

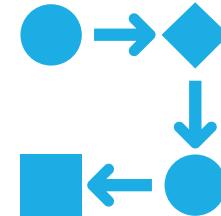


Anwendung Generativer KI

OKTOBER 2025

Lizenz

- Diese Kursmaterialien stehen unter der Creative Commons Lizenz [CC BY 4.0](#) und dürfen frei genutzt, geteilt und bearbeitet werden, solange die Urheber*innen genannt werden.
- Teile der Materialien, einschließlich einiger Texte, Notebooks und Grafiken, wurden mit Hilfe generativer KI erstellt.
- Bitte beachten Sie, dass KI-generierte Inhalte aufgrund der fehlenden menschlichen Urheberschaft nicht alle urheberrechtlichen Schutzrechte abdecken können.
- Dennoch wurde bei der Erstellung darauf geachtet, die Inhalte sorgfältig zu prüfen und anzupassen.
- Für eine korrekte Nutzung und Weitergabe ist die Nennung des Urhebers und der Lizenz weiterhin verpflichtend.



Kurs-Organisation

ZEITPLANUNG

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

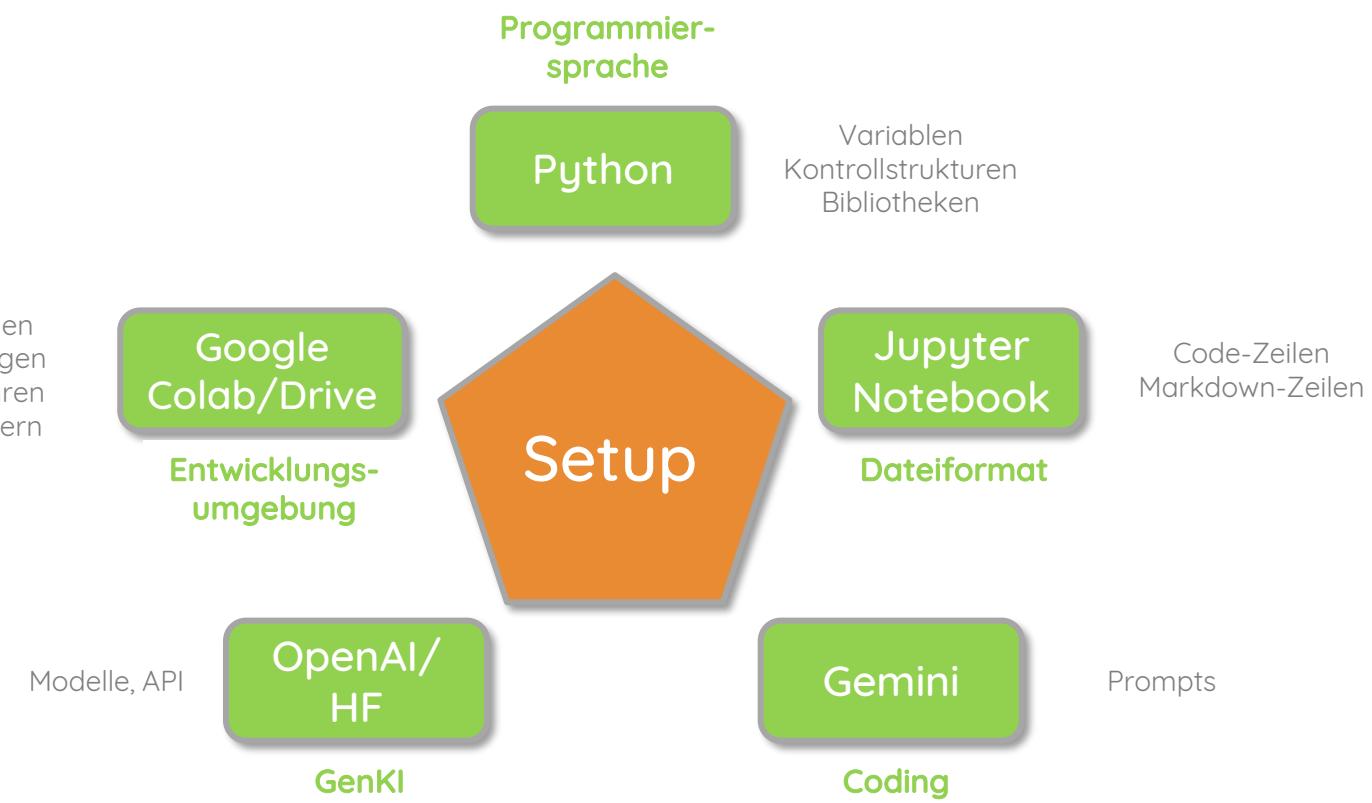
VORGEHEN

- Grundlagen/Basiswissen
- Beispiele
- Training/Fallstudien

VERSCHIEDENES

- Pinboard

IT-Setup



Pinboard

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

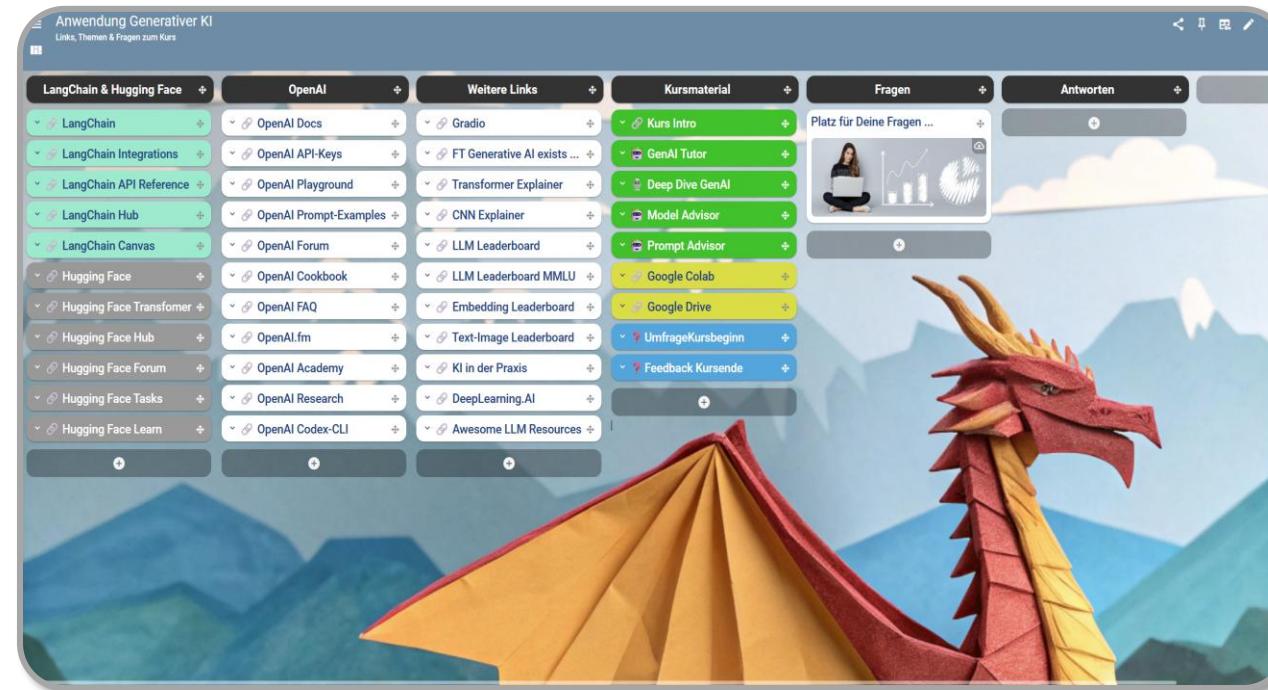


Bild mit Recraft erstellt



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.

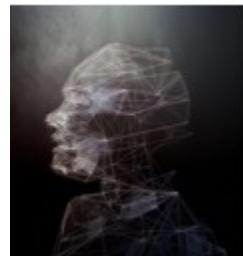
CPU = Central Processing Unit, GPU = Graphics Processing Unit, TPU = Tensor Processing Unit



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.

M00a - Module



Anwendung Generativer KI

Stand: 10.2025

1 Basismodule (1-12)

Die Basismodule bilden das Fundament des Kurses und vermitteln grundlegende Konzepte und Werkzeuge der generativen KI:

- **Einführende Grundlagen:** Allgemeine Einführung in generative KI, Modellsteuerung und fundamentale Frameworks (Module 1-4)
- **Technische Grundlagen:** Vertiefung in Transformer-Architektur, Memory-Konzepte und Output-Parser (Module 5-7)
- **Praktische Anwendungen:** Einführung in RAG, multimodale Bildverarbeitung, Agenten, Gradio und lokale Modelle (Module 8-12)

Diese Module stellen sicher, dass Sie über ein solides Grundverständnis der generativen KI-Technologien verfügen.

2 Erweiterungsmodule (13-23)

Die Erweiterungsmodule bauen auf den Grundlagen auf und bieten fortgeschrittene Konzepte und Spezialisierungen:

- **Erweiterte multimodale Anwendungen:** SQL RAG, Multimodal RAG, Audio- und Videoverarbeitung (Module 13-16)

- **Modelloptimierung:** MCP, Fine-Tuning, Modellauswahl und Evaluation (Module 17-19)
- **Fortgeschrittene Methoden:** Advanced Prompt Engineering und Context Engineering (Modul 20-21)
- **Regulatorische Aspekte:** EU AI Act und Ethik (Modul 22)
- **KI-Challenge:** Praktische Anwendung und Integration der Kursmodule (Modul 23)

Diese Module vertiefen spezifische Anwendungsbereiche und bieten fortgeschrittene Techniken für professionelle KI-Anwendungen.

Die Basismodule sind obligatorisch, die Erweiterungsmodule sind fakultativ.

3 Übersicht nach Basis/Erweiterung

Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
1	Basis	✓	✓	Einführung GenAI	<ul style="list-style-type: none">• Kursüberblick• Überblick Generative AI• Einführung OpenAI• Einführung Hugging Face• Einführung LangChain	Fundamentales Verständnis der Gen-AI-Technologien als Grundlage für den Kurs.
2	Basis	✓		Modellsteuerung -und optimierung	<ul style="list-style-type: none">• Überblick Prompting, Context Engineering, RAG, Fine-Tuning• Einsatzszenarien• Entscheidungskriterien• Trade-offs	Essentielles Verständnis der verschiedenen Ansätze zur Modellansteuerung.

Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
3	Basis	✓	✓	Codieren mit GenAI	<ul style="list-style-type: none"> Prompting für Codegenerierung Revisionsprompts Debugging mit LLMs Prozessintegration Claude Code 	Praktische Fähigkeiten für LLM-gestützte Programm-Entwicklung.
4	Basis	✓	✓	LangChain 101	<ul style="list-style-type: none"> Was ist LangChain & Architektur Kernkonzepte (Chains, Models, Prompts) Best Practices & Design Patterns 	Fundamentales Verständnis von LangChain als zentrales Framework für die Entwicklung von LLM-Anwendungen.
5	Basis	✓	✓	Large Language Models und Transformer	<ul style="list-style-type: none"> Foundation Model Transformer-Architektur Textgenerierung Textzusammenfassung Textklassifizierung LLM schreibt ein Buch 	Vertieftes Verständnis der Transformer-Architektur und LLM-Funktionsweise.
6	Basis	✓		Chat und Memory	<ul style="list-style-type: none"> Übersicht Kurzeit-Memory Langzeit-Memory Externes Memory 	Memory-Typen für Flexibilität und Skalierbarkeit, um Konversationen effizient zu verwalten.
7	Basis	✓		Output Parser	<ul style="list-style-type: none"> Structured Output Parser JSON DatetimeOutputParser Custom Output Parser 	Parser-Typen für robuste und flexible Verarbeitung von Modellantworten.

Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
8	Basis	✓	✓	Retrieval Augmented Generation	<ul style="list-style-type: none"> • Einführung in RAG • ChromaDB • Embeddings • Q&A über Dokumente • Embedding-Datenbanken 	Schlüsseltechnologie für Informationsverarbeitung mit unstrukturierten Daten.
9	Basis	✓	✓	Multimodal Bild	<ul style="list-style-type: none"> • Bildgenerierung • In-/Outpainting • Bildklassifizierung • Objekterkennung • Bildbeschreibung 	Erweiterung um visuelle Komponenten.
10	Basis	✓		Agents	<ul style="list-style-type: none"> • Grundlagen von KI-Agenten • Agentenarchitekturen • Planung und Zielverfolgung • Multi-Agenten-systeme 	Entwicklung autonomer KI-Systeme.
11	Basis	✓		Gradio	<ul style="list-style-type: none"> • Grundkonzepte von Gradio • Installation und Setup • Zentrale Komponenten • Praktische Beispiele • Fortgeschrittene Konzepte • Integration mit KI-Modellen • Best Practices • Deployment und Sharing 	UI-Entwicklung für KI-Anwendungen.

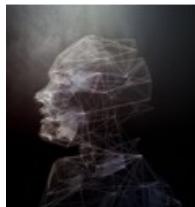
Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
12	Basis	✓		Lokale und Open Source Modelle	<ul style="list-style-type: none"> • Einführung • Ollama • Lokale Modelle und LangChain • Open Source vs. Closed Source • Beispiele Open Source Modelle • Lizenzierung und rechtliche Aspekte • Auswahlkriterien Open Source • Zukunftstrends bei Open Source 	Einsatz von KI-Modellen ohne Cloud-Abhängigkeit.
13	Erweiterung	✓		SQL RAG	<ul style="list-style-type: none"> • Einführung in SQL RAG • Vergleich zu RAG • Integration von LLMs mit Datenbanken • SQL-Generierung mit LLMs • Datenvierlidierung und Sicherheitsaspekte • Praktische Anwendungsfälle • SQL RAG mit LangChain 	Schlüsseltechnologie für die Arbeit mit strukturierten Daten und Datenbanken.

Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
14	Erweiterung	✓		Multimodal RAG	<ul style="list-style-type: none"> • Intro • Enhanced Document Processor • Multimodal Embeddings • Cross-Modale Suche 	Schlüsseltechnologie für die Arbeit mit Texten und Bildern.
15	Erweiterung	✓		Multimodal Video	<ul style="list-style-type: none"> • Video-zu-Text (VTT) • Text-zu-Video (TTV) • Bild-zu-Video (ITV) • Videoanalyse • Video-Objekterkennung 	Integration von Videoverarbeitung und -analyse für multimodale KI-Anwendungen.
16	Erweiterung	✓	✓	Multimodal Audio	<ul style="list-style-type: none"> • Speech-to-Text (STT) • Text-to-Speech (TTS) • Sprachanalyse • Audio-Summary • Audio-Pipeline • Podcast 	Erschließung von Sprachanwendungen.
17	Erweiterung	✓		MCP - Modell Context Protocol	<ul style="list-style-type: none"> • Intro • MCP-Server erstellen • MCP-Client erstellen • AI-Assistant mit MCP • MCP in der Praxis 	Optimierung und effiziente Anpassung von LLM Tooleinsatz durch ein standardisiertes Protokoll
18	Erweiterung	✓	✓	Fine-Tuning	<ul style="list-style-type: none"> • Fine-Tuning mit Dashboard • Fine-Tuning mit Code • Parameter Efficient Fine-Tuning • Bewerten des FT-Modells 	Optimierung und effiziente Anpassung für spezifische Anwendungsfälle.

Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
19	Erweiterung		✓	Modellauswahl und Evaluation	<ul style="list-style-type: none"> • KI-Modelle • Bewertung von Modellen • Modellauswahlprozess • Auswahlkriterien • Benchmarks und Modellvergleiche 	Fundierte Modellauswahl und Bewertung.
20	Erweiterung		✓	Prompt Engineering	<ul style="list-style-type: none"> • Einführung Prompt-Engineering • Few-Shot und Chain-of-Thought • Persona- und Rollenmuster • Frage- und Prüfmuster • Inhalt- und Struktur-Muster • Chat- und Reasoning-Modelle 	Fortgeschrittene Prompt-Strategien.
21	Erweiterung		✓	Context Engineering	<ul style="list-style-type: none"> • Grundstrategien • Häufige Fehler • Tools und Techniken • Praktische Anwendung 	Strategien für das Context Management.
22	Erweiterung		✓	EU AI Act /Ethik	<ul style="list-style-type: none"> • Risikobasierte Einstufung • Hochrisiko-Anwendungen • Strenge Auflagen für Hochrisiko-KI • Transparenz- und Informationspflichten • Ethische Grundsätze 	Der EU AI Act ist seit 2025 das zentrale Regelwerk für den vertrauenswürdigen, sicheren und ethischen Einsatz von KI in Europa.

Modul-Nr.	Modultyp	NB	PDF	Inhalt	Themen	Relevanz
23	Erweiterung		✓	KI-Challenge	<ul style="list-style-type: none">• Projektoptionen• Integration mehrerer Technologien• Entwicklung einer E2E-Anwendung• Gradio-Benutzeroberfläche• Evaluation und Dokumentation	Anwendung und Integration der erlernten Konzepte in einer Gesamtlösung.

M00b - Bibliothek genai_lib



Anwendung Generativer KI

Stand: 10.2025

Allgemeines:

Im Rahmen dieses Einführungskurses werden wir uns nicht durch hunderte von Zeilen Code durcharbeiten können, das wäre viel zu aufwendig und würde den Kern des Themas verfehlten. Stattdessen werden wir mit Bibliotheken arbeiten, die uns einen Teil der komplexen, technischen Details abnehmen. So können wir uns darauf konzentrieren, was wirklich wichtig ist: nämlich das Verständnis der generativen KI-Konzepte und ihre Anwendung. Kurz gesagt: Ihr müsst nicht alles von Grund auf selbst programmieren.

1 🔧 utilities.py

Übersicht

Eine Sammlung von Standard-Hilfsfunktionen für KI- und LangChain-Projekte in Python-Entwicklungsumgebungen, insbesondere für Google Colab.

`check_environment()`

- Zeigt die aktuelle Python-Version an
- Listet alle installierten LangChain-Bibliotheken auf
- Unterdrückt störende Deprecation-Warnungen

`install_packages(modules)`

- Installiert Python-Module automatisch mit `uv pip install`
- Prüft vorher, ob Module bereits verfügbar sind
- Optimiert für Google Colab mit ruhiger Installation

`get_ipinfo()`

- Ruft Geoinformationen zur aktuellen öffentlichen IP-Adresse ab

- Zeigt IP, Standort, Provider und weitere Netzwerkdaten an

```
setup_api_keys(key_names, create_globals=True)
```

- Lädt API-Keys aus Google Colab userdata
- Setzt sie als Umgebungsvariablen und optional als globale Variablen
- Unterstützt mehrere Keys gleichzeitig (OpenAI, Anthropic, Hugging Face, etc.)

```
mprint(text)
```

- Gibt Text als formatiertes Markdown in Jupyter-Notebooks aus
- Nutzt IPython's `display()` und `Markdown()` für bessere Darstellung

```
process_response(response)
```

- Extrahiert strukturierte Informationen aus LLM-Antworten
- Parst Token-Nutzungsdaten (Prompt, Completion, Total)
- Gibt bereinigten Text und Metadaten als Dictionary zurück

Verwendung

```
from utilities import check_environment, setup_api_keys, mprint,
install_packages
# Umgebung prüfen
check_environment()

# API-Keys setzen
setup_api_keys(["OPENAI_API_KEY", "ANTHROPIC_API_KEY"])

# Markdown ausgeben
mprint("# Hallo **Welt**!")

# Installation fehlender Module
install_packages(['langchain', 'langchain_openai', 'langchain_community',
'openai', 'gradio'])
```

2 prepare_prompt.py

```
apply_prepare_framework(task, role, tone, word_limit)
```

- **Zweck:** Erstellt strukturierte Prompts nach dem PREPARE-Framework
- **Parameter:** Aufgabe, Rolle, Tonfall, Wortlimit
- **Ausgabe:** Vollständig formatierter Prompt mit 7 Komponenten (P-R-E-P-A-R-E)

- **Anwendung:** Systematische Prompt-Erstellung für bessere LLM-Ergebnisse

3 show_md.py

Markdown-Display-Funktionen für Jupyter Notebooks:

- `show_md(text, prefix)` - Basis-Markdown-Anzeige mit optionalem Prefix
- `show_title(text)` - Zeigt Titel mit  -Emoji (# Titel -Symbol
- `show_warning(text)` - Warnung mit -Symbol
- `show_success(text)` - Erfolgsmeldung mit -Symbol

Anwendung: Strukturierte und visuell ansprechende Notebook-Ausgaben

Typische Anwendung

```
# 1. Umgebung prüfen und API-Keys laden
check_environment()
setup_api_keys(["OPENAI_API_KEY"])

# 2. Strukturierte Notebook-Ausgaben
show_title("Mein KI-Projekt")
show_info("Experiment gestartet")

# 3. PREPARE-Prompts erstellen
prompt = apply_prepare_framework(
    task="Erkläre maschinelles Lernen",
    role="KI-Tutor",
    tone="verständlich"
)

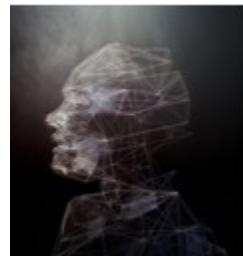
# 4. LLM-Antworten verarbeiten
result = process_response(llm_response)
show_success(f"Antwort erhalten: {result['tokens_total']} Tokens")
```

Ideal für: KI-Kurse, Jupyter-Notebooks, LangChain-Projekte, strukturierte Prompt-Engineering

Umgebung einrichten

```
!uv pip install --system --prerelease allow -q
git+https://github.com/ralf-42/Python_Modules
from genai_lib.utilities import check_environment, get_ipinfo,
setup_api_keys, mprint
setup_api_keys(['OPENAI_API_KEY', 'HF_TOKEN'], create_globals=False)
print()
check_environment()
print()
get_ipinfo()
```

M03 - Claude Code



Anwendung Generativer KI

Stand: 10.2025

In der heutigen Softwareentwicklung gewinnen KI-gestützte Codeassistenten immer mehr an Bedeutung. Besonders die Kommandozeilen-Tools Claude Code von Anthropic, OpenAI Codex CLI und Gemini CLI von Google bieten Entwicklern leistungsstarke Unterstützung beim Schreiben, Analysieren und Automatisieren von Code direkt im Terminal. Trotz ähnlicher Grundfunktionalitäten unterscheiden sich diese Tools in Architektur, Kontextmanagement, Plattformunterstützung und Zielgruppenfokus deutlich. Der folgende Vergleich verschafft einen schnellen Überblick über Stärken, Besonderheiten und Einsatzgebiete der drei führenden CLI-Codeassistenten.

1 Auswahl Codeassistenten

[Claude Code](#)

Merkmal	Claude Code (Anthropic)	OpenAI Codex CLI	Gemini CLI (Google)
Architektur & Betrieb	Lokales Terminal-Tool, agentenbasiert mit Multi-Step-Autonomie, starkes Kontextmanagement (bis ca. 100k Tokens). Läuft auf Unix-ähnlichen Systemen, Windows nur via WSL.	Hauptsächlich lokal, Sandbox-Umgebung zur sicheren Codeausführung, Open Source, läuft auf Linux/macOS, Windows via WSL.	Terminal-Tool nativ plattformübergreifend (Windows, Mac, Linux), Open Source, große Tokenfenster (~1 Million Tokens), native Windows-Unterstützung.
KI-Modell & Kontext	Claude 2 und 3/3.7, sehr gute Reasoning-Fähigkeiten, bis ca. 100k Tokens Kontextgröße, Wissensgraph zur Kontextverwaltung.	GPT-4/GPT-3.5 (OpenAI), Kontexte bis 8k-32k Tokens, flexibler Modus (Suggest, Auto-Edit, Full Auto).	Gemini 2.5 Pro, multimodal (Text & Bilder), riesiges Kontextfenster (~1 Mio. Tokens), starke Modellleistung für komplexe Projekte.
Funktionalität	Autonome Aufgaben, Codegenerierung, Analyse, automatische Git-Integration, lange und komplexe Workflows möglich, Test-Scripte, Deployments (Enterprise).	Vorschläge, automatische Code-Edits, Sandbox-Ausführungen, Integration mit GitHub Copilot, verschiedene Autonomiegrade.	Schnelle Codegenerierung und Test mit Multi-Tool-Integration, Google Cloud Dienste eingebunden, Web-/Suchintegration.
Datenschutz & Sicherheit	Lokal mit umfangreicher Datenkontrolle, keine permanente Cloudverbindung nötig, Enterprise-Optionen für Compliance.	Lokale Sandbox, Quellcode bleibt meist lokal, Cloud-API optional, Sicherheit durch Sandbox.	Cloud-hybrid, Sicherheitsmechanismen, Nutzerbestätigung bei Aktionen, datenschutzbewusst, Open Source für Transparenz.
Benutzerfreundlichkeit & UX	Sehr gute UX, interaktive CLI, stabile Laufscripts, starke Community-Unterstützung.	Solide CLI, flexible Nutzung, teilweise etwas weniger intuitiv bei komplexen Abläufen.	Minimalistisch, schnell, aber noch Verbesserungspotenzial bei UX und Fehlertoleranz.

Merkmal	Claude Code (Anthropic)	OpenAI Codex CLI	Gemini CLI (Google)
Kosten & Lizenzierung	Kommerziell (Anthropic), Abo oder Pay-As-You-Go, Probeversionen mit Limits, verfügbar über Cloud und eigene Instanzen.	Open Source, API-basierte Abrechnung bei OpenAI, benötigt API-Key, kleine kostenlose Credits beim Start.	Open Source, kostenlos mit großzügigem Nutzungskontingent, Google-Konto erforderlich für volle Nutzung.
Besondere Stärken	Hervorragende Codequalität, komplexe autonome Abläufe, großes Kontextwissen, native GitHub-Integration, auch für Enterprise geeignet.	Flexibel, vielfältige API-Unterstützung, robuste Sandbox-Ausführung, gut für Entwickler mit DevOps-Integration.	Sehr große Kontextfenster, native Multimodalität, enge Integration ins Google-Ökosystem, schnelles Prototyping.
Plattform-Unterstützung	macOS/Linux offiziell, Windows via WSL empfohlen.	macOS/Linux, Windows via WSL.	Windows/Mac/Linux nativ (kein WSL nötig).

Summary

- **Claude Code** überzeugt durch seine Tiefe in autonomer Codebearbeitung, starke Nutzung großer Kontexte, ausgezeichnete Codequalität und Enterprise-Tauglichkeit, läuft jedoch primär auf Unix-ähnlichen Systemen und erfordert Windows-Nutzer WSL.
- **OpenAI Codex CLI** ist ein etabliertes, Open-Source-Tool mit guter Flexibilität, starker DevOps-Integration und Sandbox-Sicherheit, allerdings mit kleinerem Kontextfenster und laufender Cloud-Anbindung.
- **Gemini CLI** bietet die größte Kontextkapazität mit multimodalen Fähigkeiten und plattformübergreifender nativer Unterstützung, ideal für Google-Ökosystemnutzer, ist aber bei UX und Stabilität noch im Wachstum.

Derzeit führen Nutzer- und Testberichte **Claude Code** als das ausgereiftere und qualitativ hochwertigere CLI-Tool, während Gemini CLI mit seiner Geschwindigkeit und extrem großem Kontextfenster punktet und Codex CLI als solide, flexible Option gilt.

2 Claude Code Commands

2.1 Die 10 wichtigsten Commands

#	Command	Funktion	Anwendung
1	/init	CLAUDE.md automatisch erstellen	Bei Projekt-Start
2	#	Regel zur CLAUDE.md hinzufügen	# Use Python type hints
3	/compact	Kontext komprimieren, Wichtiges behalten	Bei >50k Tokens
4	Shift+Tab	Modus wechseln (Sonnet/Opus)	Workflow anpassen
5	/plan	Feature-Plan für große Tasks	Features >100 Zeilen
6	/commit	Git Commit mit Auto-Message	Nach jedem Feature
7	/test	Tests ausführen	Vor jedem Commit
8	/review	Automatischer Code-Review	Vor Pull Request
9	/config	Projekt konfigurieren	Setup, Templates
10	Custom	Eigene Workflows in .claude/commands/	Wiederkehrende Tasks

2.2 Standard Commands

2.2.1 Basis-Commands

- `/help` - Zeigt alle verfügbaren Commands
- `/clear` - Löscht Chat-Verlauf für neuen Kontext
- `/compact` - Komprimiert Kontext: `/compact Focus on authentication`
- `/docs` - Öffnet offizielle Dokumentation
- `/settings` - Zeigt/bearbeitet Settings
- `/cancel` - Bricht aktuelle Operation ab

- `/undo` - Macht letzte Änderung rückgängig

2.2.2 Kontext & Memory

- `#` - Fügt Instruktionen zur CLAUDE.md hinzu
- `/memory` - Zeigt gespeicherten Kontext
- `/forget` - Entfernt bestimmte Informationen
- `/context` - Zeigt aktuellen Token-Verbrauch

2.3 Modell-Steuerung

Modell	Verwendung	Wechsel
Sonnet 4.5	Schnell & effizient für Standard-Tasks	Shift+Tab oder <code>/model sonnet</code>
Opus 4.1	Leistungsstark für komplexe Aufgaben	Standard oder <code>/model opus</code>

Empfehlung: Sonnet für schnelle Implementierung, Opus für komplexe Planung

2.4 Code-Analyse & Testing

Command	Funktion	Wann nutzen
<code>/test</code>	Führt Tests aus	Vor jedem Commit
<code>/coverage</code>	Zeigt Test-Coverage	Code-Abdeckung prüfen
<code>/lint</code>	Führt Linter aus	Code-Qualität
<code>/format</code>	Formatiert Code	Style anwenden
<code>/typecheck</code>	Type-Checking	TypeScript validieren

Command	Funktion	Wann nutzen
/analyze	Code-Qualität analysieren	Performance-Check
/review	Code-Review	Qualitätssicherung

2.5 Debugging & Monitoring

- `/debug` - Startet Debug-Modus für Fehlersuche
- `/logs` - Zeigt Logs für Fehleranalyse
- `/trace` - Execution Trace für Performance-Analyse
- `/profile` - Profiling für Bottleneck-Suche
- `--mcp-debug` - MCP-Konfiguration debuggen

2.6 Dokumentation

- `/explain` - Erklärt Code oder Konzepte
- `/document` - Generiert Dokumentation (z.B. API-Docs)
- `/readme` - Erstellt/updated README
- `/comment` - Fügt Code-Kommentare hinzu

2.7 Projekt-Management

- `/plan` - Erstellt detaillierten Projektplan für große Features
- `/task` - Verwaltet Tasks
- `/todo` - Zeigt TODO-Liste
- `/milestone` - Definiert Projekt-Meilensteine

2.8 Custom Commands

Erstellen: Datei in `.claude/commands/fix-issue.md`

```
Please analyze and fix the GitHub issue: $ARGUMENTS
```

Follow these steps:

1. Use `'gh issue view'` to get issue details
2. Understand the problem
3. Search codebase for relevant files
4. Implement changes
5. Run tests to verify

Aufruf: `/fix-issue #123`

2.9 Wichtige Dateien & Verzeichnisse

Datei	Scope	Git	Zweck
CLAUDE.md	Projekt	✓	Projekt-Kontext & Regeln
CLAUDE.local.md	Lokal	✗	Persönliche Notes
~/.claude/CLAUDE.md	Global	-	User Defaults
.claude/settings.json	Projekt	✓	Team-Konfiguration
.claude/settings.local.json	Lokal	✗	Persönliche Config
.claude/commands/*.md	Projekt	✓	Team Commands
~/.claude/commands/*.md	Global	-	User Commands

Datei	Scope	Git	Zweck
.claude/agents/*.md	Projekt	✓	Projekt Subagents
.claudeignore	Projekt	✓	Exclude Files/Dirs
.mcp.json	Projekt	✓	MCP Server Config

2.10 Workflow-Beispiele

2.10.1 Entwicklungs-Workflow

```
claude          # Starte Claude Code
/init           # Erstelle CLAUME.md
# Use Python type hints      # Füge Regel hinzu
"Create user authentication" # Gib Aufgabe
/test           # Teste Implementation
/commit         # Committe Änderungen
```

2.10.2 Debug-Workflow

```
/logs          # Prüfe Logs
/debug         # Starte Debug-Modus
"Analyze error in auth.py" # Analysiere Fehler
/test          # Verifiziere Fix
```

2.10.3 Code-Review-Workflow

```
/diff          # Zeige Änderungen  
/review        # Führe Review durch  
/lint          # Prüfe Code-Style  
/test          # Führe Tests aus  
/commit        # Committe wenn OK
```

2.11 Konfiguration & Templates

```
/config         # Verwaltet Konfigurationen  
/config apply --template react    # Wendet Template an  
/config show      # Zeigt aktuelle Config  
/config reset      # Setzt zurück  
/config validate    # Validiert Settings
```

2.12 Performance-Tipps

Situation	Command	Grund
Lange Konversation	/compact	Token sparen
Stuck in Loop	/clear + /plan	Neustart mit Plan
Schnelle Iteration	Shift+Tab (Sonnet)	Geschwindigkeit
Komplexes Problem	Opus	Besseres Reasoning
Token-Limit erreicht	/compact oder neuer Chat	Kontext reduzieren

2.13 Kontext-Verwaltung

Befehl	Token-Limit	Empfehlung
Normaler Chat	~200k Tokens	Standard-Tasks
/compact	Reduziert Kontext	Bei Überlauf
/clear	Löscht alles	Neustart nötig
Neuer Chat	Frischer Start	Großes Feature fertig

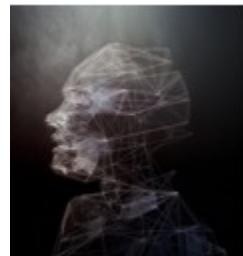
2.14 MCP (Model Context Protocol)

- **Was ist MCP?** Universal-Connector für externe Tools
- **Konfiguration:** `.claude/settings.json`
- **Beispiel-Server:** GitHub, Puppeteer, Supabase, Filesystem
- **Installation:** Via npm packages
- **Debug:** `claude --mcp-debug`

2.15 Ressourcen

- **Offizielle Docs:** <https://docs.claude.com/en/docs/claude-code>
- **Best Practices:** <https://www.anthropic.com/engineering/claude-code-best-practices>
- **Templates:** <https://github.com/davila7/claude-code-templates>
- **Awesome List:** <https://github.com/hesreallyhim/awesome-claude-code>

M04 - LangChain101



Anwendung Generativer KI

Stand 10.2025

1 Einleitung

Die Entwicklung von Applikationen, die auf großen Sprachmodellen (Large Language Models, LLMs) basieren, hat in den letzten Jahren rasant an Fahrt aufgenommen. Für Entwickler stellt sich dabei oft die zentrale Frage nach den richtigen Werkzeugen und Frameworks. Zwei prominente Optionen im Jahr 2025 sind die direkte Nutzung der Programmierschnittstelle (API) von OpenAI und der Einsatz des LangChain-Frameworks, typischerweise mit einem OpenAI-Modell als Backend. Angesichts der Schnelllebigkeit in diesem Technologiefeld ist ein aktueller Vergleich unerlässlich, um fundierte Entscheidungen treffen zu können.

Dieser Bericht stellt die aktuellen Funktionalitäten der direkten OpenAI API und des LangChain-Frameworks (bei Verwendung mit OpenAI-Modellen) gegenüber. Der Fokus liegt auf den Kernaspektren Prompting-Mechanismen, Output-Parsing und dem Aufruf von Sprachmodellen. Ergänzend werden die kritischen Themen Tool/Function Calling sowie Fehlerbehandlung und Debugging beleuchtet, da sie für die praktische Entwicklung von LLM-Applikationen von entscheidender Bedeutung sind. Eine wichtige Neuerung seitens OpenAI, die **Responses API**, wird ebenfalls in die Betrachtung einbezogen, da sie potenziell signifikante Auswirkungen auf den Vergleich und die Art und Weise hat, wie Entwickler mit den Modellen interagieren.

2 LangChain vs OpenAI

Die folgende Tabelle bietet eine erste, hochrangige Zusammenfassung der Kernunterschiede zwischen der direkten Nutzung der OpenAI API und der Verwendung von LangChain mit einem OpenAI-Backend. Sie dient als Referenzpunkt für die nachfolgenden detaillierten Abschnitte.

Aspekt	OpenAI API (Direkt)	LangChain (mit OpenAI)
Prompting-Mechanismen	messages -Array (Chat Completions API) ; input & instructions (Responses API). Manuelle Strukturierung.	PromptTemplate , ChatPromptTemplate . Flexible Template-Erstellung und -Verwaltung. LCEL für Verkettung.
Output-Parsing	Manuelles Parsen von Text; tool_calls für strukturiertes JSON.	Dedizierte OutputParser (z.B. StrOutputParser , JsonOutputParser , PydanticOutputParser). Automatische Strukturierung und Validierung.
Aufruf von Sprachmodellen	client.chat.completions.create() , client.responses.create() . Direkter API-Zugriff. Streaming via stream=True .	ChatOpenAI().invoke() , ChatOpenAI().stream() . Abstrahierte, konsistente Schnittstelle.
Tool/Function Calling	tools -Parameter mit JSON-Schema-Definition. tool_choice zur Steuerung. Responses API mit Built-in Tools.	bind_tools mit Pydantic-Modellen oder Funktionen. Orchestrierung innerhalb von Agents.
Abstraktionsebene	Gering. Direkte Kontrolle über API-Parameter.	Hoch. Vereinfachung durch Abstraktionen und standardisierte Komponenten.
Flexibilität (Modellagnostik)	Bindung an OpenAI-Modelle.	Hoch. Prinzipiell einfacher Wechsel des LLM-Providers möglich.

Fehlerbehandlung & Debugging	Standard API-Fehlercodes und Exceptions. Manuelle Implementierung von Retries.	Erweiterte Parser-Retries (<code>RetryOutputParser</code>). Observability-Plattform LangSmith für Tracing und Debugging.
Zustandsmanagement (Conversation State)	Manuell mit Chat Completions API (gesamter Verlauf muss gesendet werden). 1 Serverseitig mit Responses API (<code>store: true</code>).	LangChain Memory-Module für verschiedene Speicherstrategien.
Komplexität der Implementierung	Geeignet für einfache, direkte Aufgaben. Potenziell geringere Einstiegshürde für Basisfunktionen.	Geeignet für komplexe, mehrstufige Workflows (Chains, Agents). Kann bei einfachen Aufgaben Overhead erzeugen.

Diese Tabelle verdeutlicht, dass die direkte OpenAI API maximale Kontrolle und Direktheit bietet, während LangChain durch Abstraktion, Modularität und spezialisierte Werkzeuge die Entwicklung komplexerer Anwendungen vereinfachen kann.

3 Einsatzszenarien

Die Entscheidung zwischen der direkten Nutzung der OpenAI API und dem Einsatz von LangChain hängt stark von den spezifischen Anforderungen des Projekts, der Komplexität der Anwendung und den Präferenzen des Entwicklungsteams ab.

3.1 Wann OpenAI API?

- Einfache, klar definierte Aufgaben:** Wenn die Anwendung eine überschaubare Aufgabe erfüllt, wie z.B. eine einfache Frage-Antwort-Funktion oder Textgenerierung basierend auf einem statischen Prompt, und minimale Latenz sowie volle Kontrolle über den API-Request entscheidend sind.¹⁹ Die geringere Anzahl an Abstraktionsebenen kann hier zu einer besseren Performance führen.²⁰
- Primäre Nutzung von OpenAI-Modellen:** Wenn ausschließlich OpenAI-Modelle zum Einsatz kommen und keine Notwendigkeit für Abstraktionen besteht, die einen Wechsel des LLM-Providers erleichtern würden.¹³
- Schnelles Prototyping mit minimalem Setup:** Für sehr schnelle Prototypen oder Experimente, bei denen der Overhead eines zusätzlichen Frameworks vermieden werden soll und die Komplexität gering ist.¹³

- **Vermeidung von Framework-Abhängigkeiten:** Wenn Entwickler die "Magie" oder die zusätzlichen Abstraktionsebenen von LangChain als hinderlich empfinden oder eine direkte, transparente Kontrolle über jeden Aspekt der API-Interaktion bevorzugen.³⁹
- **Nutzung neuester OpenAI API-Features:** Wenn die neuesten Funktionen der OpenAI API (z.B. die Responses API mit serverseitigem State Management und Built-in Tools 1) die benötigte Funktionalität bereits nativ und zufriedenstellend abdecken.

3.2 Wann LangChain?

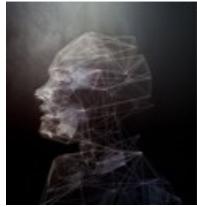
- **Komplexe Anwendungen:** Für Anwendungen, die komplexe Logik erfordern, wie die Verkettung mehrerer LLM-Aufrufe (Chains), die Implementierung von Agenten-Logik mit Entscheidungsfindung und Tool-Orchestrierung, oder anspruchsvolles Memory-Management für langanhaltende Konversationen.¹³
- **Retrieval Augmented Generation (RAG):** Wenn RAG ein Kernbestandteil der Anwendung ist, um LLMs mit externem Wissen anzureichern. LangChain bietet hierfür umfassende Komponenten (Document Loaders, Text Splitters, Vector Stores, Retrievers).¹⁴
- **Modellagnostik:** Wenn die Anwendung so konzipiert werden soll, dass potenziell verschiedene LLM-Provider (neben OpenAI auch Anthropic, Cohere, lokale Modelle etc.) genutzt oder einfach ausgetauscht werden können.¹³
- **Fortgeschrittenes Output-Parsing und Fehlerbehandlung:** Wenn robuste Mechanismen für das Parsen strukturierter Daten (z.B. mit Pydantic-Validierung) und automatische Fehlerkorrekturversuche (z.B. RetryOutputParser) benötigt werden.¹⁰
- **Observability und Debugging mit LangSmith:** Wenn die Nachvollziehbarkeit, das Monitoring und das Debugging komplexer LLM-Ketten und Agenten für die Entwicklung und Wartung essenziell sind. LangSmith ist hier ein mächtiges Werkzeug.¹⁷
- **Beschleunigung der Entwicklung:** Durch die Nutzung vorgefertigter Komponenten, Standardisierungen und Abstraktionen kann LangChain die Entwicklungszeit für bestimmte Anwendungsfälle verkürzen.²⁰

4 LangChain Version v0.3 und v1.0

Aspekt	LangChain v0.3	LangChain v1.0
Stabilität	Verbesserte Fehlerbehandlung, Ressourcenoptimierung	Fokus auf Production-Readiness, robuste Kernkomponenten

Aspekt	LangChain v0.3	LangChain v1.0
Nachrichtenstruktur	Klassische Nachrichten-/Message-Struktur	Neue .content_blocks für strukturierte und typisierte Nachrichten
Abwärtskompatibilität	Migrationshinweise für veraltete Chains, viele Komponenten deprecated	Kleine, stabile Kern-Namespace mit klarer Trennung, Backward Kompatibilität zu den wichtigsten Schnittstellen
Agenten-Entwicklung	Bereits agentische Muster, aber weniger orchestriert	Einbettung von LangGraph als Standard für Agenten, besseres Tooling und orchestration
API/Ökosystem	Split in Community- und Integrations-Pakete, z. T. separate Loader	Vereinheitlichter API-Zugang, zentrale Dokumentations-Hubs in Python & JS
Modell-Integration	Viele Integrationen, aber nicht vollständig einheitlich	Mehr Uniformität, bessere Unterstützung für alle großen LLMs, API-ready für neue Modelle
Installation/Dependencies	Wechsel zu Pydantic 2 (Python), Peer-Dependency für @langchain/core (JavaScript)	Optimiert für minimalen Kern, nur essenzielle Abhängigkeiten nötig
Performance	Reduzierte Latenz und optimierte Algorithmen	Weitere Performance-Verbesserungen, Ready für hochskalierende Anwendungen
Dokumentation	Getrennte Doku für Teilprojekte, teils fragmentiert	Zentrale Docs, vereinheitlichte Developer Guides

M05a - Transformer



Anwendung Generativer KI

Stand: 07.2025

1 Was ist ein Transformer?

Stellen wir uns vor, jemand liest einen Text und versucht ihn zu verstehen. Dabei schaut man nicht nur auf ein Wort nach dem anderen, sondern das Gehirn verbindet alle Wörter miteinander - manche sind wichtiger für das Verständnis als andere.

Ein **Transformer** macht genau das: Es ist ein Computer-Programm, das Texte "liest" und dabei automatisch erkennt, welche Wörter zusammengehören und wichtig sind.

2 Die Grundidee

Nehmen wir den Satz: „*Der Hund bellt laut.*“

- Ein Mensch versteht sofort: „**Hund**“ ist das Subjekt der Handlung
- „**bellt**“ ist das Verb und wird durch „**laut**“ näher beschrieben
- Das Wort „**Der**“ gehört grammatisch zu „**Hund**“, ist aber weniger wichtig für die Bedeutung von „**bellt**“

Ein Transformer-Modell erkennt diese Zusammenhänge über den **Self-Attention-Mechanismus**:

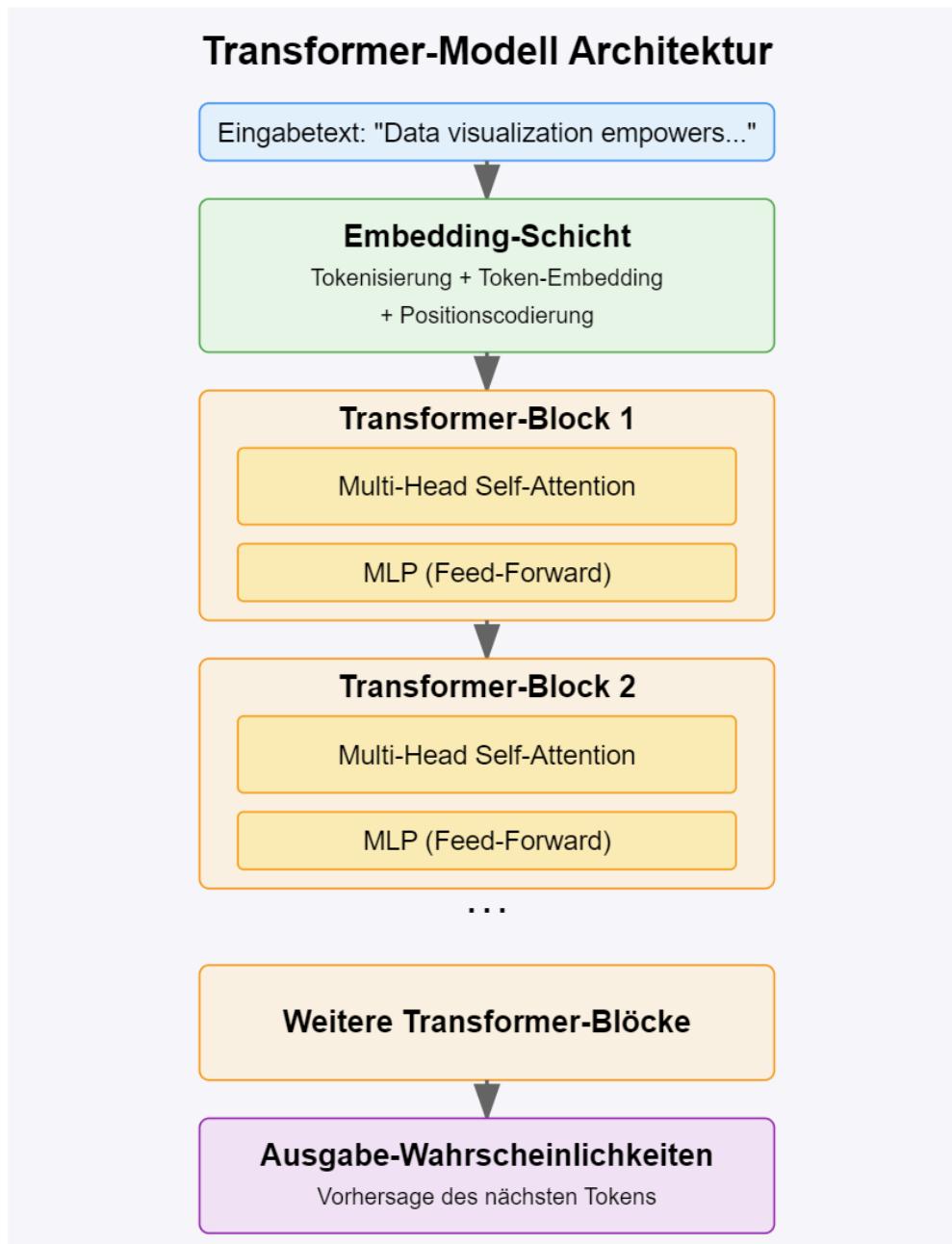
Beim Verarbeiten von „bellt“ wird **besonders auf „Hund“ und „laut“ geachtet**, weil sie für die Bedeutung relevant sind.

„**Der**“ wird zwar berücksichtigt, erhält aber ein geringeres Gewicht.

Ein Transformer lernt diese Verbindungen automatisch durch **Aufmerksamkeit**:

- Es schaut sich alle Wörter gleichzeitig an
- Es berechnet, wie stark jedes Wort mit jedem anderen zusammenhängt
- Wichtige Verbindungen bekommen mehr "Aufmerksamkeit"

3 Wie funktioniert das technisch?



3.1 Wörter werden zu Zahlen → Embedding

- Computer können nur mit Zahlen arbeiten
- Jedes Wort wird in eine Liste von Zahlen umgewandelt (wie ein Fingerabdruck des Wortes)

note Details siehe Skript: M08 - Embeddings

3.2 Position ist wichtig → Positional Encoding

- "Der Lehrer fragt den Schüler." bedeutet etwas anderes als "Den Lehrer fragt der Schüler."

- Der Transformer fügt jedem Wort eine "Positionsnummer" hinzu

3.3 Aufmerksamkeit berechnen → Self-Attention

Self-Attention ist ein Mechanismus, bei dem jedes Wort in einem Satz mit allen anderen Wörtern – einschließlich sich selbst – verglichen wird, um deren inhaltliche Ähnlichkeit oder Relevanz zu bestimmen. Diese Ähnlichkeitswerte helfen dem Modell dabei zu entscheiden, wie stark jedes Wort in die Repräsentation eines anderen Wortes einfließen soll.

Beispiel:

Im Satz „**Der große Hund bellt laut**“ wird das Wort „**bellt**“ mit allen anderen Wörtern verglichen:

- Mit „**Hund**“ besteht eine starke inhaltliche Verbindung, da der Hund die Handlung ausführt.
- „**große**“ beschreibt den Hund und ist damit indirekt relevant.
- „**laut**“ beschreibt das Bellen und ist somit direkt mit „**bellt**“ verbunden.
- „**Der**“ ist ein Artikel und hat weniger inhaltliches Gewicht.

Das Modell erkennt diese Zusammenhänge und gibt den relevanteren Wörtern (z. B. „Hund“ und „laut“) ein höheres Gewicht beim Kodieren von „**bellt**“. So entsteht eine kontextabhängige Repräsentation des Wortes „**bellt**“, die den Gesamtzusammenhang des Satzes besser widerspiegelt.

Um diese Aufmerksamkeit gezielt zu steuern, nutzt der Transformer für jedes Wort eine spezielle Fragetechnik. Dabei wird nicht nur geschaut, **wie stark Wörter miteinander verbunden sind**, sondern auch **in welcher Rolle** sie zueinander stehen.

Dazu stellt sich der Transformer bei jedem Wort drei zentrale Fragen:

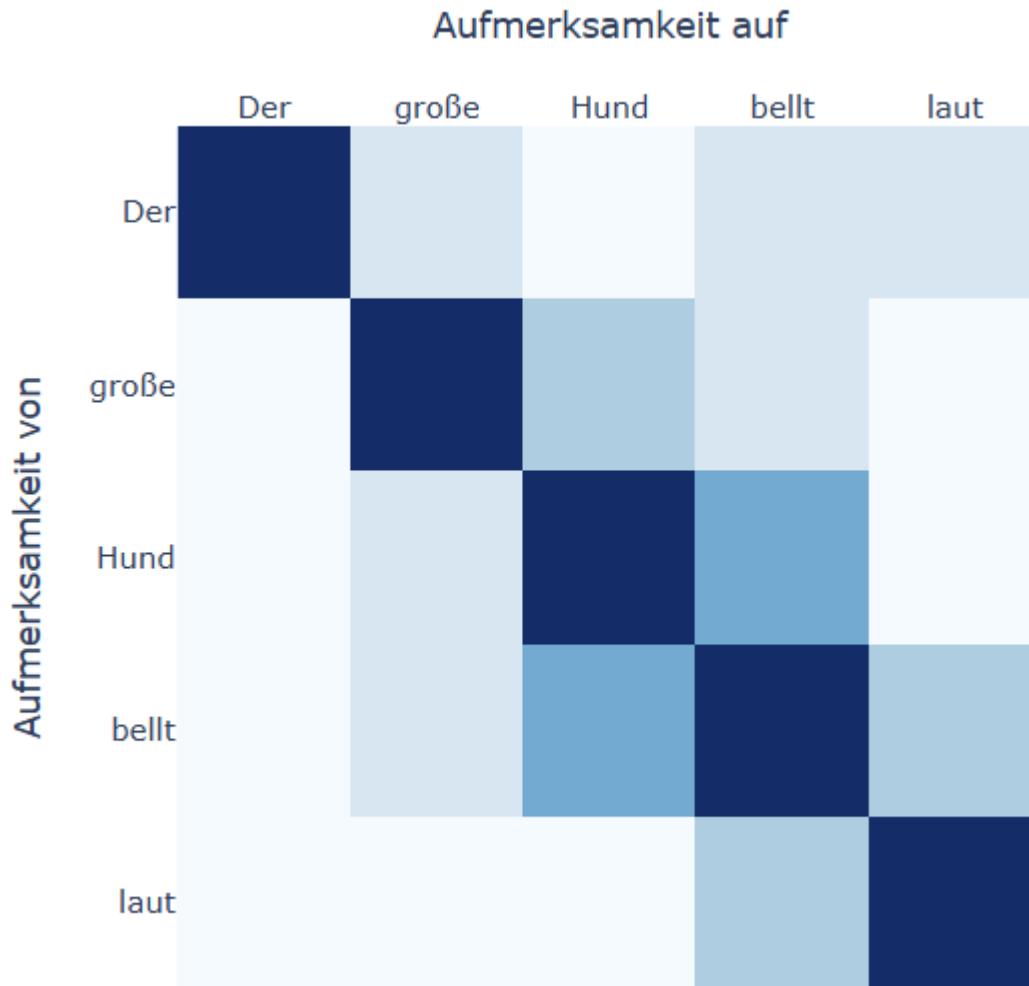
- "**Was suche ich?**" – das ist die **Query**,
- "**Was biete ich an?**" – das ist der **Key**,
- "**Was ist mein Inhalt?**" – das ist der **Value**.

Beispiel mit dem Satz: "Der **große Hund** **bellte laut**."

Das Wort "bellte" analysiert:

- **Query (Was suche ich?)**: "Ich bin ein Verb und suche nach meinem Subjekt - wer macht die Handlung?"
- **Key (Was biete ich an?)**: Von "**Hund**": "Ich bin ein Substantiv und kann Subjekt sein!"
- **Value (Was ist mein Inhalt?)**: Von "**Hund**": "Ich bin ein Tier, männlich, mit der Eigenschaft 'groß'"

Ergebnis: Der Transformer erkennt die starke Verbindung zwischen "Hund" und "bellt" und versteht: "Der Hund führt die Handlung des Bellens aus."



Das passiert gleichzeitig für alle Wörter, sodass der Transformer alle wichtigen Beziehungen im Satz erkennt.

3.4 Mehrere "Köpfe" gleichzeitig → Multi-Head-Attention

Multi-Head-Attention bedeutet, dass der Transformer **mehrere Self-Attention-Mechanismen gleichzeitig ausführt – mit unterschiedlichen Perspektiven (Köpfen)**.

Statt nur **eine Sichtweise** darauf zu berechnen, wie z. B. „bellt“ mit den anderen Wörtern zusammenhängt, berechnet das Modell **mehrere parallele Sichtweisen**, sogenannte „**Heads**“. Jeder Head hat eigene Query-, Key- und Value-Vektoren – mit unterschiedlichem Fokus.

Beispiel mit mehreren Köpfen:

Angenommen, es werden **3 Attention-Heads** verwendet – dann könnten sie sich jeweils auf unterschiedliche Aspekte konzentrieren:

Head	Möglicher Fokus
1	Wer tut etwas? – Beziehung zu „Hund“
2	Wie wird etwas getan? – Beziehung zu „laut“
3	Grammatikalische Struktur – z. B. Artikel

Jeder Head rechnet eigene Attention-Gewichte und neue Repräsentationen für „bellt“. Am Ende werden **alle Ergebnisse zusammengeführt** (konkatenieren und lineare Transformation), um eine **reichhaltige, vielschichtige Repräsentation** zu erzeugen.

Warum ist das sinnvoll?

Ein einzelner Attention-Mechanismus (ein „Kopf“) kann nur eine begrenzte Art von Beziehung gut erfassen. Mit **Multi-Head-Attention** lernt das Modell:

- **verschiedene semantische Beziehungen gleichzeitig zu erkennen**
- **kontextuelle Nuancen** besser zu verarbeiten
- **feinere Unterscheidungen** im Satzbau zu erfassen

3.5 In die Zukunft schauen verboten → Masked-Self-Attention

Problem beim Text-Generieren:

Beim Generieren von Text (z. B. mit GPT) erzeugt das Modell **ein Wort nach dem anderen**.

Dabei soll es **nur auf das schauen dürfen, was bereits gesagt wurde – nicht auf zukünftige Wörter**, die es ja gerade erst selbst erzeugen soll!

Ohne Maske (normale Self-Attention):

Das Modell würde bei der Vorhersage von Wort $n+1$ schon sehen, welches Wort an Position $n+2$ oder $n+3$ kommt. Das wäre **Schummeln**, weil das Modell dann keine echte Vorhersage trifft, sondern aus dem bereits bekannten Ausgang „abliest“.

Mit Masked Self-Attention:

Das Modell wird gezwungen, **zukünftige Wörter zu ignorieren**, indem sie **maskiert (ausgeblendet)** werden.

Es sieht beim Vorhersagen nur den bisherigen Kontext, genau wie ein Mensch, der einen Satz schreibt.

Masked Self-Attention ist notwendig, damit ein Sprachmodell **authentisch, Schritt für Schritt Text erzeugt**, ohne zukünftige Wörter vorab zu kennen.
Es simuliert damit echtes Sprachverständnis im Schreibprozess.

Self-Attention



Self-Attention kann Wörter vor und nach dem Wort von Interesse betrachten

Masked-Self-Attention



...und Masked-Self-Attention ignoriert alles, was nach dem Wort des Interesses kommt.

4 Drei Haupttypen von Transformern

[Transformer](#)

[Autoregressives Modell](#)

4.1 Verstehen - Encoder-Only (wie BERT)

Fachbegriff: Bidirectional Encoder Representations

- Liest den ganzen Text und versteht ihn
- Kann Fragen zum Text beantworten
- Wie ein sehr guter Leser, der alles durchdenkt

Beispiel: "In welchem Jahr wurde Einstein geboren?" → Findet die Antwort im Text

4.2 Schreiben - Decoder-Only (wie ChatGPT)

Fachbegriff: Autoregressive Language Models

- Schreibt Text Wort für Wort
- Jedes neue Wort basiert auf allen vorherigen
- Wie ein Autor, der eine Geschichte weitererzählt

Beispiel: "Es war einmal..." → "Es war einmal ein kleiner Drache, der fliegen lernen wollte..."

4.3 Übersetzen - Encoder-Decoder (wie T5)

Fachbegriff: Sequence-to-Sequence Models

- Liest Input vollständig, dann schreibt Output
- Verbindet Verstehen und Schreiben
- Wie ein Übersetzer, der erst alles versteht, dann übersetzt

Beispiel: "Hello world" → "Hallo Welt"

5 Warum sind Transformer so revolutionär?

5.1 Vorher (alte Methoden):

- Computer lasen Texte Wort für Wort von links nach rechts
- Langsam und vergaßen oft den Anfang des Textes
- Wie jemand, der nur ein Wort nach dem anderen lesen kann

5.2 Transformer:

- Sehen alle Wörter gleichzeitig
- Verstehen Zusammenhänge über weite Strecken
- Viel schneller, weil parallel verarbeitet wird
- Wie jemand, der den ganzen Text auf einmal erfasst

6 Top 10 Post-Transformer

Was kommt als Nächstes? Die KI-Forschung entwickelt sich schnell weiter. Hier sind einige neue Ansätze, die möglicherweise die Zukunft prägen, aber noch nicht weit verbreitet sind:

6.1 Übersicht

Rang	Modelltyp	Was ist anders?	Besonderheit	Transf. basierl
1	State Space Models (SSM)	Sequenzmodell ohne klassische Attention; lineare Skalierbarkeit $O(n)$ statt $O(n^2)$	5× schnellere Inferenz, Verarbeitung von Millionen-Token-Sequenzen	✗ Nein
2	Neural Memory Architectures	Dynamische Speicher-Module mit Kurz- und Langzeitgedächtnis, lernen zur Testzeit	Skalierung auf 2M+ Token, "Needle-in-Haystack" Aufgaben, überraschungsbasiertes Lernen	⚠ Hybride
3	Self-Adaptive Models	Dynamische Gewichtsanpassung zur Laufzeit ohne Retraining	Singular Value Fine-tuning (SVF), Spezialisierung durch "Expert"-Vektoren	✓ Ja
4	Mixture-of-Experts (MoE)	Viele Teilmodelle ("Experten"), nur wenige aktiv pro Input	Riesige Skalierbarkeit bei geringeren Kosten pro Anfrage	✓ Ja (pot.)
5	Continuous Computation Models	Zeitdynamik auf Neuron-Ebene, interne "Denkschritte" unabhängig von Input	Biologisch inspiriert, sequentielle Verarbeitung wie im Gehirn	✗ Nein
6	Test-Time Compute Models	Zusätzliche Rechenzeit während Inferenz für bessere Ergebnisse	"Denken" länger über schwierige Probleme, Chain-of-Thought zur Laufzeit	✓ Ja
7	Multimodale Modelle	Text, Bild, Audio, Video in einheitlicher Architektur verarbeitet	Kombinierte Verständnis & Generierung über Datentypen hinweg	✓ Ja
8	Retrieval-Augmented Generation	Externe Wissensquellen dynamisch zur Laufzeit einbinden	Zugriff auf aktuelle Informationen, reduziert Halluzinationen	✓ Ja (pot.)

Rang	Modelltyp	Was ist anders?	Besonderheit	Transformer-basiert
9	Diffusionsmodelle für Text	Schrittweise Rauschentfernung statt direktes Sampling	Präzise Kontrolle über Stil, Struktur und Textgenerierung	⚠️ Hybrid
10	Neurosymbolische Ansätze	Kombination neuronaler Netze mit symbolischer KI und Logik	Verbesserte Reasoning-Fähigkeiten und logische Schlussfolgerungen	⚠️ Hybrid

Legende:

Transformer-basiert:

- **Ja:** Basiert vollständig auf Transformer-Architektur
- **Nein:** Komplett neue Architektur ohne Transformer-Komponenten
- **Hybrid:** Kombiniert Transformer mit anderen Ansätzen

Entwicklungsstatus:

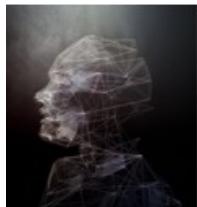
- **Produktiv:** Bereits in kommerziellen Produkten verfügbar
- **Experimentell:** Funktionsfähige Prototypen, noch nicht produktionsreif
- **Forschung:** Frühe Forschungsphase, Konzeptnachweis

6.2 Trends für 2025:

- **Memory-Revolution:** Architekturen mit neuralen Gedächtnis-Modulen
- **SSM-Adoption:** Lineare Skalierung wird Standard für lange Sequenzen
- **Runtime-Adaptation:** Modelle passen sich dynamisch ohne Retraining an
- **Hybrid-Ansätze:** Kombination verschiedener Architekturen für optimale Leistung

Warum ist das wichtig? Diese neuen Ansätze könnten in Zukunft die Transformer-Dominanz herausfordern, besonders wenn es um Effizienz und Geschwindigkeit geht.

M05b - Small Language Models



Anwendung Generativer KI

Stand: 07.2025

1 Was sind Small Language Models?

Small Language Models (SLMs) sind kompakte Versionen großer Sprachmodelle, die darauf optimiert sind, mit deutlich weniger Rechenressourcen effizient zu arbeiten. Während große Modelle wie GPT-4 oder Claude Hunderte von Milliarden Parameter haben, kommen SLMs meist mit 1-20 Milliarden Parametern aus.

2 Hauptmerkmale von SLMs

Kompaktheit: SLMs sind so gestaltet, dass sie auf Standard-Hardware wie Laptops, Smartphones oder kleineren Servern laufen können. Sie benötigen weniger Speicher und Rechenleistung.

Effizienz: Trotz ihrer geringeren Größe können SLMs für spezifische Aufgaben sehr gute Ergebnisse erzielen. Sie sind oft schneller in der Verarbeitung als ihre großen Pendants.

Lokale Ausführung: Ein großer Vorteil ist, dass SLMs lokal ausgeführt werden können, ohne Internetverbindung oder Cloud-Services zu benötigen.

3 Beliebte SLM-Beispiele

Llama 2 7B: Metas kompaktes Modell mit 7 Milliarden Parametern, das sich gut für lokale Anwendungen eignet.

Mistral 7B: Ein effizientes französisches Modell, das trotz seiner Größe starke Leistung zeigt.

Phi-3: Microsofts kleine Modellfamilie, die speziell für mobile Geräte optimiert wurde.

Gemma: Googles kompakte Modelle in verschiedenen Größen (2B, 7B).

DistilBERT: 40% kleiner als BERT, behält 97% der Leistung.

TinyBERT: Noch kompakter mit innovativen Distillation-Techniken.

MobileBERT: Optimiert für mobile Geräte mit BERT-ähnlicher Leistung.

4 Vorteile von SLMs

Datenschutz: Da die Modelle lokal laufen, verlassen sensible Daten nie das eigene System.

Kosteneffizienz: Keine laufenden API-Kosten oder Cloud-Gebühren.

Latenz: Schnellere Antwortzeiten, da keine Netzwerkkommunikation nötig ist.

Anpassbarkeit: SLMs lassen sich leichter für spezifische Anwendungsfälle fine-tunen.

5 Nachteile und Grenzen

Begrenzte Fähigkeiten: SLMs haben weniger Allgemeinwissen und können komplexe Aufgaben nicht so gut bewältigen wie große Modelle.

Weniger Sprachen: Oft sind sie nur auf wenige Sprachen trainiert.

Spezialisierung nötig: Für optimale Ergebnisse müssen SLMs oft für spezifische Aufgaben nachtrainiert werden.

Leistungsverlust: Kleinere Modelle haben weniger Kapazität, komplexe Aufgaben leiden mehr unter Verkleinerung.

6 Vergleich: LLM vs. SLM

Aspekt	Large Language Models (LLM)	Small Language Models (SLM)
Parameterzahl	100+ Milliarden Parameter	1-20 Milliarden Parameter
Modellgröße	50-500+ GB	1-15 GB
Hardware-Anforderungen	High-End GPU/TPU, 40+ GB VRAM	Standard-GPU, 8-16 GB VRAM
Ausführungsort	Cloud/Datacenter	Lokal (Laptop, Smartphone)
Internetverbindung	Erforderlich für API-Zugriff	Optional, lokale Ausführung möglich

Aspekt	Large Language Models (LLM)	Small Language Models (SLM)
Antwortgeschwindigkeit	Langsamer (Netzwerk-Latenz)	Schneller (lokale Verarbeitung)
Kosten	Hohe API-Kosten pro Anfrage	Einmalige Hardware-/Lizenzkosten
Allgemeinwissen	Sehr umfangreich	Begrenzt, spezialisiert
Mehrsprachigkeit	100+ Sprachen	Wenige Hauptsprachen
Komplexe Aufgaben	Exzellent	Gut bei spezifischen Aufgaben
Datenschutz	Daten verlassen das System	Vollständige lokale Kontrolle
Fine-Tuning	Sehr aufwendig und teuer	Einfacher und kostengünstiger
Energieverbrauch	Sehr hoch	Niedrig bis moderat
Verfügbarkeit	Abhängig von Service-Provider	24/7 verfügbar
Beispiele	GPT-4, Claude, Gemini Ultra	Llama 2 7B, Mistral 7B, Phi-3

7 Anwendungsbereiche

Edge Computing: Intelligente Geräte ohne Internetverbindung.

Unternehmen: Verarbeitung sensibler Daten ohne externe Services.

Entwicklung: Lokale Prototypenerstellung ohne API-Abhängigkeiten.

Mobile Apps: KI-Funktionen direkt auf Smartphones.

8 Methoden zur Erstellung von SLMs

8.1 Übersicht der wichtigsten Techniken

Methode	Beschreibung	Ergebnis
Knowledge Distillation	Ein kleines Modell („Student“) lernt, die Ausgaben eines großen Modells („Teacher“) nachzuahmen.	→ SLM wie DistilBERT, TinyBERT

Methode	Beschreibung	Ergebnis
Pruning	Entfernt unwichtige Gewichte/Neuronverbindungen im Netzwerk.	→ Reduziert Modellgröße und Laufzeit
Quantisierung	Repräsentiert Gewichte mit weniger Bits (z. B. 8-bit statt 32-bit).	→ Effizienteres Modell ohne große Einbußen
Low-Rank Approximation	Reduziert die Rechenkomplexität von Matrizenoperationen.	→ Schnelleres, kompakteres Modell
LoRA (Low-Rank Adaptation)	Fügt kleinen Zusatz-Layern zur Effizienzsteigerung beim Finetuning hinzu.	→ Günstiges Finetuning, kleinere Varianten
Weight Sharing	Nutzt dieselben Gewichte mehrfach im Modell (z. B. ALBERT).	→ Geringerer Speicherbedarf

8.2 Knowledge Distillation (Wissensdestillation)

Konzept: Ein großes "Lehrermodell" trainiert ein kleineres "Schülermodell", indem es dessen Ausgaben imitiert.

Vorgehensweise:

- Das große Modell generiert Antworten auf Trainingsdaten
- Das kleine Modell lernt, diese Antworten zu reproduzieren
- Zusätzlich zu den ursprünglichen Labels lernt das Schülermodell von den "weichen" Wahrscheinlichkeitsverteilungen des Lehrers

8.3 Pruning (Beschneidung)

Strukturiertes Pruning: Entfernung ganzer Neuronen, Schichten oder Attention-Heads basierend auf ihrer Wichtigkeit.

Unstrukturiertes Pruning: Entfernung einzelner Gewichte, die nahe null sind oder geringe Bedeutung haben.

Graduelle Pruning: Schrittweise Entfernung von Parametern während des Trainings.

8.4 Quantization (Quantisierung)

Konzept: Reduzierung der Präzision von Modellparametern von 32-Bit auf 16-Bit, 8-Bit oder sogar 4-Bit.

Typen:

- Post-Training Quantization: Quantisierung nach dem Training
- Quantization-Aware Training: Training mit Quantisierung von Anfang an

Effekt: Drastische Reduktion der Modellgröße bei meist geringem Leistungsverlust.

8.5 Low-Rank Approximation

Prinzip: Zerlegung großer Gewichtsmatrizen in kleinere Matrizen mit niedrigem Rang.

Techniken:

- Singular Value Decomposition (SVD)
- LoRA (Low-Rank Adaptation)
- Tensor-Zerlegung

8.6 Architecture Search

Neural Architecture Search (NAS): Automatische Suche nach effizienten Architekturen.

MobileNet-Prinzipien: Verwendung von depthwise separable convolutions und anderen effizienten Bausteinen.

9 Praktische Implementierungsschritte

9.1 Phase 1: Analyse des Ursprungsmodells

- Identifikation der wichtigsten Schichten und Parameter
- Bewertung der Leistung auf relevanten Benchmarks
- Bestimmung der Zielgröße und Leistungsanforderungen

9.2 Phase 2: Auswahl der Verkleinerungsmethode

- Knowledge Distillation für allgemeine Komprimierung
- Pruning für gezielte Parameterreduktion
- Quantization für Speicheroptimierung

9.3 Phase 3: Schrittweise Verkleinerung

- Beginnen mit weniger aggressiven Methoden
- Kontinuierliche Evaluierung der Leistung
- Anpassung der Hyperparameter

9.4 Phase 4: Fine-Tuning

- Nachtraining auf spezifischen Datensätzen
- Wiederherstellung verlorener Fähigkeiten
- Optimierung für Zielanwendung

10 Herausforderungen und Kompromisse

10.1 Leistungsverlust

- Kleinere Modelle haben weniger Kapazität
- Komplexe Aufgaben leiden mehr unter Verkleinerung
- Balance zwischen Größe und Qualität schwierig

10.2 Spezialisierung erforderlich

- SLMs funktionieren oft besser bei spezifischen Aufgaben
- Allgemeine Intelligenz geht verloren
- Domain-spezifisches Training nötig

10.3 Technische Komplexität

- Verschiedene Methoden erfordern unterschiedliche Expertise
- Optimale Kombination von Techniken nicht trivial
- Hardware-spezifische Optimierungen nötig

11 Zukunftsaussichten

SLMs werden immer wichtiger, da sie eine praktische Balance zwischen Leistung und Effizienz bieten. Die Entwicklung geht in Richtung noch effizienterer Architekturen und besserer Optimierungstechniken. Für viele Anwendungen sind sie bereits heute eine ausgezeichnete Alternative zu großen Modellen.

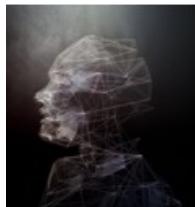
Die Forschung bewegt sich in Richtung noch effizienterer Kompressionstechniken, einschließlich automatisierter Pipelines und hardwarespezifischer Optimierungen. Neue Ansätze wie "Mixture of Experts" ermöglichen es, große Modelle zu haben, die sich wie kleine verhalten, indem nur relevante Teile aktiviert werden.

Die Verkleinerung von LLMs zu SLMs ist eine praktikable Lösung für viele Anwendungen, erfordert aber sorgfältige Planung und oft mehrere Iterationen, um die richtige Balance zwischen Größe und Leistung zu finden.

12 Fazit

Die Wahl zwischen SLMs und großen Modellen hängt vom spezifischen Anwendungsfall ab: Brauchen Sie maximale Intelligenz oder ist Effizienz und lokale Kontrolle wichtiger? SLMs bieten eine ausgezeichnete Lösung für viele praktische Anwendungen, bei denen Datenschutz, Kosteneffizienz und lokale Ausführung wichtiger sind als die absolute Leistungsstärke großer Modelle.

M05c - Mixture of Experts



Anwendung Generativer KI

Stand: 07.2025

1 Was ist Mixture of Experts?

Mixture of Experts (MoE) ist ein fortschrittliches **Machine Learning-Konzept**, bei dem mehrere spezialisierte Modelle (die sogenannten "Experten") zusammenarbeiten, um komplexe Probleme effizienter zu lösen. Anstatt ein einziges, monolithisches Modell zu verwenden, das alles können muss, teilt MoE die Aufgabe auf verschiedene Experten auf, die jeweils auf bestimmte Bereiche oder Datentypen spezialisiert sind.

2 Die Grundidee

Stellen Sie sich vor, Sie haben eine schwierige Frage und können zwischen verschiedenen Fachexperten wählen:

- Für medizinische Fragen fragen Sie einen **Arzt**.
- Für rechtliche Probleme konsultieren Sie einen **Anwalt**.
- Für technische Fragen wenden Sie sich an einen **Ingenieur**.

Genau so funktioniert MoE: Ein "**Gating Network**" (man könnte es als "Türsteher" oder "Verteiler" bezeichnen) entscheidet, welche Experten für eine bestimmte Eingabe am besten geeignet sind und diese dann zur Beantwortung der Frage heranzieht.

3 Hauptkomponenten eines MoE-Systems

Ein MoE-System besteht im Wesentlichen aus drei Schlüsselkomponenten:

3.1 Die Experten

- Dies sind **mehrere spezialisierte Modelle**, oft neuronale Netze.
- Jeder Experte fokussiert sich auf bestimmte Eingabemuster oder Teilaufgaben.

- Sie können unterschiedliche Architekturen haben, je nachdem, worauf sie spezialisiert sind.

3.2 Das Gating Network

- Es ist das **Gehirn** des MoE-Systems.
- Entscheidet dynamisch, **welche Experten aktiviert** werden sollen.
- Bestimmt die **Gewichtung** der Vorhersagen der ausgewählten Experten.
- Lernt im Laufe des Trainings, die beste Kombination von Experten für jede Eingabe zu finden.

3.3 Die Kombination

- Die **finale Vorhersage** des Systems wird durch eine gewichtete Summe der Ausgaben der aktivierten Experten gebildet.
- Das Gating Network bestimmt, wie stark die Meinung jedes Experten in die Endentscheidung einfließt.

4 Vorteile von MoE

MoE-Architekturen bieten mehrere signifikante Vorteile:

- **Spezialisierung:** Jeder Experte kann sich auf das konzentrieren, was er am besten kann, statt ein "Allesköninger" sein zu müssen. Dies führt oft zu einer höheren Genauigkeit in spezifischen Bereichen.
- **Skalierbarkeit:** Neue Experten können hinzugefügt werden, um das System zu erweitern, ohne das gesamte System von Grund auf neu trainieren zu müssen. Dies erleichtert die Anpassung an neue Daten oder Aufgaben.
- **Effizienz:** Nur die relevantesten Experten werden für eine gegebene Eingabe aktiviert, was Rechenressourcen spart und die Inferenzzeit verkürzen kann.
- **Interpretierbarkeit:** Es ist oft leichter nachzuvollziehen, welche Experten für welche Entscheidungen verantwortlich waren, was die Transparenz des Modells erhöht.

5 Einfaches Beispiel

Betrachten wir ein **Bildklassifizierungssystem**:

- **Experte 1:** Spezialisiert auf die Erkennung von **Tieren**.
- **Experte 2:** Spezialisiert auf die Erkennung von **Fahrzeugen**.
- **Experte 3:** Spezialisiert auf die Erkennung von **Pflanzen**.

Wenn ein Bild eines Hundes eingegeben wird, erkennt das Gating Network, dass Experte 1 am relevantesten ist, und gewichtet dessen Vorhersage am stärksten, um das Bild

korrekt als "Hund" zu klassifizieren.

6 Moderne Anwendungen

MoE wird heute in vielen großen und leistungsstarken KI-Systemen eingesetzt, insbesondere dort, wo es um die Verarbeitung riesiger Datenmengen oder die Lösung sehr komplexer Aufgaben geht:

- **Large Language Models (LLMs)**: Modelle wie GPT-4 und PaLM verwenden MoE-Architekturen, um verschiedene Arten von Texten besser zu verstehen, zu generieren und zu verarbeiten.
- **Empfehlungssysteme**: Verschiedene Experten können für unterschiedliche Produktkategorien oder Nutzergruppen zuständig sein, um personalisierte Empfehlungen zu liefern.
- **Computer Vision**: Experten für verschiedene Objekttypen, Bildqualitäten oder Szenarien können die Genauigkeit und Robustheit von Bilderkennungssystemen verbessern.

7 Herausforderungen

Trotz seiner Vorteile birgt MoE auch einige Herausforderungen:

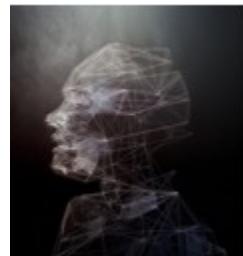
- **Training-Komplexität**: Das gleichzeitige Training mehrerer Experten und des Gating Networks ist anspruchsvoll und erfordert oft spezielle Algorithmen und Techniken.
- **Load Balancing**: Es ist wichtig sicherzustellen, dass alle Experten gleichermaßen genutzt werden und sich das System nicht auf wenige konzentriert. Ungleichmäßige Lastverteilung kann die Effizienz mindern.
- **Overfitting**: Experten können sich zu stark auf ihre spezifischen Trainingsdaten spezialisieren, was zu einer geringeren Generalisierungsfähigkeit bei neuen, ähnlichen Daten führen kann.

8 Fazit

Mixture of Experts ist ein eleganter und leistungsstarker Ansatz im Machine Learning, der die Vorteile der **Spezialisierung** mit der **Flexibilität kombinierter Systeme** verbindet. Es ermöglicht die Lösung komplexer Probleme durch die Zusammenarbeit spezialisierter Komponenten – ein Prinzip, das sowohl in der Natur als auch in menschlichen Organisationen weit verbreitet ist.

Für Einsteiger ist MoE ein faszinierendes Beispiel dafür, wie Machine Learning-Systeme durch clevere architektonische Entscheidungen sowohl leistungsfähiger als auch effizienter werden können. Es zeigt auf eindrucksvolle Weise, dass die Summe der Teile manchmal weit größer sein kann als die einzelnen Komponenten für sich.

M08a - Tokenizing & Chunking



Anwendung Generativer KI

Stand: 05.2025

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.
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.

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
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.
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 gruppiert 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)	Größere 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.

 **Tipp**

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.

Tipp

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 1: Checkliste Parameterwahl

- **1. Analyse der Dokumenteigenschaften**

- 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 Modelleistung.
- 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.

6 Anhang 2: Vom Wort zur Zahl

Warum ist es sinnvoll bei der Verarbeitung von natürlicher Sprache Worte in Zahlen umzuwandeln.

Zahlen sind direkt maschinenlesbar

Computer arbeiten intern mit binären Zahlen (0 und 1). Jede Zahl lässt sich direkt als Bitfolge darstellen. Zum Beispiel:

- Die Zahl **5** ist im Binärsystem **101**.
- Diese Darstellung ist eindeutig, kompakt und sofort verwendbar.

Worte sind komplexe Zeichenfolgen

Worte müssen zunächst **kodiert** werden, bevor der Computer sie verarbeiten kann. Dafür nutzt man z. B.:

- **ASCII** oder **Unicode** zur Umwandlung von Buchstaben in Zahlen.
- „Hallo“ → [72, 97, 108, 108, 111] (im ASCII-Code)

Worte haben Mehrdeutigkeit

Sprache ist für Menschen gemacht – sie ist:

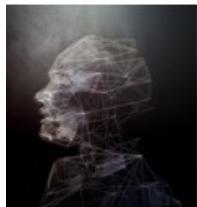
- **mehrdeutig** („Bank“ = Sitzmöbel oder Geldinstitut)
- **kontextabhängig**
- **grammatisch komplex** (Zeitformen, Fälle, Satzbau usw.)

Das macht die Verarbeitung von Sprache viel schwieriger als die von Zahlen, bei denen jede Zahl klar definiert ist.

Zahlen folgen klaren Regeln

Mit Zahlen kann der Computer sofort rechnen, vergleichen oder sortieren. Sie folgen mathematischen Gesetzen, die leicht implementiert sind.

M08b - Embeddings



Anwendung Generativer KI

Stand: 10.2025

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.

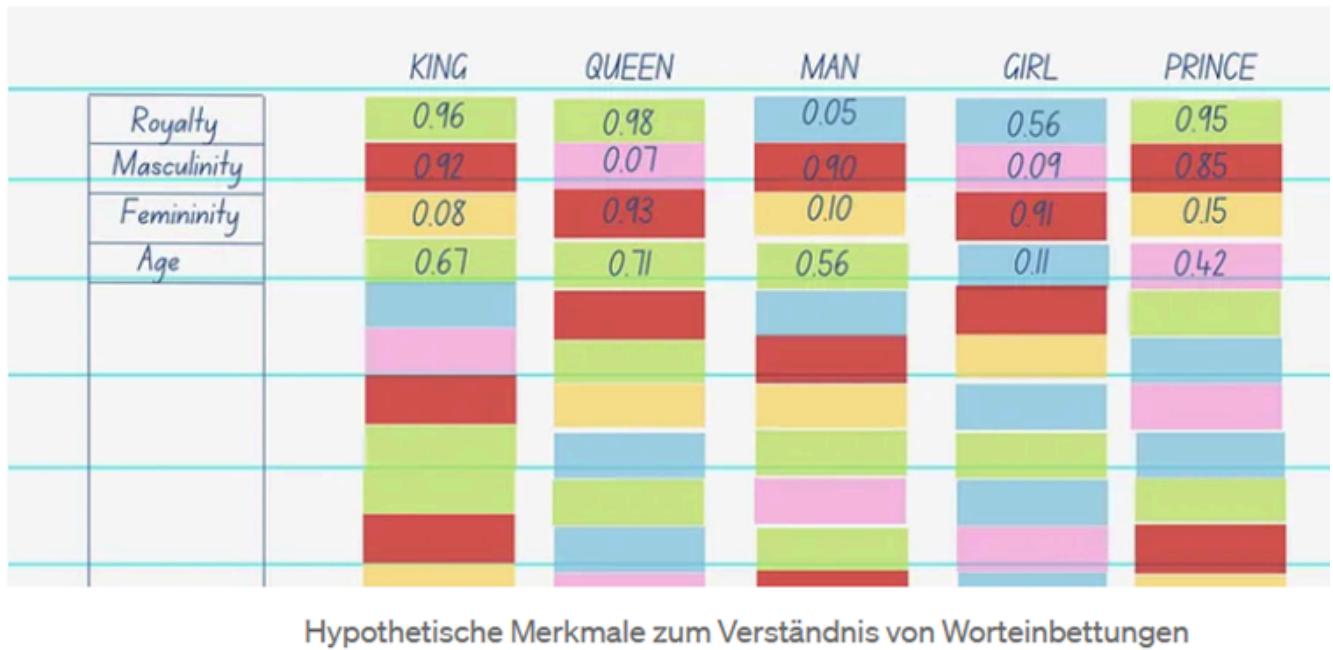
→ Die Zahlen von „King“ und „Queen“ sind **ähnlicher** als die von „Man“ und „Girl“. Dies zeigt, dass die KI die inhaltliche Nähe dieser Begriffe versteht.

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](#)

[Embedding-Beispiel Fahrzeug](#)

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

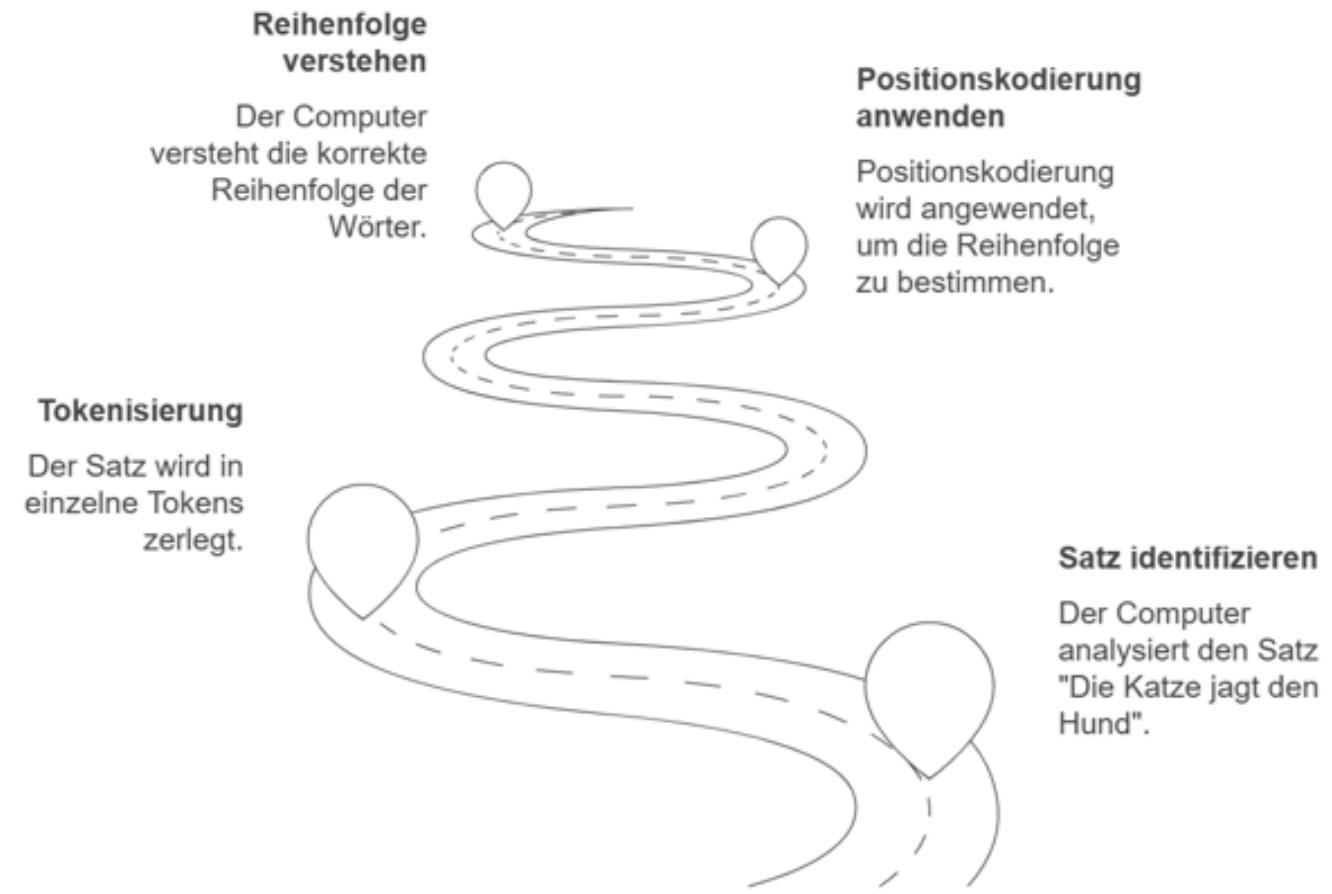


Bild mit napkin.ai erstellt

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.

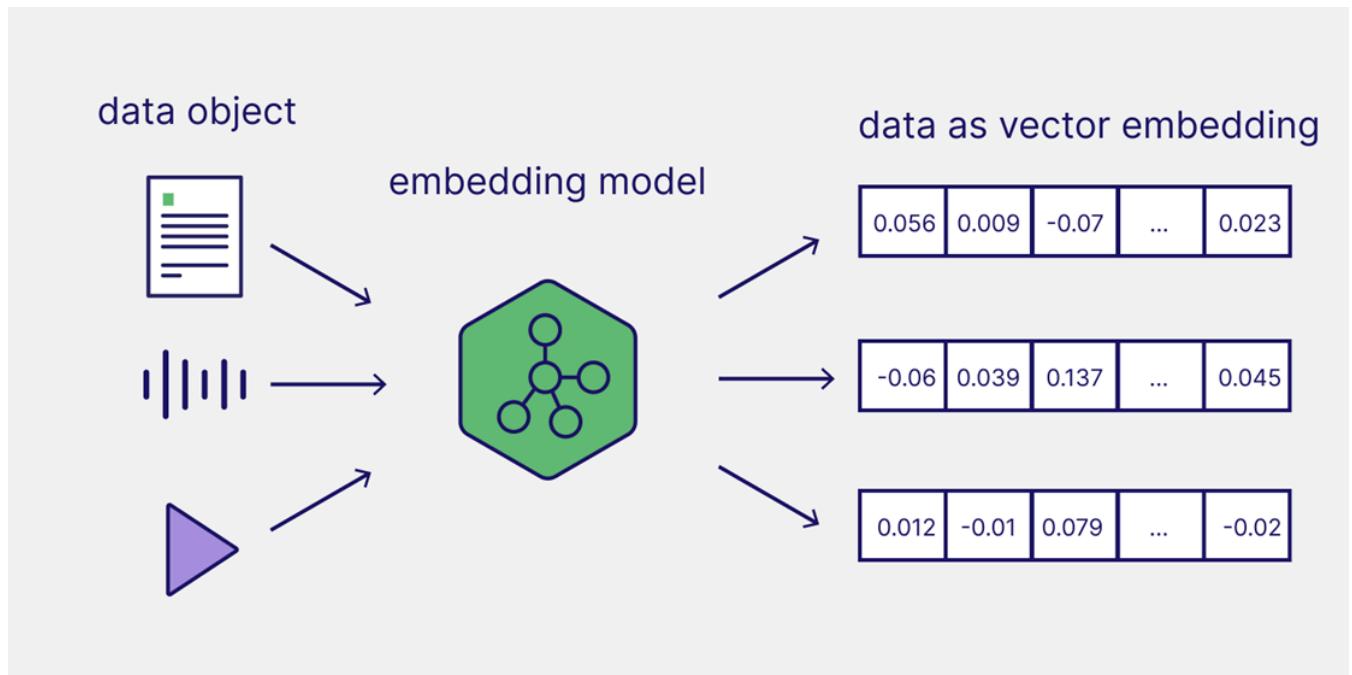


Bild: [Step-by-Step Guide to Choosing the Best Embedding Model for Your Application](#) | [Weaviate - Vector Database](#)

Übersicht Einbettungsmodelle:

Einbettungsvektor	Typische Größen	Einsatzbereich
Word2Vec	100-300 Dimensionen	Wort- und Satzähnlichkeiten, 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

Einbettungsvektor	Typische Größen	Einsatzbereich
Node2Vec	64-256 Dimensionen	Graphenanalysen, soziale Netzwerke
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 Kombi: Embedding - Token - Chunk

Hier ist eine tabellarische Übersicht, die ausgehend vom **Embedding-Modell** zeigt, welche **Tokenizer** und **Chunking-Strategien** zulässig oder üblich sind

Embeddingmodell	Zulässiger Tokenizer
text-embedding-ada-002	<code>tiktoken.encoding_for_model("text-embedding-ada-002")</code>
text-embedding-3-small	<code>tiktoken.encoding_for_model("text-embedding-3-small")</code>
text-embedding-3-large	<code>tiktoken.encoding_for_model("text-embedding-3-large")</code>

Embeddingmodell	Zulässiger Tokenizer
all-MiniLM-L6-v2	<code>transformers.AutoTokenizer.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")</code>
sentence-transformers/...	<code>transformers.AutoTokenizer.from_pretrained(...)</code>
e5-base-v2	<code>transformers.AutoTokenizer.from_pretrained("intfloat/e5-base-v2")</code>

7 Warum sind Embeddings so wichtig?

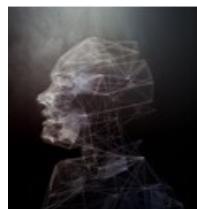
- **Sprachverarbeitung:** Chatbots, Übersetzungen und Textanalysen basieren auf Embeddings.
- **Bilderkennung:** KI kann ähnliche Bilder oder Objekte erkennen.
- **Suche & Empfehlungssysteme:** Personalisierte Vorschläge auf Plattformen wie Netflix, Spotify oder YouTube nutzen Embeddings.
- **Musik- 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.

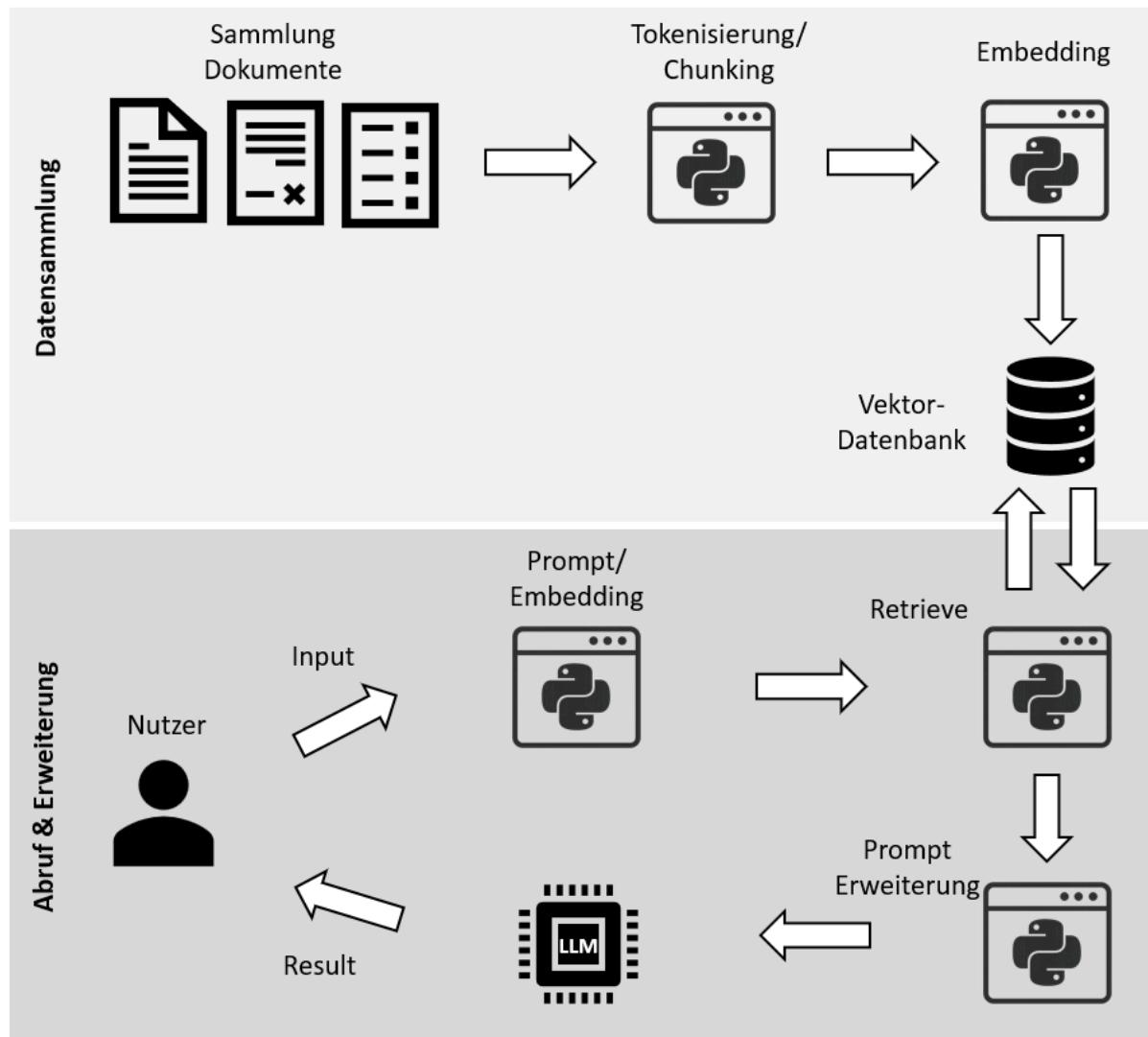
M08c - RAG-Optimierungsansätze



Anwendung Generativer KI

Stand: 07.2025

1 RAG-Prozess



2 Chunking & Preprocessing

Adaptive Chunking: Dynamische Anpassung der Chunk-Größe basierend auf Dokumententyp und Inhaltsstruktur. Berücksichtigt semantische Grenzen statt starrer Zeichenlimits.

Hierarchical Chunking: Mehrstufige Segmentierung mit Parent-Child-Beziehungen zwischen Chunks. Ermöglicht sowohl detaillierte als auch übergeordnete Kontextinformationen.

Overlapping Chunks: Überlappende Textabschnitte zur Vermeidung von Informationsverlust an Chunk-Grenzen. Typischerweise 10-20% Überlappung zwischen benachbarten Chunks.

Document Structure Awareness: Berücksichtigung von Dokumentenstruktur wie Überschriften, Absätze und Listen beim Chunking. Erhält logische Zusammenhänge und semantische Einheiten.

Chunk Merging (Auto-Merging Retrieval): Organisiert Chunks in Parent-Child-Beziehungen und fügt bei Bedarf Kontext hinzu. Besonders effektiv wenn relevante Informationen in geclusterten Bereichen vorkommen. Ermöglicht dynamische Anpassung des Kontextumfangs basierend auf Retrieval-Ergebnissen.

Metadata Inclusion: Anreicherung von Chunks mit strukturierten Metadaten wie Quelle, Datum, Autor oder Dokumenttyp. Ermöglicht präzises Filtern und verbessert die Retrieval-Genauigkeit durch Eingrenzung des Suchraums.

3 Embedding-Optimierung

Fine-tuned Embeddings: Anpassung von Embedding-Modellen auf domänenspezifische Daten. Verbessert die Repräsentation fachspezifischer Terminologie und Konzepte.

Multi-Modal Embeddings: Integration verschiedener Datentypen (Text, Bilder, Tabellen) in einheitliche Vektorrepräsentationen. Ermöglicht umfassendere Dokumentenverarbeitung.

Embedding Ensemble: Kombination mehrerer Embedding-Modelle zur Verbesserung der Robustheit. Reduziert Bias einzelner Modelle und erhöht Recall.

Contextual Embeddings: Berücksichtigung des Dokumentenkontexts bei der Embedding-Erstellung. Verbessert die Disambiguierung mehrdeutiger Begriffe.

Embedding Adapters: Externe Adapter zur Optimierung der Alignment zwischen Retriever-Outputs und LLM-Präferenzen. Transformiert Query-Embeddings in einen

aufgabenspezifischen latenten Raum. Ermöglicht flexible Anpassung ohne vollständiges Retraining.

4 Retrieval & Search

Hybrid Search: Kombination von semantischer (Vektor-) und lexikalischer (BM25) Suche. Nutzt Stärken beider Ansätze für verbesserte Relevanz.

Query Expansion: Erweiterung der Benutzeranfrage um verwandte Begriffe und Synonyme. Erhöht die Wahrscheinlichkeit, relevante Dokumente zu finden.

Multi-Query Generation: Generierung mehrerer Varianten der ursprünglichen Anfrage. Verbessert die Abdeckung verschiedener Formulierungsmöglichkeiten.

Hypothetical Document Embedding (HyDE): Generierung hypothetischer Antworten zur Verbesserung der Suchqualität. Überbrückt die Lücke zwischen Frage und Antwort-Stil.

Dense Passage Retrieval (DPR): Verwendet separate Encoder für Queries und Dokumente, die speziell für maximale Retrieval-Performance trainiert werden. Übertrifft oft traditionelle Embedding-Ansätze bei der semantischen Suche.

Sentence-Level Retrieval (Sentence Window Retrieval): Speichert Embeddings auf Satzebene, erweitert aber bei der Generierung den Kontext um umgebende Sätze. Verbessert Präzision bei gleichzeitiger Beibehaltung des Kontexts.

5 Ranking & Reranking

Cross-Encoder Reranking: Verwendung spezialisierter Modelle zur Neubewertung der Top-K Retrieval-Ergebnisse. Verbessert die Rangfolge durch paarweise Vergleiche.

Diversity-Aware Ranking: Berücksichtigung der Diversität bei der Auswahl relevanter Chunks. Verhindert Redundanz und verbessert Informationsabdeckung.

Temporal Reranking: Bevorzugung aktueller Informationen bei zeitkritischen Anfragen. Besonders relevant für sich schnell ändernde Domänen.

User Context Reranking: Personalisierung der Ergebnisse basierend auf Benutzerkontext und -historie. Verbessert die Relevanz für individuelle Bedürfnisse.

6 Context Management

Context Compression: Intelligente Kürzung und Verdichtung des abgerufenen Kontexts. Reduziert Rauschen und fokussiert auf relevante Informationen.

Context Windowing: Dynamische Anpassung der Kontextfenstergröße basierend auf Anfragekomplexität. Optimiert Balance zwischen Vollständigkeit und Effizienz.

Multi-Hop Reasoning: Iterative Retrieval-Zyklen für komplexe Anfragen. Ermöglicht schrittweise Informationsbeschaffung und -verknüpfung.

Context Deduplication: Entfernung redundanter Informationen aus dem zusammengestellten Kontext. Verbessert Fokus und reduziert Verwirrung.

7 Generation & Prompting

Prompt Engineering: Optimierung der Systemprompts für bessere Antwortqualität. Berücksichtigt Kontext-Integration und Antwortstruktur.

Chain-of-Thought Prompting: Anleitung des Modells zu schrittweisem Denken. Verbessert Nachvollziehbarkeit und Qualität komplexer Antworten.

Self-Reflection: Integration von Selbstbewertungsmechanismen in die Generierung. Ermöglicht Qualitätskontrolle und Korrektur von Fehlern.

Citation Generation: Automatische Generierung von Quellenangaben für Antworten. Verbessert Vertrauenswürdigkeit und Nachverfolgbarkeit.

8 Evaluation & Monitoring

Automated Evaluation: Verwendung von Metriken wie RAGAS, BLEU und semantischer Ähnlichkeit. Ermöglicht kontinuierliche Qualitätsmessung.

Human-in-the-Loop Evaluation: Integration menschlicher Bewertung für kritische Anwendungen. Validiert automatische Metriken und deckt Edge Cases auf.

A/B Testing: Systematischer Vergleich verschiedener RAG-Konfigurationen. Identifiziert optimale Parameterkombinationen datengetrieben.

Failure Analysis: Systematische Analyse von Fehlerfällen und Verbesserungsmöglichkeiten. Ermöglicht gezielte Optimierungsmaßnahmen.

9 System Architecture

Modular RAG Architecture: Aufbau flexibler, austauschbarer Komponenten. Ermöglicht einfache Anpassung und Skalierung einzelner Teile.

Caching Strategies: Implementierung intelligenter Caching-Mechanismen für Embeddings und Ergebnisse. Reduziert Latenz und Rechenaufwand.

Parallel Processing: Parallelisierung von Retrieval und Embedding-Operationen. Verbessert Durchsatz und Antwortzeiten.

Incremental Updates: Effiziente Aktualisierung der Wissensbasis ohne vollständige Neuindizierung. Ermöglicht zeitnahe Integration neuer Informationen.

10 Zusätzliche wichtige Aspekte

Knowledge Graph Integration: Einbindung strukturierter Wissensrepräsentationen zur Verbesserung der Retrieval-Qualität. Ermöglicht bessere Berücksichtigung von Beziehungen zwischen Konzepten.

Multi-Agent RAG: Verwendung mehrerer spezialisierter Agenten für verschiedene Aufgaben. Verbessert Effizienz durch Aufgabenteilung und Expertenwissen.

Feedback Loop Integration: Kontinuierliche Verbesserung durch Nutzer-Feedback und Interaktionsdaten. Ermöglicht selbstlernende Systeme.

Diese Übersicht deckt die wesentlichen Optimierungsansätze für RAG-Systeme ab und bietet eine strukturierte Grundlage für gezielte Verbesserungsmaßnahmen.

11 Prozesszuordnung

Optimierungsansatz	Data Collection Process	Inference Process	Einsteiger
Chunking & Preprocessing	✓		✓
Adaptive Chunking	✓		
Hierarchical Chunking	✓		
Overlapping Chunks	✓		
Document Structure Awareness	✓		
Chunk Merging (Auto-Merging Retrieval)	✓		
Metadata Inclusion	✓		✓
Embedding-Optimierung	✓		✓
Fine-tuned Embeddings	✓		
Multi-Modal Embeddings	✓		
Embedding Ensemble	✓		
Contextual Embeddings	✓	(✓ bei Query)	
Embedding Adaptors	✓		

Optimierungsansatz	Data Collection Process	Inference Process	Einstieger
Retrieval & Search		✓	✓
Hybrid Search		✓	
Query Expansion		✓	
Multi-Query Generation		✓	
Hypothetical Document Embedding (HyDE)	✓ (*)	✓ (*)	
Dense Passage Retrieval (DPR)		✓	
Sentence-Level Retrieval		✓	
Ranking & Reranking		✓	
Cross-Encoder Reranking		✓	
Diversity-Aware Ranking		✓	
Temporal Reranking		✓	
User Context Reranking		✓	
Context Management		✓	✓
Context Compression		✓	
Context Windowing		✓	✓
Multi-Hop Reasoning		✓	
Context Deduplication		✓	
Generation & Prompting		✓	✓
Prompt Engineering		✓	✓
Chain-of-Thought Prompting		✓	
Self-Reflection		✓	
Citation Generation		✓	(✓)
Evaluation & Monitoring	✓ (Monitoring)	✓ (Evaluation)	✓
Automated Evaluation	✓	✓	
Human-in-the-Loop Evaluation	✓	✓	
A/B Testing	✓	✓	
Failure Analysis	✓	✓	
System Architecture	✓	✓	
Modular RAG Architecture	✓	✓	
Caching Strategies	✓	✓	
Parallel Processing	✓	✓	

Optimierungsansatz	Data Collection Process	Inference Process	Einsteiger
Incremental Updates	✓		
Zusätzliche Aspekte	✓	✓	
Knowledge Graph Integration	✓	✓	
Multi-Agent RAG	✓	✓	
Feedback Loop Integration	✓	✓	

(*) HyDE kann sowohl im Data Collection Process (Vorbereitung) als auch im Inference Process genutzt werden.

(✓) Citation Generation ist für Einsteiger optional hilfreich, aber kein Muss.

M09 - Multimodal Bild



Anwendung Generativer KI

Stand: 03.2025

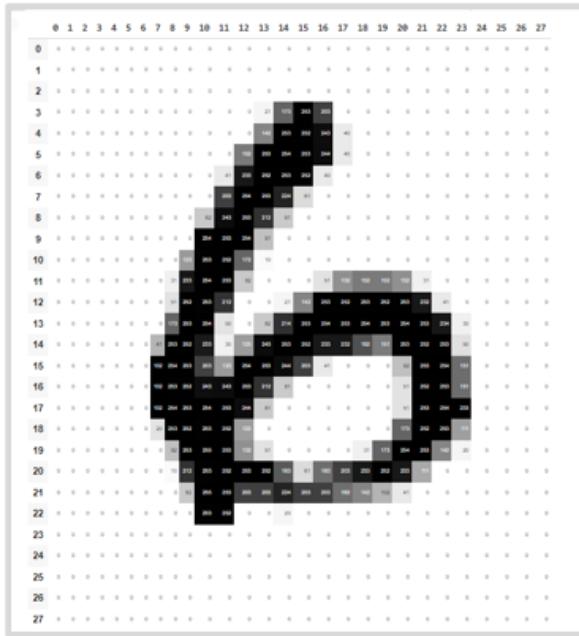
1 Grundlagen Bilderkennung

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.

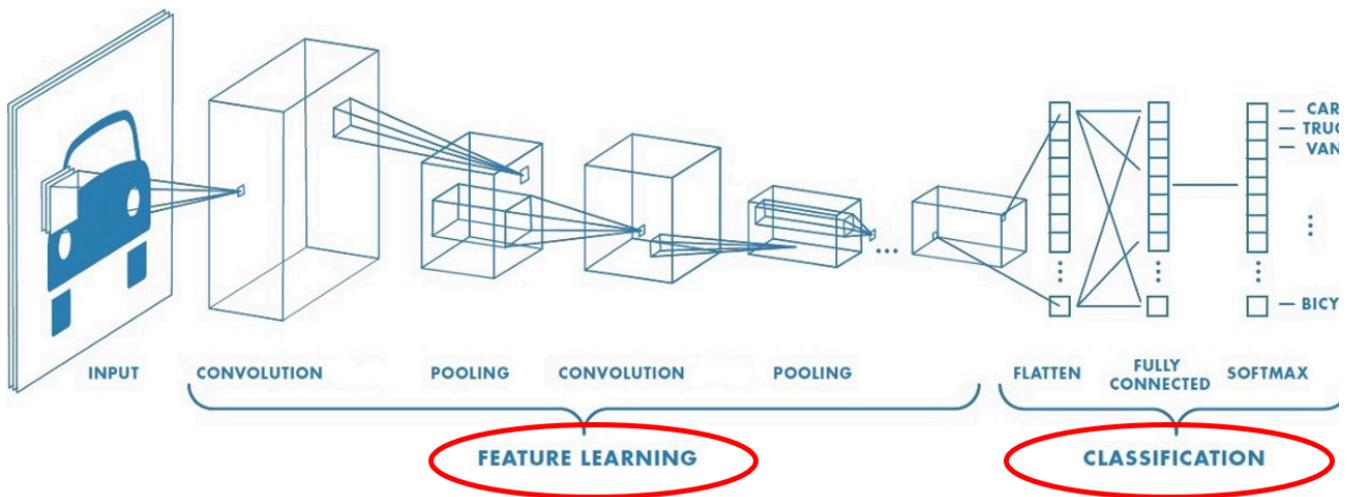


Bild: [A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science](#)

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:



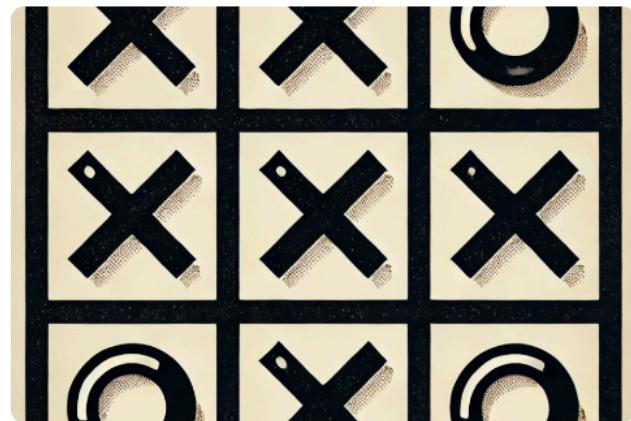
3.2 Multimodale Modelle

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.



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.

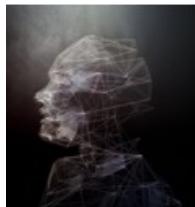
4 Image-Embeddings

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](#)

M14 - Bibliothek - multimodal_rag_modul



Anwendung Generativer KI

Stand: 10.2025

Allgemeines:

Im Rahmen dieses Einführungskurses werden wir uns nicht durch hunderte von Zeilen Code durcharbeiten können, das wäre viel zu aufwendig und würde den Kern des Themas verfehlten. Stattdessen werden wir mit Bibliotheken arbeiten, die uns einen Teil der komplexen, technischen Details abnehmen. So können wir uns darauf konzentrieren, was wirklich wichtig ist: nämlich das Verständnis der generativen KI-Konzepte und ihre Anwendung. Kurz gesagt: Ihr müsst nicht alles von Grund auf selbst programmieren.

1 Übersicht

Das **Multimodale RAG Modul** ist ein produktionsreifes Retrieval-Augmented Generation (RAG) System, das Text- und Bilddokumente in einer einheitlichen Vektordatenbank verwaltet und intelligent durchsucht.

Hauptmerkmale:

- 🔎 **Hybride Suche:** Kombination aus Text-Embeddings (OpenAI) und Bild-Embeddings (CLIP)
- 🖼 **Automatische Bildbeschreibungen:** KI-generierte Beschreibungen via GPT-4o-mini
- ⚡ **Multimodale Suchrichtungen:** Text→Text, Text→Bild, Bild→Bild, Bild→Text
- 📁 **ChromaDB Backend:** Persistente Vektordatenbank mit separaten Collections
- 🏗️ **Funktionale Architektur:** Klare Trennung von Konfiguration, Komponenten und Logik

2 Installation

```
# Erforderliche Pakete
pip install langchain langchain-openai langchain-chroma
pip install sentence-transformers chromadb
pip install markitdown pillow openai
```

3 Schnellstart

```
from multimodal_rag_modul import (
    init_rag_system_enhanced,
    process_directory,
    multimodal_search,
    search_text_by_image,
    search_similar_images
)

# 1. System initialisieren
rag = init_rag_system_enhanced()

# 2. Dokumente und Bilder verarbeiten
stats = process_directory(
    rag,
    './files',
    include_images=True,
    auto_describe_images=True
)

# 3. Multimodale Suche durchführen
result = multimodal_search(rag, "Was sind Cyborgs?")
print(result)
```

4 Suchfunktionen

4.1 Text → Text/Bild Suche

```
# Klassische textbasierte Suche mit optionaler Bildanreicherung
result = multimodal_search(
    rag,
    query="Roboter mit Emotionen",
    k_text=3,           # Anzahl Text-Ergebnisse
    k_images=3,         # Anzahl Bild-Ergebnisse
    enable_cross_modal=True
)
```

Liefert:

- LLM-generierte Antwort basierend auf Textdokumenten
 - Relevante Textquellen mit Ähnlichkeitswerten
 - Visuell passende Bilder via CLIP
 - Cross-Modal gefundene Bilder über Textbeschreibungen
-

4.2 Bild → Bild Suche

```
# Finde visuell ähnliche Bilder
similar_images = search_similar_images(
    rag,
    query_image_path="./mein_roboter.jpg",
    k=5
)

for img in similar_images:
    print(f"{img['filename']}: {img['similarity']}")
    print(f"  Beschreibung: {img['description']}")
```

Nutzt:

- CLIP-Embeddings für visuelle Ähnlichkeit
 - Cosinus-Ähnlichkeit im Embedding-Raum
-

4.3 Bild → Text Suche (NEU in v3.0)

```
# Finde Textinformationen zu einem Bild
result = search_text_by_image(
    rag,
    query_image_path="./cyborg_bild.png",
    k=3,           # Anzahl ähnlicher Bilder
    k_text=3       # Anzahl Text-Dokumente
)
```

Funktionsweise:

1. Findet visuell ähnliche Bilder via CLIP
2. Holt deren Beschreibungen aus der Datenbank
3. **NEU:** Nutzt Beschreibungen für semantische Textsuche
4. Findet relevante Text-Dokumente basierend auf Bildinhalten
5. Generiert Zusammenfassung aus Bildern UND Texten

Ausgabe:

- LLM-Zusammenfassung (Bilder + Texte)
- Ähnliche Bilder mit Beschreibungen
- Relevante Text-Dokumente mit Ähnlichkeitswerten

5 Architektur

5.1 Datenmodell

```
ChromaDB (./multimodal_rag_db/)
├── texts_collection (OpenAI Embeddings)
│   ├── Text-Dokumente (doc_type: "text_document")
│   │   └── Chunks mit Metadaten
│   └── Bildbeschreibungen (doc_type: "image_description")
│       └── GPT-4o-mini generierte Beschreibungen
|
└── images_collection (CLIP Embeddings)
    └── Bilder mit Cross-References zu text_collection
```

5.2 Komponenten

RAGConfig

- Zentrale Konfigurationsklasse
- Anpassbare Parameter (chunk_size, models, thresholds)

RAGComponents

- Container für alle System-Komponenten
- Text-Embeddings, CLIP-Model, LLMs, Collections

Hauptfunktionen

- `init_rag_system_enhanced()` : System-Initialisierung
 - `process_directory()` : Bulk-Import von Dateien
 - `add_text_document()` : Einzelnes Dokument hinzufügen
 - `add_image_with_description()` : Bild mit Auto-Beschreibung
 - `search_texts()` : Text-Suche inkl. Bildbeschreibungen
 - `search_images()` : CLIP-basierte Bildsuche
 - `search_similar_images()` : Bild→Bild Ähnlichkeitssuche
 - `search_text_by_image()` : Bild→Text Suche (NEU)
 - `multimodal_search()` : Erweiterte multimodale Suche
-

6 Unterstützte Dateiformate

6.1 Text-Dokumente

- `.txt` - Plain Text
- `.md` - Markdown
- `.pdf` - PDF-Dokumente
- `.docx` - Word-Dokumente
- `.html` - HTML-Dateien

6.2 Bilder

- `.jpg` / `.jpeg`
 - `.png`
 - `.gif`
 - `.bmp`
-

7 Konfigurationsoptionen

```

from multimodal_rag_modul import RAGConfig

# Benutzerdefinierte Konfiguration
config = RAGConfig(
    chunk_size=200,                                # Text-Chunk-Größe
    chunk_overlap=20,                               # Overlap zwischen Chunks
    text_threshold=1.2,                            # Text-Ähnlichkeits-Schwellwert
    image_threshold=0.8,                           # Bild-Ähnlichkeits-Schwellwert
    clip_model='clip-ViT-B-32',                   # CLIP-Modell
    text_model='text-embedding-3-small',           # OpenAI Embedding-Modell
    llm_model='gpt-4o-mini',                      # LLM für Textgenerierung
    vision_model='gpt-4o-mini',                   # Vision-LLM für Bildbeschreibungen
    db_path='./custom_rag_db'                     # Datenbank-Pfad
)

rag = init_rag_system_enhanced(config)

```

8 Modalitätsmatrix

Eingabe (Query)	Ausgabe	Funktion	Status
Text	Text + Bilder	multimodal_search()	✓
Text	Nur Text	search_texts()	✓
Text	Nur Bilder	search_images()	✓
Bild	Ähnliche Bilder	search_similar_images()	✓
Bild	Text + Bilder	search_text_by_image()	✓

9 Performance-Optimierungen

9.1 Implementiert in v3.0

1. **Batch-Retrieval:** Vermeidung von N+1 Query-Problem
 - Cross-Modal-Retrieval in einem einzigen Batch-Call

2. **Effiziente Filterung:** Trennung von Text-Dokumenten und Bildbeschreibungen
 - Direkte Filterung über Metadaten
 3. **Score-Normalisierung:** ChromaDB L2-Distanz → Ähnlichkeitswert
 - Konsistente Ähnlichkeitswerte (0-1 Range)
 4. **Optimierte Embeddings:** Wiederverwendung von bereits berechneten Embeddings
 - Keine doppelte Embedding-Berechnung
-

10 Anwendungsfälle

10.1 Multimodale Wissensdatenbank

- Verwalte Produktkataloge mit Bildern und Beschreibungen
- Durchsuche technische Dokumentation mit Diagrammen
- Bilde-zu-Text Retrieval für wissenschaftliche Paper

10.2 Content Discovery

- "Finde ähnliche Produkte zu diesem Bild"
- "Welche Textinformationen passen zu diesem Screenshot?"
- "Zeige verwandte Artikel mit Bildern"

10.3 Forschung & Analyse

- Visuelle Ähnlichkeitsanalyse in Bildarchiven
 - Semantische Suche über Multimodale Daten
 - Cross-Modal Information Retrieval
-

11 Fehlerbehebungen (v3.0)

11.1 ChromaDB Where-Klausel Fix

Problem: ChromaDB erwartet `$and` Operator bei mehreren Filterbedingungen

Lösung:

```
# Vorher (fehlerhaft)
components.text_collection.get()
```

```
    where={"source": path, "doc_type": "image_description"}  
)  
  
# Nachher (korrekt)  
components.text_collection.get(  
    where={  
        "$and": [  
            {"source": path},  
            {"doc_type": "image_description"}  
        ]  
    }  
)
```

12 API-Referenz

12.1 System-Management

```
init_rag_system_enhanced(config=None)
```

- Initialisiert das RAG-System
- Parameter: Optional RAGConfig Instanz
- Returns: RAGComponents

```
get_system_status(components)
```

- Gibt System-Status zurück
- Returns: Dict mit Statistiken

```
cleanup_database(db_path='./multimodal_rag_db')
```

- Löscht die Datenbank komplett

12.2 Dokument-Verarbeitung

```
add_text_document(components, file_path)
```

- Fügt ein Text-Dokument hinzu
- Returns: bool (Erfolg)

```
add_image_with_description(components, image_path, auto_describe=True)
```

- Fügt Bild mit Beschreibung hinzu

- Returns: (success: bool, text_doc_id: str)

```
process_directory(components, directory, include_images=True,  
auto_describe_images=True)
```

- Verarbeitet alle Dateien rekursiv
 - Returns: Dict mit Statistiken
-

13 Lizenz

MIT License - Copyright (c) 2025 Ralf

14 Version History

14.1 v3.0 (Oktober 2025)

- ⚡ NEU: `search_text_by_image()` mit Text-Dokumenten-Suche
- 🐛 FIX: ChromaDB where-Klausel für mehrere Bedingungen
- ⚡ Performance: N+1 Query Problem behoben
- 🎨 Erweiterte LLM-Prompts für bessere Zusammenfassungen

14.2 v2.0

- Bild→Bild Suche via CLIP
- Automatische Bildbeschreibungen
- Cross-Modal Retrieval

14.3 v1.0

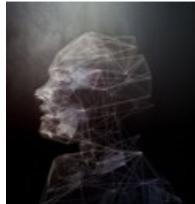
- Basis RAG-System
 - Text-Suche
 - ChromaDB Integration
-

15 Support & Kontakt

- **Repository:** [GenAI/01_ipynb/multimodal_rag_modul.py](#)

- **Autor:** Enhanced by Claude
- **Datum:** Oktober 2025

M16 - Multimodel Audio



Anwendung Generativer KI

Stand: 05.2025

1 Technische Grundlagen

Bevor wir in die praktische Anwendung von Audio-KI eintauchen, ist es wichtig, die grundlegenden technischen Konzepte zu verstehen, die hinter diesen Modellen stehen.

1.1 Von der Schallwelle zum digitalen Signal

Audio ist physikalisch betrachtet eine Schallwelle, die durch Druckschwankungen in der Luft entsteht. Um mit Computern verarbeitet zu werden, muss dieser analoge Schall in ein digitales Signal umgewandelt werden:

1. **Sampling (Abtastung)**: Der kontinuierliche Schall wird in regelmäßigen Zeitabständen gemessen. Die **Abtastrate** (Sampling Rate) gibt an, wie viele Messungen pro Sekunde durchgeführt werden. CD-Qualität verwendet z.B. 44.100 Messungen pro Sekunde (44,1 kHz).
2. **Quantisierung**: Jeder gemessene Wert wird in eine Zahl umgewandelt. Die **Bittiefe** bestimmt, wie genau diese Umwandlung ist. 16-Bit-Audio kann 65.536 verschiedene Lautstärkewerte darstellen.

[Audio_Viz](#)

[MediaPipe](#)

1.2 Wie funktionieren Audio-KI-Modelle?

1.3 Speech-to-Text (Whisper)

OpenAI's Whisper nutzt eine **Encoder-Decoder-Architektur** mit Transformer-Technologie:

- Der **Encoder** wandelt das Audiosignal in eine kompakte Repräsentation um

- Der **Decoder** übersetzt diese Repräsentation in Text

Whisper wurde mit über 680.000 Stunden mehrsprachiger Audiodaten trainiert, wodurch es verschiedene Sprachen, Akzente und Umgebungsgeräusche verarbeiten kann.

1.4 Text-to-Speech (TTS-1)

TTS-1 verwendet ebenfalls eine komplexe neuronale Netzwerkarchitektur:

1. **Text-Encoder**: Wandelt Text in linguistische Merkmale um
2. **Prosody-Predictor**: Bestimmt Betonung, Rhythmus und Melodie
3. **Vocoder**: Erzeugt aus diesen Informationen naturgetreue Sprachsignale

Diese Komponenten arbeiten zusammen, um Text in natürlich klingende Sprache umzuwandeln, die Emotionen und Betonungen enthält.

1.5 Von der Audiowelle zum Verständnis

Wie "verstehen" KI-Modelle Audioinhalte? Der Prozess umfasst mehrere Schritte:

1. **Feature-Extraktion**: Aus dem Audiosignal werden charakteristische Merkmale extrahiert, z.B. durch Spektrogramme (visuelle Darstellungen der Frequenzanteile über Zeit)
2. **Musterkennung**: Neuronale Netze erkennen Muster in diesen Merkmalen
3. **Kontext-Analyse**: Durch Aufmerksamkeitsmechanismen wird der Kontext berücksichtigt
4. **Ausgabe-Generierung**: Erzeugung der Transkription oder der synthetisierten Sprache

Diese technischen Grundlagen erklären, warum moderne Audio-KI-Modelle so leistungsfähig sind und warum sie in der Lage sind, auch komplexe Audioinhalte zu verarbeiten und zu generieren.

2 Herausforderungen und Grenzen

Obwohl moderne Audio-KI-Systeme beeindruckende Ergebnisse erzielen, stoßen sie in bestimmten Situationen an ihre Grenzen. Diese Herausforderungen zu verstehen ist wichtig, um realistische Erwartungen zu setzen und die Qualität der Ergebnisse zu verbessern.

2.1 Herausforderungen bei Speech-to-Text (STT)

2.2 Sprachvariationen

- **Akzente und Dialekte:** Regionale Sprachvarianten können die Erkennungsgenauigkeit erheblich beeinflussen
- **Sprechgeschwindigkeit:** Sehr schnelles oder langsames Sprechen erschwert die korrekte Erkennung
- **Umgangssprache und Slang:** Informelle Ausdrücke werden oft nicht korrekt erkannt

2.3 Umgebungs faktoren

- **Hintergrundgeräusche:** Lärm, Musik oder andere Gespräche können die Qualität der Transkription beeinträchtigen
- **Halleffekte:** In halligen Räumen aufgenommene Sprache ist schwieriger zu transkribieren
- **Mikrofonqualität:** Niedrige Aufnahmegerätequalität führt zu schlechteren Transkriptionsergebnissen

2.4 Inhaltliche Komplexität

- **Fachbegriffe:** Spezialisierte Terminologie wird oft falsch transkribiert
- **Eigennamen:** Ungewöhnliche Namen werden häufig falsch erkannt
- **Homophone:** Wörter, die gleich klingen aber unterschiedlich geschrieben werden, führen zu Fehlern

2.5 Grenzen bei Text-to-Speech (TTS)

- **Emotionale Nuancen:** Subtile emotionale Ausdrücke sind schwer zu reproduzieren
- **Sprechpausen und Rhythmus:** Natürliches Sprechtempo ist eine Herausforderung
- **Aussprache seltener Wörter:** Ungewöhnliche oder fremdsprachige Begriffe werden oft falsch ausgesprochen
- **Kontextuelle Anpassung:** Die Anpassung an den inhaltlichen Kontext (z.B. Frage vs. Aussage) ist begrenzt

2.6 Strategie zur Verbesserung der Ergebnisse

2.6.1 Transkriptionen:

1. **Qualität der Aufnahme optimieren:** Ruhige Umgebung, gutes Mikrofon, angemessener Abstand zum Mikrofon
2. **Deutlich sprechen:** Gleichmäßiges Tempo, klare Aussprache
3. **Fachbegriffe bereitstellen:** Bei Bedarf eine Liste spezieller Begriffe vorbereiten

2.6.2 Sprachsynthese:

1. **Textformatierung anpassen:** Interpunktions für natürliche Pausen nutzen
2. **Aussprache-Hinweise:** Für schwierige Wörter phonetische Schreibweisen verwenden
3. **Stimme passend wählen:** Verschiedene Stimmen für unterschiedliche Inhalte

2.7 Ethische und praktische Grenzen

- **Stimmimitation:** Die Fähigkeit, Stimmen zu imitieren, wirft Fragen bezüglich Identitätsdiebstahl auf
- **Mehrsprachigkeit:** Die Qualität variiert stark zwischen verschiedenen Sprachen
- **Ressourcenverbrauch:** Hochwertige Audio-KI-Modelle benötigen erhebliche Rechenressourcen
- **Datenschutz:** Die Verarbeitung von Audiodaten erfordert besondere Sorgfalt im Umgang mit persönlichen Informationen

Das Bewusstsein für diese Herausforderungen hilft, Audio-KI-Technologien realistisch einzuschätzen und in geeigneten Kontexten effektiv einzusetzen.

3 Grundbegriffe für Einsteiger

Bevor wir uns mit der KI-basierten Audioverarbeitung beschäftigen, ist es wichtig, einige grundlegende Konzepte der digitalen Audioverarbeitung zu verstehen.

3.1 Was ist digitales Audio?

Audio besteht physikalisch aus Schallwellen – Druckschwankungen in der Luft, die unser Ohr wahrnimmt. Computer können jedoch nur mit digitalen Daten arbeiten, daher muss Schall für die Verarbeitung in Zahlen umgewandelt werden.

3.2 Der Digitalisierungsprozess

1. **Aufnahme:** Ein Mikrofon wandelt Schallwellen in elektrische Signale um
2. **Analog-Digital-Wandlung:** Diese kontinuierlichen Signale werden in diskrete Zahlenwerte umgewandelt
3. **Speicherung:** Die Zahlenwerte werden als Datei gespeichert
4. **Verarbeitung:** Die gespeicherten Werte können nun durch Programme verarbeitet werden

3.3 Wichtige Audio-Parameter

3.3.1 Abtastrate (Sampling Rate)

- **Definition:** Anzahl der Messungen pro Sekunde, gemessen in Hertz (Hz)

- **Typische Werte:**
 - 44.100 Hz (CD-Qualität)
 - 48.000 Hz (Professionelles Audio)
 - 8.000 Hz (Telefonie)
- **Auswirkung:** Höhere Abtastraten können höhere Frequenzen erfassen (gemäß dem Nyquist-Theorem)

3.3.2 Bittiefe (Bit Depth)

- **Definition:** Anzahl der Bits pro Abtastwert, bestimmt die Anzahl möglicher Lautstärkestufen
- **Typische Werte:**
 - 16 Bit (CD-Qualität, 65.536 Stufen)
 - 24 Bit (Professionelles Audio, über 16 Millionen Stufen)
- **Auswirkung:** Höhere Bittiefe verbessert den Dynamikumfang und reduziert Quantisierungsrauschen

3.3.3 Kanäle

- **Definition:** Anzahl der gleichzeitig aufgezeichneten Audiospuren
- **Typische Werte:**
 - Mono (1 Kanal)
 - Stereo (2 Kanäle)
 - Surround (5.1, 7.1, etc.)
- **Auswirkung:** Mehr Kanäle ermöglichen räumliches Audio

3.3.4 Audioformate

- **Unkomprimiert:** WAV, AIFF (verlustfreie Speicherung, große Dateien)
- **Komprimiert verlustbehaftet:** MP3, AAC, OGG (kleinere Dateien, etwas reduzierte Qualität)
- **Komprimiert verlustfrei:** FLAC, ALAC (reduzierte Dateigröße bei voller Qualität)

3.4 Audio-Eigenschaften

3.4.1 Amplitude

- **Definition:** Die Stärke des Audiosignals, entspricht der wahrgenommenen Lautstärke
- **Messung:** Dezibel (dB)

3.4.2 Frequenz

- **Definition:** Die Anzahl der Schwingungen pro Sekunde, bestimmt die Tonhöhe
- **Messung:** Hertz (Hz)
- **Menschliches Hören:** Etwa 20 Hz bis 20.000 Hz

3.4.3 Spektrum

- **Definition:** Die Verteilung der Energie über verschiedene Frequenzen
- **Darstellung:** Spektrogramm (Zeit-Frequenz-Darstellung)

3.5 Audioqualität und KI-Verarbeitung

Die Qualität der Audioeingabe beeinflusst direkt die Ergebnisse der KI-Verarbeitung:

- **Hochwertige Aufnahmen:**
 - Klare Sprache ohne Hintergrundgeräusche
 - Angemessene Lautstärke (weder zu leise noch übersteuert)
 - Geeignete Abtastrate und Bitrate
- **Faktoren, die die Qualität beeinflussen:**
 - Mikrofonqualität und -platzierung
 - Akustik des Aufnahmeraums
 - Vermeidung von Übersteuerung und Verzerrung

4 Probleme & Hacks Audio-API

4.1 API-Fehler bei OpenAI

Symptome:

- Fehlermeldung: "Rate limit exceeded"
- Timeout-Fehler

Lösungsansätze:

```
import openai
import time
import backoff

# Exponential Backoff-Funktion für Wiederholungsversuche
@backoff.on_exception(backoff.expo,
                      (openai.RateLimitError, openai.APITimeoutError),
                      max_tries=5)

def transcribe_with_retry(file_path):
```

```
with open(file_path, "rb") as audio_file:
    try:
        response = openai.audio.transcriptions.create(
            model="whisper-1",
            file=audio_file
        )
        return response.text
    except Exception as e:
        print(f" Fehler bei der Transkription: {e}")
        # Warten vor dem nächsten Versuch
        time.sleep(2)
        raise e
```

4.2 Unnatürliche Aussprache

Symptome:

- Falsche Betonung von Wörtern
- Abgehackte Sätze
- Falsche Aussprache von Fachbegriffen

Lösungen:

1. Interpunktionsanpassungen

- Kommas für kurze Pausen einfügen
- Punkte für längere Pausen verwenden

2. Aussprache-Hinweise verwenden

```
# Beispiel für Aussprache-Hinweise
text = "Der Patient leidet an Pneumonie (ausgesprochen: noi-mo-nie)."

# Alternative: Phonetische Schreibweise verwenden
text = "Python kann für verschiedene Aufgaben verwendet werden."
```

3. Satzstruktur vereinfachen

- Komplexe, verschachtelte Sätze in kürzere Sätze aufteilen

4.3 Fehlende Emotionalität

Lösungen:

1. Passende Stimme wählen

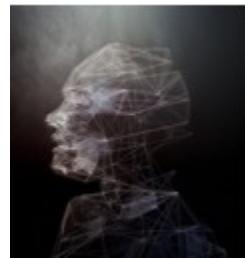
- Verschiedene Stimmen für unterschiedliche Stimmungen testen
- "Nova" für freundliche Inhalte, "Onyx" für ernstere Themen

2. Text mit Emotionshinweisen anreichern

```
# Emotionale Hinweise im Text
text = "Wow! Das ist eine fantastische Nachricht!" # Begeisterung

# Oder durch Beschreibungen
text = "[begeistert] Das ist eine fantastische Nachricht!
[/begeistert]"
```

M17 - Bibliothek - mcp_modul



Anwendung Generativer KI

Stand: 10.2025

Allgemeines:

Im Rahmen dieses Einführungskurses werden wir uns nicht durch hunderte von Zeilen Code durcharbeiten können, das wäre viel zu aufwendig und würde den Kern des Themas verfehlten. Stattdessen werden wir mit Bibliotheken arbeiten, die uns einen Teil der komplexen, technischen Details abnehmen. So können wir uns darauf konzentrieren, was wirklich wichtig ist: nämlich das Verständnis der generativen KI-Konzepte und ihre Anwendung. Kurz gesagt: Ihr müsst nicht alles von Grund auf selbst programmieren.

1 Überblick

Das Modul `mcp_modul.py` bietet eine vollständige, funktionale Implementierung der drei Hauptkomponenten einer MCP-Architektur: **Server**, **Client** und **AI-Assistent**. Es dient primär zu Demonstrationszwecken, um den Kommunikationsfluss und die Logik hinter dem Protokoll zu verdeutlichen. Die Wahl eines funktionalen Ansatzes, anstelle eines klassischen objektorientierten Designs, fördert dabei die **Klarheit**, **Testbarkeit** und **Wartbarkeit** der einzelnen Komponenten. Dies macht das Modul ideal für das Prototyping und das Verständnis der MCP-Spezifikation.

2 Konfiguration

Die Basiskonfigurationen definieren die Identitäten und Fähigkeiten der Komponenten innerhalb der simulierten MCP-Umgebung.

Komponente	Name	Version	Wichtigste Tools (Server)	Detail und Zweck
Server	file-server	1.0.0	read_file , write_file , list_files , get_system_info	Stellt grundlegende System- und Dateizugriffe bereit. Er fungiert als Datendienst-Connector .
Client	demo-client	1.0.0	-	Verwaltet die Netzwerkverbindungen und die Tool-Liste des Servers. Er kapselt die MCP-Kommunikationslogik.
Assistant	Functional-MCP-Assistant	1.0.0	Nutzt das konfigurierte OpenAI-Modell (gpt-4o-mini)	Die zentrale Intelligenz, die menschliche Sprache in strukturierte MCP-Aufrufe übersetzt und die Ergebnisse für den Nutzer aufbereitet.

3 Server-Funktionen (Connector)

Der Server implementiert die Logik zur Verarbeitung von **JSON-RPC**-basierten MCP-Anfragen. Er ist die einzige Komponente, die direkt mit den externen Ressourcen (hier: das Dateisystem und das Betriebssystem) interagiert.

Funktion	Beschreibung	Kernaufgabe
handle_mcp_request(request)	Haupt-Handler: Verarbeitet alle eingehenden MCP-Anfragen (initialize , tools/list , tools/call). Er validiert das JSON-RPC-Format und sorgt für eine konsistente Fehlerbehandlung.	Leitet Requests an die entsprechenden Tool-Funktionen weiter.

<code>get_server_info()</code>	Gibt die Metadaten des Servers (Name, Version) und die Liste der verfügbaren Tool-Namen zurück.	Statusabfrage und Discovery-Unterstützung für den Client.
<code>register_new_tool(name, description, parameters, function)</code>	Erweitert den Server zur Laufzeit um neue Tools und deren Implementierung. Dies ermöglicht eine dynamische Server-Anpassung an neue Datenquellen oder APIs.	Erweiterbarkeit des Tool-Sets ohne Neustart des Servers.

Implementierte Tools (`tool_functions`):

- `read_file_tool(filepath)` : Liest den Inhalt einer Datei. Gibt neben dem Inhalt auch Metadaten wie Dateipfad, Größe und den Zeitstempel des Zugriffs zurück.
- `list_files_tool(directory)` : Listet die Dateien im angegebenen Verzeichnis auf. Das Tool liefert detaillierte Informationen wie Dateiname, Pfad, Größe und ob es sich um eine Datei oder ein Verzeichnis handelt.
- `write_file_tool(filepath, content)` : Schreibt Inhalt in eine Datei. Dies demonstriert die **bidirektionale Natur** von MCP (Lesen und Schreiben von Daten), im Gegensatz zu reinen Retrieval-Ansätzen.
- `get_system_info_tool()` : Gibt grundlegende Informationen über das laufende System zurück (z. B. Betriebssystem, Python-Version). Dies dient als einfacher Test für die Tool-Ausführungsumgebung.

4 Client-Funktionen (API-Wrapper)

Der Client fungiert als zuverlässige Vermittlungsschicht. Seine Hauptaufgabe ist die **Schnittstellen-Normalisierung** und das Management des Server-Zustands.

Funktion	Beschreibung	Workflow-Schritt

<code>setup_full_connection(server_name, server_handler)</code>	Führt die vollständige Verbindung in einem Schritt aus: <code>connect_to_server</code> (Verbindung herstellen), <code>initialize_server_connection</code> (Protokoll-Handshake durchführen) und <code>discover_server_tools</code> (Tool-Liste abrufen).	1. Connect → 2. Initialize → 3. Discover Tools
<code>call_server_tool(server_name, tool_name, arguments)</code>	Erstellt eine <code>tools/call</code> -Anfrage im standardisierten MCP-Format und sendet sie asynchron an den Server-Handler. Die Funktion kapselt die Fehlerbehandlung des RPC-Aufrufs.	Tool-Ausführung.
<code>list_connected_servers()</code>	Listet alle aktuell verbundenen Server auf, die für den Client verfügbar sind.	Status.
<code>get_available_tools()</code>	Gibt eine aggregierte Übersicht aller vom Client entdeckten Tools zurück. Diese Liste wird verwendet, um den AI-Assistenten über seine externen Fähigkeiten zu informieren.	Status.

5 AI-Assistant-Funktionen (LLM-Logik)

Der Assistent ist das Herzstück der Interaktion. Er nutzt das LLM in einem **Multi-Step-Reasoning-Prozess**, um auf MCP-Tools gestützte Antworten zu liefern.

Funktion	Beschreibung	Prozess-Schritt
<code>process_user_query(user_query, use_mcp=True)</code>	Haupt-Loop: Verarbeitet die Anfrage in zwei Schritten:	Die Kernlogik des Assistenten.
	1. Initial Call (Reasoning): Das LLM erhält den <code>system_prompt</code> mit der Anweisung, MCP-Aufrufe zu generieren. Die rohe LLM-Antwort (mit potenziellen Tool-Calls) wird abgerufen.	Reasoning.

	2. Execute Calls (Tool-Nutzung): Die Funktion <code>extract_mcp_calls_from_text</code> parst die rohe Antwort. Die extrahierten Aufrufe werden synchron/asynchron über den Client ausgeführt, um reale Daten zu erhalten.	Tool-Nutzung.
	3. Final Call (Ergebnisgenerierung): Die ursprüngliche Benutzeranfrage und die Ergebnisse der Tool-Aufrufe (<code>mcp_results</code>) werden dem LLM in einem neuen Prompt als Kontext präsentiert. Die resultierende Antwort ist die endgültige, natürliche und fundierte Ausgabe an den Benutzer.	Ergebnisgenerierung.
<code>setup_assistant_mcp_connection(server_name)</code>	Konfiguriert den Assistenten, um einen bestimmten, bereits verbundenen Server für Tool-Aufrufe zu nutzen, und aktiviert den MCP-Modus.	Einbindung.
<code>get_assistant_status()</code>	Gibt den aktuellen Zustand des Assistenten zurück, einschließlich des verbundenen Servers, der Aktivierung des MCP-Modus und einer Liste aller dem Assistenten bekannten Tools.	Status.
<code>toggle_mcp_mode(enabled)</code>	Schaltet die MCP-Nutzung für den Assistenten ein oder aus. Im deaktivierten Zustand antwortet das LLM nur mit seinem internen Wissen.	Steuerung.

6 Kommunikations-Konventionen

Das Modul verwendet die folgende Konvention für den LLM-Output, um MCP-Aufrufe zu identifizieren und zu parsen. Diese Formatierung ist kritisch, da sie es dem Code ermöglicht, die **Aktionsabsicht** des Modells zu erkennen.

```
[MCP_CALL: <tool_name>({<arguments_json>})] [/MCP_CALL]
```

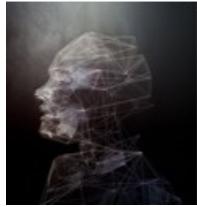
Erklärung der Syntax:

- [MCP_CALL: ...] und [/MCP_CALL] : Beginnt und beendet den Aufruf-Block.
- <tool_name> : Entspricht einem der im Server registrierten Tool-Namen (z. B. write_file).
- {<arguments_json>} : Ein gültiges JSON-Objekt, das die notwendigen Parameter für das aufgerufene Tool enthält.

Beispiel:

```
[MCP_CALL: write_file({"filepath": "todos.txt", "content": "1. Meeting vorbereiten"})] [/MCP_CALL]
```

M18 - Fine-Tuning



Anwendung Generativer KI

Stand: 05.2025

1 Intro

Fine-Tuning ist eine bewährte Technik, um ein vortrainiertes KI-Modell gezielt auf spezifische Aufgaben oder Datensätze anzupassen. Dabei werden die bereits gelernten Strukturen und Muster eines bestehenden Modells genutzt und weiterverfeinert. Dieser Prozess spart nicht nur Rechenressourcen und Zeit, sondern führt auch bei kleineren Datenmengen zu beeindruckenden Ergebnissen.

In der Praxis bedeutet Fine-Tuning, dass Entwickler:innen lediglich einen Bruchteil der Rechenleistung und Datenmenge benötigen, die für das Training eines Modells von Grund auf erforderlich wären. Gleichzeitig wird die Möglichkeit geschaffen, eigene oder sensible Daten gezielt einzusetzen. Fine-Tuning ist somit nicht nur ein technisches Werkzeug, sondern auch ein strategischer Ansatz zur Effizienzsteigerung, Individualisierung und Qualitätssicherung von KI-Anwendungen.

Zudem ist Fine-Tuning ein zentraler Bestandteil des sogenannten Model Optimization Workflows. Dabei wird die Modellleistung durch eine Kombination aus **Evals**, **Prompt Engineering** und **Fine-Tuning** in einem iterativen Zyklus optimiert. Ziel ist es, die Qualität der Modellantworten durch Feedback und gezielte Anpassung kontinuierlich zu verbessern. Dieser sogenannte Optimierungs-Flywheel ermöglicht es, systematisch bessere Prompts, Trainingsdaten und daraus resultierende Modelle zu entwickeln.

2 Fine-Tuning-Ansätze

2.1 Transfer Learning

- **Grundprinzip:** Ein vortrainiertes Modell wird als Ausgangspunkt verwendet. Die allgemeinen Merkmale der frühen Schichten bleiben erhalten.

- **Vorgehen:** Die letzten Schichten werden ersetzt oder angepasst, um spezifische Aufgaben zu lösen.
- **Einsatzgebiete:** Bildklassifikation, Verarbeitung natürlicher Sprache (NLP), Computer Vision.
- **Vorteile:** Schneller Einstieg, geringer Datenbedarf, bewährte Basismodelle.
- **Vergleich zum Pre-Training:** Während das Pre-Training ein Modell mit großen Datenmengen (oft Billionen von Tokens) trainiert, um allgemeine Sprachmuster zu lernen, benötigt Fine-Tuning nur einen Bruchteil der Daten und Rechenkapazität für eine spezifische Aufgabe.

2.2 Parameter-effizientes Fine-Tuning (PEFT)

- **Prinzip:** Anpassung nur weniger Parameter, während das Basismodell unverändert bleibt.
- **Methoden:**
 - **LoRA (Low-Rank Adaptation):** Kompakte Matrizen für effiziente Gewichtsänderung. Reduziert den Rechenaufwand erheblich durch Low-Rank-Approximation der Gewichtsänderungen.
 - **QLoRA:** Eine Erweiterung von LoRA, die Gewichtsparameter auf 4-Bit-Präzision quantisiert, wodurch der Speicherbedarf weiter reduziert wird.
 - **DoRA (Weight-Decomposed Low-Rank Adaptation):** Zerlegt Gewichte in Größen- und Richtungskomponenten für präzisere Updates bei gleichbleibender Effizienz.
 - **Adapter:** Zusätzliche Module zwischen bestehenden Schichten.
 - **Prompt Tuning:** Anpassung durch trainierbare Prompts.
- **Vorteile:** Spart Ressourcen, wiederverwendbar, ideal für verschiedene Aufgaben.
- **Anwendungsbereich:** Besonders geeignet für ressourcenbeschränkte Umgebungen und für mehrere Spezialisierungsaufgaben mit demselben Basismodell.

2.3 Instruction Fine-Tuning

- **Ziel:** Das Modell lernt, auf klare, natürlichsprachliche Anweisungen zu reagieren.
- **Daten:** Input-Output-Paare mit expliziten Instruktionen.
- **Anwendungen:** Sprachassistenten, automatisierte Kommunikation, LLM-basierte Tools.
- **Formatbeispiel:** Oft in diesem Format: "##Human: < *InputQuery* > ##Assistant: < *GeneratedOutput* >"

2.4 Supervised Fine-Tuning (SFT)

- **Ansatz:** Feinabstimmung mit handverlesenen, hochwertigen Beispielen.

- **Zweck:** Optimierung für bestimmte Anforderungen oder Zielgruppen.
- **Typisch kombiniert mit:** Reinforcement Learning from Human Feedback (RLHF).
- **Datenerfordernisse:** Benötigt mindestens 10 Beispiele, empfohlen sind 50-100 qualitativ hochwertige Demonstrationen. Die Qualität der Daten ist entscheidender als die Menge.
- **Prozess:** Besteht aus Datenvorbereitung, Upload der Trainingsdaten, Erstellung eines Fine-Tuning-Jobs und anschließender Evaluierung des Modells.

3 Weitere Ansätze OpenAI

3.1 Direct Preference Optimization (DPO)

- **Vorgehen:** Training mit präferierten und abgelehnten Antwortpaaren.
- **Nutzen:** Verfeinert Nuancen wie Stil, Tonalität, Ausdruck.
- **Funktionsweise:** Ein effizienterer Weg als RLHF, um Modelle an menschliche Präferenzen anzupassen. Jedes Beispiel im Datensatz enthält einen Prompt, eine bevorzugte Ausgabe und eine nicht-bevorzugte Ausgabe.
- **Beta-Parameter:** Kann zwischen 0 und 2 konfiguriert werden, um zu steuern, wie streng das neue Modell am vorherigen Verhalten festhält versus sich an den neuen Präferenzen orientiert.

3.2 Reinforcement Fine-Tuning (RFT)

- **Prinzip:** Modell wird nicht mit festen Zielantworten, sondern anhand von Bewertungssignalen (Grader) trainiert.
- **Vorteile:** Besonders geeignet für komplexe, mehrdeutige Aufgaben.
- **Grader-Konzept:** RFT verwendet Grader, die die Modellantworten bewerten und ein numerisches Signal (zwischen 0 und 1) zurückgeben. Diese können als String-Check, Text-Similarity oder Model-Grader konfiguriert werden.
- **Anwendungsbereich:** Besonders effektiv bei Aufgaben, bei denen Experten in der Domäne sich über die richtigen Antworten einig sind und die Aufgabe eindeutig bewertbar ist.
- **Unterstützung:** Aktuell nur für reasoning-Modelle wie o4-mini verfügbar.

3.3 Vision Fine-Tuning

- **Zweck:** Anpassung von Modellen mit visuellen Eingaben (z. B. Bilder).
- **Anwendungen:** Bildklassifikation, visuelle Beschreibungen, Objektlokalisierung.
- **Technische Hinweise:** Unterstützt Base64-Bilder oder URLs, max. 10 Bilder pro Beispiel.
- **Besonderheiten:**

- Bilder müssen im JPEG-, PNG- oder WEBP-Format vorliegen
- Maximalgröße pro Bild: 10 MB
- Bilder mit Menschen, Gesichtern, Kindern oder CAPTCHAs werden aus Datenschutzgründen ausgeschlossen
- Der Detail-Parameter kann auf "low" gesetzt werden, um Trainingskosten zu reduzieren

3.4 Modell-Distillation

- **Konzept:** Nutzung der Ausgaben eines großen Modells, um ein kleineres Modell zu trainieren, das ähnliche Leistung für eine spezifische Aufgabe erzielt.
- **Vorteile:** Reduziert Kosten und Latenz erheblich, da kleinere Modelle effizienter sind.
- **Prozess:**
 1. Speichern qualitativ hochwertiger Ausgaben eines großen Modells mit dem Parameter `store: true`
 2. Evaluierung der gespeicherten Antworten mit dem großen und kleinen Modell
 3. Auswahl relevanter Antworten als Trainingsdaten für das kleine Modell
 4. Fine-Tuning des kleinen Modells mit diesen Beispielen
 5. Evaluierung des fine-tuned kleineren Modells
- **Anwendungsbereich:** Besonders nützlich, wenn ein spezifischer, begrenzter Aufgabenbereich abgedeckt werden soll.

4 Fine-Tuning-Pipeline für LLMs

4.1 Datenvorbereitung

- Datensammlung aus verschiedenen Quellen
- Vorverarbeitung und Formatierung (z.B. JSONL-Format)
- Umgang mit unausgeglichenen Daten (Oversampling, Undersampling)
- Datensatzaufteilung (Training/Validierung/Test)

4.2 Modellinitialisierung

- Auswahl eines geeigneten vortrainierten Modells
- Einrichtung der Umgebung und Installation der Abhängigkeiten
- Laden des Modells in den Speicher

4.3 Trainingsumgebung

- Konfiguration von Hardwareressourcen (GPU/TPU)
- Definition von Hyperparametern (Lernrate, Batch-Größe, Epochen)

- Initialisierung von Optimierern und Verlustfunktionen

4.4 Fine-Tuning-Prozess

- Auswahl der Fine-Tuning-Technik (Voll, PEFT, etc.)
- Durchführung des Trainings mit regelmäßigen Validierungen
- Überwachung von Metriken und Verlustfunktionen

4.5 Evaluierung und Validierung

- Aufsetzen von Evaluierungsmetriken
- Analyse der Trainingsverlaufskurve
- Überwachung und Interpretation der Ergebnisse

4.6 Deployment

- Export des fine-tuned Modells
- Einrichtung der Infrastruktur
- API-Entwicklung für die Modellinteraktion

4.7 Monitoring und Wartung

- Kontinuierliche Überwachung der Modellleistung
- Aktualisierung des LLM-Wissens bei Bedarf
- Wiederholte Feinabstimmung bei veränderter Datenlage

5 Schlüsselkomponenten der Modelloptimierung

5.1 Evaluierungen (Evals)

- **Nutzen:** Systematische Tests zur Bewertung von Modellantworten.
- **Formate:** Multiple Choice, Klassifikation, Stringvergleich etc.
- **Grader-Typen:**
 - **String-Check-Grader:** Einfache String-Operationen (gleich, ungleich, enthält)
 - **Text-Similarity-Grader:** Bewertung der Ähnlichkeit zwischen Modellantwort und Referenz
 - **Model-Grader:** Nutzung eines separaten Modells zur Bewertung der Ausgaben
 - **Python-Grader:** Ausführung von Python-Code zur Bewertung
 - **Multi-Grader:** Kombination mehrerer Grader für komplexe Bewertungskriterien

- **Integrierter Prozess:** Evals sollten vor dem Fine-Tuning erstellt werden, um eine Baseline zu etablieren und den Fortschritt zu messen.

5.2 Prompt Engineering

- **Ziele:** Maximale Modelleistung ohne Training.
- **Methoden:** Klare Instruktionen, Kontextbereitstellung, Few-Shot-Beispiele.
- **Zusammenspiel mit Fine-Tuning:** Prompt Engineering kann Fine-Tuning ergänzen oder in manchen Fällen sogar ersetzen.
- **Beispiel:** Die Prompt-Konstruktion mit relevanten Beispielen (Few-Shot-Learning) kann die Leistung signifikant verbessern, ohne das Modell neu zu trainieren.

Embeddings spielen beim **Fine-Tuning eines Large Language Models (LLMs)** eine zentrale Rolle, da sie den **Ausgangspunkt der Verarbeitung von Eingabedaten** im Modell darstellen. Hier ist eine strukturierte Erklärung ihrer Rolle:

6 Embeddings und Fine-Tuning

6.1 Recap: Was sind Embeddings?

Embeddings sind **dichte, numerische Vektoren**, die Wörter, Tokens oder ganze Sätze in einem kontinuierlichen Vektorraum repräsentieren. Sie sind so trainiert, dass **semantisch ähnliche Begriffe nahe beieinander** im Vektorraum liegen.

6.2 Rolle beim Fine-Tuning eines LLMs

1. Initiale Repräsentation der Eingabedaten:

- Bevor Text durch die Transformer-Schichten geht, wird er in Embeddings umgewandelt.
- Diese Embeddings enthalten bereits **viele Informationen über die Bedeutung** der Tokens.

2. Anpassung an die Zielaufgabe:

- Beim Fine-Tuning werden **nicht nur die oberen Schichten** (z. B. der Decoder oder der Klassifikator), sondern häufig auch die **Embedding-Schicht selbst angepasst**.
- So kann sich das Modell an spezielle Fachterminologie oder Ausdrucksweisen der Zielanwendung gewöhnen.

3. Transferlernen durch vortrainierte Embeddings:

- Das Modell startet mit **generischen Embeddings** aus dem Pretraining.
- Beim Fine-Tuning lernen die Embeddings, sich besser an die neue Domäne anzupassen (z. B. Jura, Medizin, Technik).

4. Spezialfall: Adapter-Fine-Tuning oder LoRA:

- In Methoden wie **LoRA** oder **Adapter Layers** werden die Embeddings oft **nicht direkt verändert**, sondern nur zusätzliche Parameter eingeführt.
- Vorteil: Die ursprünglichen Embeddings bleiben erhalten → weniger Overfitting, kleinere Modelle.

6.3 Warum sind sie so wichtig?

- Sie beeinflussen maßgeblich, **wie der Text semantisch verstanden wird**.
- Eine gute Embedding-Anpassung beim Fine-Tuning verbessert die Fähigkeit des Modells, **Aufgabenkontext korrekt zu erfassen** (z. B. bei Named Entity Recognition, Sentiment Analysis, RAG-Systemen usw.).

6.4 Fazit

Die Embeddings sind die **Brücke zwischen rohem Text und neuronaler Verarbeitung**. Beim Fine-Tuning werden sie oft (aber nicht immer) mitangepasst, um eine **bessere Domänenanpassung und höhere Genauigkeit** zu erzielen.

7 Best Practices

7.1 Datenstrategie

1. Datenqualität schlägt Datenmenge

- Beginnen Sie mit 50-100 hochwertigen Beispielen
- Verwenden Sie realistische Daten aus der Zielanwendung
- Stellen Sie sicher, dass die Daten repräsentativ für die Aufgabe sind

2. Vielfältige und realistische Beispiele wählen

- Decken Sie verschiedene Szenarien, Formulierungen und Nuancen ab
- Vermeiden Sie starke Verzerrungen in den Trainingsdaten
- Berücksichtigen Sie auch Randfall-Szenarien

3. Konsistente Formatierung (z. B. JSONL)

- Verwenden Sie das korrekte Format für Ihre Fine-Tuning-Methode
- Stellen Sie sicher, dass jede Zeile ein vollständiges JSON-Objekt enthält
- Validieren Sie Ihr Datenformat vor dem Training

7.2 Trainingsstrategie

4. Schrittweises Auftauen von Schichten

- Beginnen Sie mit dem Training der obersten Schichten
- Tauen Sie schrittweise tiefere Schichten auf
- Dies führt zu stabilerem Training und verhindert Overfitting

5. Kleine Lernraten zur Stabilisierung

- Verwenden Sie Lernraten zwischen 1e-4 und 2e-4 für stabile Konvergenz
- Ein Lernraten-Schedule mit Warmup und linearem Abfall kann hilfreich sein
- Experimentieren Sie mit verschiedenen Batch-Größen

6. Regelmäßige Evaluierung und Monitoring

- Setzen Sie vor dem Fine-Tuning Evaluierungen (Evals) auf
- Überwachen Sie Trainings- und Validierungsmetriken
- Implementieren Sie Early Stopping, um Overfitting zu vermeiden

7.3 Technische Exzellenz

7. Verwendung von Checkpoints und Modellversionierung

- Speichern Sie regelmäßig Zwischenstände (alle 5-8 Epochen)
- Vergleichen Sie die Leistung verschiedener Checkpoint-Modelle
- Behalten Sie die Versionshistorie bei, um Regressionen zu erkennen

8. Hyperparameter systematisch optimieren

- Nutzen Sie automatisierte Hyperparameter-Optimierung (Random Search, Grid Search, Bayesian)
- Fokussieren Sie auf Lernrate, Batch-Größe und Epochenzahl
- Dokumentieren Sie die Ergebnisse verschiedener Konfigurationen

9. Tools wie TensorBoard, W&B, MLflow einsetzen

- Visualisieren Sie Trainingsmetriken in Echtzeit
- Verfolgen Sie Experimente und deren Ergebnisse
- Vergleichen Sie verschiedene Trainingsläufe untereinander

7.4 Sicherheit und Effizienz

10. Datenschutzkonformität beachten

- Anonymisieren Sie sensible Daten vor dem Training
- Beachten Sie rechtliche Anforderungen beim Training mit personenbezogenen Daten
- Implementieren Sie Sicherheitsprüfungen für Modelleingaben und -ausgaben

11. Kosten im Blick behalten (Token-Effizienz)

- Überwachen Sie den Token-Verbrauch während des Trainings
- Optimieren Sie Prompts für kürzere Ausgaben, wo möglich
- Verwenden Sie Modell-Distillation für häufig genutzte Aufgaben

7.5 Spezifische Techniken für erweiterte Anwendungen

12. Multi-Task Learning

- Trainieren Sie das Modell für mehrere verwandte Aufgaben gleichzeitig
- Verwenden Sie spezifische Adapter für verschiedene Aufgaben
- Kombinieren Sie bei Bedarf mehrere Adapter für komplexe Anwendungen

13. Modell-Quantisierung

- Reduzieren Sie die Präzision der Modellparameter (z.B. von 32-bit auf 8-bit)
- Verwenden Sie QLoRA für effizientes Training mit quantisierten Modellen
- Testen Sie die Leistung quantisierter Modelle gründlich

14. Multimodale Integration

- Bei Vision Fine-Tuning: Achten Sie auf Bildqualität und -größe
- Verwenden Sie den "detail"-Parameter, um Trainingskosten zu optimieren
- Testen Sie verschiedene Kombinationen von Text- und Bildeingaben

8 Herausforderungen & Perspektiven

8.1 Skalierbarkeit

- **Rechnerische Ressourcen:** Fine-Tuning großer Modelle erfordert erhebliche Rechen- und Speicherkapazitäten
- **Memory-Effizienz:** Techniken wie LoRA, QLoRA und Half Fine-Tuning adressieren diese Herausforderungen
- **Zukünftige Entwicklungen:** Co-Design von Hardware und Algorithmen speziell für LLM-Training

8.2 Ethische Überlegungen

- **Bias und Fairness:** Trainingsdaten können Verzerrungen enthalten, die sich auf das Modell übertragen
- **Datenschutz:** Umgang mit sensiblen oder proprietären Daten während des Fine-Tunings
- **Transparenz und Nachvollziehbarkeit:** Dokumentation des Fine-Tuning-Prozesses und seiner Auswirkungen

8.3 Integration mit neuen Technologien

- **Internet of Things (IoT):** LLMs können IoT-Daten in Echtzeit analysieren und Entscheidungen optimieren
- **Edge Computing:** Fine-Tuned Modelle können direkt auf Edge-Geräten eingesetzt werden
- **Federated Learning:** Training über verteilte Datenquellen ohne zentrale Datenspeicherung

9 Fazit

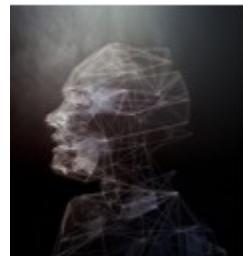
Fazit

Fine-Tuning ist mehr als nur eine technische Maßnahme – es ist ein strategischer Hebel zur Anpassung von KI-Systemen an konkrete Anforderungen, Zielgruppen und Kontexte. Die Kombination aus fundierter Datenbasis, passenden Methoden und iterativer Evaluation bildet das Fundament für erfolgreiche KI-Projekte.

Mit Plattformen wie OpenAI lassen sich diese Prozesse effizient gestalten – von der Vorbereitung über das Training bis zur Bewertung. Erweiterte Verfahren wie DPO, RFT und Vision Fine-Tuning ermöglichen zusätzlich eine feinkörnige Kontrolle und Weiterentwicklung leistungsfähiger Modelle.

Die siebenstufige Fine-Tuning-Pipeline bietet einen strukturierten Ansatz vom Datenmanagement bis zur kontinuierlichen Überwachung, während parameter-effiziente Methoden die Ressourcennutzung optimieren. Durch den gezielten Einsatz dieser Techniken können Entwickler leistungsfähige KI-Lösungen erstellen, die genau auf ihre spezifischen Anforderungen zugeschnitten sind.

M19a - Modellauswahl



Anwendung Generativer KI

Stand: 05.2025

1 KI-Modelllandschaft: Ein Überblick

Die moderne KI-Landschaft bietet verschiedene spezialisierte Modelltypen für unterschiedliche Anwendungsfälle:

- **Reasoning-Modelle:** Spezialisiert auf logisches Denken und systematische Problemlösung (z.B. o3-mini) - diese Modelle lösen komplexe Aufgaben durch schrittweises, strukturiertes Denken.
- **Sprachmodelle:** Konzipiert für natürlichsprachliche Aufgaben wie Textgenerierung, Zusammenfassungen und Konversationen (z.B. GPT-4) - sie verstehen und erzeugen menschenähnliche Texte.
- **Codex-Modelle:** Optimiert für Codegenerierung und Programmieraufgaben - diese Modelle können Code schreiben, analysieren und debuggen.
- **Bildgenerierungsmodelle:** Erzeugen Bilder aus textlichen Beschreibungen (z.B. DALL-E) - sie wandeln Textanweisungen in visuelle Ergebnisse um.
- **Sprachverarbeitungsmodelle:** Spezialisiert auf Spracherkennung und -transkription (z.B. Whisper) - sie wandeln gesprochene Sprache in Text um.

2 Vergleich wichtiger GPT-Modelle

Die Wahl des richtigen Modells ist entscheidend für optimale Ergebnisse, Ressourcenschonung und maximale Effizienz. Hier ein Überblick der wichtigsten Modelle:

Modell	Hauptmerkmale	Empfohlene Anwendungsfälle
GPT-5	Neuestes Spaltenmodell mit integriertem Denken: Automatische Reasoning-Modi, überlegene Coding-Fähigkeiten, beste Instruktionsbefolgung. 400K Context Window.	Komplexe Coding-Projekte, Agentic Tasks, Automatisierung, Frontend-Entwicklung, anspruchsvolle Schreibaufgaben. Beste Wahl für professionelle Anwendungen.
GPT-5 Mini	Kleinere GPT-5 Version: Schneller und günstiger, behält aber die meisten GPT-5 Fähigkeiten bei. Optimiert für Geschwindigkeit und Kosteneffizienz.	Hochvolumen-Anwendungen, Chatbots, Content-Generierung im großen Stil, kostenbewusste Projekte mit hohen Qualitätsansprüchen.
GPT-5 Nano	Ultraschnelle GPT-5 Variante: Niedrigste Latenz und Kosten der GPT-5 Familie. Für Anwendungen die sofortige Antworten benötigen.	Real-time Anwendungen, Live-Chat, schnelle API-Calls, mobile Apps, IoT-Geräte. Wo Geschwindigkeit wichtiger als maximale Intelligenz ist.
GPT-4o	Multimodales Allround-Modell: Versteht Text, Bilder und Audio, kann Bilder generieren. Sehr schnell und vielseitig.	Alltägliche Aufgaben, Brainstorming, Texterstellung, Content-Ideen, Bildanalysen, E-Mails, Konzepte. Gut für schnelle Dialoge und allgemeine Fragen.
GPT-4o Mini	Leichtere Version von GPT-4o: Verarbeitet Text und Bilder, ressourcenschonend und günstiger. Deutlich intelligenter als GPT-3.5-turbo.	Einfachere Aufgaben, Bildverarbeitung, schnelle und unkomplizierte Anwendungen, kostengünstige Chatbots.
o3-mini	Reasoning-Modell: Hohe Intelligenz bei niedrigen Kosten und geringer Latenz. Konzipiert für strukturiertes Denken.	Wissenschaftliche, mathematische und Programmieraufgaben, technische und logische Probleme, faktenbasierte Recherchen.
o4-mini	Kompaktes Reasoning-Modell: Optimiert für Geschwindigkeit und Kosteneffizienz. Stark in mathematischen, Programmier- und visuellen Aufgaben.	Komplexe Argumentationsstrukturen, technische Aufgaben, Programmierprojekte, visuelles Denken, wissenschaftliche Fragestellungen.

Modell	Hauptmerkmale	Empfohlene Anwendungsfälle
o3	Leistungsstärkster "Denker": Herausragend in Programmierung, Mathematik, Wissenschaft und visueller Analyse. Arbeitet mit verknüpften Einzelschritten ("Chain-of-Thought").	Komplexe Recherchen, anspruchsvolle Programmieraufgaben, Datenanalyse, strategische Planung, Code-Review und Debugging. Beste Wahl für höchste Präzision.

Schnelle Modellwahl-Hilfe

Anwendungsfall	Empfohlenes Modell
📌 Für neue Projekte (2025)	GPT-5 oder GPT-5 Mini
💰 Kostenbewusst	GPT-5 Nano oder GPT-4o Mini
🧠 Komplexes Reasoning	o3 oder o3-mini
⚡ Schnelle Antworten	GPT-5 Nano oder GPT-4o
🔧 Coding & Development	GPT-5 (beste Wahl) oder o3
🖼 Multimodale Aufgaben	GPT-4o oder GPT-5
📊 Datenanalyse	o3 oder GPT-5
💬 Chatbots	GPT-5 Mini oder GPT-4o Mini

API-Namen Übersicht

Modell	API-Name
GPT-5	gpt-5
GPT-5 Mini	gpt-5-mini

Modell	API-Name
GPT-5 Nano	gpt-5-nano
GPT-4o	gpt-4o
GPT-4o Mini	gpt-4o-mini
o3	o3
o3-mini	o3-mini
o4-mini	o4-mini

Stand: September 2025

3 Modellauswahlprozess: Schritt für Schritt

Die Auswahl des optimalen KI-Modells erfordert einen strukturierten Prozess:

3.1 Anforderungsanalyse

- **Definition der Aufgaben:** Legen Sie fest, welche spezifischen Funktionen das Modell erfüllen soll (z.B. Textgenerierung, Fragebeantwortung).
- **Qualitätskriterien:** Bestimmen Sie, welche Qualitätsstandards (Kohärenz, Genauigkeit) erfüllt werden müssen.
- **Domänenkenntnisse:** Identifizieren Sie, welches Fachwissen für Ihre Aufgabe notwendig ist.
- **Antwortgeschwindigkeit:** Definieren Sie die akzeptable Reaktionszeit des Modells.
- **Budget:** Setzen Sie einen finanziellen Rahmen für Ihre KI-Lösung.

3.2 Bewertungskriterien

- **Verständlichkeit:** Wie klar und nachvollziehbar sind die Modellausgaben?

- **Effizienz:** Wie schnell verarbeitet das Modell Eingaben und liefert Ausgaben?
- **Skalierbarkeit:** Kann das Modell mit steigenden Anforderungen mitwachsen?
- **Kosten:** Wie hoch sind die Betriebs- und Nutzungskosten des Modells?

3.3 Recherche und Vorauswahl

- Analysieren Sie verfügbare Modelle anhand Ihrer festgelegten Kriterien und erstellen Sie eine Vorauswahl geeigneter Kandidaten.

3.4 Praktische Modellbewertung

- **Quantitative Methoden:** Verwenden Sie Benchmarks und Metriken, um die Leistung objektiv zu messen.
- **Qualitative Verfahren:** Sammeln Sie Nutzerfeedback zur praktischen Verwendbarkeit.
- **Testphase:** Erproben Sie die Modelle in einer realistischen Umgebung.

3.5 Finale Auswahl und Implementierung

- Treffen Sie eine fundierte Entscheidung für das am besten geeignete Modell und integrieren Sie es in Ihre Systeme.

[Modellauswahl](#)). ☺

4 Modellkaskade: Mehrere Modelle klug kombinieren

Die Modellkaskade kombiniert mehrere KI-Modelle, um ihre jeweiligen Stärken zu nutzen und Schwächen auszugleichen:

4.1 Beispiel für eine Modellkaskade

1. **Datenanalyse mit pandas:** Analysiert große Datensätze und erstellt statistische Zusammenfassungen
2. **Logische Strukturierung mit o3-mini:** Strukturiert die Ergebnisse und erstellt eine logische Gliederung
3. **Kreative Textgenerierung mit GPT-4o:** Verfasst ansprechende Texte basierend auf der Struktur

4. **Multimodale Präsentation:** Ergänzt den Text mit visuellen Elementen

4.2 Vorteile einer Modellkaskade

1. **Effizienzsteigerung:** Jedes Modell wird für seine Stärken optimal eingesetzt
2. **Kostenoptimierung:** Ressourcenschonende Modelle für einfache Aufgaben, teurere nur wo nötig
3. **Flexibilität:** Bearbeitung unterschiedlichster Anforderungen durch spezialisierte Modelle

5 Bewertungsmethoden für KI-Modelle

5.1 Wichtige Benchmarks

- **MMLU (Massive Multitask Language Understanding):** Standard-Benchmark über 57 Fachgebiete, der die Allgemeinbildung und Fachkenntnisse von Modellen misst.

Modell	MMLU-Score
GPT-4o	88,7%
Gemini 2.0 Ultra	90,0%
Claude 3 Opus	88,2%
Llama 3.1 405B	87,3%
gpt-4o-mini	70,0%

5.2 Bewertungsdimensionen

Die Bewertung von KI-Modellen umfasst verschiedene Aspekte:

1. Wissens- und Fähigkeitsbewertung:

- Wie gut beantwortet das Modell Fragen verschiedener Schwierigkeitsgrade?
- Wie zuverlässig ergänzt es fehlendes Wissen?
- Wie gut löst es logische und mathematische Probleme?
- Wie effektiv nutzt es externe Werkzeuge?

2. Alignment-Bewertung:

- Inwieweit stimmt das Modellverhalten mit menschlichen Werten überein?
- Wie ethisch und moralisch sind die Antworten?
- Wie fair und unvoreingenommen ist das Modell?
- Wie wahrhaftig sind die gelieferten Informationen?

3. Sicherheitsbewertung:

- Wie robust ist das Modell gegenüber Störungen und Angriffen?
- Welche potenziellen Risiken birgt die Nutzung des Modells?

5.3 Konkrete Bewertungsmethoden

5.4 Automatisierte Metriken

- **BLEU**: Misst die Übereinstimmung zwischen generiertem und Referenztext durch Vergleich von Wortgruppen.
- **ROUGE**: Bewertet die Qualität von Zusammenfassungen durch Analyse übereinstimmender Wortsequenzen.

5.5 Menschliche Bewertung

- Bewertung nach Kriterien wie Grammatik, Zusammenhang, Lesbarkeit und Relevanz
- Elo-System für den direkten Vergleich verschiedener Modelle (ähnlich wie bei Schach-Ratings)

5.6 KI-basierte Bewertung

- Einsatz leistungsfähiger Modelle zur Bewertung anderer Modelle
- Automatische Erkennung von Fehlinformationen in KI-Antworten

6 Praktische Anwendungsbereiche

Die Modellevaluierung und -auswahl findet in verschiedenen Szenarien Anwendung:

6.1 Kundenservice-Chatbots

- Auswahl eines schnellen Modells mit guter Verständlichkeit und Mehrsprachigkeit
- Bewertung nach Kundenzufriedenheit und Lösungsrate

6.2 Content-Erstellung

- Nutzung kreativer Modelle für Marketing, Social Media und Blogbeiträge
- Bewertung nach Originalität, Engagement und Konversionsraten

6.3 Technische Assistenz

- Einsatz von Reasoning-Modellen für Programmierung und Fehlerbehebung
- Bewertung nach Codequalität und Lösungsgeschwindigkeit

7 Fazit

Fazit

Zusammenfassend lässt sich sagen, dass die **Evaluierung von Large Language Models (LLMs) ein wichtiges Forschungsgebiet** ist, um ihre Fähigkeiten und Grenzen zu verstehen. Die Evaluierung umfasst verschiedene **Attribute wie**

Grammatikalität, Kohäsion, Gefallen, Relevanz, Flüssigkeit und Bedeutungserhalt. Sowohl **menschliche Evaluatoren als auch LLMs selbst werden zur Bewertung eingesetzt.** Es gibt **spezifische Benchmarks und Datensätze** zur Bewertung von LLMs in verschiedenen Bereichen wie **Textgenerierung, Fragebeantwortung und Zusammenfassung.**

Ein wichtiger Aspekt der LLM-Evaluierung ist die **Sicherheitsbewertung**, die **Robustheit gegenüber adversarialen Angriffen** (manipulierte Eingaben, um LLM in die Irre zu führen) und die Identifizierung von **Risiken wie Bias und Toxizität** umfasst. Die Evaluierung kann auch auf **spezialisierte LLMs** in Bereichen wie Medizin, Recht und Finanzen zugeschnitten sein.

Verschiedene **Metriken**, darunter **Likert-Skalen und der BLEU-Score**, werden zur Quantifizierung der LLM-Leistung verwendet. Es gibt auch **Tools und Frameworks wie DeepEval**, die die Evaluierung erleichtern. Es ist wichtig zu beachten, dass **Evaluierungsbias existieren können**, beispielsweise eine Präferenz für längere Texte. Die **ethischen Aspekte** spielen ebenfalls eine Rolle bei der Entwicklung und Nutzung von LLMs.

8 A | Aufgabe

Die Aufgabestellungen unten bieten Anregungen, Sie können aber auch gerne eine andere Herausforderung angehen.

Anforderungsanalyse für ein KI-Projekt

Entwickeln Sie eine strukturierte Anforderungsanalyse für ein fiktives oder reales KI-Projekt.

Aufgabenstellung:

1. Wählen Sie einen konkreten Anwendungsfall (z.B. Kundenservice-Chatbot für eine Bank, Content-Generator für Social Media, oder Übersetzungstool für technische Dokumentation).

2. Definieren Sie:

- Die primären Funktionen, die das KI-Modell erfüllen soll
- Die spezifischen Anforderungen an das Sprachverständnis
- Notwendige Fachkenntnisse in relevanten Domänen
- Anforderungen an die Antwortgeschwindigkeit
- Budget-Rahmenbedingungen

3. Erstellen Sie eine Prioritätenliste dieser Anforderungen (unbedingt erforderlich, wichtig, wünschenswert).

4. Beschreiben Sie, welche Kompromisse Sie bei konkurrierenden Anforderungen eingehen würden.

Abgabeformat:

Erstellen Sie ein Dokument mit Ihrer Anforderungsanalyse (1-2 Seiten).

Vergleichsanalyse bekannter KI-Modelle

Führen Sie eine vergleichende Analyse von mindestens drei verschiedenen KI-Modellen anhand vorgegebener Bewertungskriterien durch.

Aufgabenstellung:

1. Wählen Sie drei KI-Modelle aus der folgenden Liste aus:

- GPT-4o
- Claude 3 Opus
- Gemini 2.0 Ultra
- Llama 3.1
- Mistral 7B
- Ein anderes aktuelles KI-Modell Ihrer Wahl

2. Recherchieren Sie die Leistungsmerkmale dieser Modelle anhand der folgenden Kriterien:

- MMLU-Score oder vergleichbare Benchmark-Ergebnisse
- Kontextfenstergröße
- Antwortlatenz
- Kosten (pro Token oder alternativer Maßstab)
- Verfügbarkeit (API, Open-Source, etc.)
- Unterstützte Sprachen
- Multimodale Fähigkeiten (falls vorhanden)

3. Erstellen Sie eine Bewertungstabelle mit den recherchierten Informationen.

4. Verfassen Sie eine begründete Empfehlung, welches dieser Modelle sich für folgende Szenarien am besten eignen würde:

- Entwicklung eines kostengünstigen Chatbots für ein kleines Unternehmen
- Erstellung von KI-generierten Inhalten für ein internationales Nachrichtenportal
- Unterstützung bei der Software-Entwicklung

Abgabeformat:

Vergleichstabelle mit Bewertungen und einer Seite mit Ihren Empfehlungen.

Konzept für die qualitative Evaluation eines Sprachmodells

Entwickeln Sie ein strukturiertes Testverfahren zur qualitativen Bewertung eines Sprachmodells.

Aufgabenstellung:

1. Entwerfen Sie ein Bewertungsschema mit 5-7 qualitativen Kategorien, die für Ihre gewählte Anwendung relevant sind (z.B. Genauigkeit, Kreativität, Nützlichkeit der Antworten, Verständnis komplexer Anweisungen, Kulturelle Sensibilität).
2. Erstellen Sie für jede Kategorie:
 - Eine klare Definition, was in dieser Kategorie bewertet wird
 - Eine Bewertungsskala (z.B. 1-5 oder 1-10)

- 2-3 konkrete Testfragen oder -aufgaben, die diese Kategorie prüfen
- Bewertungskriterien: Was wäre eine ausgezeichnete (5/5) vs. eine unzureichende (1/5) Antwort?

3. Beschreiben Sie den Evaluationsprozess:

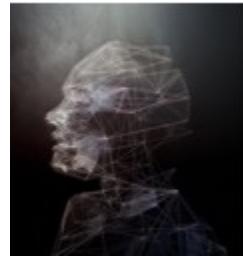
- Wie viele Bewerter sollten eingesetzt werden?
- Wie würden Sie die Bewertungen zusammenfassen?
- Welche Maßnahmen würden Sie ergreifen, um Bewertungsverzerrungen zu vermeiden?

4. Erläutern Sie, wie Sie die Ergebnisse dieser qualitativen Bewertung mit quantitativen Metriken (wie MMLU) kombinieren würden, um ein Gesamtbild der Modellleistung zu erhalten.

Abgabeformat:

Ein 2-3 seitiges Konzeptpapier mit Ihrem Evaluationsschema, den Testfragen und dem geplanten Prozess.

M19b - Reasoning Modelle



Anwendung Generativer KI

Stand: 04.2025

1 Intro

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

2 Was sind Reasoning Modelle?

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.

3 Abgrenzung zu Chat-Modellen

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.

4 Architektonische Unterschiede

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 Programminterpreter 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.

5 Der Einfluss der Trainingsdaten

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.

6 Fähigkeiten im Vergleich

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 natürlich 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.

7 Vorteile von Reasoning Modellen

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.

8 Nachteile von Reasoning Modellen

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 mehrschrittigen 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 Endergebnis 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.

9 Anwendungsfälle von Reasoning Modellen

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

10 Anwendungsfälle anderer Chat-Modelle

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

11 Aktuelle Forschungstrends

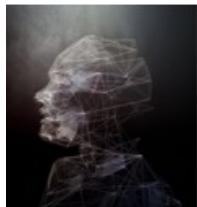
Die aktuelle Forschung im Bereich der Reasoning Modelle konzentriert sich auf die Verbesserung von Prompting-Techniken wie Tree of Thought (ToT) und Graph of Thought (GoT).⁴ Ein weiterer wichtiger Trend ist die Weiterentwicklung der Retrieval-Augmented Generation (RAG).⁴ Die Forschung zielt auch darauf 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 domänenspezifischer 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.

12 Fazit

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.

M20a - Prompt Engineering



Anwendung Generativer KI

Stand: 07.2025

1 Intro

Bei Prompt Engineering geht es darum, effektive Eingaben zu erstellen, um Sprachmodelle wie GPT-4 zur Generierung nützlicher und präziser Ergebnisse zu führen. Eine gute Beherrschung des Prompt Engineering ist entscheidend, um präzise Informationen zu erhalten, überzeugende Kommunikation zu erstellen und fundierte Entscheidungen zu treffen.

Spezifische und vollständige Anweisungen geben

Durch präzise Anweisungen versteht das KI-Modell genau, was benötigt wird. Dies reduziert Unklarheiten und erhöht die Relevanz der Antwort.

Ineffektiver Prompt:

Erkläre die Arten von Versicherungspolicen

Effektiver Prompt:

Als Versicherungsfachperson bitte die wichtigsten Unterschiede zwischen Kapitallebensversicherung und Risikolebensversicherung zusammenfassen, mit Fokus auf Leistungen, Laufzeit und typische Kundenprofile.

Ausnahmesituationen berücksichtigen

Durch Vorhersehen potenzieller Ausnahmen und Anweisungen zum Umgang mit diesen können unvollständige oder irreführende Antworten vermieden werden.

Liste die fünf besten Versicherungsanbieter in Deutschland nach Kundenzufriedenheit auf. Falls aktuelle Daten nicht verfügbar sind, bitte die neuesten Statistiken aus seriösen Quellen nennen und das Jahr der Daten angeben.

Ausgabeformat erklären

Die Definition des gewünschten Ausgabeformats erleichtert die Verwendung der Informationen und kann die Integration mit anderen Dokumenten oder Präsentationen erleichtern.

Erstelle eine Vergleichstabelle für Risikolebensversicherungen und Kapitallebensversicherungen mit den Spalten "Merkmale", "Vorteile" und "Ideal für". Präsentiere die Informationen im Markdown-Format.

2 Fünf Kern-Techniken

2.1 Kontextbasiertes Prompting

Wann anwenden: Wenn KI-Antworten zu oberflächlich sind oder den Kern der spezifischen Situation verfehlten.

Grundprinzip: Bevor die eigentliche Frage oder Aufgabe formuliert wird, werden der KI alle relevanten Hintergrundinformationen gegeben. Dies können spezifische Rahmenbedingungen, wichtige Annahmen, Auszüge aus Dokumenten oder Situationsbeschreibungen sein.

Vorteil: Die KI wird mit den notwendigen Informationen versorgt und kann eine Antwort generieren, die präzise auf die individuelle Situation zugeschnitten ist.

2.1.1 Beispiel: Versicherungsberatung mit Kontext

Ohne Kontext:

Welche Lebensversicherung ist die beste?

Mit Kontext:

KONTEXT:

- Kunde: 35-jähriger Familienvater, zwei Kinder (5 und 8 Jahre)
- Beruf: Selbständiger IT-Berater, schwankendes Einkommen
- Finanzielle Situation: 50.000€ Jahreseinkommen (Durchschnitt), 20.000€ Rücklagen
- Ziele: Absicherung der Familie bei Todesfall, zusätzlicher Vermögensaufbau
- Risikobereitschaft: Moderat
- Bestehende Absicherung: Gesetzliche Krankenversicherung, private Berufsunfähigkeitsversicherung

AUFGABE:

Empfehle eine passende Lebensversicherungslösung und begründe deine Wahl unter Berücksichtigung der spezifischen Situation.

2.1.2 Weitere Anwendungsbereiche:

- **Unternehmensberatung:** Firmenspezifische Daten vor Strategieempfehlungen

- **Technische Dokumentation:** Systemspezifikationen vor Implementierungsanleitungen
 - **Kundenservice:** Kundenhistorie vor Problemlösung
-

2.2 Rollenbasiertes Prompting

Wann anwenden: Wenn eine Antwort aus einer bestimmten Sichtweise, mit einem spezifischen Tonfall oder implizitem Fachwissen benötigt wird.

Grundprinzip: Der KI wird explizit eine Rolle zugewiesen, in die sie schlüpfen soll. Dies kann ein Beruf, eine historische Persönlichkeit oder eine fiktive Figur sein.

Vorteil: Die Perspektive, der Stil und oft auch der implizite Wissensschatz der KI werden gelenkt.

2.2.1 Persona-Muster

Du bist ein erfahrener Versicherungsberater mit 20 Jahren Erfahrung, spezialisiert auf Familienpolicen. Du sprichst verständlich und empathisch. Erkläre einem neuen Kunden die Vorteile einer Kapitallebensversicherung.

2.2.2 Zielgruppen-Persona

Du bist ein Finanzberater, der sich auf junge Erwachsene spezialisiert hat. Erkläre einem 25-jährigen Hochschulabsolventen, der gerade ins Berufsleben eingestiegen ist, warum eine Lebensversicherung sinnvoll sein könnte. Verwende eine lockere, verständliche Sprache ohne Fachjargon.

2.2.3 Rollenspiel-Prompts

ROLLENSPIEL-SETUP:

Du bist ein skeptischer Kunde, der zum ersten Mal eine Lebensversicherung abschließen möchte. Du hast Bedenken bezüglich der Kosten und zweifelst am Nutzen. Stelle kritische Fragen zu Versicherungsprodukten und äußere typische Einwände.

Beginne das Gespräch mit einem Versicherungsberater.

2.2.4 Umgedrehtes Interaktionsmuster

Du bist mein persönlicher Finanzberater mit Expertise in Versicherungsprodukten. Führe ein strukturiertes Beratungsgespräch mit mir. Stelle mir gezielte Fragen, um den besten Lebensversicherungsplan für meine Bedürfnisse zu ermitteln. Beginne mit der Analyse meiner aktuellen Situation.

2.3 Few-Shot Prompting

Wann anwenden: Wenn eine sehr spezifische Antwortstruktur, ein bestimmtes Format oder Muster in der Ausgabe benötigt wird.

Grundprinzip: Statt die KI "kalt" an eine Aufgabe heranzuführen, werden ihr einige wenige, aber aussagekräftige Beispiele direkt im Prompt mitgegeben.

Vorteil: Die KI erkennt das gewünschte Muster und kann es auf neue Anfragen anwenden.

2.3.1 Beispiel 1: Risikoklassifizierung

Klassifizierte die folgenden Antragsteller basierend auf ihrer Krankengeschichte in die Risikokategorien "Niedrig", "Mittel" oder "Hoch":

BEISPIELE:

Antragsteller 1: 45 Jahre, Raucher, leichte Hypertonie, regelmäßiger Sport

Klassifizierung: Mittleres Risiko

Begründung: Rauchen und Bluthochdruck erhöhen das Risiko, Sport wirkt kompensierend

Antragsteller 2: 30 Jahre, Nichtraucher, keine Vorerkrankungen, gesunder Lebensstil

Klassifizierung: Niedriges Risiko

Begründung: Junges Alter, keine Risikofaktoren, gesunder Lebensstil

Antragsteller 3: 55 Jahre, Diabetes Typ 2, Herzinfarkt vor 3 Jahren, Übergewicht

Klassifizierung: Hohes Risiko

Begründung: Multiple schwerwiegende Vorerkrankungen in Kombination

NEUE BEWERTUNG:

Antragsteller 4: 42 Jahre, Nichtraucher, hoher Cholesterinspiegel, familiäre Vorbelastung Herzkrankheiten

Klassifizierung:

2.3.2 Beispiel 2: Kundentyp-Analyse

Analysiere den Kundentyp anhand der Kommunikation und empfehle eine passende Beratungsstrategie:

BEISPIELE:

Kunde A: "Ich möchte schnell eine Versicherung abschließen. Was ist am günstigsten?"

Typ: Preisfokussierter Schnellentscheider

Strategie: Direkte Produktvergleiche, klare Kostenübersicht, schnelle Abwicklung

Kunde B: "Können Sie mir alle Details erklären? Ich möchte alles genau verstehen."

Typ: Analytischer Informationssammler

Strategie: Ausführliche Erklärungen, Zahlen/Daten/Fakten, Bedenkzeit einräumen

NEUE ANALYSE:

Kunde C: "Meine Familie ist mir das Wichtigste. Wie kann ich sie am besten absichern?"

Typ:

Strategie:

2.4 Chain-of-Thought Prompting

Wann anwenden: Bei komplexen Aufgaben, die logisches Denken, Schlussfolgerungen oder mehrere aufeinander aufbauende Schritte erfordern.

Grundprinzip: Die KI wird explizit aufgefordert, ihre Überlegungen und Lösungswege Schritt für Schritt offenzulegen und jeden einzelnen Schritt logisch zu begründen.

Vorteil: Erhöht Transparenz und Qualität bei komplexen, mehrstufigen Aufgaben durch strukturiertes "lautes Denken".

2.4.1 Beispiel 1: Komplexe Risikobewertung

Führe eine umfassende Risikobewertung für den folgenden Antragsteller durch und erkläre dabei jeden Schritt deiner Überlegungen:

ANTRAGSTELLER:

- 52 Jahre, männlich
- Bluthochdruck (seit 5 Jahren, medikamentös eingestellt)
- Nichtraucher (seit 10 Jahren, vorher 20 Jahre Raucher)
- Regelmäßiger Sport (3x pro Woche Joggen)
- Familiäre Vorgeschichte: Vater Herzinfarkt mit 65, Mutter Diabetes Typ 2
- Beruf: Büroangestellter, niedriger Stress
- BMI: 27 (leichtes Übergewicht)

AUFGABE:

Bewerte das Risiko Schritt für Schritt und begründe deine Einschätzung.

Berücksichtige dabei:

1. Positive und negative Risikofaktoren
2. Gewichtung der einzelnen Faktoren
3. Wechselwirkungen zwischen den Faktoren
4. Endgültige Risikoeinstufung mit Begründung

2.4.2 Beispiel 2: Optimale Versicherungsstrategie

Entwickle eine optimale Versicherungsstrategie für den folgenden Fall und dokumentiere dabei jeden Überlegungsschritt:

KUNDE:

- 35 Jahre, verheiratet, zwei Kinder (3 und 6 Jahre)
- Haushaltsnettoeinkommen: 85.000€ jährlich
- Beruf: Angestellter Ingenieur (stabiler Job)
- Partnerin: Teilzeit arbeitend, 20.000€ jährlich
- Immobilie: Eigenheim mit 250.000€ Restschuld
- Vorhandene Absicherung: Gesetzliche Krankenversicherung, BU-Versicherung (beide)
- Sparziel: 500€ monatlich für Altersvorsorge verfügbar

DENK-PROZESS:

Gehe systematisch vor:

1. Analysiere den Absicherungsbedarf
2. Identifizierte Versicherungslücken
3. Priorisiere die Absicherungsbausteine
4. Berechne angemessene Versicherungssummen
5. Empfehle konkrete Produktkombination
6. Begründe jede Entscheidung

2.5 Meta Prompting

Wann anwenden: Wenn man unsicher ist, ob der eigene Prompt bereits gut ist oder das Maximum aus der KI herausholt.

Grundprinzip: Die KI wird nicht nur als Antwortgeber, sondern auch als Berater zur Optimierung des Prompts selbst eingesetzt.

Vorteil: KI-gestütztes Coaching für das eigene Prompt-Design, um Prompts deutlich klarer, präziser und effektiver zu gestalten.

2.5.1 Beispiel 1: Prompt-Analyse und Verbesserung

AUFGABE: Analysiere den folgenden Prompt und gib konkrete Verbesserungsvorschläge:

URSPRUNGLICHER PROMPT:

"Schreibe eine E-Mail über Versicherungen für Kunden."

ANALYSE-KRITERIEN:

1. Klarheit und Spezifität
2. Zielgruppendefinition
3. Gewünschtes Ergebnis
4. Fehlende Kontextinformationen
5. Strukturierung der Anfrage

Bewerte jeden Aspekt und formuliere einen optimierten Prompt.

2.5.2 Beispiel 2: Prompt-Coaching für komplexe Aufgaben

Ich möchte einen Prompt entwickeln, um KI bei der Erstellung von personalisierten Versicherungsempfehlungen zu unterstützen.

MEIN BISHERIGER ANSATZ:

"Empfiehl eine Versicherung basierend auf Kundendaten."

HILF MIR DABEI:

1. Welche spezifischen Informationen sollte ich der KI zur Verfügung stellen?
2. Wie strukturiere ich den Prompt für optimale Ergebnisse?
3. Welche Prompt-Techniken würden hier am besten funktionieren?
4. Erstelle einen beispielhaften optimierten Prompt
5. Erkläre, warum diese Version besser funktioniert

Berücksichtige dabei, dass die Empfehlungen rechtlich korrekt und ethisch vertretbar sein müssen.

2.5.3 Beispiel 3: Iterative Prompt-Optimierung

PROMPT-OPTIMIERUNGS-WORKFLOW:

Schritt 1: Analysiere diesen Prompt

"Erstelle Verkaufstexte für Lebensversicherungen"

Schritt 2: Identifiziere die 3 größten Schwächen

Schritt 3: Formuliere 3 alternative Versionen mit unterschiedlichen Schwerpunkten:

- Version A: Fokus auf Zielgruppe
- Version B: Fokus auf Struktur
- Version C: Fokus auf Kontext

Schritt 4: Bewerte jede Version und empfiehl die beste Lösung

Schritt 5: Erkläre die wichtigsten Lernpunkte für zukünftige Prompts

3 Erweiterte Prompt-Techniken

3.1 Frage- und Prüfmuster

3.1.1 Fragenverfeinerungsmuster

Beim Fragenverfeinerungsmuster beginnt man mit einer allgemeinen Frage und verfeinert sie dann schrittweise.

Was sind die Hauptvorteile einer Lebensversicherung? Konzentriere dich nun speziell darauf, wie eine Lebensversicherung als Anlagevehikel für die Altersvorsorge dienen kann.

3.1.2 Kognitives Prüfmuster

Das kognitive Prüfmuster fordert die AI auf, Informationen zu verifizieren oder zu überprüfen.

Erkläre, wie fondsgebundene Lebensversicherungspolicen funktionieren. Verifiziere die Informationen, indem du die wichtigsten Merkmale und damit verbundenen Risiken zusammenfasst.

3.1.3 Faktenchecklisten-Muster

Liste die erforderlichen Schritte zur Umwandlung einer Risikolebensversicherung in eine Kapitallebensversicherung auf und erkläre jeden Schritt kurz.

3.1.4 Vergleichs-/Kontrast-Prompts

Vergleiche und kontrastiere fondsgebundene Lebensversicherungen und Universal-Lebensversicherungen in Bezug auf Anlageoptionen, Risiko und Flexibilität. Erstelle eine übersichtliche Gegenüberstellung.

3.2 Inhalt- und Struktur-Muster

3.2.1 Vorlagenmuster

Erstelle anhand der folgenden Vorlage eine Zusammenfassung einer Lebensversicherungspolice:

****Police-Übersicht:****

- Name der Police: [Produktnname]
- Art der Police: [Typ]
- Deckungssumme: [Betrag]
- Prämiedetails: [Kosten und Zahlweise]
- Hauptvorteile: [3-5 Punkte]
- Ausschlüsse: [Wichtigste Einschränkungen]
- Zusätzliche Bausteine: [Optionen]
- Zielgruppe: [Für wen geeignet]

3.2.2 Meta-Sprachmuster

Entwerfe eine personalisierte E-Mail an [KUNDENNAME], um ihn über die Vorteile einer Erweiterung seiner aktuellen [VERSICHERUNGSTYP] um [NEUES_FEATURE] zu informieren. Berücksichtige dabei sein Profil als [KUNDENTYP] und verwende einen [KOMMUNIKATIONSSTIL].

3.2.3 Gliederungserweiterungsmuster

Erweitere die folgenden Punkte zu einem umfassenden Artikel über Risikolebensversicherungen:

1. Definition der Risikolebensversicherung
2. Erschwinglichkeit und Einfachheit
3. Ideale Kandidaten für Risikolebensversicherungen
4. Vergleich mit Kapitallebensversicherungen
5. Wie wählt man die richtige Laufzeit
6. Häufige Fehler bei der Auswahl

Jeder Abschnitt soll 200-300 Wörter umfassen und praxisnahe Beispiele enthalten.

4 Prompting für verschiedene Modelltypen

4.1 Prompting für Chat-Modelle

Optimiert für:

- Natürliche Gespräche
- Kundenservice
- Informationsanfragen
- Kreative Inhalte

Effektive Strategien:

1. Konversationeller Ton:

Als Versicherungsberater möchte ich einen Kunden über fondsgebundene Lebensversicherungen informieren. Wie würdest du in einfacher Sprache die Vor- und Nachteile erklären?

2. Rollenbasierte Interaktionen:

Du bist ein freundlicher Versicherungsberater. Ein Kunde fragt, warum er eine Risikolebensversicherung abschließen sollte. Wie würdest du antworten?

3. Strukturierte Formatvorgaben:

Erkläre die Unterschiede zwischen Risiko- und Kapitallebensversicherung in einer übersichtlichen Tabelle mit maximal 5 Vergleichspunkten.

4.2 Prompting für Reasoning-Modelle

Optimiert für:

- Logisches Schlussfolgern
- Komplexe Berechnungen
- Mehrstufige Entscheidungsfindung
- Tiefgreifende Analysen

Effektive Strategien:

1. Schrittweise Analyse:

Analysiere die optimale Versicherungsstrategie für einen 45-jährigen Selbständigen mit zwei Kindern, der für seine Altersvorsorge und den Vermögensaufbau plant. Führe deine Überlegungen Schritt für Schritt durch und begründe jede Empfehlung.

2. Kritisches Denken:

Beurteile kritisch die Vor- und Nachteile einer fondsgebundenen Rentenversicherung im Vergleich zu direkten ETF-Investments für die langfristige Altersvorsorge. Berücksichtige dabei steuerliche Aspekte, Kostenstrukturen und Flexibilität. Begründe jede Schlussfolgerung und betrachte verschiedene Szenarien.

3. Strukturierte Gedankengänge:

Entwickle ein Entscheidungsmodell für die Auswahl einer geeigneten Lebensversicherung. Strukturiere deine Analyse in folgende Schritte:

1. Identifizierte die relevanten Kundenfaktoren
2. Analysiere die verfügbaren Versicherungsoptionen
3. Bewerte Vor- und Nachteile jeder Option
4. Entwickle Entscheidungskriterien
5. Empfehle eine begründete Vorgehensweise

4. "Let's think step by step":

Ein 40-jähriger Familienvater möchte eine Risikolebensversicherung abschließen. Welche Versicherungssumme wäre angemessen? Lass uns Schritt für Schritt überlegen.

4.3 Unterschiede im Prompting

- **Detailgrad:** Reasoning-Modelle profitieren von detaillierteren, strukturierteren Anweisungen
- **Reflexion:** Explizite Aufforderung zur kritischen Überprüfung von Annahmen
- **Mehrstufige Prompts:** Zerlegung komplexer Probleme in Teilschritte

Nachdem du die Empfehlung für die Versicherungslösung gegeben hast, hinterfrage kritisch deine eigenen Annahmen und diskutiere alternative

Szenarien oder mögliche Schwachstellen in deiner Argumentation.

5 Best Practices und Tipps

5.1 Do's

- **Seien Sie spezifisch:** Je präziser der Prompt, desto besser das Ergebnis
- **Geben Sie Kontext:** Hintergrundinformationen verbessern die Antwortqualität
- **Strukturieren Sie komplexe Anfragen:** Zerlegen Sie große Aufgaben in Teilschritte
- **Nutzen Sie Beispiele:** Few-Shot Learning funktioniert oft besser als reine Beschreibungen
- **Fordern Sie Begründungen:** Lassen Sie sich den Denkprozess erklären
- **Iterieren Sie:** Verfeinern Sie Prompts basierend auf den Ergebnissen

5.2 Don'ts

- **Vermeiden Sie Mehrdeutigkeiten:** Unklare Formulierungen führen zu unbrauchbaren Ergebnissen
- **Überlasten Sie nicht:** Zu viele Anforderungen in einem Prompt verwirren das Modell
- **Vergessen Sie nicht die Zielgruppe:** Antworten müssen für den Empfänger verständlich sein
- **Vernachlässigen Sie nicht die Ethik:** Achten Sie auf faire und ausgewogene Darstellungen
- **Ignorieren Sie nicht Feedback:** Nutzen Sie Meta-Prompting zur Verbesserung

5.3 Voreingenommenheit berücksichtigen

Prompt Engineering kann strategisch eingesetzt werden, um Voreingenommenheit in KI-generierten Antworten zu reduzieren:

Erkläre die Vor- und Nachteile verschiedener Lebensversicherungsarten. Achte dabei auf eine neutrale, ausgewogene Darstellung ohne Bevorzugung bestimmter Produkte. Berücksichtige unterschiedliche Lebenssituationen und finanzielle Möglichkeiten.

5.4 Vorschläge zum Prompt einholen

Ich möchte eine E-Mail an einen Kunden verfassen, die die Vorteile einer Zusatzversicherung für schwere Krankheiten zu seiner Lebensversicherung erklärt. Mein aktueller Prompt ist: 'Schreibe eine E-Mail über die

Ergänzung mit einer Zusatzversicherung.' Hast du Vorschläge zur Verbesserung dieses Prompts für detailliertere und überzeugendere Inhalte?

Fazit

Erfolgreiches Prompt Engineering erfordert das Verständnis der verschiedenen Techniken und deren situationsgerechte Anwendung. Die Kombination von Kontext, Rolle, Beispielen, strukturiertem Denken und kontinuierlicher Optimierung führt zu den besten Ergebnissen. Experimentieren Sie mit verschiedenen Ansätzen und nutzen Sie Meta-Prompting, um Ihre Fähigkeiten kontinuierlich zu verbessern.

6 Aufgabe

Few-Shot-Prompting für Kundenfeedback-Analyse

Ziel: Entwickeln Sie einen Few-Shot-Prompt zur Kategorisierung von Kundenfeedback.

Aufgabenstellung:

1. Formulieren Sie einen Prompt, der Kundenfeedback zu einem KI-gestützten Versicherungsprodukt in folgende Kategorien einordnet: "Benutzerfreundlichkeit", "Funktionsumfang", "Technische Probleme" und "Sonstiges"
2. Integrieren Sie mindestens drei Beispiele (Few-Shot-Ansatz)
3. Testen Sie Ihren Prompt mit fünf fiktiven Kundenfeedbacks
4. Fügen Sie eine Stimmungsbewertung hinzu (positiv, neutral, negativ)

Chain-of-Thought für ML-Modellauswahl

Ziel: Anwendung des Chain-of-Thought-Ansatzes für eine fundierte ML-Modellempfehlung.

Aufgabenstellung:

1. Erstellen Sie einen Prompt für die Empfehlung eines ML-Algorithmus
2. Szenario: "Ein Versicherungsunternehmen möchte Betrugsversuche erkennen, Schadenshöhen vorhersagen und Kundensegmente identifizieren"
3. Der Prompt soll das Modell anweisen, den Entscheidungsprozess Schritt für Schritt darzulegen
4. Identifizierung von Implementierungsherausforderungen

Persona-basierte Zielgruppen-Kommunikation

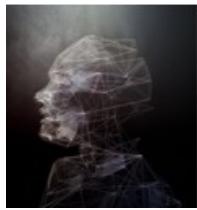
Ziel: Erstellung zielgruppenspezifischer Erklärungen mit Persona-Prompting.

Aufgabenstellung:

1. Wählen Sie ein komplexes KI-Konzept (z.B. maschinelles Lernen in der Risikomodellierung)
2. Entwickeln Sie drei Persona-Prompts für:
 - C-Level Executive ohne technischen Hintergrund
 - Versicherungsmathematiker mit Grundkenntnissen
 - IT-Entwickler für die Implementierung
3. Vergleichen Sie die Ergebnisse hinsichtlich Fachsprache, Detailtiefe und Praxisbezug

4. Fügen Sie branchenspezifische Anwendungsbeispiele hinzu

M20b - Langchain Hub



Anwendung Generativer KI

Stand: 05.2025

1 Einführung

Der LangChain Hub ist eine zentrale Plattform für die Verwaltung, Versionierung und gemeinsame Nutzung von Prompts in LangChain-Anwendungen. Diese Funktion ermöglicht es Entwicklern, ihre Prompts zu standardisieren, zu optimieren und effizient in Teams zusammenzuarbeiten.

2 Hauptfunktionen

2.1 Zentrale Prompt-Verwaltung

Der LangChain Hub ermöglicht die zentrale Speicherung und Verwaltung von Prompts, was besonders in größeren Teams oder Projekten vorteilhaft ist, um Konsistenz zu gewährleisten.

```
from langchain_hub import pull

# Einen bestimmten Prompt aus dem Hub abrufen
qa_prompt = pull("yourusername/qa_prompt")
```

2.2 Prompt-Versionierung

Ähnlich wie bei Git können Prompts versioniert werden, um Änderungen nachzuverfolgen und bei Bedarf zu früheren Versionen zurückzukehren.

```
# Eine bestimmte Version eines Prompts abrufen
qa_prompt_v2 = pull("yourusername/qa_prompt:v2")
```

2.3 Prompt-Sharing und Kollaboration

Teams können Prompts einfach teilen und gemeinsam daran arbeiten, was die Zusammenarbeit und Wiederverwendung von bewährten Prompt-Mustern fördert.

```
from langchain_hub import push
from langchain_core.prompts import PromptTemplate

# Einen neuen Prompt erstellen
my_prompt = PromptTemplate.from_template(
    "Analysiere folgendes Dataset: {dataset_description}"
)

# Prompt zum Hub hochladen
push(my_prompt, "yourusername/data_analysis_prompt", new_version=True)
```

2.4 Prompt-Bibliotheken und Community

Der Hub bietet Zugang zu einer wachsenden Sammlung von Community-erstellten Prompts für verschiedene Anwendungsfälle, die als Ausgangspunkt oder Inspiration dienen können.

```
# Einen Community-Prompt für Zusammenfassungen verwenden
summarization_prompt = pull("community/summarization:latest")
```

3 Integration in LangChain-Anwendungen

Die Hub-Prompts lassen sich nahtlos in bestehende LangChain-Anwendungen integrieren:

```
from langchain_hub import pull
from langchain.chains import LLMChain
from langchain_openai import ChatOpenAI

# Prompt aus dem Hub abrufen
classification_prompt = pull("myteam/sentiment_classification:v3")

# LLM definieren
llm = ChatOpenAI(model="gpt-4")
```

```
# Chain erstellen und ausführen
chain = LLMChain(llm=llm, prompt=classification_prompt)
result = chain.run(text="Das neue Produkt übertrifft alle meine
Erwartungen!")
```

4 Best Practices

4.1 Standardisierte Benennung

Entwickeln Sie ein konsistentes Benennungsschema für Ihre Prompts, das den Anwendungsfall und die Version klar kommuniziert.

```
teamname/anwendungsfall_prompttyp:version
```

Beispiel: dataengineers/customer_feedback_classification:v2

4.2 Dokumentation von Prompts

Fügen Sie jedem Prompt Metadaten und Dokumentation hinzu, um seinen Zweck, die erwarteten Eingaben und das AusgabefORMAT zu beschreiben.

```
from langchain_core.prompts import PromptTemplate
from langchain_hub import push

prompt = PromptTemplate(
    template="Klassifiziere das folgende Kundenfeedback: {feedback}",
    input_variables=["feedback"],
    metadata={
        "description": "Prompt zur Klassifizierung von Kundenfeedback in Kategorien",
        "expected_output": "Eine der Kategorien: Positiv, Neutral, Negativ",
        "example_inputs": {"feedback": "Ihr Produkt funktioniert großartig!"}
    }
)

push(prompt, "myteam/feedback_classification", new_version=True)
```

4.3 A/B-Tests und Prompt-Experimente

Nutzen Sie den Hub für A/B-Tests verschiedener Prompt-Formulierungen, um die effektivsten Prompts für Ihre Anwendungsfälle zu identifizieren.

```
# Variante A abrufen und testen
prompt_a = pull("myteam/product_summary:v1")
# Variante B abrufen und testen
prompt_b = pull("myteam/product_summary:experimental")

# Vergleich der Ergebnisse beider Prompts
```

5 Fortgeschrittene Anwendungen

5.1 Prompt-Chaining mit dem Hub

Komplexe Reasoning-Ketten können durch die Kombination spezialisierter Prompts aus dem Hub erstellt werden.

```
from langchain_hub import pull
from langchain.chains import SequentialChain
from langchain.chains import LLMChain
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4")

# Spezialisierte Prompts aus dem Hub abrufen
extraction_prompt = pull("myteam/data_extraction:v2")
analysis_prompt = pull("myteam/data_analysis:v1")
recommendation_prompt = pull("myteam/recommendations:v3")

# Chains erstellen
extraction_chain = LLMChain(llm=llm, prompt=extraction_prompt,
output_key="extracted_data")
analysis_chain = LLMChain(llm=llm, prompt=analysis_prompt,
output_key="analysis_results")
recommendation_chain = LLMChain(llm=llm, prompt=recommendation_prompt,
output_key="recommendations")

# Sequentielle Chain erstellen
sequential_chain = SequentialChain(
    chains=[extraction_chain, analysis_chain, recommendation_chain],
    input_variables=["raw_data"],
    output_variables=["recommendations"],
    verbose=True
```

```
)  
  
results = sequential_chain({"raw_data": customer_data})
```

5.2 Dynamische Prompt-Auswahl

Implementieren Sie eine dynamische Auswahl von Prompts basierend auf Eingabedaten oder Kontextinformationen.

```
from langchain_hub import pull  
  
def select_prompt(input_data):  
    if "technical" in input_data.lower():  
        return pull("myteam/technical_support:latest")  
    elif "billing" in input_data.lower():  
        return pull("myteam/billing_support:latest")  
    else:  
        return pull("myteam/general_support:latest")  
  
user_query = "Ich habe ein technisches Problem mit der API-Integration."  
selected_prompt = select_prompt(user_query)  
# Verwenden des ausgewählten Prompts für die Antwortgenerierung
```

6 Hub-Prompts mit Reasoning-Modellen

Für komplexe Reasoning-Aufgaben können spezialisierte Prompts aus dem Hub besonders wertvoll sein:

```
from langchain_hub import pull  
from langchain_openai import ChatOpenAI  
from langchain.chains import LLMChain  
  
# Reasoning-Prompt aus dem Hub abrufen  
reasoning_prompt = pull("myteam/step_by_step_reasoning:v2")  
  
# Modell mit erweitertem Reasoning verwenden  
reasoning_model = ChatOpenAI(model="gpt-4", temperature=0.2)  
  
# Chain erstellen  
reasoning_chain = LLMChain(llm=reasoning_model, prompt=reasoning_prompt)
```

```
# Komplexes Problem lösen mit strukturiertem Reasoning
problem = """
Ein Datensatz enthält 5000 Kundenrezensionen. 20% der Rezensionen
enthalten technische Fragen,
35% beziehen sich auf Preisgestaltung, und der Rest betrifft
Lieferprobleme.
Welcher ML-Ansatz wäre optimal für die Klassifizierung dieser Daten und
warum?
"""

analysis = reasoning_chain.run(problem=problem)
```

7 Template Repositories

LangChain Hub bietet vorgefertigte Template Repositories, die schnell in eigenen Projekten eingesetzt werden können:

```
# Community-Template für Few-Shot-Prompting abrufen
few_shot_template = pull("community/few_shot_classification:latest")

# Anpassen mit eigenen Beispielen
custom_few_shot = few_shot_template.format(
    examples=[
        {"input": "Das System reagiert nicht.", "output": "Technisches
Problem"},
        {"input": "Die Kosten sind zu hoch.", "output": "Preisbeschwerde"},
        {"input": "Tolle Funktionen!", "output": "Positives Feedback"}
    ]
)
```

8 Praktische Anwendungsfälle

8.1 Mehrsprachige Prompts

Verwalten Sie Prompts in verschiedenen Sprachen für internationale Anwendungen:

```
# Sprachspezifische Prompts abrufen
de_prompt = pull("myteam/customer_service:de")
en_prompt = pull("myteam/customer_service:en")
es_prompt = pull("myteam/customer_service:es")
```

```
# Dynamische Sprachauswahl
def get_language_prompt(language_code):
    return pull(f"myteam/customer_service:{language_code}")
```

8.2 Domänenspezifische Prompts

Erstellen Sie spezialisierte Sammlungen von Prompts für verschiedene Geschäftsbereiche:

```
# Prompts für verschiedene ML-Aufgaben
classification_prompt = pull("ml_team/text_classification:v2")
sentiment_prompt = pull("ml_team/sentiment_analysis:latest")
entity_extraction_prompt = pull("ml_team/entity_extraction:stable")
```

9 Zukunft des Prompt-Engineerings

Der LangChain Hub entwickelt sich kontinuierlich weiter und bietet Entwicklern leistungsstarke Werkzeuge für das Prompt-Engineering:

- **Automatisierte Prompt-Optimierung:** Zukünftige Versionen werden voraussichtlich Tools für automatisierte A/B-Tests und Optimierung von Prompts enthalten.
- **Erweiterte Prompt-Templates:** Komplexere Template-Strukturen mit bedingter Logik und dynamischer Formatierung.
- **KI-gestützte Prompt-Vorschläge:** Intelligente Vorschläge zur Verbesserung von Prompts basierend auf Erfolgsmetriken.

Der LangChain Hub bietet eine robuste Infrastruktur für die Verwaltung und Optimierung von Prompts, die besonders bei der Zusammenarbeit in Teams und bei der Entwicklung komplexer KI-Anwendungen wertvoll ist. Durch die Standardisierung und Versionierung von Prompts können Entwickler konsistentere und effektivere Ergebnisse erzielen.

M21 - Context Engineering



Anwendung Generativer KI

Stand: 07.2025

1 Was ist Context Engineering?

Context Engineering ist die **Kunst, KI-Systemen die richtigen Informationen zur richtigen Zeit zu geben**. Stellen Sie sich vor, Sie sind ein Berater, der einem Kunden hilft - Sie brauchen alle relevanten Informationen über den Kunden, seine Situation und seine Bedürfnisse, um eine gute Beratung zu geben.

1.1 Der Unterschied zu Prompt Engineering

Aspekt	Prompt Engineering	Context Engineering
Definition	Kunst der Formulierung optimaler Eingabeaufforderungen für KI-Modelle	Systematisches Design und Management des gesamten Kontexts für KI-Systeme
Fokus	Einzelne Prompt-Optimierung	Gesamtes Kontextmanagement und -architektur
Zeitrahmen	Kurzfristig, pro Anfrage	Langfristig, systemweit
Zielgruppe	Endnutzer, Content-Erststeller	Entwickler, Systemarchitekten
Hauptziel	Bessere Antworten durch optimierte Prompts	Konsistente, kontextbewusste KI-Systeme
Techniken	- Few-Shot Learning - Chain-of-Thought - Role-Playing - Template-Design	- RAG (Retrieval-Augmented Generation) - Vektorschreibe - Wissensgraphen - Kontext-Caching
Eingabeformat	Textuelle Anweisungen und Beispiele	Strukturierte Daten, Dokumente, Metadaten

Aspekt	Prompt Engineering	Context Engineering
Skalierbarkeit	Begrenzt auf einzelne Interaktionen	Hochskalierbar für Enterprise-Anwendungen
Wartung	Manuelle Anpassung der Prompts	Automatisiertes Kontext-Management
Fehlerbehandlung	Trial-and-Error bei Prompts	Systematische Kontext-Validierung
Messbarkeit	Qualitative Bewertung der Antworten	Quantitative Metriken (Relevanz, Genauigkeit)
Kosten	Niedrig (nur Prompt-Optimierung)	Höher (Infrastruktur, Datenmanagement)
Anwendungsbereich	- Chatbots - Content-Generierung - Übersetzungen - Kreative Aufgaben	- Wissensmanagementsysteme - Dokumentensuche - Expertensysteme - Enterprise-KI
Herausforderungen	- Prompt-Injection - Inkonsistente Ergebnisse - Begrenzte Kontextlänge	- Datenqualität - Kontext-Fragmentierung - Skalierungskosten
Erfolgsfaktoren	- Klare Anweisungen - Gute Beispiele - Strukturierte Prompts	- Hochwertige Datenquellen - Effiziente Suche - Kontext-Relevanz
Tools & Frameworks	- OpenAI Playground - LangChain PromptTemplates - Anthropic Console	- LangChain - LlamaIndex - Pinecone - Weaviate
Zukunftstrend	Integration in größere Systeme	Weiterentwicklung zu autonomen Agenten
Best Practices	- Iterative Verbesserung - A/B-Testing - Klare Rollenverteilung	- Datengovernance - Monitoring & Logging - Kontext-Versionierung

1.2 Fazit

Prompt Engineering ist ideal für schnelle, einzelne Optimierungen und kreative Anwendungen, während **Context Engineering** für robuste, skalierbare KI-Systeme in Unternehmen unerlässlich ist. Moderne KI-Anwendungen profitieren von der Kombination beider Ansätze.

1.3 Warum ist das wichtig?

- **85% aller KI-Fehler** entstehen durch fehlende oder falsche Kontextinformationen

- Ihr Prompt macht nur **0,1%** des gesamten Kontexts aus, den die KI verarbeitet
- **38% bessere Ergebnisse** durch gutes Context Engineering

2 Die vier Grundstrategien

2.1 Kontext Auswählen(Context Selection)

Die richtigen Informationen zur richtigen Zeit bereitstellen.

Beispiel - Versicherungsberatung:

Kundenkontext:

- Alter: 35 Jahre
- Familie: 2 Kinder
- Beruf: Selbständige
- Ziel: Familienabsicherung

→ KI wählt passende Produktinformationen aus

2.2 Kontext Komprimieren (Context Compression)

Nur die wichtigsten Informationen behalten.

Beispiel:

Lange Schadenshistorie (50 Seiten)

↓

Zusammenfassung: "3 Kleinschäden in 5 Jahren,
Gesamtschaden: 2.500€, keine Muster erkennbar"

2.3 Kontext Speichern (Context Memory)

Wichtige Informationen für später aufzubewahren.

Beispiel:

Kundeninteraktion 1: "Ich bevorzuge niedrige Beiträge"

↓ (gespeichert)

Kundeninteraktion 2: KI erinnert sich an Präferenz

2.4 Kontext Trennen (Context Isolation)

Verschiedene Aufgaben mit separaten Kontexten bearbeiten.

Beispiel:

Agent A: Schadensprüfung (hat Zugang zu Schadensdaten)

Agent B: Kundenberatung (hat Zugang zu Produktdaten)

3 Die drei häufigsten Fehler

3.1 Context Overload

Problem: Zu viele Informationen verwirren die KI

Lösung: Nur relevante Informationen bereitstellen

3.2 Context Conflict

Problem: Widersprüchliche Informationen

Lösung: Informationen auf Konsistenz prüfen

3.3 Context Staleness

Problem: Veraltete Informationen

Lösung: Regelmäßige Updates implementieren

4 Praktische Anwendung

4.1 Kontext analysieren

Frage: "Welche Versicherung brauche ich?"

Benötigte Kontextinformationen (nach Priorität):

✓ KRITISCH:

- Alter: 32 Jahre
- Familienstand: verheiratet, 2 Kinder (3, 7 Jahre)
- Beruf: Software-Entwickler
- Einkommen: 65.000€ brutto/Jahr

✓ WICHTIG:

- Bestehende Absicherungen: KFZ-Haftpflicht, Hausratversicherung
- Immobilienstatus: Eigenheim (Restschuld 180.000€)
- Gesundheitsstatus: keine Vorerkrankungen

✓ ERGÄNZEND:

- Risikobereitschaft: konservativ
- Finanzielle Ziele: Familienabsicherung, Altersvorsorge
- Verfügbares Budget: 200€/Monat für Versicherungen

4.2 Kontext strukturieren

PROMPT-STRUKTUR:**==== KUNDENKONTEXT ===****Demografisch:**

- Person: 32 Jahre, männlich, verheiratet
- Familie: 2 Kinder (3, 7 Jahre), Hausfrau-Ehefrau
- Wohnort: Eigenheim, Restschuld 180.000€

Finanziell:

- Einkommen: 65.000€ brutto/Jahr (alleinverdienend)
- Budget Versicherungen: 200€/Monat
- Risikobereitschaft: konservativ

==== PRODUKTKONTEXT ===**Bestehende Absicherung:**

- KFZ-Haftpflicht: vollständig
- Hausratversicherung: 50.000€ Versicherungssumme
- Keine weitere Absicherung vorhanden

Relevante Produktkategorien:

- Risikolebensversicherung (Familienabsicherung)
- Berufsunfähigkeitsversicherung (Einkommensschutz)
- Private Unfallversicherung
- Rechtsschutzversicherung

==== BERATUNGSKONTEXT ===

Anfrage: "Welche Versicherung brauche ich?"

Beratungsziel: Bedarfsanalyse und Produktempfehlung

Compliance: Versicherungsberatung nach §34d GewO

4.3 Kontext optimieren

OPTIMIERUNGSREGELN für KI-Verarbeitung:**1. TOKEN-EFFIZIENZ (Max. 500 Token für Kontext):**

✗ "Der Kunde ist 32 Jahre alt und arbeitet als Software-Entwickler..."

✓ "Kunde: 32J, Software-Dev, 65k€, verheiratet, 2 Kinder"

2. RELEVANZ-FILTERING:

Für Versicherungsberatung IMMER relevant:

- Alter, Familienstand, Beruf, Einkommen
- Bestehende Policien
- Gesundheitsstatus (wenn abgefragt)

SITUATIV relevant:

- Hobbys (nur bei Unfallversicherung)
- Immobilien (nur bei Sachversicherungen)

3. STRUKTURIERUNG für LLM:

AUFTAG: Versicherungsbedarfsanalyse KUNDE: 32J, Soft-Dev, 65k€, verheiratet, 2Ki(3,7J), Eigenheim(180k€ Schuld) BESTAND: KFZ-Haft, Hausrat(50k€) BUDGET: 200€/Monat PRÄFERENZ: konservativ ZIEL: Familien-/Einkommensabsicherung

AUFGABE: Identifiziere Versicherungslücken und empfehle passende Produkte mit Begründung.

4.4 Konsistenz-Checkliste:

- [] Gleiche Kategorien in allen Abschnitten verwendet
- [] Konkrete Beispiele statt Platzhalter
- [] Token-Limits definiert und eingehalten
- [] Relevanz-Kriterien spezifiziert
- [] Optimierung messbar (Token-Reduktion, Strukturierung)

5 Einfache Tools und Techniken

5.1 Tool 1: Context-Checkliste

- Sind alle notwendigen Informationen vorhanden?
- Sind die Informationen aktuell?

- Gibt es Widersprüche?
- Ist der Kontext nicht zu lang?
- Ist der Kontext relevant für die Aufgabe?

5.2 Tool 2: Kontext-Templates

Kundenberatung-Template:

KUNDE: [Name, Alter, Beruf]

SITUATION: [Aktuelle Lebensumstände]

ZIEL: [Was möchte der Kunde erreichen?]

BUDGET: [Verfügbare Mittel]

PRÄFERENZEN: [Besondere Wünsche]

5.3 Tool 3: Einfache Kontextregeln

1. Immer aktuellste Daten verwenden
2. Maximal 3 Hauptinformationen pro Kontext
3. Widersprüche sofort klären
4. Kundenspezifische Informationen priorisieren
5. Rechtliche Anforderungen immer beachten

6 Messbare Verbesserungen

6.1 Vorher vs. Nachher

Ohne Context Engineering:

- ✗ 45% Fehlerrate bei Empfehlungen
- ✗ 3+ Nachfragen pro Beratung
- ✗ 15 Min. Bearbeitungszeit

Mit Context Engineering:

- ✓ 12% Fehlerrate bei Empfehlungen
- ✓ 1 Nachfrage pro Beratung
- ✓ 8 Min. Bearbeitungszeit

6.2 Erfolgs-Metriken

Genauigkeit: +65%

Effizienz: +47%

Kundenzufriedenheit: +30%

7 Sofort umsetzbare Tipps

7.1 Do's

- Beginnen Sie mit einfachen Context-Checklisten
- Sammeln Sie systematisch Feedback
- Dokumentieren Sie erfolgreiche Kontextmuster
- Starten Sie mit Ihren häufigsten Anwendungsfällen
- Messen Sie Verbesserungen kontinuierlich

7.2 Don'ts

- Nicht zu kompliziert beginnen
- Nicht alle Kontextquellen auf einmal ändern
- Nicht ohne Messungen optimieren
- Nicht vergessen, das Team zu schulen
- Nicht auf Feedback verzichten

8 Nächste Schritte

8.1 Stufe 1: Grundlagen (Sie sind hier!)

- Context Engineering verstehen
- Erste Tools anwenden
- Einfache Verbesserungen umsetzen

8.2 Stufe 2: Fortgeschrittene Techniken

- Automatisierte Kontextauswahl
- KI-gestützte Kontextoptimierung
- Multi-Agenten-Systeme

8.3 Stufe 3: Expertenlevel

- Eigene Context-Engineering-Frameworks
- Komplexe Gedächtnissysteme
- Unternehmensweite Implementierung

 **Tipp**

Context Engineering ist eine erlernbare Fähigkeit, die sofort bessere KI-Ergebnisse liefert. Beginnen Sie heute mit einfachen Techniken und bauen Sie schrittweise Expertise auf!

9 Aufgabe

9.1 Aufgabe 1: Kontext-Analyse

Aufgabe: Analysieren Sie eine typische Kundenanfrage in Ihrem Bereich.

Beispiel:

Kundenanfrage: "Ich suche eine günstige Hausratversicherung"

Fehlende Kontextinformationen:

- Wohnort und Wohnungsgröße?
- Wert der Einrichtung?
- Besondere Risiken?
- Bisherige Schäden?
- Definition von "günstig"?

9.2 Aufgabe 2: Kontext-Design

Aufgabe: Erstellen Sie ein Kontext-Template für Ihre häufigste Aufgabe.

Vorlage:

AUFGABE: [Beschreibung]

BENÖTIGTE INFORMATIONEN:

1. [Primäre Info]
2. [Sekundäre Info]
3. [Ergänzende Info]

AUSSCHLUSSKRITERIEN:

- [Was nicht relevant ist]

QUALITÄTSKRITERIEN:

- [Wann ist der Kontext gut?]

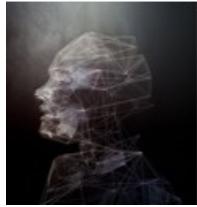
9.3 Aufgabe 3: Fehler-Identifikation

Aufgabe: Identifizieren Sie typische Kontextfehler in Ihrem Arbeitsbereich.

Häufige Fehler:

- Veraltete Produktinformationen
- Fehlende Kundenpräferenzen
- Unvollständige Risikobewertung
- Ignorierte Ausschlusskriterien
- Widersprüchliche Datenquellen

M22a - EU AI Act



Anwendung Generativer KI

Stand: 04.2025

1 Einleitung & Zielsetzung

Der EU AI Act (Verordnung (EU) 2024/1689) ist das weltweit erste umfassende Gesetz zur Regulierung von Künstlicher Intelligenz (KI). Ziel ist es, ein vertrauenswürdiges KI-Ökosystem in Europa zu schaffen, das Sicherheit, Grundrechte und europäische Werte wahrt, gleichzeitig aber Innovation und Investitionen fördert. Kern des Gesetzes ist ein risikobasierter Ansatz, der KI-Systeme in vier Risikoklassen einteilt: inakzeptabel, hoch, begrenzt und minimal. Für jede Risikoklasse gelten abgestufte Anforderungen. Der Gesetzgebungsprozess wurde durch globale politische Entwicklungen, technologische Dynamik und zunehmende gesellschaftliche Debatten beschleunigt. Der AI Act ist Teil der Digitalstrategie der EU und steht im Kontext internationaler Regulierungsbemühungen.

2 Umsetzung & Anwendbarkeit

Das Gesetz trat am 1. August 2024 in Kraft. Die Anwendung erfolgt gestaffelt bis 2030. Erste Verbote (z. B. manipulative KI, Social Scoring) gelten seit Februar 2025. Für Hochrisiko-KI-Systeme sind lange Übergangsfristen vorgesehen. Mitgliedstaaten müssen zuständige Behörden benennen, Sandkästen einrichten und über notwendige Ressourcen verfügen. Bisher zeigen sich Unterschiede im Umsetzungsstand. Auf EU-Ebene übernimmt das European AI Office zentrale Aufgaben, insbesondere bei General Purpose AI (GPAI). Die nationale Umsetzung variiert stark, was Risiken für Fragmentierung birgt. Sandkästen sollen Innovation fördern, insbesondere bei KMU.

Wichtige Meilensteine bei der Umsetzung der KI-Regulierung



3 Risikoklassen im Detail

Der risikobasierte Ansatz ist das zentrale Steuerungsinstrument des EU AI Acts. KI-Systeme werden in vier Klassen eingestuft, je nach potenzieller Auswirkung auf Sicherheit, Grundrechte und gesellschaftliche Werte:

- **Inakzeptables Risiko:** KI-Systeme, die gegen Werte der EU verstößen oder erhebliche Risiken für Individuen bergen, sind verboten. Beispiele: Social Scoring durch Behörden, Echtzeit-Gesichtserkennung im öffentlichen Raum (außer in engen Ausnahmefällen).
- **Hohes Risiko:** Systeme in sensiblen Bereichen wie Bildung, Justiz, Strafverfolgung, Gesundheit oder kritischen Infrastrukturen. Sie unterliegen umfangreichen Anforderungen wie Risikomanagement, Dokumentationspflichten, menschlicher Aufsicht und hoher Datenqualität. Diese Systeme müssen vor dem Einsatz ein Konformitätsbewertungsverfahren durchlaufen.
- **Begrenztes Risiko:** Systeme mit potenziellen Transparenzrisiken, wie Chatbots oder Deepfake-Generatoren. Hier bestehen Informationspflichten gegenüber Nutzer:innen, z. B. Kennzeichnung, dass Inhalte KI-generiert sind.
- **Minimales Risiko:** Die Mehrheit aller KI-Anwendungen (z. B. Spamfilter, Empfehlungssysteme für Streaming-Dienste) fällt in diese Kategorie. Es gelten keine verpflichtenden Anforderungen, freiwillige Verhaltenskodizes werden jedoch gefördert.

Diese Einteilung ermöglicht eine differenzierte Regulierung: Statt alle KI-Technologien über einen Kamm zu scheren, werden Anforderungen gezielt dort angesetzt, wo die potenziellen Gefahren für Individuum und Gesellschaft am größten sind.

KI-Risikokategorien



Unakzeptables Risiko

KI-Systeme, die gegen Werte der EU verstößen oder erhebliche Risiken für Individuen bergen, sind verboten. Beispiele: Social Scoring und Echtzeit-Gesichtserkennung.



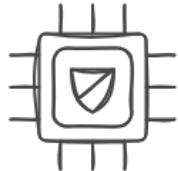
Hohes Risiko

Systeme in sensiblen Bereichen wie Bildung, Justiz oder Gesundheitswesen. Sie erfordern Risikomanagement, Dokumentation, menschliche Aufsicht und hohe Datenqualität.



Begrenztes Risiko

Systeme mit potenziellen Transparenzrisiken, wie Chatbots. Diese haben Informationspflichten gegenüber Nutzern, wie die Kennzeichnung von KI-generierten Inhalten.



Minimales Risiko

Die meisten KI-Anwendungen, wie Spamfilter, fallen in diese Kategorie. Es gibt keine verpflichtenden Anforderungen, aber freiwillige Verhaltenskodizes werden empfohlen.

Made with Naj

4 Herausforderungen & Kritikpunkte

Zentrale Herausforderungen:

- **Unklare Begriffsdefinitionen:** Begriffe wie "KI-System", "systemisches Risiko" oder "enge verfahrenstechnische Aufgabe" sind interpretationsbedürftig.
- **Risikoklassifizierung:** Die Einstufung als Hochrisiko-System ist teils unklar. Die Ausnahmeregelung nach Art. 6(3) wird als potenzielles Schlupfloch kritisiert.
- **GPAI-Regulierung:** Neue Regeln für Basismodelle wie GPT-4 sind komplex. Kritik gibt es an Schwellenwerten, Transparenzpflichten und Umsetzbarkeit.
- **Grundrechtslücken:** Zivilgesellschaftliche Organisationen bemängeln Ausnahmen für Sicherheitsbehörden und unzureichenden Schutz z. B. im Migrationskontext.
- **Durchsetzbarkeit:** Behörden mangelt es oft an Ressourcen und Expertise. Die komplexe Governance-Struktur erhöht die Anforderungen an Koordination.
- **Wirtschaftliche Sorgen:** Unternehmen kritisieren hohe Compliance-Kosten, vage Formulierungen und potenzielle Innovationshemmnisse.

5 Potenziale & Chancen

Trotz der Kritik birgt der AI Act bedeutende Potenziale:

- **Rechtssicherheit:** Einheitliche Regeln erleichtern die Planung und Investition für Unternehmen.
- **Marktchancen:** Es entsteht ein Markt für vertrauenswürdige KI-Produkte, der als Qualitätsmerkmal dienen kann.
- **Neue Dienstleistungen:** Nachfrage nach Compliance-Tools, Auditierung, Governance-Frameworks und Ethikberatung steigt.
- **Standardisierung:** Harmonisierte technische Normen könnten auch international prägend wirken.
- **Weltweite Vorreiterrolle:** Der "Brussels Effect" – die globale Strahlkraft europäischer Regulierung – könnte KI-Regelsetzung weltweit beeinflussen.

6 Best Practices & Empfehlungen

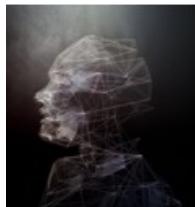
Empfohlene Maßnahmen für die Umsetzung:

- **Frühzeitige Systeminventur:** Unternehmen sollten ihre KI-Systeme erfassen und frühzeitig risikobasiert einstufen.
- **Governance-Modelle etablieren:** Aufbau interner Compliance-Strukturen, idealerweise unter Einbeziehung bestehender DSGVO-Frameworks.
- **Transparenz und Dokumentation:** Verfahrensdokumentation, Modellkarten, Bias-Tests und menschliche Aufsicht sind essenziell.
- **Nutzung von Sandkästen:** Besonders für KMU eine Möglichkeit, Innovation rechtskonform zu testen.
- **Offene Kommunikation:** Stakeholder-Dialoge und partizipative Gestaltung erhöhen Akzeptanz und Wirksamkeit.

7 Ausblick

Der langfristige Erfolg des EU AI Acts wird davon abhängen, ob es gelingt, die Balance zwischen Regulierung und Innovationsförderung zu halten. Die praktische Durchsetzbarkeit, die Kohärenz mit anderen EU-Gesetzen (z. B. DSGVO, Data Act, DSA) sowie die internationale Anschlussfähigkeit werden entscheidend sein. Wichtig wird auch sein, wie flexibel der AI Act auf technologische Entwicklungen reagieren kann und ob er Vertrauen in KI langfristig stärkt.

M22b - Ethik und Generative KI



Anwendung Generativer KI

Stand: 04.2025

1 Ethische Dimensionen

Definition & Abgrenzung:

- Generative KI (GenAI) erzeugt neue Inhalte (Texte, Bilder, Musik, Code) auf Basis gelernter Muster und stellt damit eine kreative, wenn auch nicht bewusste, Nachbildung menschlicher Ausdrucksformen dar.
- Abgrenzung zu anderen KI-Typen:
 - *Analytische oder prädiktive KI* analysiert bestehende Daten, um Vorhersagen zu treffen (z. B. Kredit-Scoring). Ethikfragen betreffen hier v. a. Fairness und Nachvollziehbarkeit der Entscheidungskriterien.
 - *Regelbasierte oder symbolische KI* folgt festen, menschenkodierten Entscheidungsregeln (z. B. Expertensysteme) und ist meist gut erklärbar.
 - *AGI* (Artificial General Intelligence) beschreibt eine hypothetische KI mit menschenähnlicher, allgemeiner Intelligenz. Sie existiert aktuell nicht, ist aber zentrales Thema in der KI-Ethik-Forschung.

Die Abgrenzung ist wichtig, da **generative KI besonders intransparente, kreative Outputs erzeugt**, was neue ethische Fragestellungen aufwirft – etwa zur Originalität, Verantwortung und Manipulationsgefahr.

Zentrale ethische Prinzipien:

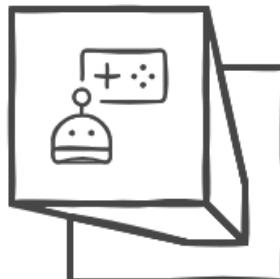
- **Verantwortung:** Wer haftet bei Fehlentscheidungen? Eine klare juristische und ethische Zuweisung ist meist schwierig.
- **Fairness:** Gefahr der Reproduktion sozialer Ungleichheiten durch Daten-Bias; bedarf systematischer Überprüfung.
- **Transparenz:** Undurchschaubarkeit der Modelle verhindert Vertrauen und kontrollierte Anwendung.

- **Datenschutz:** Besonders problematisch bei sensiblen Daten wie Gesundheitsdaten oder intimen Nutzereingaben.
- **Autonomie:** Nutzer:innen dürfen nicht entmündigt werden; Systeme müssen überschreibbar bleiben.
- **Sicherheit:** Technisch wie gesellschaftlich müssen Risiken minimiert werden, etwa durch Missbrauchsprävention.
- **Akteure:** Technologieunternehmen (oft marktgetrieben), Forschung (wissenstrieben), Politik (regulierend), Zivilgesellschaft (wertorientiert), Nutzer:innen (praktisch-orientiert) prägen gemeinsam das öffentliche Verständnis und die Entwicklungspfade von KI.

Ethische Prinzipien in der KI

Autonomie

Autonomie ist einfach umzusetzen, hat aber großen Einfluss auf Nutzer.



Datenschutz

Datenschutz erfordert komplexe Lösungen mit hoher gesellschaftlicher Wirkung.



Transparenz

Transparenz ist leicht zu erreichen, aber mit geringer Wirkung.



Verantwortung

Verantwortung ist komplex, aber mit begrenzter direkter Auswirkung.



Made with Napkin

2 Rahmenwerke & Praxis

Regulatorische Grundlagen:

- Der **EU AI Act** ist das weltweit erste umfassende Gesetz zur Regulierung von KI und unterteilt Systeme in vier Risikokategorien. Besonders generative KI mit hohem Einfluss auf Meinungsbildung und Kreativbereiche steht dabei unter besonderer Beobachtung.

- **OECD- und UNESCO-Richtlinien** setzen wichtige normative Standards, die Fairness, Erklärbarkeit und Rechenschaftspflicht als universelle Prinzipien formulieren.

Umsetzung in der Industrie:

- Unternehmen wie OpenAI, Google und Meta haben ethische Leitlinien entwickelt, die Aspekte wie Moderation, Red Teaming und Prompt-Engineering einschließen.
- Tools wie SynthID oder Wasserzeichen-Lösungen fördern Transparenz und Fälschungsschutz.
- Trotzdem bleibt die Selbstverpflichtung oft hinter regulatorischen Anforderungen zurück.

Organisatorische Umsetzung:

- Interne Ethikboards, Compliance-Beauftragte und Prozesse zur Ethikfolgenabschätzung gewinnen an Bedeutung.
- Wichtig ist nicht nur die Existenz, sondern die Integration ethischer Reflexion in agile Entwicklungsprozesse.

Rolle von Bildung & Forschung:

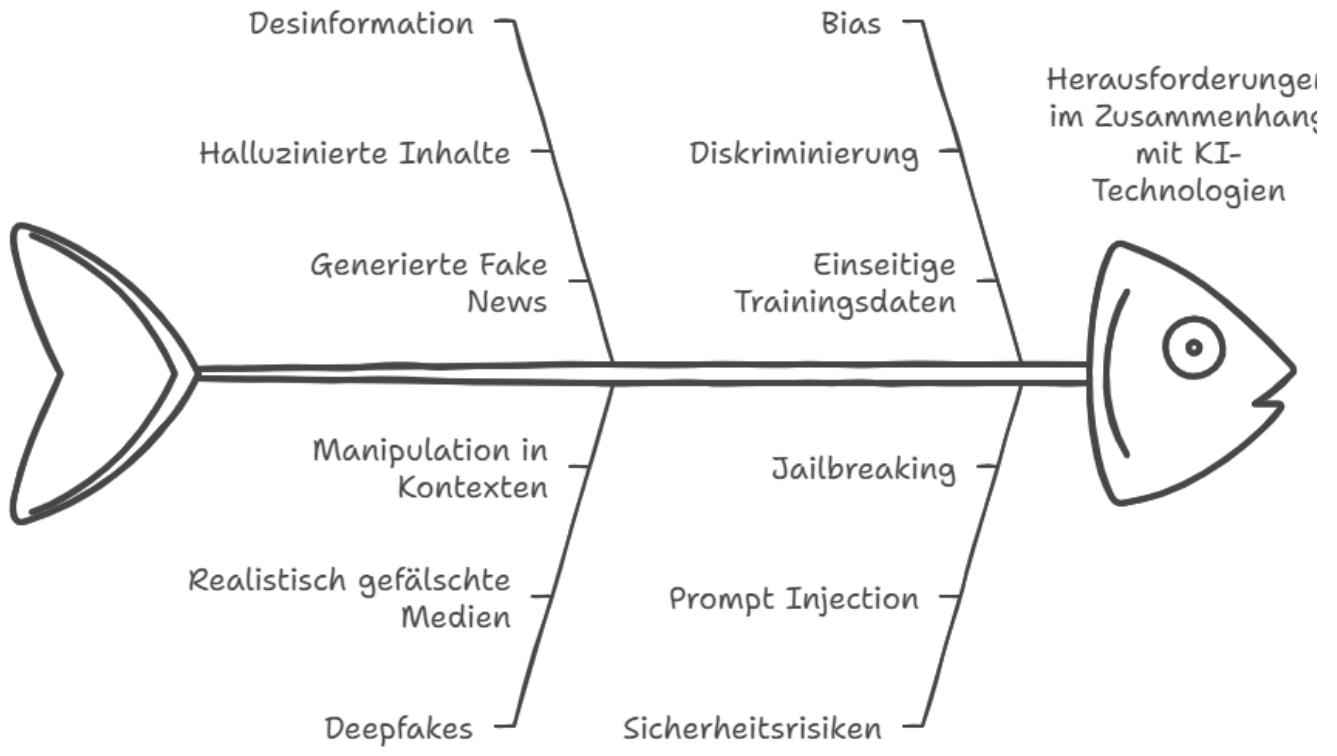
- Hochschulen und Ausbildungsinstitutionen entwickeln spezialisierte Curricula zu KI-Ethik.
- Praxisnahe Formate wie Fallanalysen, Planspiele oder interdisziplinäre Projektarbeit sind besonders wirksam.

3 Risiken & Fehlerquellen

Kernrisiken:

- **Desinformation:** Halluzinierte Inhalte und generierte Fake News untergraben Vertrauen in Informationen.
- **Deepfakes:** Realistisch gefälschte Medien können zur Manipulation in politischen oder wirtschaftlichen Kontexten führen.
- **Bias:** Diskriminierung aufgrund einseitiger Trainingsdaten ist ein zentrales Problem.
- **Rechtsunsicherheit:** Unklarheiten bei Urheberrecht und Datenschutz hemmen klare Verantwortungszuweisung.
- **Sicherheitsrisiken:** Prompt Injection, Jailbreaking oder Training mit vergifteten Daten sind reale Angriffsvektoren.

Risiken im Zusammenhang mit KI



Made with Nap

Ethische Spannungsfelder:

- Innovation vs. Regulierung
- Transparenz vs. Datenschutz oder geistiges Eigentum
- Open Source vs. Missbrauch
- Automatisierung vs. Arbeitsplatzverlust

Fehlerquellen im Lebenszyklus:

- *Daten*: Verzerrung durch schlechte oder einseitige Quellen.
- *Modell*: Fehlende Robustheit, Halluzinationen, Black-Box-Verhalten.
- *Prozess*: Mangelnde Tests, nicht-diverse Teams, unklare Verantwortlichkeiten.
- *Nutzung*: Missbrauch für Desinformation, unreflektierte Übernahme von KI-Outputs.

Verantwortung:

- Verteilte Verantwortung in Wertschöpfungsketten erschwert Haftung und Governance.
- Neue regulatorische Ansätze wie der AI Act definieren Rollen und Pflichten neu.

4 Chancen & Potenziale

Gesellschaftlicher Mehrwert:

- **Bildung:** Automatisierte Lernpfade, intelligente Nachhilfe und adaptive Lernumgebungen.
- **Barrierefreiheit:** Text-zu-Sprache, visuelle Erkennung, einfache Sprache für mehr Teilhabe.
- **Wissenschaft:** Hypothesengenerierung, Datenanalyse, automatisierte Literaturauswertung.
- **Kreativität:** Unterstützung für künstlerische Prozesse und Demokratisierung kreativer Mittel.
- **Wirtschaft:** Automatisierung von Routineaufgaben, Entlastung von Fachkräften.
- **Nachhaltigkeit:** Umweltmonitoring, Klimamodellierung, Analyse von ESG-Daten.

Ethics by Design:

- Ethische Reflexion bereits in der Designphase einbetten.
- Interdisziplinäre Teams, Impact Assessments und diverse Perspektiven sind zentral.

Gemeinwohlorientierte KI:

- Open-Source-Modelle, öffentliche KI-Infrastrukturen (z. B. EU AI Factories), transparente Standards.
- Ziel: Technologische Souveränität und gerechter Zugang zu KI.

5 Best Practices

Technische Maßnahmen:

- **Explainable AI (XAI):** Methoden wie LIME, SHAP, RAG-basierte Erklärungen fördern Transparenz.
- **Bias-Mitigation:** In allen Phasen (Pre-, In-, Postprocessing), begleitet von Fairness-Audits.
- **Sicherheit:** Red Teaming, Input-Validierung, Zugriffskontrollen, Inhaltsfilter.
- **Datenschutz:** Anonymisierung, Pseudonymisierung, differenzielle Privatsphäre.
- **Transparenz:** Dokumentation, Wasserzeichen, klare Kommunikation der KI-Nutzung.

Organisatorische Strategien:

- Etablierung klarer Verantwortlichkeiten für KI im Unternehmen.
- Entwicklung und Pflege von KI-Ethik-Kodizes.

- Integration von ethischer Reflexion in Produktentwicklung und -bewertung.

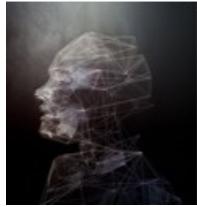
Bildungs- und Schulungsinitiativen:

- Schulung für Entwickler (z. B. Bias, Datenschutz, XAI).
- Aufklärung von Anwendern über Grenzen, Risiken und verantwortungsvollen Umgang mit KI.

Rahmenwerke und Tools:

- Nutzung internationaler Standards (z. B. NIST AI RMF, EU AI Act, ISO 42001).
- Checklisten für ethisch orientierte Entwicklung und Deployment.

M23 - KI-Challenge



Anwendung Generativer KI

Stand: 07.2025

1 | Überblick KI-Challenge

Die KI-Challenge dient als praktische Anwendung und Integration der in den Kursmodulen erlernten Konzepte und Techniken. Ziel ist es, eine funktionsfähige KI-Anwendung zu entwickeln, die mehrere Aspekte der generativen KI kombiniert und einen praktischen Nutzen bietet.

1.1 Lernziele

- Integration mehrerer Technologien aus den Basismodulen
- Praktische Anwendung von LLM-basierten Lösungen
- Entwicklung einer vollständigen End-to-End-Anwendung
- Präsentation und Dokumentation der eigenen Lösung

1.2 Voraussetzungen

- Abschluss der Basismodule (Module 1-12)
- Module aus dem Bereich Erweiterung
- Kenntnisse in Python und LangChain
- Zugriff auf API-Keys (OpenAI, Hugging Face)
- Grundlegende Vertrautheit mit Gradio für UI-Entwicklung

2 | Projektoptionen

Zur Auswahl stehen vier verschiedene Projekttypen, die jeweils unterschiedliche Aspekte der generativen KI betonen. Wählen Sie eine Option aus oder kombinieren Sie Elemente verschiedener Optionen.

2.1 Dokumentenanalyse-Assistent

Beschreibung: Ein System, das PDF-Dokumente, Word-Dateien oder Textdateien verarbeitet und intelligente Zusammenfassungen, Antworten auf Fragen oder strukturierte Analysen liefert.

Kernelemente:

- RAG-Pipeline mit Vektordatenbank (ChromaDB)
- Dokumentenverarbeitung und Chunking
- Intelligentes Prompting für die Analyse
- Benutzeroberfläche mit Gradio

Erwartete Module:

- Modul 4 (LangChain)
- Modul 7 (Output Parser)
- Modul 8 (RAG)
- Modul 11 (Gradio)

2.2 Multimodaler Assistent

Beschreibung: Ein Assistent, der Bild, Text und optional Audio verarbeiten kann, um komplexe Aufgaben zu erfüllen oder Informationen zu analysieren.

Kernelemente:

- Integration von Bild- und Texterkennung
- Multimodale Prompt-Strategien
- Kontextbewusste Antworten
- Interaktive Benutzeroberfläche

Erwartete Module:

- Modul 5 (LLMs und Transformer)
- Modul 6 (Chat und Memory)
- Modul 9 (Multimodal Bild)
- Modul 14 (optional: Multimodal Audio)

2.3 Agentenbasiertes System

Beschreibung: Ein System mit mehreren spezialisierten Agenten, die zusammenarbeiten, um komplexe Aufgaben zu lösen oder Workflow-Prozesse zu automatisieren.

Kernelemente:

- Multi-Agenten-Architektur
- Werkzeugintegration (APIs, Datenbanken)
- Planung und Zielverfolgung
- Benutzerinteraktion und Transparenz

Erwartete Module:

- Modul 3 (Codieren mit GenAI)
- Modul 10 (Agents)
- Modul 12 (Lokale Modelle)
- Modul 18 (optional: Advanced Prompt Engineering)

2.4 Domänen Fachexperte

Beschreibung: Ein spezialisierter Assistent für ein bestimmtes Fachgebiet (z.B. Recht, Medizin, Finanzen, Marketing), der tiefgreifendes Fachwissen bereitstellt und domänenspezifische Aufgaben löst.

Kernelemente:

- Fachspezifische Wissensdatenbank
- Spezialisierte Prompts und Output-Strukturen
- Benutzeroberfläche für Fachexperten
- Optional: Feinabstimmung eines bestehenden Modells

Erwartete Module:

- Modul 2 (Grundlagen Modellansteuerung)
- Modul 8 (RAG)
- Modul 16 (optional: Fine-Tuning)
- Modul 19 (optional: EU AI Act/Ethik)

3 | Projekt-Setup

Hier finden Sie den Code für das grundlegende Setup Ihres Projekts, ähnlich wie in den Kursmodulen.

```
#@title
#@markdown    <p><font size="4" color='green'> Colab-Umfeld</font> </p>
# Installierte Python Version
import sys
```

```
print(f"Python Version: ", sys.version)
# Installierte LangChain Bibliotheken
print()
print("Installierte LangChain Bibliotheken:")

!pip list | grep '^langchain'
# Unterdrückt die "DeprecationWarning" von LangChain für die Memory-
Funktionen
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning,
module="langsmith.client")
```

```
#@title
#@markdown <p><font size="4" color='green'> SetUp API-Keys
(setup_api_keys)</font> <br></p>
def setup_api_keys():
    """Konfiguriert alle benötigten API-Keys aus Google Colab userdata"""
    from google.colab import userdata
    import os
    from os import environ

    # Dictionary der benötigten API-Keys
    keys = {
        'OPENAI_API_KEY': 'OPENAI_API_KEY',
        'HF_TOKEN': 'HF_TOKEN',
        # Weitere Keys bei Bedarf
    }

    # Keys in Umgebungsvariablen setzen
    for env_var, key_name in keys.items():
        environ[env_var] = userdata.get(key_name)

    return {k: environ[k] for k in keys.keys()}

# Verwendung
all_keys = setup_api_keys()
# Bei Bedarf einzelne Keys direkt zugreifen
# WEATHER_API_KEY = all_keys['WEATHER_API_KEY']
```

4 | Projektstruktur

Ein erfolgreiches Abschlussprojekt sollte folgende Komponenten enthalten:

4.1 Problemdefinition und Anforderungen

- Klare Beschreibung des Problems oder der Aufgabe
- Definition der Anforderungen und Erfolgskriterien
- Abgrenzung des Projektumfangs

4.2 Datenstrukturen und Modellauswahl

- Auswahl und Begründung der verwendeten Modelle
- Datenstrukturen und Datenvorbereitung
- Embedding-Strategien (bei RAG-Anwendungen)

4.3 Kernfunktionalität

- LangChain-Pipelines oder -Ketten
- Prompt-Engineering und Templates
- Integration mit externen APIs oder Datenquellen

4.4 Benutzeroberfläche und Interaktion

- Gradio-Interface für die Interaktion
- Benutzerführung und Feedback
- Fehlerbehandlung und Robustheit

4.5 Evaluation und Tests

- Testfälle für verschiedene Szenarien
- Bewertung der Modellleistung
- Benutzerfeedback und Verbesserungen

4.6 Dokumentation und Präsentation

- Projektdokumentation (Markdown oder PDF)
- Code-Kommentare und Erklärungen
- Präsentation der Ergebnisse

5 | Bewertungskriterien

Die KI-Challenge wird anhand folgender Kriterien bewertet:

Kriterium	Beschreibung	Gewichtung
Funktionalität	Die Anwendung erfüllt die definierten Anforderungen und funktioniert zuverlässig	30%
Integration	Erfolgreiche Kombination mehrerer Technologien und Module aus dem Kurs	25%
Code-Qualität	Sauberer, lesbarer und gut strukturierter Code mit angemessenen Kommentaren	15%
Innovation	Kreative Lösungsansätze und eigenständige Weiterentwicklung der Konzepte	15%
Dokumentation	Vollständige und verständliche Dokumentation des Projekts	15%

6 | Beispielprojekt: Doku-Assi

Als Orientierung dient hier ein vereinfachtes Beispiel für einen Dokumentenanalyse-Assistenten:

```
# Import der benötigten Bibliotheken
import os
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import Chroma
from langchain.chat_models import ChatOpenAI
from langchain.retrievers.multi_query import MultiQueryRetriever
from langchain.chains import ConversationalRetrievalChain
import gradio as gr

# API-Keys einrichten
os.environ["OPENAI_API_KEY"] = "Ihr-OpenAI-Key"

# Funktion zum Laden und Verarbeiten von Dokumenten
def load_and_process_document(file_path):
    """
    Lädt ein PDF-Dokument und bereitet es für die Verarbeitung vor

    Args:
        file_path: Pfad zur PDF-Datei
    """
    # Hier kommt der Code, der das Dokument laden und in Teile unterteilen
    # sowie Vektoren erstellen wird
    pass
```

Returns:

```
    Chroma-Vektordatenbank mit den Dokumentenchunks
"""
# PDF laden
loader = PyPDFLoader(file_path)
pages = loader.load()

# Text in Chunks aufteilen
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=100
)
chunks = text_splitter.split_documents(pages)

# Embeddings erstellen und Vektorstore initialisieren
embeddings = OpenAIEmbeddings()
vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=embeddings
)

return vectorstore

# Chat-Modell und Retrieval-Kette initialisieren
def setup_qa_chain(vectorstore):
    """
    Erstellt eine Konversations-Retrieval-Kette für Frage-Antwort-
    Interaktionen
    """

    Args:
        vectorstore: Chroma-Vektordatenbank

    Returns:
        ConversationalRetrievalChain für QA
    """
    # LLM initialisieren
    llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")

    # Retriever mit Multi-Query-Strategie
    retriever = MultiQueryRetriever.from_llm(
        vectorstore.as_retriever(search_kwargs={"k": 3}),
        llm=llm
    )

    return ConversationalRetrievalChain(
        retriever=retriever,
        llm=llm,
        question_generator=TextPromptGenerator(
            prompt="Frage: "
        ),
        message_history=MessageHistory(),
        verbose=True
    )
```

```
    llm
)

# QA-Kette erstellen
qa_chain = ConversationalRetrievalChain.from_llm(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)

return qa_chain

# Gradio-Interface für die Benutzerinteraktion
def create_interface():
    """
    Erstellt ein Gradio-Interface für die Benutzerinteraktion

    Returns:
        Gradio-Interface
    """
    # Zustandsvariablen
    state = {
        "qa_chain": None,
        "chat_history": []
    }

    # PDF-Upload-Funktion
    def upload_pdf(file):
        try:
            vectorstore = load_and_process_document(file.name)
            state["qa_chain"] = setup_qa_chain(vectorstore)
            state["chat_history"] = []
            return "Dokument erfolgreich geladen und verarbeitet!"
        except Exception as e:
            return f" Fehler beim Laden des Dokuments: {str(e)}"

    # Frage-Antwort-Funktion
    def ask_question(question):
        if state["qa_chain"] is None:
            return "Bitte laden Sie zuerst ein Dokument hoch."

        try:
```

```
        result = state["qa_chain"](
            {"question": question, "chat_history": state["chat_history"]})
    )

    # Chat-Historie aktualisieren
    state["chat_history"].append((question, result["answer"]))

    # Quellen hinzufügen
    sources = set()
    for doc in result["source_documents"]:
        page_content = doc.page_content[:150] + "..." if
len(doc.page_content) > 150 else doc.page_content
        sources.add(f"Quelle (Seite {doc.metadata.get('page',
'N/A')}) : {page_content}")

    sources_text = "\n\n".join(sources)
    full_response = f"{result['answer']}\n\n---\n\nVerwendete
Quellen:\n{sources_text}"

    return full_response
except Exception as e:
    return f"Fehler bei der Verarbeitung der Frage: {str(e)}"

# Gradio-Interface erstellen
with gr.Blocks(title="Dokumentenanalyse-Assistent") as interface:
    gr.Markdown("# 📄 Dokumentenanalyse-Assistent")
    gr.Markdown("Laden Sie ein PDF-Dokument hoch und stellen Sie
Fragen dazu.")

    with gr.Row():
        with gr.Column():
            file_input = gr.File(label="PDF-Dokument hochladen")
            upload_button = gr.Button("Dokument verarbeiten")
            status_text = gr.Textbox(label="Status",
interactive=False)

        with gr.Column():
            question_input = gr.Textbox(label="Ihre Frage zum
Dokument", placeholder="Stellen Sie eine Frage zum Inhalt des
Dokuments...")
            answer_output = gr.Textbox(label="Antwort",
```

```
interactive=False, lines=15)
    ask_button = gr.Button("Frage stellen")

    # Ereignisbehandlung
    upload_button.click(upload_pdf, inputs=[file_input], outputs=
[status_text])
    ask_button.click(ask_question, inputs=[question_input], outputs=
[answer_output])

return interface

# Hauptfunktion
def main():
    interface = create_interface()
    interface.launch(share=True)

# Ausführung
if __name__ == "__main__":
    main()
```

7 | Ressourcen und Hilfestellung

Folgende Ressourcen können bei der Entwicklung des Abschlussprojekts hilfreich sein:

- **Dokumentation:**
 - [LangChain Dokumentation](#)
 - [OpenAI API Dokumentation](#)
 - [Hugging Face Dokumentation](#)
 - [Gradio Dokumentation](#)
- **Beispielprojekte und Tutorials:**
 - LangChain Cookbook im GitHub-Repository
 - Beispiel-Implementierungen aus den Kursmodulen
 - Hugging Face Spaces für Beispielanwendungen
- **Online-Tools:**
 - GenAI Tutor
 - ChatBots, wie ChatGPT, Gemini, ...
 - ...

Bei Fragen oder Problemen während der Projektentwicklung können Sie das [Kurs-Forum](#) nutzen.



Viel Erfolg!