannt
Gemini 1.5 Flash	Google	Closed	1M	4K	Unbekannt
Claude 3 Opus	Anthropic	Closed	200K	4K	Unbekannt
Claude 3 Sonnet	Anthropic	Closed	200K	4K	~400B
Claude 3 Haiku	Anthropic	Closed	200K	4K	Unbekannt
Llama 3.1 405B	Meta	Open	512K	4K	405B
Llama 3.1 70B	Meta	Open	256K	4K	70B
Llama 3.1 8B	Meta	Open	128K	4K	8B
Mistral 7B	Mistral.AI	Open	32K	4K	~7B

Auszug OpenAI Modelle: (Stand 02.2025)

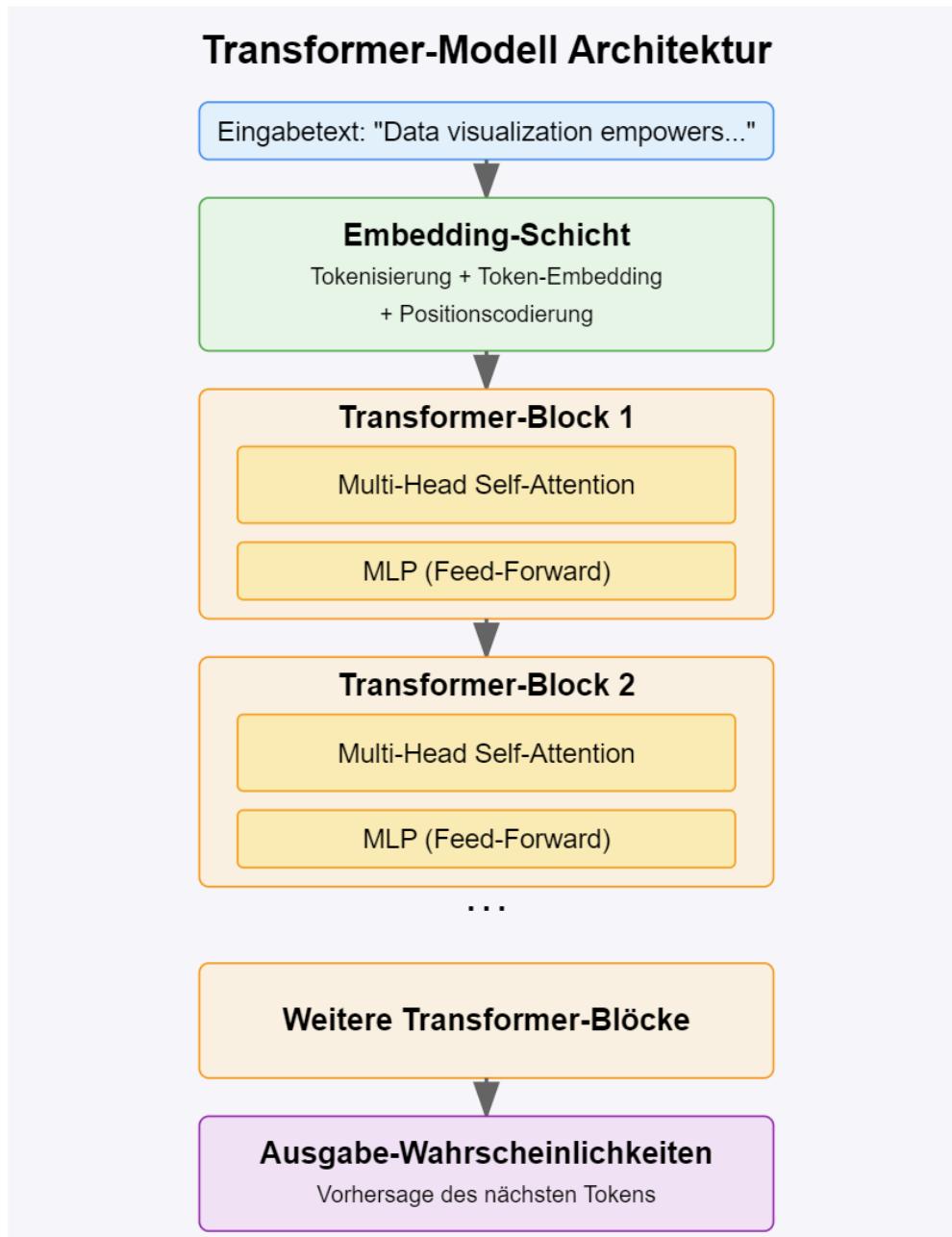
Modell	Anwendungsbereiche
gpt-4o	Textgenerierung, Übersetzungen, Zusammenfassungen, breite Anwendungen in der natürlichen Sprachverarbeitung.
gpt-4o-audio-preview	Generierung und Verarbeitung von audio-basierten Inhalten, wie automatisierte Transkription, Spracherkennung und Musikkomposition.

Modell	Anwendungsbereiche
gpt-4o-realtime-preview	Echtzeitanwendungen wie interaktive Chatbots, Echtzeitübersetzungen, schnelle Antwortzeiten erfordernd.
gpt-4o-mini	Leichte Textgenerierung, geeignet für weniger rechenintensive Anwendungen, z.B. in Embedded-Systemen.
gpt-4o-mini-audio-preview	Einfache Spracherkennung, Soundeffekt-Generierung, weniger komplexe Audioverarbeitungsaufgaben.
gpt-4o-mini-realtime-preview	Echtzeitanwendungen mit schnellen Antwortzeiten, geeignet für mobile oder IoT-Geräte.
o1	Anspruchsvolle Berechnungen, komplexe Simulationen, detaillierte Datenanalysen, umfangreiche Sprachmodelle.
o3-mini	Kostengünstige NLP- und ML-Aufgaben, ideal für akademische Forschung oder Startups mit begrenztem Budget.
o1-mini	Mittelschwere Aufgaben in Machine Learning und Datenverarbeitung, weniger ressourcenintensive Alternative zu Vollversionen.

M05b - Transformer-Explainer

Stand: 03.2025

[Transformer Explainer: LLM Transformer Model Visually Explained](#)



1 Was ist ein Transformer?

Ein Transformer ist eine spezielle Art von neuralem Netzwerk, das seit 2017 die KI-Welt revolutioniert hat. Diese Architektur steckt hinter bekannten Textgeneratoren wie ChatGPT, Llama und Gemini. Transformer werden auch für Bilder, Audio und andere Anwendungen genutzt.

2 Textgenerierende Transformer

Sie arbeiten nach dem Prinzip "Was kommt als nächstes?": Wenn du einen Text eingibst, berechnet das Modell, welches Wort mit der höchsten Wahrscheinlichkeit folgen sollte. Der wichtigste Teil ist der **Self-Attention-Mechanismus**, der es dem Modell erlaubt, Beziehungen zwischen allen Wörtern in einem Text zu verstehen.

Ein bekanntes Beispiel ist GPT-2 (small) mit 124 Millionen Parametern - kein Riese nach heutigen Standards, aber ein gutes Modell zum Verständnis der Grundlagen.

3 Aufbau eines Transformer-Modells

Ein Transformer besteht aus drei Hauptteilen:

1. **Embedding-Schicht**: Wandelt Text in Zahlen um
2. **Transformer-Blöcke**: Verarbeiten diese Zahlen
3. **Ausgabe-Schicht**: Berechnet Wahrscheinlichkeiten für das nächste Wort

3.1 Embedding - Text in Zahlen umwandeln

Bevor ein Transformer arbeiten kann, muss Text in Zahlen umgewandelt werden:

1. **Tokenisierung**: Der Text wird in Einzelteile (Token) zerlegt. Diese können ganze Wörter oder Wortteile sein.
2. **Token-Embedding**: Jedem Token wird ein Zahlenvektor zugewiesen (bei GPT-2 small sind das 768 Zahlen pro Token).
3. **Positionscodierung**: Dem Modell wird mitgeteilt, an welcher Position jedes Token steht.

Beispiel: "Data visualization empowers users to..." wird in Token zerlegt und jedes Token bekommt einen eigenen Zahlenvektor.

3.2 Transformer-Blöcke: Das Herzstück

Hier findet die eigentliche Verarbeitung statt. Jeder Block enthält:

- **Multi-Head Self-Attention:** Ermöglicht dem Modell, auf relevante Teile des Textes zu "achten"
- **MLP (Mehrschichtiges Perzeptron):** Ein kleines neuronales Netzwerk zur Weiterverarbeitung

3.3 (Multi-Head) Self-Attention

Dies ist der Kern eines Transformers. Für jeden Token werden drei Arten von Vektoren berechnet:

- **Query (Q):** "Was suche ich?" - ähnlich einer Suchanfrage
- **Key (K):** "Wo könnte es sein?" - ähnlich Seitentiteln
- **Value (V):** "Was ist der Inhalt?" - die eigentliche Information

Durch Berechnungen mit diesen Vektoren bestimmt das Modell, wie stark jedes Wort auf andere Wörter "achten" soll. GPT-Modelle verwenden dabei "maskierte Attention", was bedeutet, dass ein Wort nur auf vorangegangene Wörter achten darf, nicht auf zukünftige.

Die Attention wird auf mehrere "Köpfe" (Heads) aufgeteilt, damit das Modell unterschiedliche Beziehungen zwischen Wörtern lernen kann.

3.4 Ausgabe-Wahrscheinlichkeiten

Am Ende berechnet das Modell Wahrscheinlichkeiten für jedes mögliche nächste Wort. Mit verschiedenen Parametern kann man die Textgenerierung steuern:

- **Temperatur:** Steuert, wie "kreativ" oder wie "sicher" die Antworten sind
 - Niedrige Temperatur: vorhersehbare, sichere Antworten
 - Hohe Temperatur: kreativere, überraschendere Antworten
- **Top-k-Sampling:** Beschränkt die Auswahl auf die k wahrscheinlichsten Wörter
- **Top-p-Sampling:** Wählt aus einer dynamischen Anzahl wahrscheinlicher Wörter aus

4 Zusätzliche wichtige Elemente

Für ein stabiles Training braucht ein Transformer noch:

- **Layer-Normalisierung:** Hält die Zahlen im Modell in einem sinnvollen Bereich
- **Dropout:** Schaltet zufällig Teile des Netzwerks ab, damit es nicht auswendig lernt
- **Residual-Verbindungen:** Ermöglichen ein besseres Training tiefer Netzwerke

Hier siehst du eine vereinfachte Darstellung der Transformer-Architektur. Diese Visualisierung zeigt den Fluss eines Textes durch das Modell - von der Eingabe über die Embedding-Schicht durch mehrere Transformer-Blöcke bis zur endgültigen Vorhersage des nächsten Wortes.

5 Python-Beispiel für einen einfachen Transformer

Hier ist ein einfaches Beispiel, wie man mit der Hugging Face Transformers-Bibliothek einen vortrainierten Transformer verwenden kann:

```
# Einfaches Beispiel für die Verwendung eines Transformer-Modells
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Modell und Tokenizer laden
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# Eingabetext
eingabetext = "Data visualization empowers users to"

# Text in Token umwandeln
input_ids = tokenizer.encode(eingabetext, return_tensors='pt')

# Parameter für die Textgenerierung
temperatur = 0.7 # Kreativität (höher = kreativer)
top_k = 50       # Nur die 50 wahrscheinlichsten Wörter betrachten
max_laenge = 30  # Maximale Textlänge

# Text generieren
ausgabe = model.generate(
    input_ids,
    max_length=max_laenge,
    temperature=temperatur,
    top_k=top_k,
    do_sample=True
)

# Generiertes Ergebnis dekodieren und anzeigen
generierter_text = tokenizer.decode(ausgabe[0], skip_special_tokens=True)
print(generierter_text)
```

M05c - Embeddings

Stand: 03.2025

Damit Künstliche Intelligenz (KI) sinnvoll mit Sprache, Bildern oder anderen Inhalten arbeiten kann, muss sie deren Bedeutung erfassen. Allerdings verarbeitet ein Computer keine Wörter oder Bilder direkt, sondern nur Zahlen. **Embeddings** sind eine Methode, um solche Inhalte als Zahlen zu kodieren, sodass die KI Zusammenhänge und Bedeutungen erkennen kann.

1 Was sind Embeddings?

Ein **Embedding** ist eine mathematische Darstellung eines Wortes, Satzes oder Bildes in Form eines Vektors, also einer Zahlenliste. Diese Zahlen erfassen Ähnlichkeiten und Zusammenhänge zwischen verschiedenen Konzepten.

Beispiel für Sprache:

- Das Wort „King“ könnte als Zahlenvektor **[0.96, 0.92, 0.08, 0.67]** dargestellt werden.
- Das Wort „Queen“ könnte **[0.98, 0.07, 0.93, 0.71]** haben.
- Das Wort „Girl“ könnte **[0.56, 0.09, 0.91, 0.11]** haben.

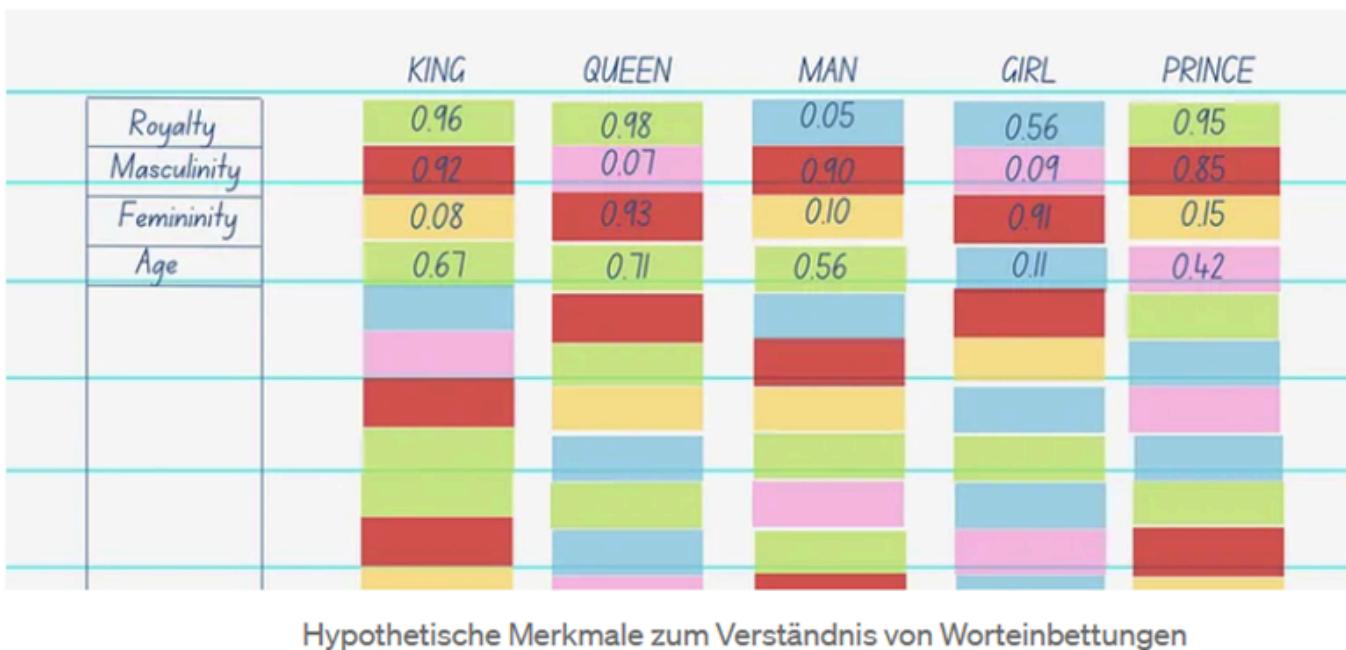
→ Die Zahlen von „King“ und „Queen“ sind **ähnlicher** als die von „Man“ und „Girl“. Dies zeigt, dass die KI die inhaltliche Nähe dieser Begriffe versteht.

Beispiel für Bilder:

- Ein Bild von einem Hund wird in Zahlen umgewandelt.
- Ein ähnliches Bild erhält einen ähnlichen Zahlenvektor.
- Dadurch kann die KI visuelle Ähnlichkeiten erkennen.

Embeddings werden nicht nur für Sprache und Bilder genutzt, sondern auch in Empfehlungssystemen für Musik, Filme oder sogar in der medizinischen Forschung zur Mustererkennung.

Hypothetisches Beispiel für die Embeddings:



Quelle: [Ein Leitfaden für Anfänger zu Word2Vec. Grundlagen von Word2Vec + Implementierung... | von Manan Suri | Medium](#)

2 Wie entstehen Embeddings?

Embeddings werden mit **künstlichen neuronalen Netzen** oder **statistischen Methoden** erzeugt. Dabei durchläuft der Prozess mehrere Schritte:

1 Daten sammeln

- Sprachmodelle nutzen große Mengen an Texten aus Büchern, Webseiten oder Artikeln.
- Bilderkennungsmodelle analysieren Millionen von Fotos mit passenden Beschreibungen.
- Musik- oder Videoplattformen sammeln Daten zu Nutzerverhalten und Inhaltsmerkmalen.

2 Daten in Zahlen umwandeln

- Wörter werden als **Vektoren** dargestellt, die Bedeutungsähnlichkeiten widerspiegeln.
- Bilder werden in **Pixelwerte** und Merkmale wie Kanten oder Farben umgerechnet.
- Musik wird anhand von Frequenzmustern und Rhythmen analysiert.

3 Neuronale Netze trainieren

- Modelle wie **Word2Vec**, **GloVe** oder **FastText** für Sprache sowie **ResNet** oder **VGG** für Bilder lernen, welche Begriffe oder Objekte ähnlich sind.
- Empfehlungssysteme analysieren, welche Songs oder Filme Nutzer häufig zusammen konsumieren.

4 Ähnlichkeiten erkennen

- Begriffe mit ähnlicher Bedeutung liegen im Zahlenraum nahe beieinander.
- Beispiel: Das Embedding für „König“ liegt näher an „Königin“ als an „Banane“.
- Bilder von Hunden liegen näher an Wölfen als an Autos.

5 Feinabstimmung (Fine-Tuning)

- Embeddings können für spezifische Anwendungen optimiert werden.
 - Beispiel: Eine KI für medizinische Analysen trainiert spezielle Embeddings für Fachbegriffe.
 - Streaming-Dienste passen ihre Embeddings an individuelle Nutzerpräferenzen an.
-

3 Positional Encoding

Die Positions kodierung fügt jedem Token-Vektor (aus der Einbettungsmatrix) Informationen über seine Position in der Sequenz hinzu. Dies geschieht durch die Kombination von Positionsinformationen und den ursprünglichen Token-Einbettungen. Ohne zusätzliche Information gäbe es keinen Unterschied zwischen:

- *Die Katze jagt den Hund* und
- *Den Hund jagt die Katze*

Die Positions kodierung ist wie ein kleiner Hinweiszettel, der sagt, welches Wort an welcher Stelle steht.

Positionskodierung im maschinellen Verständnis

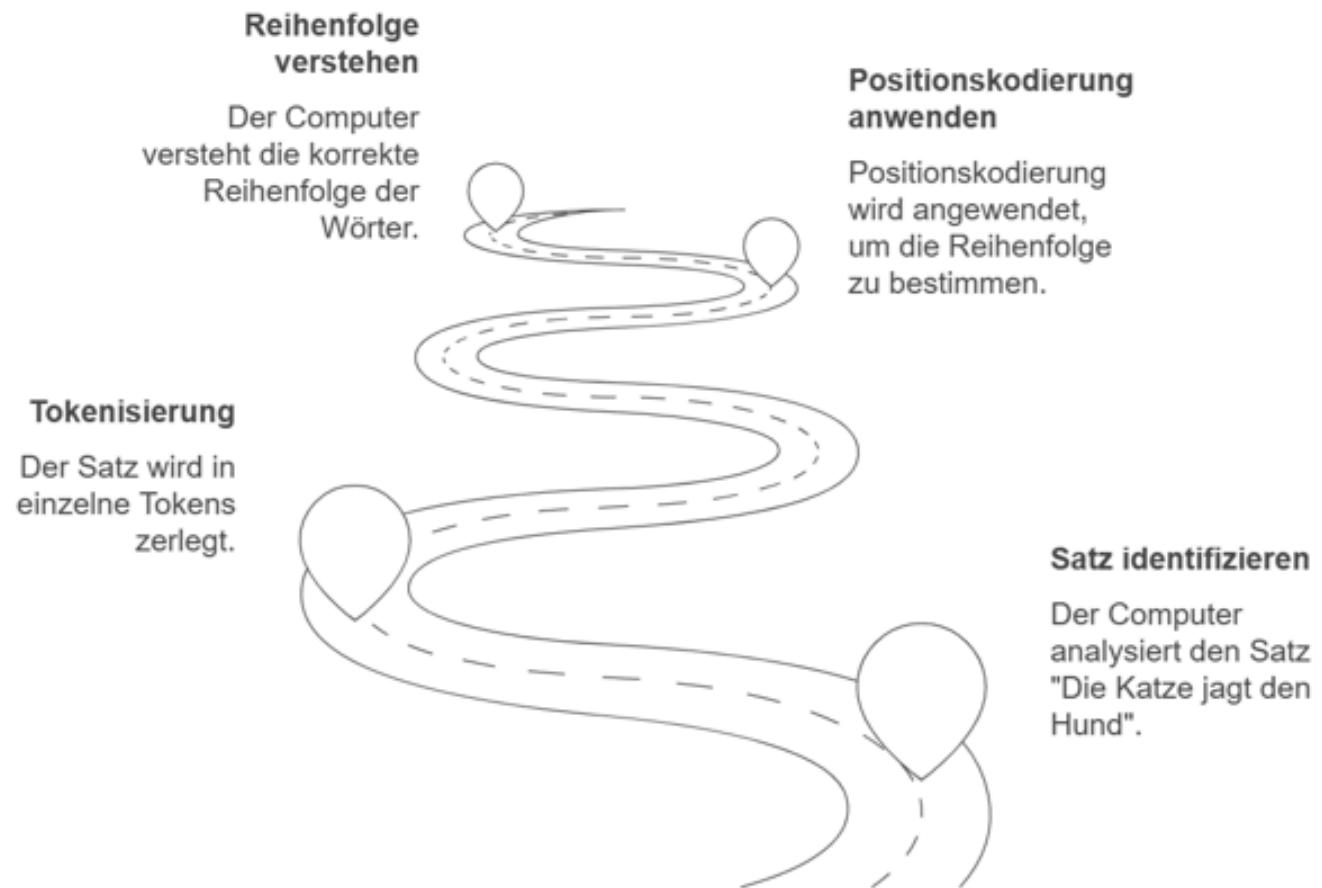


Bild mit napkin.ai erstellt

4 Embedding-Modelle

Es gibt verschiedene Einbettungsmodelle wie Word2Vec, GloVe und FastText für Wortrepräsentationen, BERT für kontextuelle Einbettungen sowie Node2Vec und LSTM-basierte Modelle für Netzwerke und Sequenzen, die jeweils auf spezifische Anwendungsfälle und Datenstrukturen optimiert sind.

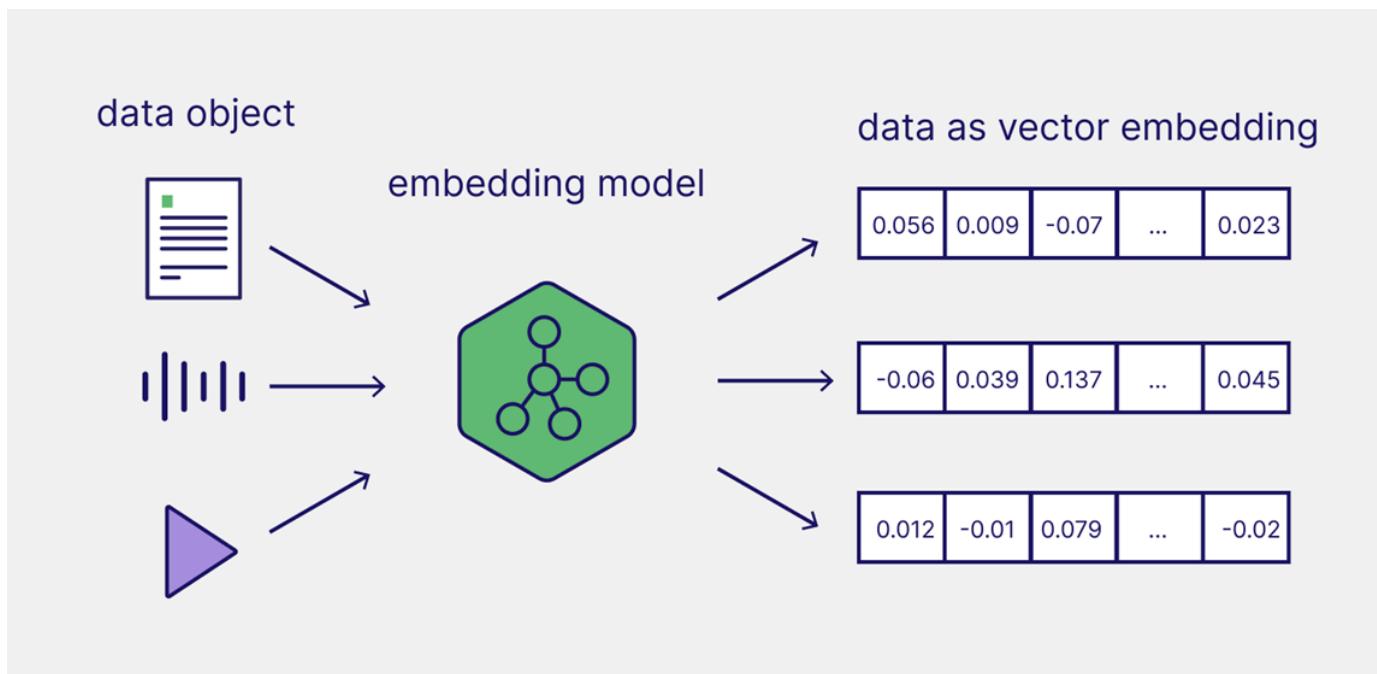


Bild: [Step-by-Step Guide to Choosing the Best Embedding Model for Your Application](#) | [Weaviate - Vector Database](#)

Übersicht Einbettungsmodelle:

Einbettungsvektor	Typische Größen	Einsatzbereich
Word2Vec	100-300 Dimensionen	Wort- und Satzhähnlichkeiten, NLP
GloVe	50, 100, 200, 300 Dimensionen	Semantische Wortbeziehungen, NLP
FastText	100-300 Dimensionen	OOV-Wortbehandlung, NLP
BERT (Basisversion)	768 Dimensionen	Kontextuelle Textverarbeitung, NLP
BERT (Large-Version)	1024 Dimensionen	Fortgeschrittene NLP-Anwendungen
text-embedding-ada-002 (OpenAI)	1536 Dimensionen	Hochqualitative semantische Suche, RAG
sentence-transformers	384-768 Dimensionen (modellabhängig)	Semantische Ähnlichkeit, Clustering, RAG
Benutzer- und Produkteinbettungen	50-200 Dimensionen	Empfehlungssysteme, Personalisierung
Einbettungen aus CNNs (VGG16)	4096 Dimensionen (für FC-Schichten)	Bildverarbeitung, Objekterkennung
Einbettungen aus CNNs (ResNet)	Variiert (tiefer mit unterschiedlichen Größen)	Bildanalyse, Feature-Extraktion
Node2Vec	64-256 Dimensionen	Graphenanalysen, soziale Netzwerke

Einbettungsvektor	Typische Größen	Einsatzbereich
LSTM-basierte Sequenzeinbettungen	50-500 Dimensionen	Zeitreihen, Sprachmodellierung, NLP

5 Training von Embedding-Modellen

Das Training von Embedding-Modellen wie Word2Vec basiert auf der Idee, dass Wörter, die in ähnlichen Kontexten vorkommen, ähnliche Bedeutungen haben. Hier wird detaillierter beschrieben, wie dieses Prinzip im Training umgesetzt wird:

Algorithmus-Auswahl

Word2Vec bietet zwei grundlegende Modelle zur Generierung von Wort-Embeddings:

1. **CBOW (Continuous Bag of Words)**: Hierbei wird das Zielwort basierend auf einem umgebenden Wortkontext vorhergesagt. Das Modell bekommt mehrere Wörter als Eingabe (den Kontext) und versucht, das Wort in der Mitte (das Zielwort) zu vorherzusagen.
2. **Skip-Gram**: Hier wird der umgekehrte Ansatz verfolgt. Ausgehend von einem Zielwort versucht das Modell, die umgebenden Kontextwörter vorherzusagen.

Training

Das Training von Word2Vec kann wie folgt zusammengefasst werden:

- **Initialisierung**: Zuerst werden Vektoren für jedes Wort zufällig initialisiert.
- **Durchlauf durch den Korpus**: Das Modell geht durch den gesamten Textkorpus, nimmt jedes Wort zusammen mit seinen Nachbarwörtern (innerhalb eines bestimmten Fensters) und führt Trainingsiterationen durch.
- **Verlustfunktion**: Die Hauptaufgabe beim Training ist die Optimierung der Verlustfunktion. Für CBOW und Skip-Gram wird oft eine Funktion verwendet, die die logarithmische Wahrscheinlichkeit maximiert, korrekte Wörter basierend auf ihren Kontexten vorherzusagen.
 - Bei **CBOW** wird der Verlust berechnet, indem die Differenz zwischen dem vorhergesagten Zielwort und dem tatsächlichen Zielwort über die Softmax-Funktion gemessen wird.
 - Beim **Skip-Gram** wird der Verlust für jedes vorhergesagte Kontextwort berechnet.
- **Backpropagation**: Mit Hilfe des Gradientenabstiegs oder ähnlicher Optimierungsalgorithmen werden die Gewichte (Wortvektoren) so angepasst, dass die Verlustfunktion minimiert wird. Dies bedeutet, dass die Wortvektoren nach und nach angepasst werden, um den wahren Kontext besser widerzuspiegeln.

Ergebnis

Das Ergebnis des Trainings ist ein Set von Vektoren, eines für jedes Wort im Vokabular. Wörter, die in ähnlichen Kontexten vorkommen, enden nahe beieinander im Vektorraum, was ihre semantische Ähnlichkeit widerspiegelt. Diese Vektoren können dann in verschiedenen nachgelagerten maschinellen Lernaufgaben verwendet werden, z.B. in der Sentiment-Analyse, bei der Klassifikation von Dokumenten oder anderen NLP-Aufgaben, die eine numerische Repräsentation von Text erfordern.

Evaluierung

Um die Qualität der Embeddings zu überprüfen, werden oft qualitative Tests wie die Suche nach den nächsten Nachbarn (ähnliche Wörter finden) oder quantitative Benchmarks (z.B. auf Datensätzen für analoge Aufgaben) durchgeführt. Diese Evaluierungen helfen dabei festzustellen, ob das Modell die Wortbedeutungen effektiv erfasst hat.

6 Warum sind Embeddings so wichtig?

- Sprachverarbeitung:** Chatbots, Übersetzungen und Textanalysen basieren auf Embeddings.
- Bilderkennung:** KI kann ähnliche Bilder oder Objekte erkennen.
- Suche & Empfehlungssysteme:** Personalisierte Vorschläge auf Plattformen wie Netflix, Spotify oder YouTube nutzen Embeddings.
- Musik- und Videovorschläge:** Streaming-Dienste berechnen Nutzerpräferenzen basierend auf Embeddings.
- Medizinische Diagnosen:** KI analysiert Krankheitsbilder und medizinische Muster durch Embeddings.
- Generative KI:** Sprachmodelle wie ChatGPT nutzen Embeddings, um kontextbezogene Antworten zu generieren.

Fazit

Embeddings sind ein zentrales Konzept in der modernen KI. Sie ermöglichen Maschinen, Bedeutungen zu erfassen, Muster zu erkennen und personalisierte Inhalte zu liefern. Ohne Embeddings wären viele heutige KI-Technologien nicht denkbar – von Chatbots über Bilderkennung bis hin zu Streaming-Diensten. Sie sind das **unsichtbare Gerüst**, das intelligente Systeme erst möglich macht.

M05d - Multi-Head-Attention

Stand: 03.2025

1 Fünf mögliche Köpfe und ihre Spezialisierungen:

1.1 Kopf 1 – Grammatikalische Struktur

- Erkennt die Beziehungen zwischen Subjekt, Prädikat und Objekt.
- Z. B. "Katze" → "jagt" (Subjekt-Prädikat), "jagt" → "Maus" (Prädikat-Objekt).

Beispiel Attention-Matrix:

Die	Katze	jagt	eine	kleine	Maus
Die	□□□□□				
Katze	□	□□□□			
jagt	□□	□□□			
eine	□□□□□				
kleine	□□□□	□			
Maus	□□□□	□			

👉 Fokus auf Verb-Beziehungen ("Katze jagt", "jagt Maus").

1.2 Kopf 2 – Semantische Beziehungen (Synonyme, Kategorien)

- Erkennt, dass "Katze" und "Maus" beides Tiere sind.
- Verstärkte Verbindungen zwischen ähnlichen Wörtern.

Beispiel Attention-Matrix:

Die	Katze	jagt	eine	kleine	Maus
Die	□□□□□				
Katze	□	□□□	□	□	□
jagt	□□□□□				
eine	□□□□□				

kleine	□□□□□
Maus	□ □ □ □ □

👉 "Katze" und "Maus" haben eine semantische Verbindung.

1.3 Kopf 3 – Bestimmung von Modifikatoren (Adjektive & Artikel)

- Achtet auf die Beziehungen zwischen Hauptwort und seinen Modifikatoren.
- Z. B. "eine" und "kleine" beziehen sich auf "Maus".

Beispiel Attention-Matrix:

Die	Katze	jagt	eine	kleine	Maus
Die	■	□□□□□			
Katze	□□□□□				
jagt	□□□□□				
eine	□□	■	□□		
kleine	□□□	■	□		
Maus	□□□□		■		

👉 "eine" und "kleine" sind wichtig für "Maus".

1.4 Kopf 4 – Fokus auf das Verb (wichtige Handlung)

- Achtet auf das Verb und dessen direkte Verknüpfung mit dem Subjekt und Objekt.
- "Katze jagt Maus" bekommt hohe Gewichtung.

Beispiel Attention-Matrix:

Die	Katze	jagt	eine	kleine	Maus
Die	□□□□□				
Katze	□□	■	□□		
jagt	□	■	■	□□	■
eine	□□□□□				
kleine	□□□□□				
Maus	□□	■	□□		

👉 Das Verb "jagt" verbindet "Katze" mit "Maus".

1.5 Kopf 5 – Langfristiger Kontext (Zusammenhang im ganzen Satz)

- Verteilt die Aufmerksamkeit gleichmäßig, um den gesamten Satzkontext zu behalten.
- Hilft z. B. dabei, das Hauptthema zu erkennen (eine Jagdszene).

Beispiel Attention-Matrix:

	Die	Katze	jagt	eine	kleine	Maus	
Die	■■■■■■						
Katze		■■■■■■					
jagt			■■■■■■				
eine				■■■■■■			
kleine					■■■■■■		
Maus						■■■■■■	

👉 Jedes Wort erhält Kontext zu allen anderen Wörtern.

1.6 Fazit

Jeder Attention-Kopf lernt eine andere Beziehung zwischen Wörtern:

1. **Grammatik** (Subjekt-Prädikat-Objekt)
2. **Semantische Ähnlichkeit** (Bedeutung von Wörtern)
3. **Adjektiv- und Artikel-Beziehungen** (Bestimmungswörter)
4. **Handlungsfokus** (Verb & beteiligte Akteure)
5. **Globaler Kontext** (Verbindung zwischen allen Wörtern)

Durch diese verschiedenen Blickwinkel wird das Modell viel leistungsfähiger, als wenn es nur eine einzige Gewichtungsmatrix hätte.

2 Code-Beispiel für eine echte Visualisierung

```
import torch
from transformers import BertTokenizer, BertModel
import matplotlib.pyplot as plt
import numpy as np

# Vortrainiertes BERT-Modell und Tokenizer laden
model_name = "bert-base-german-cased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertModel.from_pretrained(model_name, output_attentions=True)

# Beispiel-Satz
sentence = "Die Katze jagt eine kleine Maus."

# Tokenisierung und Modell-Eingabe vorbereiten
inputs = tokenizer(sentence, return_tensors="pt")
```

```

with torch.no_grad():
    outputs = model(**inputs)

# Attention-Matrizen extrahieren (Layer 0, weil es Multi-Head Attention auf
# Token-Ebene ist)
attentions = outputs.attentions[0] # Erste Schicht
attentions = attentions.squeeze().numpy() # Entferne Batch-Dimension

# Tokens für Achsenlabels
tokens = tokenizer.convert_ids_to_tokens(inputs["input_ids"].squeeze())

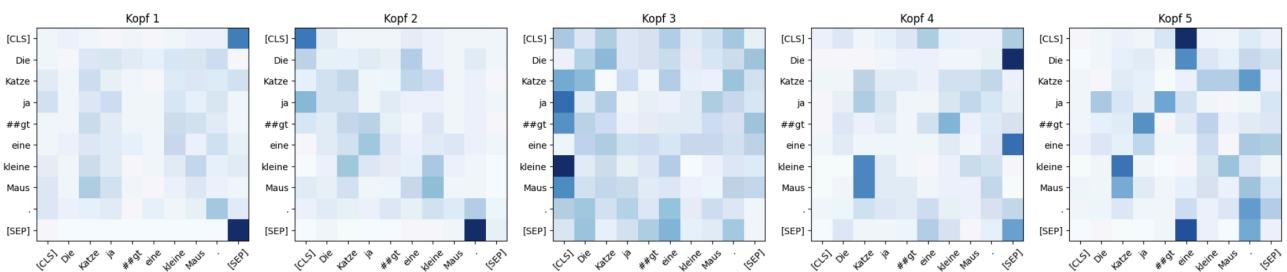
# Visualisierung der Attention-Gewichte für 5 Köpfe
fig, axes = plt.subplots(1, 5, figsize=(20, 5))

for i in range(5): # Erste 5 Köpfe visualisieren
    ax = axes[i]
    ax.imshow(attentions[i], cmap="Blues")
    ax.set_xticks(range(len(tokens)))
    ax.set_yticks(range(len(tokens)))
    ax.set_xticklabels(tokens, rotation=45)
    ax.set_yticklabels(tokens)
    ax.set_title(f"Kopf {i+1}")

plt.tight_layout()
plt.show()

```

Visual



M08a - RAG - Prozess

Stand: 03.2025

Ein RAG-System erweitert ein Large Language Model (LLM) durch den Zugriff auf externe Wissensquellen. Dies ermöglicht es, spezifische und aktuelle Informationen aus externen Dokumenten bereitzustellen, die nicht direkt im LLM enthalten sind. In diesem Kontext werden Datenschutz- und Datensicherheitsdokumente im PDF-Format über LangChain verarbeitet, um eine fundierte und faktenbasierte Generierung von Antworten zu gewährleisten.

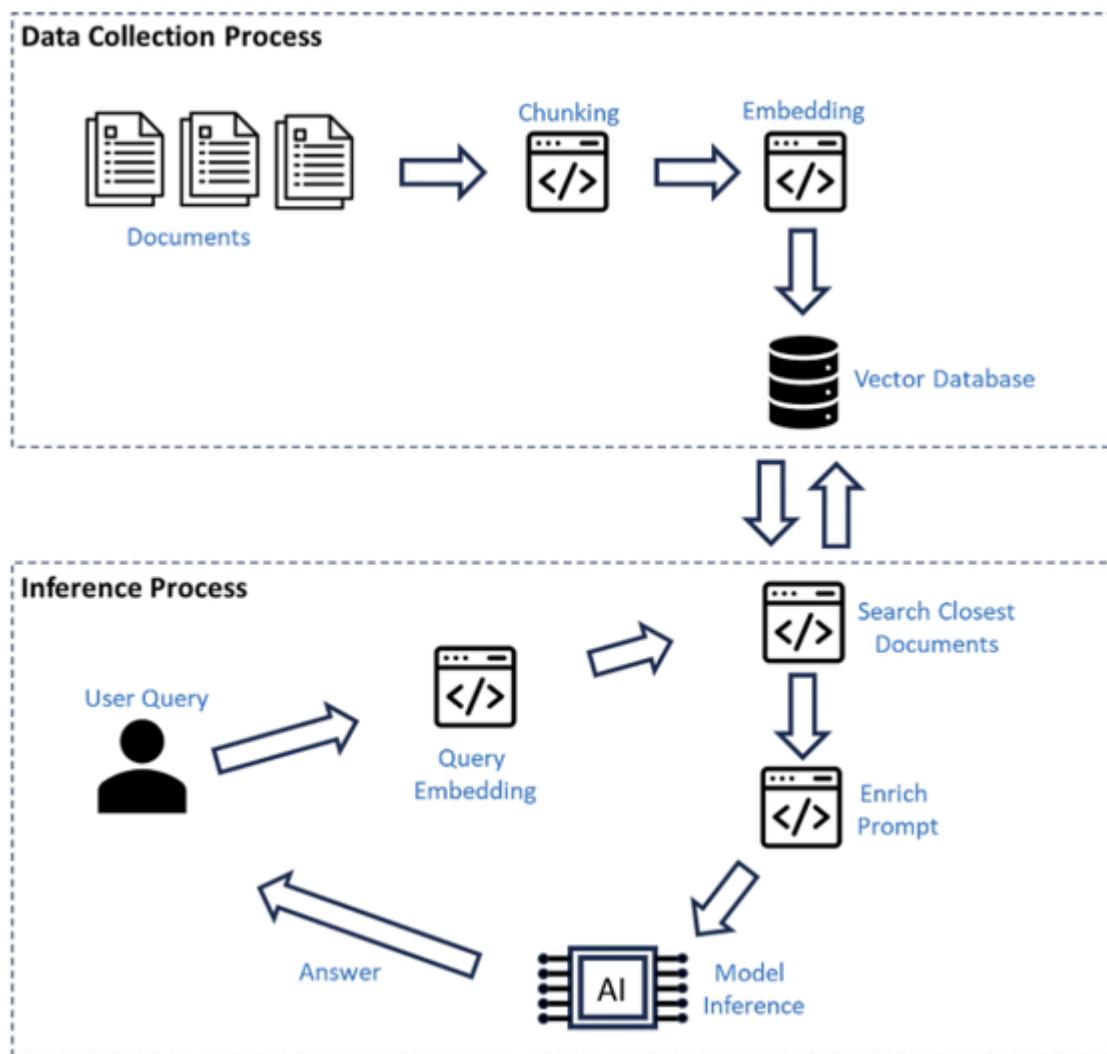


Bild: [4 Strategies to Optimize Retrieval-Augmented Generation \(RAG\). | by Carlos Jose Garces Rodrigues | AI Advances](#)

1 Textvorverarbeitung

Damit die Dokumente effizient durchsuchbar und für das LLM nutzbar gemacht werden, erfolgt eine mehrstufige Vorverarbeitung:

1. **Textnormalisierung:** Sonderzeichen werden entfernt, Zeichensetzung und Groß-/Kleinschreibung harmonisiert sowie irrelevante Informationen herausgefiltert. Dies reduziert die Komplexität und verbessert die Qualität der späteren Embeddings.
2. **Tokenisierung:** Der Text wird in Tokens zerlegt, die jeweils eine eindeutige ID erhalten. Dies ist notwendig, um die Texte numerisch weiterverarbeiten zu können.
3. **Chunking:** Die Dokumente werden in kleinere Segmente (Chunks) unterteilt, um den Kontext möglichst zu erhalten. Der **Recursive Character Text Splitter** z.B. stellt sicher, dass die Chunks eine sinnvolle Länge aufweisen, um relevante Inhalte effizient abrufbar zu machen.

2 Embeddings

Um semantische Ähnlichkeiten zwischen Textpassagen zu berechnen, werden die Chunks in numerische Vektoren umgewandelt. Hierfür wird ein **Embedding-Modell** wie `text-embedding-ada-002` oder `sentence-transformers` genutzt. Diese Embeddings werden in einer **Vektordatenbank** (z. B. FAISS) gespeichert und können mittels Ähnlichkeitsberechnungen effizient durchsucht werden.

Optimierungsmöglichkeiten:

- **Hybrid-Suche:** Kombination dichten und spärlichen Vektorrepräsentationen für präzisere Abrufresultate.
- **Dynamische Chunk-Anpassung:** Optimierung der Segmentgrößen basierend auf Textstruktur und Relevanzbewertung.

3 Retrieval und Prompting

Bei einer Benutzeranfrage wird der eingegebene Text ebenfalls in einen Vektor umgewandelt und mit den gespeicherten Embeddings verglichen. Die relevantesten Chunks werden abgerufen und als Kontext an das LLM übergeben. Die Qualität der Antwort hängt entscheidend von der **Prompt-Strukturierung** ab.

Verbesserungsstrategien:

- **Query Expansion:** Ergänzung der Suchanfrage um Synonyme oder verwandte Begriffe zur Verbesserung der Trefferquote.
- **Re-Ranking:** Gewichtung und Neuanordnung der abgerufenen Chunks nach Relevanz, um nur die relevantesten Informationen an das LLM weiterzugeben.

4 Generierung der Antwort durch das LLM

Das LLM verarbeitet die übergebenen Informationen und generiert eine fundierte Antwort. Die **Temperatur-Einstellung** bestimmt dabei die Kreativität der Antwort:

- Niedrige Werte (z. B. 0.2) → faktenbasierte, deterministische Antworten.
- Höhere Werte (z. B. 0.8) → kreativere und diversere Antworten.

Erweiterte Techniken wie **Chain-of-Thought-Prompting** können die Argumentationsführung des LLM verbessern, indem die Antwort schrittweise hergeleitet wird.

5 Komponenten der RAG-Pipeline

Ein RAG-System besteht aus mehreren miteinander verknüpften Modulen:

- **Retrieval-Modul:** Sucht relevante Textpassagen aus der Wissensquelle.
- **Prompting-Modul:** Strukturiert den Kontext für das LLM.
- **Generierungs-Modul:** Erstellt die endgültige Antwort basierend auf den abgerufenen Informationen.
- **Feedback-Modul:** Nutzerbewertungen zur Verbesserung der Antwortqualität und zur iterativen Optimierung der Pipeline.

6 Evaluierung und Optimierung

Die Qualität eines RAG-Systems wird anhand von zwei Hauptmetriken bewertet:

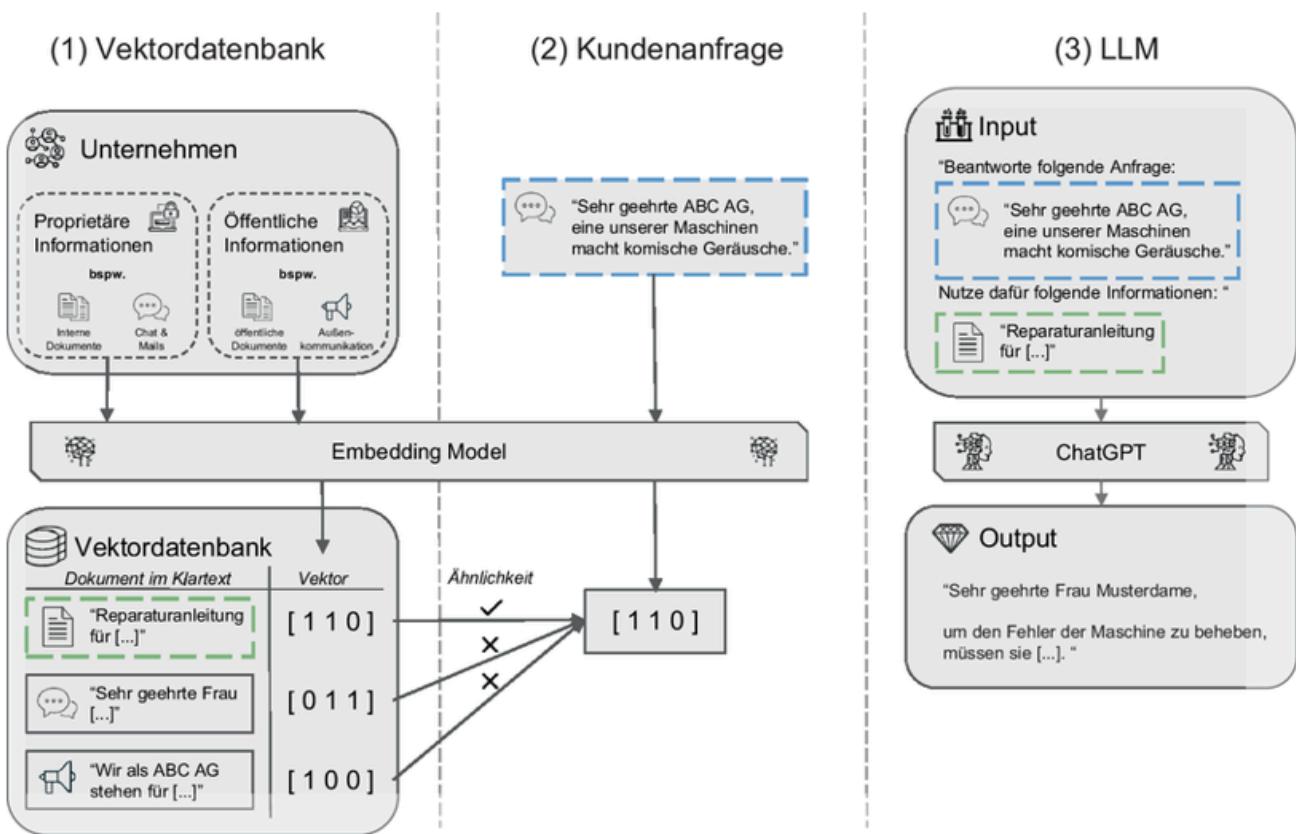
1. **Retrieval-Qualität:** Präzision und Vollständigkeit der abgerufenen Textsegmente.
2. **Generierungsqualität:** Korrektheit und Relevanz der durch das LLM erstellten Antworten.

Zusätzliche Optimierungen können durch **A/B-Tests mit verschiedenen Embedding-Modellen**, Anpassung der Chunk-Größen und Kombination verschiedener Retrieval-Strategien erreicht werden.

Fazit

Ein RAG-System bietet erhebliche Vorteile für die Generierung fundierter, aktueller Antworten durch die Integration externer Wissensquellen. Die Qualität hängt von der Wahl der Embedding-Modelle, der Retrieval-Methoden und der Prompt-Strategien ab. Durch kontinuierliche Optimierung und Evaluierung lässt sich ein effizientes und präzises System für den jeweiligen Anwendungsfall entwickeln.

7 Praxisbeispiel



Quelle: [Buxmann et al.: Die Nutzung von ChatGPT in Unternehmen, 2024](#)

M08b - Tokenizing & Chunking

Stand: 02.2025

Die effiziente Textverarbeitung beruht auf drei zentralen Elementen: der Wahl des richtigen Tokenizers, der optimalen Chunk-Größe und einer passenden Chunking-Strategie. Diese Faktoren bilden die Basis für eine erfolgreiche Dokumentenanalyse in NLP-Anwendungen. Im Folgenden erfahren Sie, wie Sie diese Parameter systematisch an die Eigenschaften Ihres Dokuments und die Anforderungen der Anwendung (z. B. Fragebeantwortung, Zusammenfassung, Code-Verarbeitung) anpassen und optimieren können.

1 Tokenizer-, Chunking- & Strategieauswahl

1.1 Dokumenttypen

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
Lange Texte	SentencePiece oder BPE	512–1024	20–30%	Semantisches & embeddingbasiertes Chunking	Diese Tokenizer zerlegen den Text in kleinere, semantisch sinnvolle Einheiten. Größere Chunks helfen, den Kontext beizubehalten und logische Einheiten in dichten Texten zu bewahren.

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
Mittel-lange Texte	WordPiece	256–512	10–20%	Semantisches Chunking	WordPiece verarbeitet gemischte Sprache gut. Semantisches Chunking fasst narrative und strukturierte Abschnitte optimal zusammen, ohne den Text zu stark zu fragmentieren.
Kurze Texte	Whitespace-/Symbol-basierte Tokenizer	50–200	0–5%	Rekursives Zeichen-Chucking (bei unklaren Grenzen)	Kurze, oft stark strukturierte Texte profitieren von kleinen Chunks. Rekursives Zeichen-Chucking kann helfen, bei fehlenden klaren Grenzen die Struktur zu wahren.

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
Code & Technische Dokumente	Whitespace- oder benutzerdefinierte symbolbasierte Tokenizer (mit funktionsspezifischen Regeln)	Basierend auf logischen Blöcken (z. B. pro Funktion oder Absatz, ca. 256 Tokens)	Variabel (idealerweise minimale Überlappung oder blockgrenzenangepasst)	Agentisches Chunking (unter Einbeziehung logischer und syntaktischer Strukturen)	Die strukturelle Integrität ist entscheidend, um die Semantik des Codes zu erhalten. Agentisches Chunking berücksichtigt funktionale Zusammenhänge und stellt die Intaktheit der Blöcke sicher.

1.2 Anwendungsszenarien

Szenario	Ziel	Empfohlenes Chunking	Empfohlene Strategie	Begründung
Antworten auf Fragen	Exakte Extraktion relevanter Passagen	Moderat bis große Chunks (512 Tokens bei langen Texten) mit hoher Überlappung (30–50%)	Kombination aus semantischem und embeddingbasiertem Chunking	Hohe Überlappung stellt sicher, dass der Kontext zwischen den Chunks nicht verloren geht. Semantische Grenzen und embeddingbasierte Analysen erfassen relevante Abschnitte präzise.
Zusammenfassungen	Verdichtung des Inhalts bei Beibehaltung der Kernaussagen	Mittlere Chunks (256 Tokens) mit moderater Überlappung (10–20%)	Semantisches Chunking	Semantisches Chunking bewahrt komplett Sinnabschnitte, sodass die Kernaussagen klar extrahiert werden können, ohne den Kontext zu verlieren.
Informationsretrieval (RAG)	Effiziente Auffindbarkeit relevanter Abschnitte	Chunks von 256–512 Tokens mit moderater Überlappung (10–20%)	Embeddingbasiertes Chunking	Embeddingbasiertes Chunking gruppier semantisch verwandte Inhalte. So werden relevante Informationen leichter auffindbar und retrieval-technisch optimal aufbereitet.

Szenario	Ziel	Empfohlenes Chunking	Empfohlene Strategie	Begründung
Named Entity Recognition (NER)	Identifikation wichtiger Entitäten (Namen, Daten usw.)	Chunks, die an Satzgrenzen ausgerichtet sind (ca. 256 Tokens) und minimale Überlappung (5–15%)	Semantisches Chunking (ggf. kombiniert mit embeddingbasierten Ansätzen)	Durch an Satzgrenzen ausgerichtete Chunks wird vermieden, dass Entitäten aufgespalten werden. Eine embeddingbasierte Analyse kann zusätzlich helfen, zusammengehörige Entitäten zu erfassen.
Textklassifikation	Zuweisung von Labels zu Dokumenten oder Abschnitten	Größere, grobere Chunks (gesamtes Dokument oder 512 Tokens) mit wenig bis keiner Überlappung	Semantisches Chunking (optional mit reduzierter Granularität)	Gröbere Unterteilungen verhindern Rauschen, während semantische Einheiten erhalten bleiben, die für die Klassifikation relevant sind.
Code-Kommentierung/Erklärung	Verständnis und Erklärung von Codeabschnitten	Chunks, die durch logische Blöcke definiert sind (z. B. pro Funktion, Modul) mit Überlappung nur, wenn notwendig (blockgrenzenbezogen)	Agentisches Chunking	Agentisches Chunking berücksichtigt syntaktische und semantische Aspekte des Codes. So bleiben logische Zusammenhänge, wie Funktionsdefinitionen, erhalten und können optimal erklärt werden.



Bevor Sie eine konkrete Implementierung starten, sollten Sie Ihre Dokumente genau analysieren, um die für Ihren Anwendungsfall optimale Kombination aus Tokenizer, Chunk-Größe, Überlappung und Chunking-Strategie auszuwählen. Eine Pilotphase mit verschiedenen Einstellungen kann helfen, den besten Ansatz zu ermitteln.

2 Beispiel

```
● ● ●

# Original Text
text = "Maschinelles Lernen ist ein spannendes Thema."

# Schritt 1: Text zu Token
text_tokens = ["Masch", "inelles", "_Lernen", "_ist", "_ein", "_spannendes", "_Thema", "."]

# Schritt 2: Token zu IDs
token_ids = [2847, 1123, 892, 345, 287, 4561, 1876, 13]

# Schritt 3: Chunking (Chunk-Größe = 4)
chunks = [
    # Chunk 1: ["Masch", "inelles", "_Lernen", "_ist"]
    [2847, 1123, 892, 345], 

    # Chunk 2: ["_ist", "_ein", "_spannendes", "_Thema"]
    [345, 287, 4561, 1876]
]
```

- Tokenizing:
 - Zerlegt Text in kleinste Einheiten (Token)
 - Diese Token werden in Zahlen (IDs) umgewandelt
 - Ein Token kann ein Wort, Teil eines Wortes oder ein Satzzeichen sein
- Chunking:
 - Gruppert die Token in verarbeitbare Blöcke
 - Beispiel: Bei max. 4096 Token pro Anfrage werden längere Texte in Chunks aufgeteilt
 - Jeder Chunk behält dabei genug Überlappung (hier 1) zum vorherigen Chunk für Kontexterhalt
- Zusammenspiel:

- Text wird erst tokenisiert (in kleinste Einheiten zerlegt)
- Die Token werden dann in Chunks gruppiert (für Verarbeitung)
- Chunks werden nacheinander verarbeitet
- LLM behält Kontext zwischen Chunks durch Überlappungen

3 Parameter- und Strategieauswahl

- **Tokenizer-Auswahl:**
 - **SentencePiece/BPE** sind ideal für lange, unstrukturierte Texte, da sie feine Subworteinheiten erzeugen und dabei semantische Bedeutung beibehalten.
 - **WordPiece** ist optimal für hybride Texte, in denen technische sowie allgemeine Sprache vorkommen.
 - **Whitespace-/Symbol-basierte Tokenizer** (oder speziell angepasste Tokenizer für Code) gewährleisten, dass die Struktur, beispielsweise in kurzen Texten oder Quellcode, erhalten bleibt.
- **Chunk-Größe und Überlappung:**
 - Die **Chunk-Größe** wird so gewählt, dass jeweils eine komplette logische Einheit erfasst wird. Längere Texte benötigen größere Chunks, während bei kurzen Texten kleinere, präzisere Segmente ausreichend sind.
 - **Überlappung** hilft dabei, Kontextinformationen am Rand der Chunks nicht zu verlieren. Für komplexe Aufgaben (wie präzise Fragebeantwortung) ist eine höhere Überlappung vorteilhaft, wohingegen bei Aufgaben wie Klassifikation geringere Überlappungen ausreichend sind.
- **Zusätzliche Chunking-Strategien:**
 - **Semantisches Chunking** zielt darauf ab, thematisch und inhaltlich zusammenhängende Abschnitte zu bilden.
 - **Rekursives Zeichen-Chucking** eignet sich, wenn keine klaren sprachlichen Grenzen vorliegen oder bei sehr strukturierten, kurzen Dokumenten.
 - **Embeddingbasiertes Chunking** nutzt Ähnlichkeiten im Einbettungsraum, um semantisch verwandte Inhalte zu gruppieren, was insbesondere bei Retrieval-Aufgaben nützlich ist.
 - **Agentisches Chunking** verwendet agentenbasierte Verfahren, um logische und syntaktische Zusammenhänge zu identifizieren – ein Ansatz, der besonders bei Code und technischen Dokumenten Vorteile bietet.
- **Praktische Rahmenbedingungen:**

- **Speicherverbrauch und Verarbeitungsgeschwindigkeit** lassen sich durch Anpassung der Chunk-Größe steuern. Kleinere Chunks reduzieren den Speicherbedarf und beschleunigen die Verarbeitung, was vor allem bei großen Datenmengen von Bedeutung ist.
- **Kosten** können durch die Optimierung der Überlappung minimiert werden. Eine zu hohe Überlappung erhöht Redundanzen und Rechenaufwand, sodass hier ein ausgewogenes Verhältnis gefunden werden muss.

Tip:



Erstellen Sie eine Checkliste für die Parameterwahl, die alle wesentlichen Aspekte – von Tokenizer-Auswahl über Chunk-Größe bis hin zur konkreten Chunking-Strategie – abdeckt. Dies unterstützt Sie dabei, systematisch vorzugehen und sicherzustellen, dass alle praktischen Rahmenbedingungen berücksichtigt werden.

4 Optimierung für verschiedene Modellgrößen

- **Große Modelle:**
 - Können größere Chunk-Größen (bis zu 1024 Tokens) verarbeiten, wodurch mehr Kontext erhalten bleibt.
 - Profitieren in Szenarien wie der Fragebeantwortung von einer höheren Überlappung (30–50%) und einer Kombination aus semantischem und embeddingbasiertem Chunking.
- **Kleinere Modelle:**
 - Sollten kleinere Chunk-Größen (z. B. 256–512 Tokens) verwenden, um den Speicherbedarf und die Verarbeitungsgeschwindigkeit zu optimieren.
 - Eine geringere Überlappung (10–20%) ist empfehlenswert, um unnötige Redundanz zu vermeiden, während dennoch ausreichend Kontext für die jeweilige Aufgabe erhalten bleibt.
- **Iterative Feinabstimmung:**
 - Es empfiehlt sich, anhand von Metriken wie F1-Score (bei Fragebeantwortung), ROUGE (bei Zusammenfassungen) und Recall@K (bei Retrieval-Aufgaben) verschiedene Einstellungen zu evaluieren und schrittweise zu optimieren.
 - Szenariospezifische Experimente können helfen, die Wahl des Tokenizers, die Chunk-Größe, die Überlappung und die jeweils verwendete Chunking-Strategie iterativ anzupassen.



Nutzen Sie automatisierte Tests und Monitoring-Tools, um die Leistung Ihrer Modelle kontinuierlich zu überwachen und dynamisch Anpassungen an den Chunking-Parametern vorzunehmen. Eine regelmäßige Evaluation ermöglicht es, auf Veränderungen im Input oder in den Anforderungen schnell zu reagieren.

5 Anhang: Checkliste Parameterwahl

- 1. Analyse der Dokumenteneigenschaften
 - Dokumenttyp identifizieren:** Lange Texte, mittel-lange Texte, kurze Texte, Code/technische Dokumente.
 - Typische Eigenschaften erfassen:** Länge, Struktur, Informationsdichte, spezielle Formatierungen (z. B. Tabellen, Codeblöcke).
- 2. Auswahl des Tokenizers
 - Sprachliche und technische Anforderungen prüfen:**
 - Lange, unstrukturierte Texte: SentencePiece oder BPE
 - Gemischte Inhalte (technisch und allgemein): WordPiece
 - Stark strukturierte Daten oder Code: Whitespace-/Symbol-basierte Tokenizer oder angepasste Tokenizer
 - Spezifische Anforderungen an Subwort-Einheiten bewerten.**
- 3. Festlegung der Chunk-Größe
 - Ziel der Chunking-Einheit definieren:** Soll ein vollständiger Satz, Absatz oder logische Einheit abgebildet werden?
 - Dokumenttyp berücksichtigen:**
 - Lange Texte: 512–1024 Tokens
 - Mittel-lange Texte: 256–512 Tokens
 - Kurze Texte: 50–200 Tokens
 - Code/technische Dokumente: Abhängig von logischen Blöcken (z. B. pro Funktion)
 - Praktische Rahmenbedingungen einbeziehen:** Speicherverbrauch und Verarbeitungsgeschwindigkeit.
- 4. Definition der Überlappung
 - Ziel des Überlappungsgrades festlegen:** Sicherstellung des Kontext-Erhalts, ohne unnötige Redundanz.
 - Empfohlene Überlappungswerte anpassen:**
 - Hohe Überlappung (30–50%) für kontext-sensitive Aufgaben wie Q&A.
 - Geringere Überlappung (0–5% bis 10–20%) für Klassifikation oder strukturierte Daten.
- 5. Auswahl der konkreten Chunking-Strategie
 - Strategie zur Erhaltung semantischer Einheiten prüfen:**

- Semantisches Chunking, um zusammenhängende inhaltliche Blöcke zu bilden.
- Alternativen in Betracht ziehen, falls keine klaren Grenzen vorliegen:**
- Rekursives Zeichen-Chucking.
- Spezialfälle für Retrieval und NER:**
- Embeddingbasiertes Chunking, um semantisch verwandte Abschnitte zu gruppieren.
- Besondere Anforderungen bei Code:**
- Agentisches Chunking, um logische und syntaktische Zusammenhänge zu berücksichtigen.
- **6. Evaluation und iterative Feinabstimmung**
 - Metriken definieren:** F1-Score, ROUGE, Recall@K usw.
 - Pilotphase durchführen:** Verschiedene Einstellungen testen und Ergebnisse vergleichen.
 - Parameter anpassen:** Basierend auf den Evaluierungsergebnissen systematisch justieren.
 - **7. Monitoring und praktische Rahmenbedingungen**
 - Automatisierte Tests und Monitoring einrichten:** Zur kontinuierlichen Überwachung der Modellleistung.
 - Kosten und Ressourcenverbrauch im Blick behalten:** Speicher, Rechenleistung und Kosten optimieren.
 - Regelmäßige Überprüfung:** Auf Veränderungen im Input oder in den Anforderungen reagieren und Parameter entsprechend anpassen.

Diese Checkliste unterstützt Sie dabei, einen strukturierten Ansatz zu verfolgen, sodass alle relevanten Parameter und praktischen Rahmenbedingungen systematisch berücksichtigt werden.

M08c - Vektordatenbanken

Stand: 02.2025

Hier ist eine vergleichende Übersicht zu **Vektordatenbanken**, die häufig im Kontext von KI-Anwendungen, insbesondere für die Speicherung und Abfrage von Embeddings (Vektoren), verwendet werden:

1 Übersicht

Aspekt	Pinecone	Weaviate	Milvus	FAISS (Facebook AI Similarity Search)	Chroma
Definition	Cloud-native Vektordatenbank für Echtzeit-Suche und Empfehlungssysteme.	Open-Source-Vektordatenbank mit integrierter NLP- und KI-Funktionalität.	Open-Source-Vektordatenbank für skalierbare Vektorähnlichkeitssuche.	Bibliothek für effiziente Ähnlichkeitssuche in großen Vektordatensätzen.	Open-Source-Vektordatenbank für KI-Anwendungen, einfach zu integrieren.
Zweck	Echtzeit-Suche und Empfehlungssysteme.	Kombiniert Vektorsuche mit strukturierten Daten und NLP-Funktionen.	Skalierbare Vektorähnlichkeitssuche für große Datensätze.	Effiziente Ähnlichkeitssuche in großen Vektordatensätzen.	Einfache Integration von Vektorsuche in KI-Anwendungen.

Aspekt	Pinecone	Weaviate	Milvus	FAISS (Facebook AI Similarity Search)	Chroma
Modellunterstützung	Unterstützt verschiedene Embedding-Modelle.	Unterstützt verschiedene Embedding-Modelle und NLP-Modelle.	Unterstützt verschiedene Embedding-Modelle.	Unterstützt verschiedene Embedding-Modelle.	Unterstützt verschiedene Embedding-Modelle.
Flexibilität	Hoch, da cloud-nativ und für Echtzeit-Anwendungen optimiert.	Hoch, durch Kombination von Vektorsuche und strukturierten Daten.	Hoch, durch Skalierbarkeit und Unterstützung großer Datensätze.	Hoch, als Bibliothek in bestehende Anwendungen integrierbar.	Hoch, durch einfache Integration und Nutzung.
Anwendungsfälle	Empfehlungssysteme, personalisierte Suche, Echtzeit-Analyse.	KI-gestützte Suche, Wissensgraphen, NLP-Anwendungen.	Skalierbare Ähnlichkeitssuche, Bild- und Textsuche.	Effiziente Suche in großen Vektordatensätzen, Forschung und Entwicklung.	KI-Anwendungen, semantische Suche, Chatbots.
Community & Support	Kommerzieller Support, wachsende Community.	Aktive Open-Source-Community, kommerzielle Unterstützung verfügbar.	Große Open-Source-Community, kommerzielle Unterstützung verfügbar.	Große Community, da von Facebook entwickelt, aber keine kommerzielle Unterstützung.	Wachsende Open-Source-Community.
Integration	Einfache Integration mit Cloud-Diensten und KI-Frameworks.	Integration mit NLP-Tools, KI-Frameworks und Datenbanken.	Integration mit KI-Frameworks und Big-Data-Tools.	Integration in bestehende Anwendungen als Bibliothek.	Einfache Integration mit KI-Frameworks wie LangChain.
Open Source	Nein	Ja	Ja	Ja	Ja

Aspekt	Pinecone	Weaviate	Milvus	FAISS (Facebook AI Similarity Search)	Chroma
Lernkurve	Mittel, aufgrund der Cloud-Integration und Echtzeit-Funktionen.	Mittel, durch Kombination von Vektorsuche und strukturierten Daten.	Mittel bis hoch, aufgrund der Skalierbarkeit und Komplexität.	Mittel, als Bibliothek mit Fokus auf Effizienz.	Niedrig bis mittel, dank einfacher Integration und Nutzung.
Beispiele	Personalisierte Produktempfehlungen, Echtzeit-Suche.	KI-gestützte Wissensgraphen, semantische Suche.	Bild- und Textsuche in großen Datensätzen.	Forschung und Entwicklung, effiziente Suche in großen Datensätzen.	Semantische Suche in Chatbots, KI-gestützte Anwendungen.

2 Zusammenfassung:

- **Pinecone**: Ideal für Echtzeit-Anwendungen und kommerzielle Nutzung, aber nicht Open Source.
- **Weaviate**: Kombiniert Vektorsuche mit strukturierten Daten und NLP, gut für KI-gestützte Anwendungen.
- **Milvus**: Skalierbare Lösung für große Datensätze, geeignet für komplexe Anwendungen.
- **FAISS**: Effiziente Bibliothek für die Suche in großen Vektordatensätzen, ideal für Forschung und Entwicklung.
- **Chroma**: Einfache Integration und Nutzung, ideal für KI-Anwendungen und Einsteiger.

Die Wahl der Vektordatenbank hängt von Ihren spezifischen Anforderungen ab:

- **Pinecone** für Echtzeit-Anwendungen und kommerzielle Nutzung.
- **Weaviate** für KI-gestützte Anwendungen mit strukturierten Daten.
- **Milvus** für skalierbare Lösungen mit großen Datensätzen.
- **FAISS** für effiziente Suche in der Forschung.
- **Chroma** für einfache Integration und KI-Anwendungen.

Fazit

Chroma überzeugt durch seine einfache Integration, intuitive Python-API und geringe Einstiegshürden, was es zur idealen Vektordatenbank für Einsteiger und schnelle Prototypen macht. Als Open-Source-Lösung bietet es zudem Kostenvorteile und Flexibilität, während es gleichzeitig leistungsfähig genug für die meisten KI-Anwendungsfälle bleibt.

M09 - Multimodal - Bild

Stand: 03.2025

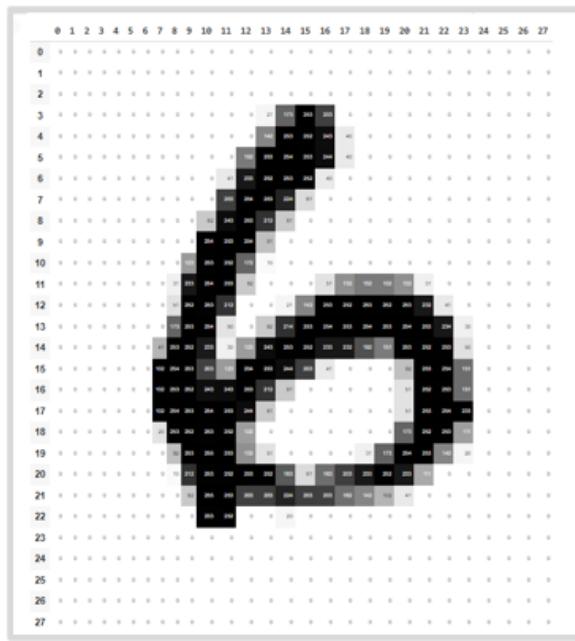
1 Grundlagen Bilderkennung

Ein Computer "sieht" ein Bild nicht wie ein Mensch. Für ihn stellt ein Bild lediglich eine Matrix aus Zahlenwerten dar, wobei jeder Wert die Intensität eines Pixels in verschiedenen Farbkanälen (meist Rot, Grün, Blau) repräsentiert. Diese Zahlenwerte werden von Algorithmen verarbeitet, um Muster und Strukturen zu erkennen, die für die Bilderkennung essenziell sind.

Der Prozess der Bilderkennung umfasst folgende Schritte:

1. **Bilderfassung:** Ein Bild wird in eine numerische Darstellung umgewandelt. Dabei werden Bildformate wie JPEG oder PNG in ein Raster aus Pixelwerten umgerechnet.
2. **Vorverarbeitung:** Das Bild wird normalisiert, skaliert und aufbereitet. Dies kann Schritte wie Rauschunterdrückung, Kontrastanpassung oder Farbnormalisierung umfassen.
3. **Merkmalsextraktion:** Wichtige Merkmale wie Kanten, Formen oder Farbverteilungen werden identifiziert. Hier können Methoden wie Histogramm-basierte Ansätze oder Kantendetektion (z. B. Sobel-Operator) angewendet werden.
4. **Klassifikation:** Basierend auf den extrahierten Merkmalen erfolgt eine Klassifikation des Bildes. Dies geschieht mithilfe von maschinellen Lernmodellen oder regelbasierten Systemen.

Raster aus Pixelwerten



2 Methoden

2.1 Traditionelle Methoden

Bei traditionellen Methoden müssen explizit Merkmale definiert werden, die als relevant gelten (z. B. Kanten, Farben, Texturen). Klassische Verfahren beinhalten Methoden wie:

- **Kantendetektion** mittels Sobel- oder Canny-Operator
- **Merkmalsvektoren** wie SIFT (Scale-Invariant Feature Transform) oder HOG (Histogram of Oriented Gradients)
- **Template Matching**, um spezifische Muster in Bildern zu finden

Diese Verfahren erfordern umfassendes domänenspezifisches Wissen und sind oft anfällig für Variationen in Beleuchtung, Perspektive oder Bildrauschen.

Merkmals-Filter: <https://editor.p5js.org/ralf.bendig/rb/full/zLXqi5u6f>

Merkmals-Filter-Anwendung: <https://editor.p5js.org/ralf.bendig.rb/full/Xi2uabjR9>

2.2 Deep Learning

Moderne Ansätze setzen auf neuronale Netze, insbesondere Convolutional Neural Networks (CNNs), die eigenständig lernen, welche Merkmale relevant sind. CNNs bestehen aus mehreren Schichten, die folgende Aufgaben erfüllen:

- **Faltungsschichten (Convolutional Layers)**: Extrahieren Merkmale durch das Anwenden von Filtern
- **Pooling-Schichten**: Reduzieren die Dimensionen und verallgemeinern die Merkmale
- **Voll verbundene Schichten (Fully Connected Layers)**: Nutzen die extrahierten Merkmale zur Klassifikation

Deep-Learning-Modelle werden auf große Datensätze trainiert, wodurch sie eine hohe Generalisierungsfähigkeit erreichen und in der Lage sind, komplexe Muster autonom zu lernen.

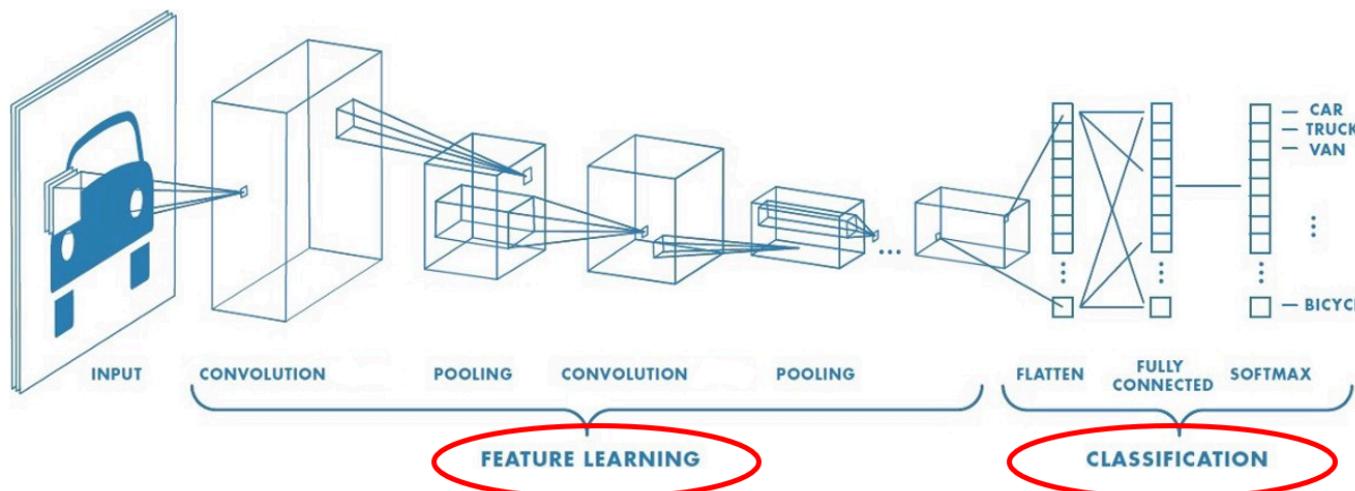


Bild: [A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science](https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a1c)

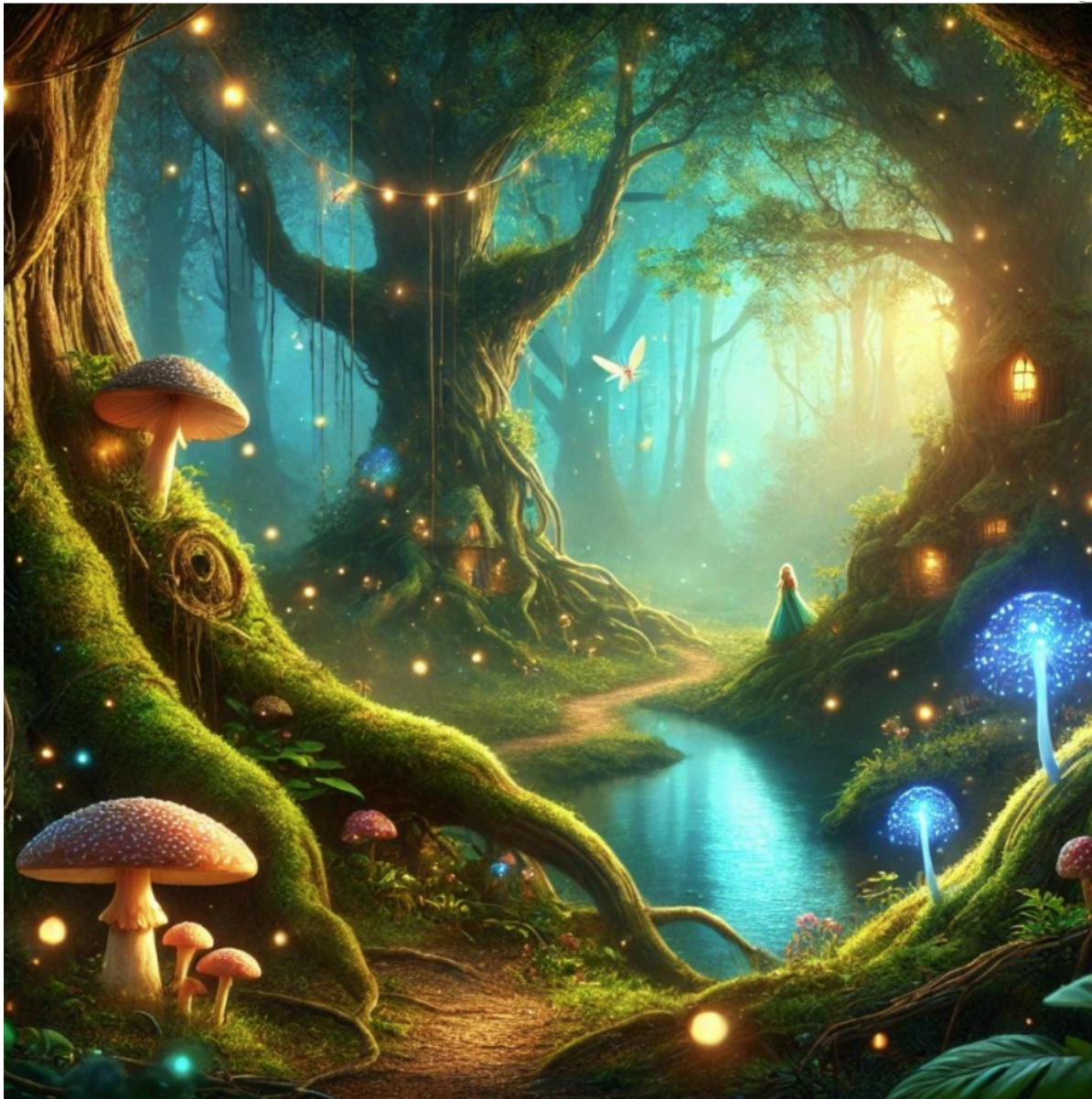
3 Bild-Modelle

3.1 Text-zu-Bild-Modelle

In diesem Abschnitt wird der Fokus auf Text-zu-Bild-Modelle gelegt, einem innovativen Bereich der künstlichen Intelligenz, der es Maschinen ermöglicht, Bilder basierend auf Textbeschreibungen zu erzeugen. Diese Modelle überbrücken die Kluft zwischen Sprache und visuellen Inhalten, indem sie eine natürliche Spracheingabe in eine vollständige Bilddarstellung umsetzen. Die Generierung von Bildern aus Text basiert auf Deep-Learning-Methoden, insbesondere durch die Kombination von natürlicher Sprachverarbeitung (NLP) und Computer Vision.

Ein zentrales Merkmal dieser Modelle ist ihre Fähigkeit, Zusammenhänge zwischen sprachlichen und visuellen Elementen zu erfassen. Durch das Training mit umfangreichen Datensätzen, die Texte mit den dazugehörigen Bildern verknüpfen, lernen sie, sprachliche Beschreibungen wie „eine Katze sitzt auf einem Fensterbrett“ mit relevanten visuellen Eigenschaften wie Formen, Texturen, Farben und räumlichen Anordnungen zu verbinden. Modelle wie DALL·E, MidJourney und Stable Diffusion haben das kreative Potenzial dieser Technologie eindrucksvoll demonstriert, indem sie sowohl fotorealistische als auch künstlerische Bilder direkt aus textlichen Vorgaben generiert haben.

Hier wird veranschaulicht, wie DALL·E ein Bild mit einem zauberhaften Märchenwald generiert:

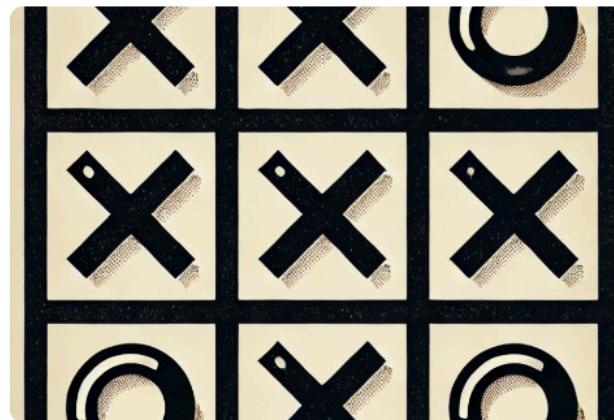


3.2 Multimodale Modelle

In diesem Abschnitt wird die spannende Welt multimodaler Modelle untersucht, die verschiedene Datentypen wie Text, Bilder, Audio und Video verarbeiten und zu einem ganzheitlichen Verständnis verknüpfen. Diese Modelle markieren einen bedeutenden Fortschritt in der KI, da sie Eingaben aus unterschiedlichen Quellen kombinieren und dadurch komplexere Aufgaben bewältigen können – ähnlich wie der Mensch Informationen aus verschiedenen Sinneseindrücken zusammenführt.

Während spezialisierte Modelle, etwa Text-zu-Bild-Systeme, bereits einzelne Verbindungen zwischen Modalitäten ermöglichen, gehen multimodale Modelle einen Schritt weiter. Sie analysieren und verstehen die Beziehungen zwischen verschiedenen Informationsarten, was vielfältige Anwendungen ermöglicht. Dazu zählen das Generieren von Bildunterschriften, das Beantworten visueller Fragen oder die Erstellung von Bildern auf Basis von Text und anderen Medieneingaben.

Ein anschauliches Beispiel für multimodale Fähigkeiten ist die Analyse eines handgezeichneten Tic-Tac-Toe-Bretts. Ein solches Modell kann ein Bild des Spiels interpretieren, die Platzierung von X und O erkennen und basierend auf den Spielregeln den Gewinner bestimmen – ohne zusätzliche textliche Informationen über das Spielfeld zu benötigen.



Wer hat das Spiel gewonnen?

Das Bild zeigt ein Tic-Tac-Toe-Spielbrett mit einem 3x3-Raster. Die meisten Felder sind mit "X" markiert, während drei Felder ein "O" enthalten.

Um den Gewinner zu bestimmen, prüfen wir, ob eine der Parteien drei ihrer Symbole in einer Reihe, Spalte oder Diagonale angeordnet hat:

- Horizontale Reihen: Alle Reihen enthalten gemischte Symbole oder nur "X".
- Vertikale Spalten: Keine Spalte enthält ausschließlich "O".
- Diagonale Linien: Keine vollständige Diagonale besteht nur aus "O".

Da in keiner Reihe, Spalte oder Diagonale drei "O" erscheinen, aber eine horizontale Reihe oder Spalte vollständig mit "X" belegt ist, hat "X" das Spiel gewonnen.



Durch die Kombination verschiedener Datentypen ermöglichen multimodale Modelle eine fortschrittlichere und intuitivere Interaktion zwischen Mensch und KI. Dies macht sie zu einem vielseitigen Werkzeug mit breiten Anwendungsmöglichkeiten – von Gesundheitswesen und Bildung bis hin zu Unterhaltung und Design.

4 Image-Embeddings

Ähnlich wie bei Text-Embeddings, die Wörter oder Sätze in einer Weise kodieren, dass semantische Ähnlichkeiten erhalten bleiben, transformieren Image-Embeddings visuelle Merkmale in eine für Maschinen lernbare Form.

Mithilfe neuronaler Netze – typischerweise Convolutional Neural Networks (CNNs) oder Transformer-Modelle wie CLIP – werden hochdimensionale Bilddaten in kompakte Vektoren umgewandelt. Diese Embeddings ermöglichen Aufgaben wie Bildähnlichkeitssuche, Clustering oder die Kombination von Bild- und Textdaten für multimodale Modelle.

[Image-Embeddings](#)

M10 - Agenten

Stand: 03.2025

1 Was sind KI-Agenten?

KI-Agenten sind eine Weiterentwicklung traditioneller KI-Systeme, insbesondere von Large Language Models (LLMs). Im Gegensatz zu reaktiven Chatbots können KI-Agenten **proaktiv und autonom Aufgaben übernehmen und ausführen**.

Laut den Texten unterscheidet Anthropic zwischen zwei Haupttypen:

- **Workflows**: Systeme mit vordefinierten Codepfaden, bei denen LLMs und Tools orchestriert werden
- **Agenten**: Systeme, in denen LLMs ihre Prozesse und Tool-Nutzung dynamisch steuern und selbstständig Aufgaben erfüllen

2 Kernkomponenten eines KI-Agenten

1. **Wahrnehmung**: Fähigkeit, Informationen aus der Umgebung aufzunehmen
2. **Gehirn**: Das LLM für logisches Denken und Schlussfolgerungen
3. **Aktionen**: Fähigkeit, Aktionen auszuführen (z.B. durch API-Aufrufe)
4. **Planung**: Möglichkeit, komplexe Aufgaben zu analysieren und Lösungsstrategien zu entwickeln
5. **Anpassungsfähigkeit**: Fähigkeit zur Weiterentwicklung durch Lernen

3 Muster für agentische Systeme

Anthropic identifiziert folgende gängige Muster:

1. **Augmentiertes LLM**: Ein LLM mit Abruf, Tools und Speicher
2. **Prompt Chaining**: Zerlegung von Aufgaben in Sequenzen
3. **Routing**: Klassifizierung und Weiterleitung von Eingaben an spezialisierte Module
4. **Parallelisierung**: Mehrere LLMs arbeiten gleichzeitig an einer Aufgabe
5. **Orchestrator-Worker**: Zentrales LLM verteilt Aufgaben an Worker-LLMs
6. **Evaluator-Optimizer**: Ein LLM generiert, ein anderes bewertet
7. **Agenten**: Eigenständige Planung, Tool-Verwendung und Feedback-Integration

4 Verfügbare Frameworks

Es gibt verschiedene Frameworks zur Entwicklung von KI-Agenten:

- **LangChain**: Populäres Framework mit vielen Tools und Integrationen
- **Semantic Kernel (Microsoft)**: SDK für LLM-Integration und zielorientierte Agenten
- **AGI2**: Open-Source-Framework für Multi-Agenten-Systeme
- **Swarm (OpenAI)**: Experimentelles Framework zur Orchestrierung
- **CrewAI**: Framework für rollenbasierte Multi-Agenten-Zusammenarbeit
- **smolagents**: Leichtgewichtige Bibliothek mit Fokus auf Open-Source-Modellen
- **AutoGPT**: Open-Source-Projekt für autonome Aufgaben

5 Praktische Anwendungsbereiche

KI-Agenten werden bereits in verschiedenen Bereichen eingesetzt:

- Kundenservice
- Gesundheitswesen
- Finanzwesen
- E-Commerce
- Forschung
- Unternehmensautomatisierung (Buchhaltung, Vertragsprüfung, Personalwesen)

6 Implementierungsempfehlungen

Basierend auf den Texten sind folgende Prinzipien wichtig:

1. **Einfachheit**: Mit der einfachsten Lösung beginnen und Komplexität nur bei Bedarf erhöhen
2. **Transparenz**: Klare Planungsschritte und nachvollziehbare Entscheidungen
3. **Strukturierte Schnittstellen**: Saubere Dokumentation und Tests
4. **Kontinuierliche Bewertung**: Iterative Verbesserung durch Feedback und Tests

7 Herausforderungen

Die Texte nennen folgende Herausforderungen:

- Variierende Genauigkeit der einzelnen Schritte
- Ethische Überlegungen bezüglich Arbeitsmarktauswirkungen
- Standardisierungsbedarf bei Schnittstellen
- Datenschutz und Sicherheit
- Transparenz und Erklärbarkeit

8 Beispiel: Einfacher KI-Agent

Hier ein einfaches Python-Beispiel, wie du mit LangChain einen KI-Agenten erstellen kannst:

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain.agents import AgentExecutor, create_openai_tools_agent
from langchain_community.tools.tavily_search import TavilySearchResults

# LLM initialisieren
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

# Tool definieren (hier: Suchfunktion)
search_tool = TavilySearchResults(max_results=3)
tools = [search_tool]

# Prompt-Template erstellen
prompt = ChatPromptTemplate.from_template("""
Du bist ein hilfreicher Assistent, der Recherche-Aufgaben durchführt.
Nutze die bereitgestellten Tools, um die Anfrage des Nutzers zu beantworten:
{input}
""")

# Agent erstellen
agent = create_openai_tools_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

# Agent ausführen
def run_agent(query):
    result = agent_executor.invoke({"input": query})
    return result["output"]

# Beispieldurchführung
if __name__ == "__main__":
    user_query = "Was sind die neuesten Entwicklungen bei KI-Agenten?"
    result = run_agent(user_query)
    print(result)
```

9 Fazit

Fazit

KI-Agenten bieten großes Potenzial für die Automatisierung komplexer Aufgaben und die intelligente Verarbeitung von Informationen. Sie können das volle Potenzial von LLMs ausschöpfen und ermöglichen personalisierte, adaptive Interaktionen.

Wie von Anthropic empfohlen, ist es sinnvoll, mit einfachen Lösungen zu beginnen und diese schrittweise zu erweitern. Der Schlüssel zum Erfolg liegt nicht in der Komplexität, sondern in der Wahl des richtigen Systems für die jeweilige Anwendung.

Referenz:

[Building Effective AI Agents | Anthropic \ Anthropic](#)

M13 - SQL RAG

Stand: 03.2025

1 Intro

Retrieval Augmented Generation (RAG) hat sich als eine revolutionäre Technik im Bereich Natural Language Processing (NLP) und Large Language Models (LLMs) erwiesen. RAG kombiniert traditionelle Sprachmodelle mit einem innovativen Retrieval-Mechanismus, der es den Sprachmodellen ermöglicht, auf eine riesige Wissensbasis zuzugreifen, um die Qualität und Relevanz ihrer Antworten zu verbessern. RAG ist besonders wertvoll, wenn detaillierte und aktuelle Informationen benötigt werden.

Die Landschaft der datengesteuerten Technologien entwickelt sich ständig weiter, und verschiedene Methoden und Techniken werden eingesetzt, um die Leistungsfähigkeit von Daten zu nutzen. Daten können entweder **strukturiert** (z. B. in SQL-Datenbanken) oder **unstrukturiert** (z. B. Textdokumente) sein. RAG-Systeme nutzen diese unterschiedlichen Datenarten, um umfassende Antworten zu generieren.

2 Was ist RAG?

RAG ist der Prozess des **Abrufens** relevanter Kontextinformationen aus einer Datenquelle und des Weitergebens dieser Informationen zusammen mit der Eingabeaufforderung des Benutzers an ein großes Sprachmodell. Diese Informationen werden verwendet, um die Ausgabe des Modells (generierter Text oder Bilder) durch **Erweitern** des Basiswissens des Modells zu **verbessern**.

RAG ist wertvoll für Anwendungsfälle, in denen das Modell Wissen benötigt, das nicht in den Informationen enthalten ist, die das Modell gelernt hat. Wenn Sie beispielsweise einen GPT erstellen, um Ihr Support-Team bei der Beantwortung von Kundenanfragen zu unterstützen, kann GPT-4 zwar Kundenprobleme mithilfe seines Basiswissens bearbeiten, aber nicht die neuesten Fakten zu Ihrem spezifischen Produkt oder Ihrer Dienstleistung kennen. Durch den Zugriff auf Ihr Ticketsystem kann ein GPT vergangene Tickets zu ähnlichen Problemen abrufen und diesen Kontext verwenden, um relevantere Antworten zu generieren.

3 Warum SQL für RAG?

Das Erstellen eines Retrieval-Augmented Generation (RAG)-Systems bringt mehrere Herausforderungen mit sich, aber SQL könnte helfen, diese zu bewältigen:

- **SQL kann helfen, komplexe Daten abzurufen** Das Abrufen relevanter Informationen aus riesigen und vielfältigen Datensätzen kann komplex sein, insbesondere beim Umgang mit unstrukturierten oder semistrukturierten Datenquellen wie Textdokumenten, Bildern oder Multimedia. Die Integration effizienter Retrieval-Mechanismen, die diese Komplexität bewältigen können, ist eine bedeutende Herausforderung. Die Abfragefunktionen von SQL ermöglichen den effizienten Abruf relevanter Informationen aus diesen Datenquellen. Durch das Generieren von SQL-Abfragen, die auf bestimmte Kriterien zugeschnitten sind, und die Nutzung erweiterter Suchfunktionen kann SQL den Datenabrufprozess optimieren und so die Komplexität des Zugriffs auf verschiedene Datensätze bewältigen.
- **SQL kann helfen, Qualitätsdaten abzurufen** Die Sicherstellung der Qualität und Relevanz der abgerufenen Daten ist entscheidend für die Generierung genauer und sinnvoller Antworten. Verrauchte oder veraltete Daten sowie irrelevante Informationen können die Leistung des RAG-Systems jedoch negativ beeinflussen. Die Entwicklung von Algorithmen zum effektiven Filtern und Ranking abgerufener Daten ist eine Herausforderung. SQL bietet Mechanismen zum Filtern und Ranking abgerufener Daten basierend auf verschiedenen Kriterien wie Zeitstempeln, Kategorien oder Relevanzwerten.
- **SQL bietet Skalierbarkeit und Flexibilität** Da Datensätze an Größe und Komplexität zunehmen, wird Skalierbarkeit zu einer großen Herausforderung für RAG-Systeme. Die Sicherstellung, dass das System mit zunehmenden Datenmengen umgehen kann und gleichzeitig Leistung und Reaktionsfähigkeit aufrechterhält, erfordert ein effizientes Architekturdesign und Optimierungsstrategien. SQL-Datenbanken sind darauf ausgelegt, riesige Mengen strukturierter Daten effizient zu verwalten. Die Integration von SQL in RAG-Systeme adressiert eine der wichtigsten Herausforderungen im Bereich der KI: die Skalierung des Retrieval-Mechanismus zur Handhabung umfangreicher Datensätze, ohne die Leistung zu beeinträchtigen. Darüber hinaus ermöglicht die Flexibilität von SQL bei der Formulierung von Abfragen RAG, komplexe Informationen abzurufen und dabei die Breite und Tiefe der während des Generierungsprozesses berücksichtigten Daten anzupassen.
- **SQL hilft beim Abrufen von Echtzeitdaten** Die Bereitstellung von Echtzeitantworten ist für viele Anwendungen von RAG-Systemen, wie z. B. Chatbots oder virtuelle Assistenten, von entscheidender Bedeutung. Das Erreichen niedriger Latenzzeiten bei gleichzeitiger Aufrechterhaltung der Qualität der generierten Inhalte stellt eine Herausforderung dar, insbesondere in Szenarien mit strengen Latenzanforderungen. Die Optimierungstechniken von SQL, wie z. B. Query-Caching und Indizierung, können die

Query-Verarbeitungszeiten erheblich reduzieren und es RAG-Systemen ermöglichen, Echtzeitantworten bereitzustellen.

4 RAG mit SQL-Workflow

Der LLM wandelt die Frage in natürlicher Sprache in eine SQL-Abfrage um, die in der SQL-Datenbank ausgeführt wird. Die Datenbank gibt Datensätze zurück, die dann vom LLM verarbeitet werden, um die endgültige Antwort zu generieren.

5 Praktische Schritte zur Integration von RAG mit Ihrer SQL-Datenbank

- **Bewerten Sie die Datenqualität:** Beginnen Sie mit der Bewertung der Qualität und Relevanz Ihrer Quelldaten und stellen Sie sicher, dass sie die Standards für effektive RAG-Interaktionen erfüllen.
- **Modellintegration:** Integrieren Sie das RAG-Modell in Ihre SQL-Datenbank und stellen Sie Verbindungen für den Echtzeit-Datenabruf und die -Analyse her.
- **JavaScript-Interaktion:** Entwickeln Sie JavaScript-Schnittstellen, die Benutzerinteraktionen mit dem integrierten RAG-System ermöglichen und natürliche Sprachabfragen ohne direkte SQL-Beteiligung ermöglichen.

6 Tipps und Tricks zur Beherrschung der RAG-SQL-Integration

Um die Leistung Ihrer RAG-SQL-Integration zu verbessern, sollten Sie die Implementierung von Caching-Mechanismen in Betracht ziehen, um häufig aufgerufene Daten vorübergehend zu speichern. Durch das Cachen von Abfrageergebnissen können Sie die Antwortzeiten verkürzen und die Gesamteffizienz des Systems verbessern. Darüber hinaus kann die Optimierung der Indizierung für wichtige Spalten innerhalb Ihrer Datenbank die Datenabrufprozesse erheblich beschleunigen und schnelle Antworten auf Benutzerabfragen gewährleisten.

7 Lokale SQL-Generierungsarchitektur

Die vorgeschlagene Architektur unterscheidet sich grundlegend von den herkömmlichen Retrieval-Augmented Generation (RAG)-Modellen. Anstatt sich auf Embedding-Vektoren zu verlassen, um Informationen abzurufen, dreht sich unser Modell um die Fähigkeit des LLM, Benutzerabfragen zu interpretieren und in SQL-Befehle zu übersetzen. Diese Befehle werden auf dem lokalen Gerät des Benutzers ausgeführt, und nur die bereinigten Ergebnisse werden an das LLM zurückgegeben.

1. **Benutzerabfrageeingabe:** Die Reise beginnt mit einer Benutzerabfrage in natürlicher Sprache, die von einfachen Anfragen bis hin zu komplexen Anweisungen für die

- Datenbankinteraktion reicht.
2. **LLM als Vermittler:** Das LLM tritt ein, nicht als Datenverarbeiter, sondern als Übersetzer, der die natürliche Sprachabfrage in einen SQL-Befehl umwandelt. Das Modell ist darauf ausgelegt, die Absicht und die erforderlichen Datenoperationen zu verstehen, ohne auf tatsächliche Daten zugreifen zu müssen.
 3. **Ausführung des SQL-Befehls lokal:** Der generierte SQL-Befehl wird in der lokalen Umgebung des Benutzers ausgeführt – sei es eine Cloud-basierte Datenbank oder eine lokale Browserdatenbank (Alasql). Diese lokale Ausführung stärkt die Sicherheit der Benutzerdaten, indem sie innerhalb der vom Benutzer gewählten Umgebung gehalten werden.
 4. **Lokale Daten zu informativer Antwort:** Nach der Ausführung empfängt das LLM die Ergebnisse, die es dann verwendet, um eine informative und benutzerfreundliche Antwort zu formulieren.
 5. **Kontinuierliches Lernen:** Das Modell ist darauf ausgelegt, aus jeder Interaktion zu lernen und das Benutzerfeedback zu nutzen, um die Genauigkeit der SQL-Übersetzung zu verfeinern, ohne persönliche Daten zu speichern.

M14 - Multimodel Audio

Stand: 03.2025

1 Grundlagen Audioerkennung

Ein Computer "hört" ein Audiosignal nicht wie ein Mensch. Für ihn stellt ein Audiosignal lediglich eine Sequenz von Zahlenwerten dar, wobei jeder Wert die Amplitude des Schalldrucks zu einem bestimmten Zeitpunkt repräsentiert. Diese Zahlenwerte werden von Algorithmen verarbeitet, um Muster und Strukturen zu erkennen, die für die Audioerkennung essenziell sind.

Der Prozess der Audioerkennung umfasst folgende Schritte:

1. **Audioerfassung:** Ein Audiosignal wird in eine numerische Darstellung umgewandelt. Dabei werden Audioformate wie WAV oder MP3 in eine Sequenz von Amplitudenwerten umgerechnet.
2. **Vorverarbeitung:** Das Signal wird normalisiert, gefiltert und aufbereitet. Dies kann Schritte wie Rauschunterdrückung, Amplitudenanpassung oder Frequenzfilterung umfassen.
3. **Merkmalsextraktion:** Wichtige Merkmale wie Frequenzspektren, Tonhöhen oder Rhythmusmuster werden identifiziert. Hier können Methoden wie Fast Fourier Transform (FFT), **Mel-Frequency Cepstral Coefficients (MFCCs)** oder Spektrogramme angewendet werden.

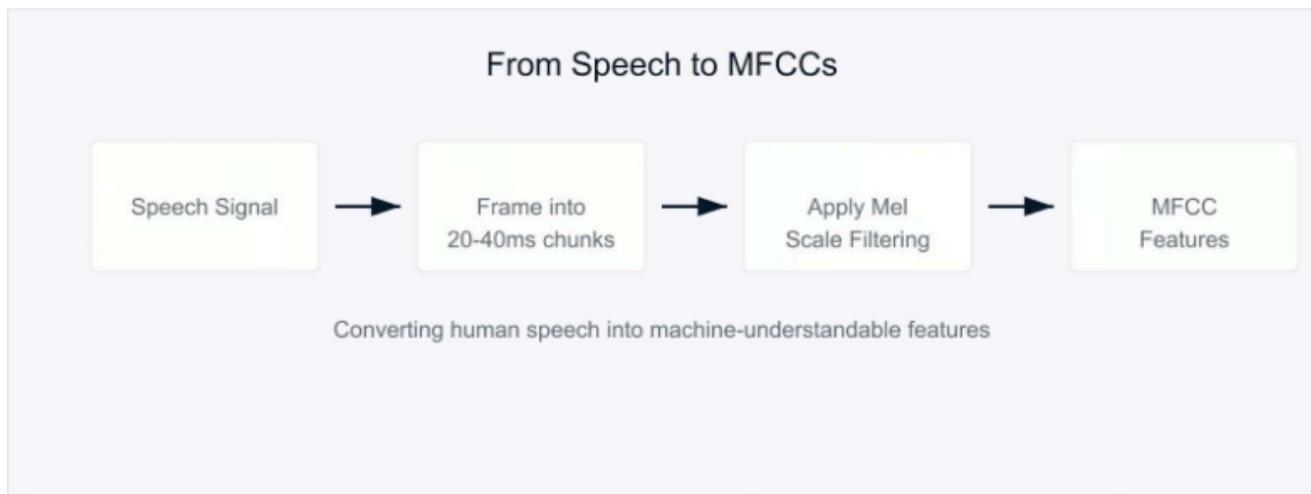


Bild: [Merkalsextraktion beim maschinellen Lernen: Ein vollständiger Leitfaden | DataCamp](#)

4. **Klassifikation:** Basierend auf den extrahierten Merkmalen erfolgt eine Klassifikation des Audiosignals. Dies geschieht mithilfe von maschinellen Lernmodellen oder regelbasierten Systemen. [MediaPipe](#))

2 Methoden

2.1 Traditionelle Methoden

Bei traditionellen Methoden müssen explizit Merkmale definiert werden, die als relevant gelten (z.B. Frequenzspektrum, Rhythmus, Tonhöhe). Klassische Verfahren beinhalten Methoden wie:

- **Frequenzanalyse** mittels Fourier-Transformation oder Wavelet-Transformation
- **Merkmalsvektoren** wie MFCCs (Mel-Frequency Cepstral Coefficients) oder Chroma-Features
- **Dynamic Time Warping (DTW)**, um zeitliche Muster in Audiosignalen zu vergleichen

Diese Verfahren erfordern umfassendes domänenspezifisches Wissen und sind oft anfällig für Variationen in Lautstärke, Tonhöhe oder Hintergrundgeräuschen.

2.2 Deep Learning

Moderne Ansätze setzen auf neuronale Netze, insbesondere Convolutional Neural Networks (CNNs) und Recurrent Neural Networks (RNNs), die eigenständig lernen, welche Merkmale relevant sind. Diese Netzarchitekturen bestehen aus mehreren Schichten, die folgende Aufgaben erfüllen:

- **CNNs mit 1D-Faltung oder 2D-Faltung auf Spektrogrammen:** Extrahieren lokale Merkmale im Zeit- oder Frequenzbereich
- **RNNs (LSTM, GRU):** Modellieren zeitliche Abhängigkeiten im Audiosignal
- **Attention-Mechanismen:** Fokussieren auf relevante Teile des Audiosignals
- **Transformer-Architekturen:** Verarbeiten lange Sequenzen mit globaler Kontext erfassung

Deep-Learning-Modelle werden auf große Datensätze trainiert, wodurch sie eine hohe Generalisierungsfähigkeit erreichen und in der Lage sind, komplexe Muster autonom zu lernen.

3 Audio-Modelle

3.1 Text-zu-Audio-Modelle

In diesem Abschnitt wird der Fokus auf Text-zu-Audio-Modelle gelegt, einem innovativen Bereich der künstlichen Intelligenz, der es Maschinen ermöglicht, Audioinhalte basierend auf Textbeschreibungen zu erzeugen. Diese Modelle überbrücken die Kluft zwischen Sprache

und akustischen Inhalten, indem sie eine natürliche Spracheingabe in eine vollständige Audiodarstellung umsetzen. Die Generierung von Audio aus Text basiert auf Deep-Learning-Methoden, insbesondere durch die Kombination von natürlicher Sprachverarbeitung (NLP) und Audio-Signalverarbeitung.

Ein zentrales Merkmal dieser Modelle ist ihre Fähigkeit, Zusammenhänge zwischen sprachlichen und akustischen Elementen zu erfassen. Durch das Training mit umfangreichen Datensätzen, die Texte mit den dazugehörigen Audiodaten verknüpfen, lernen sie, sprachliche Beschreibungen wie „ein Gewitter mit starkem Regen“ mit relevanten akustischen Eigenschaften wie Klangfarbe, Rhythmus, Tonhöhe und zeitlichen Mustern zu verbinden. Modelle wie AudioLM, MusicLM und AudioGen haben das kreative Potenzial dieser Technologie eindrucksvoll demonstriert, indem sie sowohl realistische Umgebungsgeräusche als auch musikalische Kompositionen direkt aus textlichen Vorgaben generiert haben.

Hier wird veranschaulicht, wie ein KI-Modell ein Audiobeispiel mit Naturgeräuschen eines Waldes generiert:

3.2 Multimodale Modelle

In diesem Abschnitt wird die spannende Welt multimodaler Modelle untersucht, die verschiedene Datentypen wie Text, Bilder, Audio und Video verarbeiten und zu einem ganzheitlichen Verständnis verknüpfen. Diese Modelle markieren einen bedeutenden Fortschritt in der KI, da sie Eingaben aus unterschiedlichen Quellen kombinieren und dadurch komplexere Aufgaben bewältigen können – ähnlich wie der Mensch Informationen aus verschiedenen Sinneseindrücken zusammenführt.

Während spezialisierte Modelle, etwa Text-zu-Audio-Systeme, bereits einzelne Verbindungen zwischen Modalitäten ermöglichen, gehen multimodale Modelle einen Schritt weiter. Sie analysieren und verstehen die Beziehungen zwischen verschiedenen Informationsarten, was vielfältige Anwendungen ermöglicht. Dazu zählen das Generieren von Audiobeschreibungen zu Bildern, das Synchronisieren von Lippenbewegungen in Videos mit Audioinhalten oder die Erstellung von Soundtracks auf Basis von visuellen und textuellen Eingaben.

Ein anschauliches Beispiel für multimodale Fähigkeiten ist die Analyse eines Videos mit einem musikalischen Auftritt. Ein solches Modell kann die visuellen Elemente (Instrumente, Spielbewegungen), den Audioinhalt (Melodie, Rhythmus) und kontextuelle Informationen (Musikstil, Umgebung) erfassen und diese zu einem umfassenden Verständnis der Szene verbinden.

Durch die Kombination verschiedener Datentypen ermöglichen multimodale Modelle eine fortschrittlichere und intuitivere Interaktion zwischen Mensch und KI. Dies macht sie zu einem vielseitigen Werkzeug mit breiten Anwendungsmöglichkeiten – von Gesundheitswesen (Erkennung von Krankheitssymptomen in Sprache) und Barrierefreiheit

(automatische Audiodeskription) bis hin zu Unterhaltung (adaptive Soundtracks) und Musikproduktion (KI-unterstützte Komposition).

4 Audio-Embeddings

Ähnlich wie bei Text-Embeddings und Image-Embeddings, die Textinhalte oder Bilder in einer Weise kodieren, dass semantische Ähnlichkeiten erhalten bleiben, transformieren Audio-Embeddings akustische Merkmale in eine für Maschinen lernbare Form.

Mithilfe neuronaler Netze – typischerweise Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) oder Transformer-Modelle wie Wav2Vec2 oder HuBERT – werden komplexe Audiodaten in kompakte Vektoren umgewandelt. Diese Embeddings ermöglichen Aufgaben wie Audioähnlichkeitssuche, Clustering oder die Kombination von Audio- und anderen Daten für multimodale Modelle.

Audio-Embeddings finden Anwendung in:

- Spracherkennung und Sprecheridentifikation
- Musikgenre-Klassifikation und Musikempfehlungssystemen
- Umgebungsgeräuscherkennung
- Audiosuche und -retrieval
- Multimodaler Verarbeitung mit Text und visuellen Daten

Die Qualität dieser Embeddings hängt maßgeblich von der Trainingsdatendiversität und der Modellarchitektur ab. Moderne Ansätze nutzen selbstüberwachtes Lernen, um auch aus ungelabelten Audiodaten robuste Repräsentationen zu extrahieren.

[Audio_Viz](#)

M17 - Modellauswahl - Reasoning Modelle

Stand: 04.2025

Die rasante Entwicklung der künstlichen Intelligenz (KI) und des Natural Language Processing (NLP) hat in den letzten Jahren zu bemerkenswerten Fortschritten im Bereich der Sprachmodelle geführt. Insbesondere Large Language Models (LLMs) haben anfänglich durch ihre Fähigkeit zur Textgenerierung und Mustererkennung beeindruckt.¹ Nun zeichnet sich mit dem Aufkommen von sogenannten Reasoning Modellen eine neue Phase ab, in der die Fähigkeit zum logischen Denken und zur Problemlösung in den Vordergrund rückt.¹ Dieser Bericht zielt darauf ab, Reasoning Modelle zu definieren, sie von anderen Chat-Modellen zu unterscheiden, ihre Vor- und Nachteile zu analysieren, typische Anwendungsfälle beider Modelltypen zu beleuchten und aktuelle Forschungstrends im Bereich der Reasoning Modelle zu untersuchen. Die Evolution von reiner Textgenerierung hin zu einem strukturierten Denkprozess kennzeichnet einen bedeutenden Paradigmenwechsel in den Fähigkeiten von KI-Systemen, der über bloße Sprachausgabe hinausgeht und eine Form des „Denkens“ ermöglicht. Die Beobachtung, dass die Leistungssteigerung durch bloße Skalierung traditioneller LLMs an ihre Grenzen stößt, hat die Entwicklung von Reasoning Modellen als einen vielversprechenden neuen Weg für zukünftige Fortschritte in der KI vorangetrieben.¹

1 Was sind Reasoning Modelle? Eine Definition

Reasoning Modelle sind KI-Systeme, die Natural Language Processing mit strukturierten Denkfähigkeiten verbinden.³ Ihr Design zielt nicht nur auf die Generierung von Sprache ab, sondern darauf, Probleme mit größerer Tiefe und Präzision zu durchdenken.¹ Im Kern versuchen diese Modelle, logische Prozesse zu simulieren, um Schlussfolgerungen zu ziehen oder Entscheidungen zu treffen.⁵ Dabei greifen sie häufig auf explizite Wissensrepräsentationen und Inferenzmechanismen zurück.⁵ Obwohl der Begriff „Reasoning Modell“ nicht streng definiert ist, bezieht er sich im Allgemeinen auf Modelle, die explizit strukturierte Fähigkeiten zur Problemlösung demonstrieren.⁶ Zu diesen Fähigkeiten gehören logisches Schließen (deduktiv/induktiv), mehrschrittige Problemlösung (z. B. in Mathematik, Programmierung, Rätseln), Common-Sense-Denken (Verständnis impliziten Kontexts) und kausales Denken (Verknüpfung von Ursachen und Wirkungen).⁶ Moderne Modelle erreichen dies durch architektonische Innovationen und Training auf Datensätzen, die mit Denkaufgaben angereichert sind.⁶ Der wesentliche Unterschied liegt in der Verlagerung von der reinen Vorhersage hin zu einem Prozess, bei dem Reasoning Modelle sich auf das „Wie“ und nicht nur auf das „Was“ der Antwortgenerierung konzentrieren.¹ Während traditionelle LLMs das wahrscheinlichste nächste Token vorhersagen, generieren Reasoning Modelle durch Techniken wie Chain-of-Thought Prompting explizit eine Abfolge

von Denkschritten, die menschliche Problemlösung nachahmen. Dieser interne Prozess ermöglicht die Fehlererkennung und Selbstkorrektur. Die Entwicklung von Reasoning Modellen markiert somit einen Schritt hin zu einer KI, die Aufgaben bewältigen kann, die mehr als nur Mustererkennung und Informationsabruft erfordern.¹ Die Fähigkeit, logische Schlüsse zu ziehen, Kausalitäten zu verstehen und mehrschrittige Probleme zu lösen, deutet auf ein höheres kognitives Niveau im Vergleich zu Modellen hin, die primär auf Sprachgenerierung ausgerichtet sind.

2 Was sind Chat-Modelle? Eine Abgrenzung

Chat-Modelle sind Sprachmodelle, die eine Sequenz von Nachrichten als Eingabe verwenden und Nachrichten als Ausgabe zurückgeben.⁷ Ihr Hauptaugenmerk liegt auf der Generierung von menschenähnlichem Text für Konversationszwecke.⁴ Sie sind darauf ausgelegt, menschliche Sprache oder Schrift zu verstehen und darauf zu reagieren, wodurch sie menschenähnliche Gespräche nachahmen.⁸ Diese Modelle werden auf riesigen Textkorpora trainiert, um statistische Muster zu erlernen.⁶ Typische Aufgaben von Chat-Modellen umfassen Übersetzung, Textgenerierung, Sentimentanalyse, Zusammenfassung und Beantwortung von Fragen.⁶ Obwohl sie ein gewisses Maß an Denkfähigkeit zeigen können, ist dies oft implizit und nicht das primäre Ziel ihres Designs.⁴ Chat-Modelle zeichnen sich durch ihre Fähigkeit aus, flüssigen und kontextrelevanten Text zu generieren, es fehlt ihnen jedoch möglicherweise die strukturierte Denkweise und die Fähigkeit zum logischen Schlussfolgern von Reasoning Modellen.⁹ Die primäre Funktion eines Chat-Modells besteht darin, menschenähnliche Sprache zu produzieren. Obwohl fortgeschrittene Modelle Fragen beantworten und Texte zusammenfassen können, basiert ihr zugrunde liegender Mechanismus hauptsächlich auf Mustererkennung. Reasoning Modelle hingegen sind speziell darauf ausgelegt, logische Operationen durchzuführen und Probleme in einer Reihe von Schritten zu lösen. Der Aufstieg der generativen KI hat Chat-Modelle erheblich verbessert, sie menschenähnlicher gemacht und in die Lage versetzt, ein breiteres Spektrum von Anfragen zu bearbeiten.⁸ Generative KI-Techniken, die von LLMs angetrieben werden, haben es Chat-Modellen ermöglicht, über einfache regelbasierte Antworten hinauszugehen und personalisiertere und kontextbewusstere Antworten zu generieren. Dies entspricht jedoch nicht unbedingt dem eigentlichen Denken, wie es Reasoning Modelle einsetzen.

3 Architektonische Unterschiede: Reasoning Modelle versus andere Chat-Modelle

Reasoning Modelle bauen oft auf der Architektur von LLMs auf, führen aber neue Verhaltensweisen wie strukturiertes Denken ein.¹ Ein wesentlicher Unterschied liegt in der Verwendung von Prompting-Techniken wie „Chain of Thought“ (CoT), „Tree of Thought“ (ToT) und „Graph of Thought“, um das Denken zu ermöglichen.¹ CoT fordert das Modell auf, eine Frage zu beantworten, indem es zunächst eine Kette von Denkschritten generiert.⁴ ToT verallgemeinert CoT, indem es das Modell anweist, einen oder mehrere „mögliche nächste

Schritte“ zu generieren und das Modell dann auf jeden dieser Schritte anzuwenden.⁴ Graph of Thought stellt eine weitere Verallgemeinerung dar, bei der die Denkschritte einen gerichteten azyklischen Graphen bilden.⁴ Darüber hinaus nutzen Reasoning Modelle häufig Retrieval-Augmented Generation (RAG), um Informationen aus externen Wissensquellen zu integrieren und so ihre Denkfähigkeit zu verbessern.⁴ Die Möglichkeit zur Werkzeugnutzung, die es Modellen erlaubt, externe Methoden wie Taschenrechner oder Programminterprete aufzurufen, ist ein weiteres wichtiges architektonisches Merkmal.⁴ Moderne Reasoning Modelle verwenden auch Techniken wie spärliche Aufmerksamkeit (z. B. Mistral) oder Mixture-of-Experts-Ansätze (z. B. DeepSeek), um ihre Effizienz und Leistungsfähigkeit zu steigern.⁶ Im Gegensatz dazu konzentriert sich die allgemeine Architektur anderer Chat-Modelle primär auf Transformer-Netzwerke für Sequenz-zu-Sequenz-Aufgaben.⁹ Reasoning Modelle erweitern somit die Standardarchitektur von LLMs durch spezifische Mechanismen, die darauf ausgelegt sind, strukturierte Denkprozesse zu erleichtern.¹ Techniken wie CoT sind nicht inhärent in der Basisarchitektur von LLMs vorhanden, sondern werden durch Prompting oder Feinabstimmung angewendet. Diese explizite Anleitung ermutigt das Modell, einen Denkprozess zu simulieren, was ein wesentliches Unterscheidungsmerkmal zu Standard-Chat-Modellen darstellt, die primär auf den inhärenten Mustererkennungsfähigkeiten der Transformer-Architektur beruhen. Die Fähigkeit, externe Werkzeuge und Wissensdatenbanken zu nutzen, verbessert die Denkfähigkeiten dieser Modelle erheblich.⁴ Durch die Integration mit Werkzeugen wie Taschenrechnern oder Suchmaschinen können Reasoning Modelle Einschränkungen in ihrem internen Wissen und ihren Rechenfähigkeiten überwinden. RAG erweitert dies weiter, indem es ihnen ermöglicht, auf riesige Mengen externer Informationen zuzugreifen und diese zu verarbeiten, wodurch ihre Antworten auf faktischen Daten basieren.

4 Der Einfluss der Trainingsdaten: Einblicke in die Entwicklung beider Modelltypen

Reasoning Modelle werden oft auf Datensätzen trainiert, die mit Denkaufgaben angereichert sind, wie z. B. mathematischen Problemen, Logikrätseln und Programmieraufgaben.⁶ Um die Denkfähigkeiten weiter zu verbessern, werden häufig Supervised Fine-Tuning (SFT) und Reinforcement Learning (RL) eingesetzt.³ Beim RL spielen Belohnungsmodelle eine wichtige Rolle, um den Trainingsprozess zu steuern.⁴ Im Gegensatz dazu werden andere Chat-Modelle auf riesigen Datensätzen mit allgemeinem Text und Code trainiert, um breite Sprachmuster zu erlernen.⁶ Für Reasoning Modelle ist die Qualität kuratierter Datensätze von entscheidender Bedeutung.⁶ Die spezialisierten Trainingsdaten, die für Reasoning Modelle verwendet werden, sind entscheidend, um die Fähigkeit zur strukturierten Problemlösung zu vermitteln.⁶ Während allgemeine Sprachmodelle aus einer breiten Palette von Texten lernen, benötigen Reasoning Modelle die Auseinandersetzung mit spezifischen Datentypen, die explizit logisches Denken und Problemlösung demonstrieren. Dieses gezielte Training ermöglicht es ihnen, die notwendigen Fähigkeiten für Aufgaben wie mathematische Inferenz oder logische Deduktion zu entwickeln. Techniken wie SFT und RL ermöglichen die Feinabstimmung der Modelle, um sich besser an das gewünschte

Denkverhalten anzupassen und die Leistung bei denkintensiven Aufgaben zu verbessern.³ Vorab trainierte Sprachmodelle bieten eine starke Grundlage, aber das weitere Training durch SFT mit Beispielen für Denkprozesse und RL mit Belohnungssignalen, die korrekte Denkschritte belohnen, trägt dazu bei, die Denkfähigkeiten dieser Modelle spezifisch zu entwickeln und zu verfeinern.

5 Fähigkeiten im Vergleich: Wo Reasoning Modelle glänzen und wo andere Modelle dominieren

Reasoning Modelle zeichnen sich durch ihre Fähigkeiten im logischen Denken, in der Problemlösung und im Ziehen von Schlussfolgerungen aus.¹ Sie zeigen eine verbesserte Genauigkeit bei komplexen Aufgaben, die mehrschrittige Inferenz erfordern.¹ Zudem sind sie in der Lage, nuancierte Probleme zu bearbeiten und implizite Kontexte zu verstehen.¹ Ihre Fähigkeit zur besseren faktischen Fundierung und zur Reduzierung von Halluzinationen bei Denkaufgaben ist ein weiterer Vorteil.¹ Im Gegensatz dazu liegen die Stärken anderer Chat-Modelle in der Generierung kreativer Inhalte, der Führung natürlicher klingender Gespräche, der Textzusammenfassung und der Sprachübersetzung.¹ Diese Modelle glänzen bei Aufgaben, die auf Mustererkennung und flüssiger Textgenerierung beruhen.⁹

Merkmal	Reasoning Modelle	Andere Chat-Modelle
Primäres Ziel	Problemlösung, logisches Schließen	Flüssige und ansprechende Konversation, Informationsgenerierung
Schlüsseltechniken	Chain of Thought, Tree of Thought, Graph of Thought, Werkzeugnutzung, RAG	Transformer-Architektur, Mustererkennung
Stärken	Genauigkeit bei komplexen Aufgaben, Erklärbarkeit, Umgang mit nuancierten Problemen	Generierung kreativer Inhalte, natürliche Konversation, Zusammenfassung, Übersetzung
Schwächen	Höhere Rechenkosten, langsamere Reaktionszeiten, potenzielle Fehler im logischen Denken	Begrenzte logische Inferenz, potenzielle faktische Ungenauigkeiten bei komplexen Aufgaben
Typische Anwendungsfälle	Mathematische Probleme, komplexe Fragenbeantwortung, Code-Debugging, juristische Analyse	Kundenservice, virtuelle Assistenten, Inhaltserstellung, Sprachübersetzung

Die Erklärbarkeit, die Reasoning Modelle durch Techniken wie CoT bieten, ist ein bedeutender Vorteil, insbesondere in sensiblen Anwendungen, in denen das Verständnis der

Begründung einer KI-Ausgabe entscheidend ist.¹ Im Gegensatz dazu konzentrieren sich andere Chat-Modelle primär auf die Erzeugung von menschenähnlichem Text. Die Integration externer Werkzeuge erweitert die Fähigkeiten von Reasoning Modellen über ihr internes Wissen hinaus und ermöglicht es ihnen, ein breiteres Spektrum komplexer Probleme zu bewältigen.⁴ Durch die Möglichkeit, auf Ressourcen wie Taschenrechner, Datenbanken oder APIs zuzugreifen und diese zu nutzen, können Reasoning Modelle spezialisierte Werkzeuge für Aufgaben einsetzen, die mit ihren reinen Sprachverarbeitungsfähigkeiten nicht möglich wären.

6 Vorteile von Reasoning Modellen: Logisches Denken und darüber hinaus

Die verbesserte Fähigkeit zum logischen Denken, zur Problemlösung und zum Ziehen von Schlussfolgerungen ist ein zentraler Vorteil von Reasoning Modellen.¹ Sie bieten eine höhere Genauigkeit bei komplexen, mehrschrittigen Aufgaben.¹ Ein weiterer wesentlicher Vorteil ist die erhöhte Erklärbarkeit und Interpretierbarkeit, die durch die Generierung von Denkschritten ermöglicht wird.¹ Dies führt zu einer besseren faktischen Fundierung und einer potenziellen Reduzierung von Halluzinationen in denkintensiven Bereichen.¹ Die Fähigkeit, externe Werkzeuge zur Verbesserung der Problemlösung zu nutzen, ist ebenfalls ein bedeutender Vorteil.⁴ Dies macht sie besonders geeignet für Bereiche, die Präzision und überprüfbare Lösungen erfordern.⁴ Die Erklärbarkeit, die Reasoning Modelle bieten, ist besonders in sensiblen Anwendungen von Bedeutung, in denen das Verständnis der Gründe für die Ausgabe einer KI entscheidend ist.¹ In Bereichen wie Medizin oder Recht, in denen Entscheidungen schwerwiegende Folgen haben, kann die Fähigkeit eines Reasoning Modells, seinen Denkprozess zu artikulieren, für die Überprüfung, Auditierung und den Aufbau von Vertrauen in das KI-System von unschätzbarem Wert sein. Die Integration externer Werkzeuge erweitert die Fähigkeiten von Reasoning Modellen über ihr internes Wissen hinaus und ermöglicht es ihnen, ein breiteres Spektrum komplexer Probleme zu bewältigen.⁴ Durch die Möglichkeit, auf Ressourcen wie Taschenrechner, Datenbanken oder APIs zuzugreifen und diese zu nutzen, können Reasoning Modelle spezialisierte Werkzeuge für Aufgaben einsetzen, die mit ihren reinen Sprachverarbeitungsfähigkeiten nicht möglich wären.

7 Nachteile von Reasoning Modellen: Komplexität und ihre Konsequenzen

Die potenziell höheren Rechenkosten und langsameren Reaktionszeiten aufgrund des Denkprozesses sind ein wesentlicher Nachteil von Reasoning Modellen.¹ Es besteht die Gefahr eines fehlerhaften Denkens, wenn die zugrunde liegende Logik oder Annahmen falsch sind.⁴ Im Vergleich zu Modellen, die speziell für offene Gesprächskontexte entwickelt wurden, kann die Generalisierbarkeit in solchen Szenarien begrenzt sein.⁴ Zudem ist die Notwendigkeit spezialisierter Trainingsdaten, die Denkschritte enthalten, ein weiterer Nachteil.⁴ Es besteht auch das Risiko erhöhter Halluzinationen in längeren Inferenzketten.¹

Selbst wenn der „Denkprozess“ angezeigt wird, kann eine gewisse Undurchsichtigkeit bestehen bleiben, da der Denkpfad plausibel, aber letztendlich falsch sein kann.¹ Die erhöhte Komplexität der Werkzeuge für Entwicklung und Einsatz ist ebenfalls ein Faktor.¹ Experimentelle Reasoning Modelle können auch bei einfacheren Aufgaben Inkonsistenzen aufweisen.⁶ Die erhöhten Rechenanforderungen von Reasoning Modellen können ein erhebliches Hindernis für ihre breite Akzeptanz darstellen, insbesondere in Anwendungen, die Echtzeitreaktionen erfordern.¹ Der mehrschrittige Charakter des Denkens erfordert oft mehr Rechenleistung und Zeit im Vergleich zur direkten Textgenerierung in anderen Chat-Modellen. Dies kann zu höheren Betriebskosten und längeren Latenzzeiten führen, was für nicht alle Anwendungsfälle akzeptabel ist. Trotz des Ziels einer verbesserten Genauigkeit kann die Komplexität des Denkens auch neue Wege für Fehler oder „Halluzinationen“ eröffnen, insbesondere in mehrschritten Inferenzprozessen.¹ Obwohl Reasoning Modelle darauf ausgelegt sind, zuverlässiger zu sein, können die längeren Denkketten Möglichkeiten für die Anhäufung von Fehlern schaffen, was zu falschen Schlussfolgerungen führt, selbst wenn einzelne Schritte logisch erscheinen.

Nachteil	Beschreibung
Höhere Rechenkosten	Der Denkprozess erfordert mehr Rechenleistung.
Langsamere Reaktionszeiten	Die Generierung und Auswertung von Denkschritten kann zeitaufwendiger sein.
Potenzielle Fehler im logischen Denken	Wenn der Denkprozess fehlerhaft ist oder auf falschen Annahmen beruht, ist das Ergebnis wahrscheinlich falsch.
Begrenzte Generalisierbarkeit in Gesprächen	Ihre Stärke liegt in Denkaufgaben, in offenen oder kreativen Gesprächsszenarien schneiden sie möglicherweise schlechter ab.
Spezialisierte Trainingsdaten erforderlich	Effektive Reasoning Modelle erfordern oft Datensätze, die nicht nur Antworten, sondern auch die beteiligten Denkschritte enthalten.
Potenzial für erhöhte Halluzinationsrisiken	Längere Inferenzketten können Fehler verstärken, wenn die ersten Schritte fehlerhaft sind.
Undurchsichtigkeit	Selbst wenn sie ihren „Denkprozess“ zeigen, können die bereitgestellten Denkpfade plausibel, aber letztendlich falsch sein.
Erhöhte Komplexität der Werkzeuge	Die Entwicklung und Bereitstellung von Anwendungen mit Reasoning Modellen erfordert eine sorgfältigere Verwaltung von Speicher, Token und Kosten.
Potenzielle Inkonsistenzen bei einfachen Aufgaben	Einige experimentelle Reasoning Modelle können bei einfacheren Aufgaben Inkonsistenzen aufweisen.

8 Anwendungsfälle von Reasoning Modellen: Wenn Denken gefragt ist

Typische Anwendungsfälle für Reasoning Modelle umfassen das Lösen mathematischer Textaufgaben, die Beantwortung komplexer Fragen in spezifischen Bereichen wie Wissenschaft oder Recht, das Debuggen von Code und Aufgaben im Bereich des Competitive Programming.⁴ Sie werden auch zur Analyse juristischer Dokumente, zur Strategieplanung, zur Diagnose komplexer technischer Probleme, zur Erkundung von Finanzmärkten und zum Verfassen analytischer Berichte eingesetzt.¹ Weitere Anwendungsbereiche sind die Unterstützung bei der Programmierung, die wissenschaftliche Forschung, das Lösen logischer Rätsel sowie komplexe Planungs- und Entscheidungsprozesse.⁶ Im Gesundheitswesen können sie bei der Diagnose und Behandlungsplanung helfen, in der Finanzbranche bei der Betrugserkennung und Risikoanalyse, und im Bereich Compliance bei der Einhaltung von Vorschriften.¹⁵ Reasoning Modelle sind besonders wertvoll in Bereichen, in denen Genauigkeit, logische Deduktion und die Fähigkeit, Komplexität zu bewältigen, von größter Bedeutung sind.¹ Die Fähigkeit von Reasoning Modellen, ihren „Denkprozess“ durch Techniken wie CoT darzustellen, macht sie auch für Bildungszwecke geeignet, da sie schrittweise Erklärungen liefern und das Lernen erleichtern können.¹⁴

9 Anwendungsfälle anderer Chat-Modelle: Vielseitigkeit im Einsatz

Andere Chat-Modelle finden typischerweise Anwendung in Kundenservice-Chatbots, virtuellen Assistenten für Terminplanung und Erinnerungen, der Generierung kreativer Texte, der Textzusammenfassung, der Sprachübersetzung und der Teilnahme an zwanglosen Gesprächen.¹ Sie werden auch zur Erstellung von Inhalten (Artikel, Blogbeiträge, Marketingtexte), zur Informationsbeschaffung und zur Lead-Generierung eingesetzt.¹ Weitere Anwendungsbereiche umfassen Mitarbeitersupport, kostenlose Übersetzung und 24/7-Support.¹⁶ Im Gesundheitswesen können sie bei der Symptomprüfung und Terminvereinbarung helfen, im Finanzwesen beim Abrufen von Kontoständen und bei Transaktionen, und im Einzelhandel bei Produktempfehlungen.¹⁷ Andere Chat-Modelle zeichnen sich durch ihre breite Anwendbarkeit aus, bei der flüssige und ansprechende Kommunikation im Vordergrund steht, und dienen oft als wertvolle Werkzeuge für die Automatisierung und Informationsverbreitung.¹ Die Vielseitigkeit anderer Chat-Modelle ergibt sich aus ihrem breiten Training auf diversen Textdaten, wodurch sie ein breites Spektrum an Themen und Aufgaben bearbeiten können.⁶

10 Aktuelle Forschungstrends: Die Zukunft der Reasoning Modelle

Die aktuelle Forschung im Bereich der Reasoning Modelle konzentriert sich auf die Verbesserung von Prompting-Techniken wie Tree of Thought (ToT) und Graph of Thought (GoT).⁴ Ein weiterer wichtiger Trend ist die Weiterentwicklung der Retrieval-Augmented Generation (RAG).⁴ Die Forschung zielt auch darauf ab, die Fähigkeit der Modelle zur Nutzung externer Werkzeuge zu verbessern.⁴ Supervised Fine-Tuning (SFT) mit

Reasoning-Traces ist ein weiterer aktiver Forschungsbereich.⁴ Reinforcement Learning (RL) wird intensiv zur Verbesserung der Denkfähigkeiten eingesetzt.⁴ Darüber hinaus werden Techniken wie Guided Sampling und Self-Consistency Decoding untersucht.⁴ Die Entwicklung anspruchsvollerer Benchmarks zur Bewertung der Denkfähigkeiten ist ebenfalls ein wichtiger Forschungstrend.⁴ Im Bereich der Architektur werden Hybridmodelle (die schnelles und langsames Denken kombinieren) und agentische Frameworks erforscht, ebenso wie die Verbesserung des kausalen Denkens, die Reduzierung von Halluzinationen, die Erhöhung der Transparenz und die Entwicklung spezialisierter Werkzeuge und domänenpezifischer Agenten.¹ Ein weiterer Fokus liegt auf der Effizienzsteigerung und Kostenreduktion.⁶ Die zunehmende Verfügbarkeit von Open-Source-Reasoning-Modellen ist ein wichtiger Trend.⁶ Schließlich wird an der Integration von neuro-symbolischer KI und Common-Sense-Wissen gearbeitet⁵, ebenso wie an dynamischen Lernmodellen, domänenübergreifendem Denken und dem Einsatz von Quantencomputing.¹⁵ Auch die Verbesserung der Erklärbarkeit und die Integration ethischer Überlegungen sind wichtige Forschungsziele.¹⁵ Die aktuellen Forschungstrends deuten stark darauf hin, dass die zukünftige Entwicklung von Reasoning Modellen auf die Steigerung ihrer Zuverlässigkeit, Effizienz und Erklärbarkeit abzielt, während gleichzeitig ihre Fähigkeiten durch die Integration von Werkzeugen und hybriden Ansätzen erweitert werden. Die zunehmende Verfügbarkeit von Open-Source-Reasoning-Modellen wird voraussichtlich die Innovation beschleunigen und den Zugang zu dieser fortschrittlichen Technologie demokratisieren.

11 Fazit: Reasoning Modelle im Kontext der modernen KI-Landschaft

Zusammenfassend lässt sich festhalten, dass Reasoning Modelle und andere Chat-Modelle sich primär in ihrem Fokus und ihren Fähigkeiten unterscheiden. Während Reasoning Modelle auf logisches Denken und Problemlösung ausgerichtet sind, konzentrieren sich andere Chat-Modelle auf flüssige und ansprechende Konversationen. Die Vorteile von Reasoning Modellen liegen in ihrer verbesserten Genauigkeit bei komplexen Aufgaben, ihrer erhöhten Erklärbarkeit und ihrer Fähigkeit, nuancierte Probleme zu bearbeiten. Zu ihren Nachteilen zählen höhere Rechenkosten und langsamere Reaktionszeiten. Typische Anwendungsfälle für Reasoning Modelle finden sich in Bereichen, die logische Deduktion und die Fähigkeit zur Bewältigung komplexer Probleme erfordern, wie z. B. in der Wissenschaft, Technik und im Finanzwesen. Andere Chat-Modelle hingegen eignen sich hervorragend für Kundenservice, die Erstellung von Inhalten und die allgemeine Sprachinteraktion. Die aktuellen Forschungstrends im Bereich der Reasoning Modelle deuten auf eine kontinuierliche Weiterentwicklung hin, mit dem Ziel, ihre Leistungsfähigkeit, Effizienz und Anwendbarkeit in verschiedenen Domänen zu verbessern. Reasoning Modelle stellen somit einen bedeutenden Schritt in der Evolution der künstlichen Intelligenz dar und haben das Potenzial, neue Dimensionen der Problemlösungsfähigkeiten zu erschließen.

12 Referenzen

1. The Rise of Reasoning Models: Unlocking the Next Phase of AI, Zugriff am April 4, 2025, <https://opendatascience.com/the-rise-of-reasoning-models-unlocking-the-next-phase-of-ai/>
2. The Rise of Reasoning Models: Unlocking the Next Phase of AI | by ODSC - Medium, Zugriff am April 4, 2025, <https://medium.com/@odsc/the-rise-of-reasoning-models-unlocking-the-next-phase-of-ai-8d82683cb5ec>
3. en.wikipedia.org, Zugriff am April 4, 2025, https://en.wikipedia.org/wiki/Reasoning_language_model#:~:text=Reasoning%20language%20models%20are%20artificial,initialized%20with%20pretrained%20language%20models.
4. Reasoning language model - Wikipedia, Zugriff am April 4, 2025, https://en.wikipedia.org/wiki/Reasoning_language_model
5. What is the Difference Between Reasoning Models and Other AI ..., Zugriff am April 4, 2025, <https://blog.stackademic.com/what-is-the-difference-between-reasoning-models-and-other-ai-models-35d0cdbfb5ae>
6. Exploring Reasoning Models in AI Marketplace, Feb 25 | dasarpAI, Zugriff am April 4, 2025, <https://dasarpai.com/dsblog/exploring-reasoning-models>
7. Chat models | 🦜 LangChain, Zugriff am April 4, 2025, <https://python.langchain.com/docs/integrations/chat/>
8. What are NLP chatbots and how do they work? - Zendesk, Zugriff am April 4, 2025, <https://www.zendesk.com/blog/nlp-chatbot/>
9. AI Reasoning Models Emerge as Key Differentiator for Business, Zugriff am April 4, 2025, <https://www.pymnts.com/artificial-intelligence-2/2025/ais-dual-nature-reasoning-models-emerge-as-key-differentiator-for-business/>
10. Advancing Reasoning in Large Language Models: Promising Methods and Approaches, Zugriff am April 4, 2025, <https://arxiv.org/html/2502.03671v1>
11. How does reasoning improve NLP models? - Milvus, Zugriff am April 4, 2025, <https://milvus.io/ai-quick-reference/how-does-reasoning-improve-nlp-models>
12. AI Reasoning Models: OpenAI o3-mini, o1-mini, and DeepSeek R1, Zugriff am April 4, 2025, <https://www.backblaze.com/blog/ai-reasoning-models-openai-o3-mini-o1-mini-and-deepseek-r1/>
13. NLP vs LLM: A Comprehensive Guide to Understanding Key ..., Zugriff am April 4, 2025, <https://medium.com/@vaniukov.s/nlp-vs-llm-a-comprehensive-guide-to-understanding-key-differences-0358f6571910>
14. Reasoning in large language models: a dive into NLP logic - Toloka, Zugriff am April 4, 2025, <https://toloka.ai/blog/reasoning-in-large-language-models-a-dive-into-nlp-logic/>
15. What is Reasoning in AI? Types and Applications in 2025 - Aisera, Zugriff am April 4, 2025, <https://aisera.com/blog/ai-reasoning/>
16. The Ultimate Guide to NLP Chatbots in 2025 - Botpress, Zugriff am April 4, 2025, [https://botpress.com/blog/nlp-chatbot](<https://botpress.com/blog/nlp-chatbot>)
17. A Comprehensive Guide to NLP Chatbots| Consensus, Zugriff am April 4, 2025, <https://www.consensus.com/blog/guide-to-nlp-chatbots/>

M17b - Reasoning models

Reasoning Models - OpenAI

Quelle: [Reasoning models - OpenAI API](#)

Explore advanced reasoning and problem-solving models.

Reasoning models, like OpenAI o1 and o3-mini, are new large language models trained with reinforcement learning to perform complex reasoning. Reasoning models [think before they answer](#), producing a long internal chain of thought before responding to the user. Reasoning models excel in complex problem solving, coding, scientific reasoning, and multi-step planning for agentic workflows.

As with our GPT models, we provide both a smaller, faster model (o3-mini) that is less expensive per token, and a larger model (o1) that is somewhat slower and more expensive, but can often generate better responses for complex tasks, and generalize better across domains.

The new [o1-pro model](#) has unique features like making multiple model generation turns before generating a response. To support this and other advanced API features in the future, this model is currently only available in the [Responses API](#).

1 Get started with reasoning

Reasoning models can be used through the [Responses API](#) as seen here.

Using a reasoning model in the Responses API

```
import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `

Write a bash script that takes a matrix represented as a string with
format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
`;

const response = await openai.responses.create({
  model: "o3-mini",
  reasoning: { effort: "medium" },
})
```

```



```

console.log(response.output_text);
```


```

```

from openai import OpenAI

client = OpenAI()

prompt = """
Write a bash script that takes a matrix represented as a string with
format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
"""

response = client.responses.create(
    model="o3-mini",
    reasoning={"effort": "medium"},
    input=[
        {
            "role": "user",
            "content": prompt
        }
    ]
)

print(response.output_text)
```

```

curl https://api.openai.com/v1/responses \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $OPENAI_API_KEY" \
-d '{
    "model": "o3-mini",
    "reasoning": {"effort": "medium"},
    "input": [
        {
            "role": "user",
            "content": "Write a bash script that takes a matrix represented as a
string with format \"[1,2],[3,4],[5,6]\" and prints the transpose in the
same format."
        }
    ]
}'
```

Reasoning effort

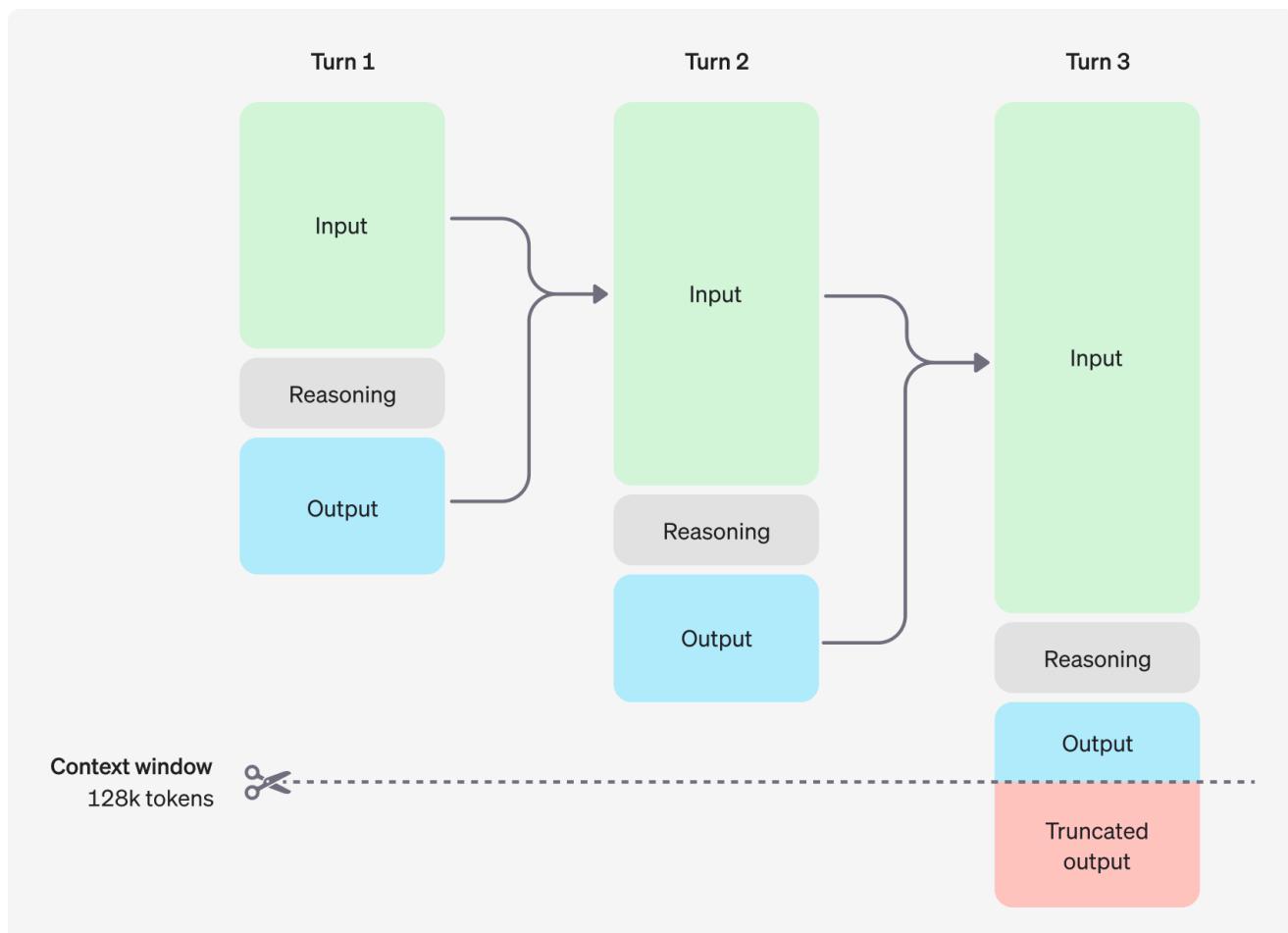
In the example above, the `reasoning.effort` parameter is used to give the model guidance on how many reasoning tokens it should generate before creating a response to the prompt.

You can specify one of `low`, `medium`, or `high` for this parameter, where `low` will favor speed and economical token usage, and `high` will favor more complete reasoning at the cost of more tokens generated and slower responses. The default value is `medium`, which is a balance between speed and reasoning accuracy.

2 How reasoning works

Reasoning models introduce **reasoning tokens** in addition to input and output tokens. The models use these reasoning tokens to "think", breaking down their understanding of the prompt and considering multiple approaches to generating a response. After generating reasoning tokens, the model produces an answer as visible completion tokens, and discards the reasoning tokens from its context.

Here is an example of a multi-step conversation between a user and an assistant. Input and output tokens from each step are carried over, while reasoning tokens are discarded.



While reasoning tokens are not visible via the API, they still occupy space in the model's context window and are billed as [output tokens](#).

Managing the context window

It's important to ensure there's enough space in the context window for reasoning tokens when creating responses. Depending on the problem's complexity, the models may generate anywhere from a few hundred to tens of thousands of reasoning tokens. The exact number of reasoning tokens used is visible in the [usage object of the response object](#), under `output_tokens_details`:

```
{
  "usage": {
    "input_tokens": 75,
    "input_tokens_details": {
      "cached_tokens": 0
    },
    "output_tokens": 1186,
    "output_tokens_details": {
      "reasoning_tokens": 1024
    },
    "total_tokens": 1261
  }
}
```

Context window lengths are found on the [model reference page](#), and will differ across model snapshots.

Controlling costs

To manage costs with reasoning models, you can limit the total number of tokens the model generates (including both reasoning and final output tokens) by using the `max_output_tokens` parameter.

Allocating space for reasoning

If the generated tokens reach the context window limit or the `max_output_tokens` value you've set, you'll receive a response with a `status` of `incomplete` and `incomplete_details` with `reason` set to `max_output_tokens`. This might occur before any visible output tokens are produced, meaning you could incur costs for input and reasoning tokens without receiving a visible response.

To prevent this, ensure there's sufficient space in the context window or adjust the `max_output_tokens` value to a higher number. OpenAI recommends reserving at least 25,000 tokens for reasoning and outputs when you start experimenting with these models. As you become familiar with the number of reasoning tokens your prompts require, you can adjust this buffer accordingly.

Handling incomplete responses

```

import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `
Write a bash script that takes a matrix represented as a string with
format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
`;

const response = await openai.responses.create({
  model: "o3-mini",
  reasoning: { effort: "medium" },
  input: [
    {
      role: "user",
      content: prompt,
    },
  ],
  max_output_tokens: 300,
});

if (
  response.status === "incomplete" &&
  response.incomplete_details.reason === "max_output_tokens"
) {
  console.log("Ran out of tokens");
  if (response.output_text?.length > 0) {
    console.log("Partial output:", response.output_text);
  } else {
    console.log("Ran out of tokens during reasoning");
  }
}

```

```

from openai import OpenAI

client = OpenAI()

prompt = """
Write a bash script that takes a matrix represented as a string with
format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
"""

response = client.responses.create(
  model="o3-mini",
  reasoning={"effort": "medium"},
  input=[
    {
      "role": "user",

```

```

        "content": prompt
    }
],
max_output_tokens=300,
)

if response.status == "incomplete" and response.incomplete_details.reason ==
"max_output_tokens":
    print("Ran out of tokens")
if response.output_text:
    print("Partial output:", response.output_text)
else:
    print("Ran out of tokens during reasoning")

```

3 Advice on prompting

There are some differences to consider when prompting a reasoning model versus prompting a GPT model. Generally speaking, reasoning models will provide better results on tasks with only high-level guidance. This differs somewhat from GPT models, which often benefit from very precise instructions.

- A reasoning model is like a senior co-worker—you can give them a goal to achieve and trust them to work out the details.
- A GPT model is like a junior coworker—they'll perform best with explicit instructions to create a specific output.

For more information on best practices when using reasoning models, [refer to this guide](#).

Prompt examples

Coding (refactoring)

OpenAI o-series models are able to implement complex algorithms and produce code. This prompt asks o1 to refactor a React component based on some specific criteria.

Refactor code

```

import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `

Instructions:
- Given the React component below, change it so that nonfiction books have red
text.
- Return only the code in your reply

```

- Do not include any additional formatting, such as markdown code blocks
- For formatting, use four space tabs, and do not allow any lines of code to exceed 80 columns

```

const books = [
  { title: 'Dune', category: 'fiction', id: 1 },
  { title: 'Frankenstein', category: 'fiction', id: 2 },
  { title: 'Moneyball', category: 'nonfiction', id: 3 },
];

export default function BookList() {
  const listItems = books.map(book =>
    <li>
      {book.title}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
` .trim();

const response = await openai.responses.create({
  model: "o3-mini",
  input: [
    {
      role: "user",
      content: prompt,
    },
  ],
});
console.log(response.output_text);

```

```
from openai import OpenAI
```

```
client = OpenAI()
```

```
prompt = """
```

Instructions:

- Given the React component below, change it so that nonfiction books have red
- text.
- Return only the code in your reply
- Do not include any additional formatting, such as markdown code blocks
- For formatting, use four space tabs, and do not allow any lines of code to exceed 80 columns

```

const books = [
  { title: 'Dune', category: 'fiction', id: 1 },
  { title: 'Frankenstein', category: 'fiction', id: 2 },
  { title: 'Moneyball', category: 'nonfiction', id: 3 },
];

export default function BookList() {
  const listItems = books.map(book =>
    <li>
      {book.title}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
"""

response = client.responses.create(
  model="o3-mini",
  input=[{
    "role": "user",
    "content": prompt,
  }]
)
print(response.output_text)

```

Coding (planning)

OpenAI o-series models are also adept in creating multi-step plans. This example prompt asks o1 to create a filesystem structure for a full solution, along with Python code that implements the desired use case.

Plan and create a Python project

```

import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `

I want to build a Python app that takes user questions and looks
them up in a database where they are mapped to answers. If there
is close match, it retrieves the matched answer. If there isn't,

```

it asks the user to provide an answer and stores the question/answer pair in the database. Make a plan for the directory structure you'll need, then return each file in full. Only supply your reasoning at the beginning and end, not throughout the code.

```
'trim();
```

```
const response = await openai.responses.create({
    model: "o3-mini",
    input: [
        {
            role: "user",
            content: prompt,
        },
    ],
});
```

```
console.log(response.output_text);
```

```
from openai import OpenAI

client = OpenAI()

prompt = """
I want to build a Python app that takes user questions and looks
them up in a database where they are mapped to answers. If there
is close match, it retrieves the matched answer. If there isn't,
it asks the user to provide an answer and stores the
question/answer pair in the database. Make a plan for the directory
structure you'll need, then return each file in full. Only supply
your reasoning at the beginning and end, not throughout the code.
"""


```

```
response = client.responses.create(
    model="o3-mini",
    input=[
        {
            "role": "user",
            "content": prompt,
        }
    ]
)

print(response.output_text)
```

STEM Research (*Science, Technology, Engineering and Mathematics*)

OpenAI o-series models have shown excellent performance in STEM research. Prompts asking for support of basic research tasks should show strong results.

Ask questions related to basic scientific research

```
import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `
What are three compounds we should consider investigating to
advance research into new antibiotics? Why should we consider
them?
`;

const response = await openai.responses.create({
  model: "o3-mini",
  input: [
    {
      role: "user",
      content: prompt,
    },
  ],
});

console.log(response.output_text);
```

```
from openai import OpenAI

client = OpenAI()

prompt = """
What are three compounds we should consider investigating to
advance research into new antibiotics? Why should we consider
them?
"""

response = client.responses.create(
  model="o3-mini",
  input=[[
    {
      "role": "user",
      "content": prompt
    }
  ]
)

print(response.output_text)
```

M17b Vergleich von GPT-4o, GPT-4-o1 und GPT-4-o3

1 Technologische Merkmale

- **Modellarchitektur & Ansatz:**

GPT-4o ist ein großer generativer Transformer (LLM), der als Haupt-ChatGPT-Modell dient. Es basiert auf der GPT-4-Architektur und wurde optimiert für hohe Allround-Intelligenz ([Pricing | OpenAI](#)). **GPT-4-o1 (OpenAI o1)** dagegen ist ein „reflective“ LLM, das *Chain-of-Thought*-Techniken nutzt – es „denkt“ zunächst intern in mehreren Schritten, bevor es antwortet ([OpenAI o1 - Wikipedia](#)). Dadurch kann es komplexe Probleme systematischer lösen (insbesondere in MINT-Bereichen) ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). **GPT-4-o3 (OpenAI o3)** ist die nächste Generation dieser Reasoning-Modelle. Es baut auf o1 auf und zielt auf noch leistungsfähigeres logisches Denken ab, jedoch mit verschiedenen Größenvarianten (z. B. **o3-mini** als kleiner, schneller Ableger) ([OpenAI o3-mini | OpenAI](#)) ([OpenAI o3-mini | OpenAI](#)). Alle drei Modelle sind Transformer-basierte KI-Sprachmodelle; o1 und o3 sind jedoch besonders auf *tiefgehende Schlussfolgerungen* trainiert, während GPT-4o auf allgemeine Vielseitigkeit optimiert ist ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)) ([Pricing | OpenAI](#)).

- **Eingabe/Output-Fähigkeiten & Multimodalität:**

GPT-4o ist **multimodal** und kann Text, Bilder und (über integrierte Tools) Audio verarbeiten. Es gilt als aktuell fortschrittlichstes multimodales Modell von OpenAI und besitzt **starke Visions-Fähigkeiten** ([Azure OpenAI Service - Pricing | Microsoft Azure](#)). (In ChatGPT Plus kann GPT-4o z. B. Bilder analysieren und per Sprachmodus Audio ausgeben.) **GPT-4-o1** beherrscht Text und – in der vollwertigen Version – ebenso Bildinputs (Visuelles Verständnis) ([OpenAI o1 explained: Everything you need to know](#)). Zum Start fehlte o1 jedoch noch die Audio-Unterstützung und Features wie Web-Browsing ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)) ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). **GPT-4-o3** (insbesondere o3-mini) fokussiert primär auf Text; die **Visions-Funktion ist hier (noch) nicht aktiviert** ([OpenAI o3-mini | OpenAI](#)). Stattdessen liegt der Schwerpunkt auf strukturierten Textausgaben und Entwicklertools. O3-mini ist das erste kleine Reasoning-Modell mit **Function Calling** und strukturierten Ausgaben ([OpenAI o3-mini | OpenAI](#)). Insgesamt bietet GPT-4o die breiteste I/O-Palette (Text, Bild, teils Audio), während o1 und o3 sich (derzeit) auf Text und logische Ausgaben konzentrieren – Bilder werden bei Bedarf über o1 verarbeitet ([OpenAI o3-mini | OpenAI](#)).

- **Geschwindigkeit & Latenz:**

GPT-4o wurde im Laufe von 2024 beschleunigt und reagiert etwa **doppelt so schnell wie frühere GPT-4-Versionen** ([OpenAI o1 explained: Everything you need to know](#)). Es

ist für Echtzeit-Dialoge in ChatGPT optimiert. **GPT-4-o1** benötigt durch das aufwändige Nachdenken etwas mehr Zeit pro Antwort ([OpenAI o1 explained: Everything you need to know](#)). Gerade bei “**o1-pro**” (der High-Compute-Variante) können Antworten länger dauern, da das Modell intensiver rechnet, um höchste Genauigkeit zu erzielen.

GPT-4-o3 adressiert die Latenz, insbesondere in der Mini-Variante: o3-mini ist **kleiner und schneller** als o1, bietet wählbare *Reasoning-Tiefen* (Low/Medium/High) je nach gewünschter Geschwindigkeit vs. Gründlichkeit ([OpenAI o3-mini | OpenAI](#)) ([OpenAI o3-mini | OpenAI](#)). In der Standardeinstellung (mittlere Denktiefe) liefert o3-mini ähnlich präzise Ergebnisse wie o1, aber **schneller** ([OpenAI o3-mini | OpenAI](#)). Für sehr komplexe Anfragen kann man o3 auch in einen High-Effort-Modus versetzen (auf Kosten höherer Latenz). Insgesamt: GPT-4o ist flott und interaktiv, o1 eher langsamer, und o3-mini kombiniert hohe Geschwindigkeit mit guter Denkleistung (o3-high wäre wiederum langsamer, aber besonders genau).

- **Kontextlänge & Speicher:**

Alle drei Modelle unterstützen sehr große Kontextfenster. **GPT-4o** kann Eingaben bis zu **128.000 Token** Kontext verarbeiten ([Azure OpenAI Service - Pricing | Microsoft Azure](#)) (deutlich mehr als ältere GPT-3.5-Modelle). **GPT-4-o1** wurde anfänglich mit ähnlichem Kontext (128K) eingeführt ([OpenAI o1 explained: Everything you need to know](#)), doch die **API-Variante von o1** bietet inzwischen sogar bis zu **200.000 Tokens Kontext** ([Pricing | OpenAI](#)). Das heißt, o1 kann extrem lange Dokumente/Chats im Prompt halten – nützlich für umfangreiche technische Analysen. Auch **GPT-4-o3** setzt diese Linie fort: o3-Modelle unterstützen laut OpenAI ebenfalls **Kontext bis 200K Token** ([Pricing | OpenAI](#)). Damit eignen sich o1/o3 hervorragend für Aufgaben, die sehr viel Input erfordern (z. B. lange Berichte, Codebasen). GPT-4o's 128K sind für die meisten Anwendungen schon groß, o1/o3 gehen im „Frontier“-Einsatz noch darüber hinaus.

- **Besondere Funktionen & Tools:**

GPT-4o als ChatGPT-Modell hat breiten Feature-Support: es kann in ChatGPT z. B. auf Web-Browsing oder Code Interpreter (eingebetteter Python) zurückgreifen und unterstützt die API-Funktion *Function Calling*, um strukturierte Ergebnisse zu liefern. **GPT-4-o1** ist ebenfalls dafür ausgelegt, mit Tools zu arbeiten – OpenAI beschreibt o1 als „**Frontier**“-Reasoning-Modell mit Tool-Unterstützung und strukturierten Outputs ([Pricing | OpenAI](#)). Allerdings standen in der Preview-Phase einige GPT-4o-Features (z. B. Surfen im Web) in o1 nicht zur Verfügung ([OpenAI o1 explained: Everything you need to know](#)). **GPT-4-o3** schließt diese Lücke weiter: o3-mini unterstützt ab Start Function Calling, strukturierte Ausgaben und Streaming ([OpenAI o3-mini | OpenAI](#)). Außerdem wurde experimentell eine **Suchfunktion** integriert, sodass o3-mini bei Bedarf Web-Ergebnisse mit Zitaten liefern kann ([OpenAI o3-mini | OpenAI](#)). Unterm Strich: GPT-4o bietet das **vollständige ChatGPT-Erlebnis** mit allen Integrationen; o1/o3 sind spezialisierter auf reasoning-lastige Aufgaben, unterstützen aber zunehmend auch Entwickler-Features (API) und werden laufend erweitert.

2 Typische Einsatzszenarien

- **GPT-4o:** Dieses Modell wird für **Alltagsanwendungen** und generelle KI-Aufgaben eingesetzt. Typische Szenarien sind z. B. das **Verfassen von Texten** (Aufsätze, Blogbeiträge, kreative Geschichten), **Beantworten von Wissensfragen** und **Zusammenfassungen**, **Übersetzungen** oder auch einfach interaktive **Dialogassistenz**. Dank seiner breiten Wissensbasis und Spracheleganz eignet sich GPT-4o auch für **kreative Aufgaben** wie das Schreiben von Gedichten oder Marketing-Texten. In der Programmierung kann GPT-4o ebenfalls helfen (Code-Snippets generieren, Erklärungen liefern), wenngleich für sehr komplexe Programmieraufgaben die speziellen o-Modelle oft noch präziser sind. Kurz: GPT-4o ist das **Universalmodell für allgemeine Zwecke** – empfohlen für die meisten Routine-Texterstellungen und Dialoge ([Pricing](#) | [OpenAI](#)).
- **GPT-4-o1:** OpenAI o1 glänzt vor allem in **anspruchsvollen, mehrstufigen Aufgaben** und technisch-wissenschaftlichen Anwendungen ([Pricing](#) | [OpenAI](#)). Typische Einsatzszenarien sind z. B. **komplexe Mathematikprobleme**, Beweise oder logische Rätsel, bei denen schrittweises Argumentieren wichtig ist. Ebenso wird o1 für **Programmierung** genutzt, insbesondere zum **Debuggen oder Lösen schwieriger Coding-Challenges**, da es systematisch den Code durchdenken kann. Im Bereich **Wissenschaft** (Physik, Chemie, Biologie) kann o1 fachliche Fragen beantworten oder Probleme lösen – OpenAI berichtet, dass o1 auf PhD-Niveau in MINT-Fragen agieren kann ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). Auch für Unternehmen, die **lange Analysen oder Berichte** durch das Modell prüfen lassen wollen, ist o1 geeignet (Stichwort: großer Kontext von 200k Tokens). Darüber hinaus bietet o1 erhöhte **Zuverlässigkeit in sensiblen Bereichen**: z. B. für medizinische oder rechtliche Auskünfte, wo man Wert auf korrekte Schritte legt, kann o1 vorteilhaft sein. Insgesamt kommt GPT-4-o1 immer dann zum Einsatz, wenn **gründliches logisches Denken** gefragt ist und einfache KI-Antworten an ihre Grenzen stoßen.
- **GPT-4-o3:** Da GPT-4-o3 die Weiterentwicklung von o1 ist, deckt es ähnliche Szenarien ab, jedoch mit Verbesserungen in **Geschwindigkeit und Zugänglichkeit**. **OpenAI o3-mini** (die kompakte Variante) wird empfohlen für **technische Domänen** wie **Coding**, **mathematische Aufgaben und logische Problemlösung** – überall dort, wo Präzision und zugleich schnelle Antworten benötigt werden ([OpenAI o3-mini](#) | [OpenAI](#)) ([OpenAI o3-mini](#) | [OpenAI](#)). Durch die effizientere Architektur eignet sich o3-mini sogar für **Echtzeit-Dialoge in technischen Support- oder Lern-Assistenten**, da es weniger Latenz hat, aber dennoch verlässliche Zwischenschritte liefert. Große Unternehmen oder Forschungsprojekte könnten ein vollausgebautes o3-Modell (mit hoher Reasoning-Tiefe) für **hochkomplexe Analysen** einsetzen – etwa KI-gestützte Forschung, die mehrere Hypothesen durchdenken soll. Ein weiterer Anwendungsfall ist der **API-Einsatz bei spezialisierten Apps**: Entwickler könnten o3-mini nehmen, um z. B. **Code-Autokomplettierungen oder mathematische Tutor-Systeme** zu bauen, da es eine gute Balance aus Leistung und Kosten hat. Zudem testet OpenAI mit o3 die Integration von **aktueller Websuche**, was es attraktiv für **Faktenfragen mit aktuellen Bezügen** macht ([OpenAI o3-mini](#) | [OpenAI](#)). Zusammengefasst dient GPT-4-o3 heute vor allem als **schneller Problemlöser** für MINT-Aufgaben und wird perspektivisch überall dort Verwendung finden, wo man die Stärken von o1 mit mehr Effizienz kombinieren möchte.

3 Vor- und Nachteile in den Einsatzszenarien

GPT-4o (Allround-ChatGPT): Vorteile: Sehr breites Allgemeinwissen und vielseitige Sprachfähigkeiten (gut für offene Dialoge, kreatives Schreiben etc.). Dank Multimodalität kann es Bilder verstehen und vielfältige Aufgaben (Übersetzen, Zusammenfassen, Texte in Ton umwandeln) in einem Tool erledigen ([Azure OpenAI Service - Pricing | Microsoft Azure](#)). Zudem ist es inzwischen recht schnell und in den meisten Anwendungen ausreichend präzise. Nicht zuletzt ist GPT-4o weit verbreitet und leicht zugänglich (ChatGPT-Plus, demnächst auch als Standard in vielen Apps) ([OpenAI o1 explained: Everything you need to know](#)). **Nachteile:** Bei komplexer Problemlösung (z. B. anspruchsvolle Mathematik oder mehrstufiges logisches Schließen) erreicht GPT-4o nicht die Tiefe der spezialisierten Modelle – es kann zu Fehlern oder Halluzinationen kommen, wo o1/o3 systematischer vorgehen würden ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). In sicherheitskritischen Fällen ist GPT-4o zwar gut, aber weniger robust gegen Jailbreaks als o1 ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). Außerdem hat GPT-4o gewisse Einschränkungen: Das Modell hält sich strikt an erlernte Sicherheitsvorgaben, was teils zu Ablehnungen führen kann, und es fehlen ihm die neuesten Spezialoptimierungen für z. B. detailliertes Planen. Auch kostet es im API-Einsatz mehr als kleinere Modelle. Zusammengefasst ist GPT-4o überall „gut bis sehr gut“, aber nicht in jeder Nische das Optimum.

GPT-4-o1 (Reasoning-Modell): Vorteile: Herausragend in Leistungsfähigkeit auf schwierigen Aufgaben – o1 erreicht z. B. ~86 % in Mathematik-Olympiade-Benchmarks (vs. 13 % bei GPT-4o) ([OpenAI o1 explained: Everything you need to know](#)). Es kann komplexe Fragen schrittweise und korrekt lösen, was in Bereichen wie Programmierung (Algorithmus-Challenges) oder Wissenschaft enorme Mehrwerte bringt ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). Auch im Bereich Safety/Alignment hat o1 Vorteile: Es wurde darauf trainiert, die OpenAI-Regeln aktiv in seine Gedankenschritte einzubeziehen, was zu höherer Robustheit gegen Fehlverhalten führt ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). Tests zeigen, dass o1 riskante Anfragen viel konsequenter ablehnt bzw. sicher beantwortet (z. B. in einem Sicherheits-Benchmark erzielte o1 eine 0,92 Nicht-Unsicherheits-Rate vs. 0,713 bei GPT-4o) ([OpenAI o1 explained: Everything you need to know](#)) ([OpenAI o1 explained: Everything you need to know](#)). Darüber hinaus kann o1 mit seinem großen Kontext sehr lange Inhalte verarbeiten – ideal, um z. B. technische Dokumentationen oder ganze Bücher zu analysieren. **Nachteile:** o1 ist langsamer und rechenintensiver – im Chat merkt man die Verzögerung durch das „Nachdenken“ ([OpenAI o1 explained: Everything you need to know](#)), was in Echtzeit-Anwendungen hinderlich sein kann. Es ist zudem stark auf STEM-Themen optimiert, sodass es bei alltäglichen Smalltalk-Themen oder kreativ-literarischen Aufgaben manchmal steifer oder weniger ausdrucksstark wirkt (das kleinere o1-mini hatte z. B. Schwächen bei breit gefächertem Weltwissen) ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). Ferner ist die Verfügbarkeit eingeschränkt: o1 war lange nur für Plus/Pro-Nutzer oder ausgewählte API-Kunden

zugänglich ([OpenAI o1 explained: Everything you need to know](#)). Auch die Kosten sind hoch – insbesondere über die API ist o1 deutlich teurer als GPT-4o ([OpenAI o1 explained: Everything you need to know](#)). Schließlich muss man beachten, dass o1 in frühen Versionen nicht alle Features (wie Surfen) bot, wobei neuere Iterationen diese Lücken teils schließen. Fazit: GPT-4-o1 ist **unschlagbar bei kniffligen Aufgaben** und in Sachen Verlässlichkeit/Sicherheit top, erkauft dies aber mit Geschwindigkeitseinbußen, höherem Preis und etwas geringerer Allgemeinflexibilität.

GPT-4-o3 (Fortgeschrittenes Reasoning-Modell): Vorteile: o3 kombiniert viele Stärken von o1 mit Verbesserungen. Die **Genauigkeit bei technischen Aufgaben** ist sehr hoch – o3-mini erreicht mit mittlerem Aufwand bereits die Leistung von o1 in Mathe, Coding und Wissenschaft ([OpenAI o3-mini | OpenAI](#)). Gleichzeitig bietet es **geringere Latenz** und höhere Effizienz, besonders in der Mini-Variante, was es alltagstauglicher macht (z. B. für interaktive Beratungsbots in Technikfragen). Entwickler profitieren von den von Anfang an integrierten Features (Function Calling, strukturierte Ausgabe), was o3 **leicht in Anwendungen integrierbar** macht ([OpenAI o3-mini | OpenAI](#)). Ein großer Vorteil ist auch die **Skalierbarkeit nach Bedarf**: Durch die wählbaren Reasoning-Modi kann man je nach Szenario Geschwindigkeit oder Gründlichkeit priorisieren ([OpenAI o3-mini | OpenAI](#)). Außerdem democratisiert OpenAI o3-mini etwas die Nutzung – erstmals bekam sogar die kostenlose ChatGPT-Version Zugang zu einem Reasoning-Modell ([OpenAI o3-mini | OpenAI](#)). Nachteile: Noch ist GPT-4-o3 **relativ neu** – die vollausgereifte große o3-Version (ohne „mini“) ist Stand Anfang 2025 ggf. noch nicht allgemein verfügbar, d.h. man arbeitet vorwiegend mit o3-mini als Vorgeschmack. Dieses kleinere Modell ist zwar flink, hat aber (wie o1-mini) einen **engeren Wissensfokus** – für breit angelegte Konversations-KI mit emotionaler Intelligenz oder Kreativität ist weiterhin GPT-4o besser geeignet. Aktuell **unterstützt o3 keine Bild-Eingaben** ([OpenAI o3-mini | OpenAI](#)), was einen Nachteil gegenüber GPT-4o und o1 darstellt, falls multimodale Fähigkeiten gefragt sind. In puncto **Kosten** liegt o3-mini zwar deutlich unter GPT-4o, allerdings erreicht es (insbesondere im High-Effort-Modus) noch nicht die absolute Spitzenleistung von einem hypothetischen großen o3-Modell – sprich, wer maximale Qualität will, müsste auf zukünftige o3-Varianten warten oder hohen Rechenaufwand investieren (im Extremfall wurden für „o3 High“ interne Kosten von bis zu \$20k pro komplexer Aufgabe berichtet ([... | Dan Leteky](#))). Zusammengefasst bietet GPT-4-o3 derzeit **exzellente Leistung bei besserer Effizienz**, ist aber in der vollständigen Ausbaustufe noch im Kommen und hat in Randbereichen (Multimodalität, Kreativität) gewisse Limitierungen.

4 Besonderheiten im Vergleich

- **Leistungsfähigkeit & Genauigkeit:** In logisch-mathematischen Aufgaben liegen GPT-4-o1 und o3 klar vor GPT-4o. O1 hat gezeigt, dass es **menschliches Expertenniveau** in vielen STEM-Benchmarks erreicht (z. B. Top-500 bei der USAMO-Qualifikation und übertrifft PhD-Level in Wissenschaftsfragen) ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). GPT-4o ist zwar stark, aber eher generalistisch – in einem direkten Vergleich schnitt o1 bei einem schwierigen Mathematiktest z. B.

sechsmal besser ab ([OpenAI o1 explained: Everything you need to know](#)). GPT-4-o3 baut die Führung weiter aus: interne Tests (ARC-AGI Evaluations) zeigten, dass ein **o3-High** Modell bis zu 88 % der Aufgaben löst, während o1 (High) nur ~32 % schaffte ([.... | Dan Leteky](#)). Wichtig ist jedoch der Kontext: Bei **kreativen oder offenen Dialogen** kann GPT-4o oft geschmeidiger und breiter antworten, da o1/o3 stark formalisiert denken. Hier behält GPT-4o die Nase vorn in Natürlichkeit und Vielfalt der Antworten. Kurz: Für *Fachprobleme* sind o1/o3 überlegen, für *Generelles* ist GPT-4o meist ausreichend bzw. passender.

- **Verfügbarkeit & Zugang:** GPT-4o ist das Standardmodell hinter ChatGPT (für Plus-Nutzer seit 2023, und via API breit zugänglich), daher **am verfügbarsten**. Viele Dienste (z. B. Bing Chat) nutzen GPT-4-Varianten, was GPT-4o allgegenwärtig macht. GPT-4-o1 wurde anfänglich nur im **Preview** an zahlende Nutzer (ChatGPT Plus/Team) ausgerollt ([OpenAI o1 - Wikipedia](#)) und im Dezember 2024 vollständig veröffentlicht ([OpenAI o1 - Wikipedia](#)) ([OpenAI o1 - Wikipedia](#)) – seitdem ist es für ChatGPT Pro-Abonnenten (mit o1-pro) und teils Plus-Nutzer verfügbar. Über die API ist o1 als kostenintensives Modell verfügbar, jedoch eher für Enterprise-Tier-Kunden oder auf Anfrage (OpenAI positioniert es als *Frontier Model*). GPT-4-o3 befindet sich noch in Einführung: Die **kleine Version o3-mini** ist seit Januar 2025 für ChatGPT Plus/Pro und sogar eingeschränkt für Free-User nutzbar ([OpenAI o3-mini | OpenAI](#)). Größere o3-Modelle sind perspektivisch zu erwarten, aber aktuell vor allem für ausgewählte Entwickler (API-Tiers 3–5) freigeschaltet ([OpenAI o3-mini | OpenAI](#)). Insgesamt gilt: GPT-4o **breit verfügbar**, o1 **limitiert (Pro/User mit Berechtigung)**, o3 **im Übergang** (Mini-Version offen für viele, Full-Version noch begrenzt).
- **Kosten (API-Nutzung):** Die unterschiedlichen Ausrichtungen spiegeln sich deutlich in den API-Preisen wider. GPT-4o ist deutlich teurer als gängige GPT-3.5-Modelle, aber immer noch günstiger als das spezialisierte o1. Als Richtwert kostet GPT-4o etwa **\$10 pro 1 Mio. Ausgabe-Tokens** ([Pricing | OpenAI](#)). OpenAI o1 liegt etwa bei **\$60/Mio Tokens (Output)** ([Pricing | OpenAI](#)) – also etwa das 6-fache von GPT-4o, was seine Premium-Nische unterstreicht ([OpenAI o1 explained: Everything you need to know](#)). OpenAI o3-mini hingegen ist sehr kostengünstig: ca. **\$4.40/Mio Tokens** ([Pricing | OpenAI](#)), also weniger als die Hälfte von GPT-4o, was es attraktiv für häufige Aufrufe macht. Diese Preise bedeuten: Für einfache Anwendungen ist GPT-4o in puncto Preis-Leistung gut, während o1 wirklich nur dort eingesetzt wird, wo die Mehrleistung die Mehrkosten rechtfertigt. O3-mini bietet einen **neuen Kompromiss** – viel Reasoning-Power für wenig Geld – und dürfte daher in 2025 an Bedeutung gewinnen. Unternehmen müssen also zwischen **Kosten und erforderlicher Genauigkeit** abwägen: Warum \$60 zahlen, wenn \$10 genügen – oder \$10 zahlen, wenn auch \$4 reichen? ([.... | Dan Leteky](#)).
- **Zusammenfassung der Unterschiede:** GPT-4o, GPT-4-o1 und GPT-4-o3 sind **komplementäre Varianten** des ChatGPT-Modellspektrums. GPT-4o ist der *Allrounder* – multimodal, allgemeinwissend, schnell und relativ günstig – ideal für die meisten Chats und Texte. GPT-4-o1 ist der *Spezialist* – langsam aber gründlich, teuer aber dafür in der Lage, Aufgaben auf expertenhaftem Niveau zu lösen (vor allem in Mathematik, Coding,

Naturwissenschaft) ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#)). GPT-4-o3 schließlich versucht, das Beste aus beiden Welten zu vereinen: *leistungsfähig wie o1, aber effizienter und zugänglicher*, besonders in der Mini-Version ([OpenAI o3-mini | OpenAI](#)). Im direkten Vergleich untereinander zeigt sich, dass keine Variante „perfekt“ für alles ist – es kommt auf den Anwendungsfall an. Für einen kreativen Dialog oder breite Wissensfragen nimmt man GPT-4o; für einen kniffligen Programmierwettbewerb oder wissenschaftlichen Problemlöser greift man zu o1/o3. OpenAI's Strategie 2024/25 spiegelt dies wider: **verschiedene Modelle für verschiedene Bedürfnisse** bereitzustellen ([Pricing | OpenAI](#)) – von schnellem, preiswertem Denken (o3-mini) bis zu maximaler Denktiefe (o1/o3-high) – wobei jedes Modell klar definierte Stärken und Schwächen im Vergleich zu den anderen aufweist.

Quellen:

Offizielle OpenAI-Blogposts und Dokumentationen von 2024/2025 sowie Berichte zum o1- und o3-Release

- [Learning to reason with LLMs | OpenAI](#)
- ([OpenAI's STEM-Optimized 'Reasoning' Model, o1, Sees Daylight -- Pure AI](#))
- ([OpenAI o1 explained: Everything you need to know](#))
- ([OpenAI o3-mini | OpenAI](#))
- ([Pricing | OpenAI](#))
- siehe referenzierte Zitate).