Anwendungsaspekte des Machine Learning

Assignment Teil 1

vorgelegt am 13. Juni 2024

Fakultät Wirtschaft und Gesundheit

Studiengang Wirtschaftsinformatik

Kurs WWI2022A

von

Luca Philipp

Inhaltsverzeichnis

1 Theoretische Recherche und Einordnung

1.1 RAG-Systeme

1.1.1 Architektur

Ein Retrieval Augmented Generation (RAG)-System verbindet ein Large Language Model (LLM) mit externen Datenquellen, um gezieltere und aktuellere Antworten zu geben.[1] Solche Systeme bestehen aus mehreren Modulen, die gemeinsam für eine präzise, sichere und nachvollziehbare Verarbeitung von Informationen sorgen. Die Architektur lässt sich anhand von sechs Funktionsbereichen beschreiben, die den Ablauf und Bestandteile eines funktionierendes RAG-System darstellen:

Dokumenten-Vorverarbeitung: Bevor ein RAG-System auf Inhalte aus externen Datenquellen zugreifen kann, müssen diese vorbereitet werden. Der Text wird z. B. aus PDFs oder Webseiten extrahiert, in kleinere Abschnitte (Chunks) aufgeteilt und anschließend können zusätzliche Informationen wie Titel oder Quellen ergänzt werden. Das Ziel des „Preprocessings“ ist es, die Inhalte gut durchsuchbar und für das Sprachmodell nutzbar zu machen.

Speichern der Inhalte: Die aufbereiteten Inhalte der Datenquellen werden in speziellen Datenbanken gespeichert, wobei beispielsweise Vektordatenbanken für klassische Textdaten und SQL-Datenbanken für strukturierte Daten genutzt werden können. Manche Systeme speichern verschiedene Inhalte in unterschiedlichen Datenbanken und greifen je nach Bedarf gezielt darauf zu.

Datenabruf: Wenn ein Nutzer eine Frage stellt, sucht das System passende Textabschnitte in den gespeicherten Daten. Es verwendet dabei moderne Suchmethoden, die auch ähnliche Formulierungen erkennen. Bei strukturierten Daten wird automatisch eine Datenbankabfrage erstellt. Das Ziel ist es, möglichst passende Informationen zur Nutzerfrage zu finden.

Routing: Bevor die Suche beginnt, entscheidet das System, welche Datenquellen überhaupt relevant sind. Das System nutzt dabei meist ein kleines LLM, um eine passende Auswahl zu treffen.

Antwortgenerierung: Im letzten Schritt werden die gefundenen Inhalte mit der ursprünglichen Frage kombiniert und an das große Sprachmodell weitergegeben. Es erstellt daraus eine Antwort. Anschließend wird diese Antwort noch in Hinblick auf die Fakten, vertraulichen Informationen und den Richtlinien geprüft. Erst dann wird die Antwort an den Nutzer zurückgegeben.

Management-Ebene: Diese Schicht kümmert sich um unterstützende Funktionen wie dem Speichern von Nutzereingaben und Antworten (z. B. Chatverläufe), der Verwaltung von Nutzerrechten, oder der Überwachung des Systems (z. B. Protokolle, Fehleranalyse). Sie ist unter anderem für Sicherheit, Kontrolle und Nachvollziehbarkeit wichtig.

---

[1] Vgl. hierzu und im Folgenden Kronsbein 2025

Zusammengefasst besteht eine RAG-Architektur also aus der Dokumentenvorverarbeitung, den angebundenen externen Datenquellen, der Speicherung in spezialisierten Datenbanken, einer Retrieval-Komponente zum Datenabruf, einem Routing-Modul zur Auswahl relevanter Datenquellen sowie der Antwortgenerierung durch ein Sprachmodell mit nachgelagerter Qualitäts- und Sicherheitsprüfung. Ergänzt wird diese Architektur durch eine Management-Ebene, die zentrale Funktionen wie Protokollierung, Zugriffskontrolle und die Verwaltung von Nutzerdaten übernimmt.

1.1.2 Unterschiede zu reinen generativen Modellen und klassichen QA-Systemen

Der Hauptunterschied von RAG-Systemen zu reinen generativen Sprachmodellen liegt darin, dass diese ausschließlich auf das im Trainingsprozess erlernte Wissen zurückgreifen, während RAG-Systeme externe Datenquellen mit einem Sprachmodell kombinieren, um wissensintensive Aufgaben präziser zu lösen.[2] In der Nutzung des Sprachmodells liegt auch der wichtigste Unterschied zu klassischen QA-Systemen, welche Textpassagen direkt aus den Dokumenten extrahieren anstatt neue Antworten basierend auf den Inhalten zu generieren.

1.1.3 Rolle von Embdeddings & Vektordatenbanken

Um für die Antworten nach relevantem Wissen zu suchen, spielen Embeddings und Vektordatenbanken eine zentrale Rolle. Embeddings wandeln Text in numerische Vektoren um, die semantische Bedeutung erfassen.[3] Diese Vektoren werden in einer Vektordatenbank gespeichert, um bei einer Nutzeranfrage ähnliche Inhalte schnell und präzise finden zu können.

1.1.4 Rolle von Prompting & Kontextkonstruktion

Prompting bezeichnet im Kontext eines RAG-Systems den Prozess, bei dem die Nutzeranfrage gemeinsam mit den abgerufenen Inhalten in eine strukturierte Eingabe (Prompt) für das Sprachmodell eingebettet wird. In einem RAG-System sorgt Prompting und Kontextkonstruktion dafür, dass die ursprüngliche Frage mit den relevanten Textabschnitten kombiniert und so formuliert wird, dass das Sprachmodell auf dieser Basis eine präzise und kontextbezogene Antwort generieren kann.[4]

---

[2] Vgl. hierzu und im Folgenden Lewis et al. 2021
[3] Vgl. hierzu und im Folgenden Schwaber-Cohen 2023
[4] Vgl. hierzu und im Folgenden Simpson 2025

1.1.5 Rolle von Modellwahl und Kostenaspekte

In RAG-Systemen beeinflusst die Modellwahl entscheidend die Qualität der Antworten, ihre Latenz und die entstehenden Kosten.[5] Große Sprachmodelle (z. B. GPT-4/Turbo) liefern meist präzisere und kontextsensible Antworten, verursachen aber deutlich höhere Token- und Inferenzkosten. Kleinere Modelle oder lokal gehostete Open-Source-Modelle bieten hingegen eine günstigere Alternative bei akzeptabler Qualität. Der ideale Kompromiss hängt von Budget, Performance-Anforderungen und Antwortqualität ab.

1.1.6 Typische Herausforderungen bei der Umsetzung

RAG-Systeme weisen aufgrund der Vielzahl beteiligter Komponenten eine hohe technische Komplexität auf, sowohl in der Implementierung als auch im laufenden Betrieb.[6] Außerdem spielen neben typischen Herausforderungen wie dem begrenzten Kontextfenster großer Sprachmodelle und potenziell hoher Latenz auch Retrieval-Qualität und Datenpflege eine zentrale Rolle. Die abgerufenen Inhalte müssen aktuell, präzise und korrekt gewichtet sein, um zuverlässige Antworten zu ermöglichen. Zusätzlich treten häufig Probleme auf wie fehlende Inhalte in der Wissensbasis, unvollständige oder falsch formatierte Ausgaben sowie technische Hürden beim Verarbeiten komplexer Dateiformate (z. B. PDFs mit Tabellen).

1.2 Agentensysteme

1.2.1 Was ist ein Agentensystem und Unterschied zur einfachen Prompt-Chain

Ein LLM-Agent ist eine Anwendung, bei der ein LLM als zentrales Steuerelement agiert und komplexe Aufgaben durch eine Kombination von Modulen wie Planung (Planning), Gedächtnis (Memory) und Tool-Nutzung bewältigt.[7] Das LLM fungiert dabei als eine Art Gehirn des Systems und steuert den Ablauf von Teilaufgaben, um ein Ziel oder eine Benutzeranfrage zu erfüllen. Während einfache Fragen eventuell noch direkt durch ein LLM oder ein RAG-System beantwortet werden können, erfordern komplexere Aufgaben ein mehrstufiges Vorgehen mit mehreren Zwischenschritten. Dazu können zum Beispiel Analysen oder Visualisierungen zählen. Ein LLM-Agent kann hierfür externe Werkzeuge wie Suchschnittstellen, Datenbanken oder Code-Interpreter nutzen, um beispielsweise Daten zu recherchieren, zu analysieren und als Diagramm darzustellen. Um solche Aufgaben zielgerichtet zu lösen, benötigt der Agent einerseits eine Planungskomponente, die Aktionen strukturiert, und andererseits eine Speicherkomponente, die den aktuellen Zustand, frühere Beobachtungen und Zwischenergebnisse

[5] Vgl. hierzu und im Folgenden Barkai 2023
[6] Vgl. hierzu und im Folgenden Wikipedia o. J.; vgl. ebenfalls Eruysaliz 2024
[7] Vgl. hierzu und im Folgenden Prompt Engineering Guide 2025

verfolgt. Die Eingabe erfolgt typischerweise über eine Nutzeranfrage, die durch ein Prompt-Template angereichert wird, wo auch definiert ist, welche Tools der LLM-Agent nutzen darf.

Im Unterschied zu einer einfachen Prompt-Chain, bei der eine Reihe fest definierter Prompts nacheinander abgearbeitet wird, zeichnet sich ein LLM-Agent durch eine höhere Eigenständigkeit und Flexibilität aus. Während eine Prompt-Chain meist statisch aufgebaut ist und keine eigenständige Entscheidungslogik enthält, trifft ein Agent dynamische Entscheidungen, wie z.B. welche Tools eingesetzt werden sollen oder wie eine komplexe Aufgabe in sinnvolle Schritte unterteilt werden kann. Dazu nutzt der Agent ein Planungsmodul, das Aufgaben in Teilaufgaben zerlegt und deren Reihenfolge festlegt. Außerdem besitzt ein LLM-Agent oft eine Tool-Integration, also die Fähigkeit, aktiv externe Funktionen wie Websuche, Datenbanken, Rechenmodule oder API-Zugriffe aufzurufen und die Ergebnisse dieser Tools wiederum im weiteren Verlauf einzubeziehen. Ein weiteres Unterscheidungsmerkmal ist der Einsatz eines Gedächtnisses (Memory). Während einfache Prompt-Chains meist ohne Speicher arbeiten, kann ein Agent den Verlauf der Konversation, Zwischenresultate oder Kontextinformationen über längere Zeiträume hinweg behalten und nutzen. Das erlaubt konsistente Antworten und eine adaptive Strategie über mehrere Interaktionen hinweg. Dadurch sind LLM-Agenten besonders geeignet für komplexe, mehrstufige Aufgaben mit dynamischem Informationsbedarf und Zustandsverfolgung, während Prompt-Chains meist für klar strukturierte, wiederholbare Abläufe ohne Abweichungen konzipiert sind.

1.2.2 Zentrale Fähigkeiten von Agentensystemen

Eine zentrale Fähigkeit von LLM-Agenten ist die Tool-Nutzung.[8] Dadurch können sie aktiv mit der Außenwelt interagieren, bspw. indem sie aktuelle Informationen abrufen, Berechnungen durchführen oder APIs aufrufen. Der Agent entscheidet, ob er zur Lösung einer Aufgabe externe Funktionen benötigt, formuliert dann strukturierte Funktionsaufrufe (z. B. JSON) und verarbeitet die Ergebnisse, um weitere Schritte abzuleiten. So wird aus einem reinen Textgenerator ein handlungsfähiger Assistent, der auch komplexe, mehrstufige Aufgaben mit externer Datenverarbeitung bewältigen kann.

Eine weitere zentrale Fähigkeit ist die Orchestrierung.[9] Sie ermöglicht es LLM-Agenten, mehrstufige Aufgaben zu planen, zu koordinieren und effizient auszuführen. Da LLMs allein Schwierigkeiten haben, komplexe Abläufe zu steuern oder kontextübergreifend zu agieren, helfen Orchestrierungs-Frameworks dabei, Prompting, Datenzugriff, API-Interaktionen und Zustandsmanagement miteinander zu verbinden. Dadurch lassen sich auch Aufgaben bewälti-

[8] Vgl. hierzu und im Folgenden TrueFoundry 2025
[9] Vgl. hierzu und im Folgenden Winland, Noble 2024

gen, die mehrere Schritte, Datenquellen oder sogar mehrere Modelle erfordern. Solche Frameworks bilden die Grundlage für robuste und skalierbare Anwendungen wie Chatbots, Entscheidungsunterstützung oder automatisierte Workflows in generativer KI.

Das Routing ist ebenfalls eine wichtige Fähigkeit von LLM-Agenten.[10] Darunter wird die gezielte Weiterleitung von Nutzeranfragen an spezialisierte Subagenten verstanden. Diese sind jeweils auf bestimmte Aufgabenbereiche wie Support oder Terminplanung ausgelegt. Statt vordefinierte Absichten oder aufwändige Trainingsdaten zu benötigen, wird der Kontext der Anfrage dynamisch analysiert und in Echtzeit entschieden, welcher Agent sie am besten bearbeiten kann. Dadurch entsteht ein System, das alle Anfragen ohne Umwege oder Verzögerungen direkt an die richtige Stelle leitet.

Wichtige Stichpunkte für die strukturierte Bearbeitung komplexer Aufgaben und gezielte Reaktion auf den Nutzerkontext sind Planung und Speicher.[11] Das Planungsmodul hilft dem Agenten, eine Benutzeranfrage in mehrere Teilaufgaben zu zerlegen und die erforderlichen Schritte entweder im Voraus oder dynamisch mit Feedback festzulegen. Gleichzeitig speichert das Speichermodul frühere Gedanken, Aktionen und Beobachtungen entweder im Kurzzeitspeicher (innerhalb des aktuellen Kontexts) oder im Langzeitspeicher über externe Vektorspeicher. Durch die Kombination dieser Komponenten kann ein Agent nicht nur konsistent auf frühere Interaktionen reagieren, sondern auch strategisch planen und sein Verhalten anpassen, um komplexe Nutzeranfragen effektiv und nachvollziehbar zu beantworten.

1.2.3 Typische Frameworks

Typische Frameworks für LLM-Agenten unterstützen Programmierer bei der Entwicklung. Solche Frameworks übernehmen zum Beispiel die Orchestrierung von Modulen wie Planung, Tool-Nutzung, Gedächtnis und Kommunikation zwischen mehreren Agenten.[12] Bekannte Beispiele sind LangChain Agents, AutoGen und CrewAI. Sie bieten Funktionen zur Aufgabenzerlegung, Rollenvergabe, Workflow-Steuerung und Tool-Integration. Die Wahl des passenden Frameworks hängt dabei stark von Faktoren wie Flexibilität, Nutzerfreundlichkeit und Integrationsfähigkeit ab. Weitere wichtige Auswahlkriterien sind unterstützte Modelle, Erweiterbarkeit, Skalierbarkeit, Sicherheitsanforderungen und Kosten.

[10] Vgl. hierzu und im Folgenden Kargwal 2025a
[11] Vgl. hierzu und im Folgenden Prompt Engineering Guide 2025
[12] Vgl hierzu und im Folgenden Kargwal 2025b

1.2.4 Vergleich LangChain vs. AutoGen

LangChain und AutoGen sind zwei beliebte kostenfreie Open-Source-Frameworks, verfolgen jedoch unterschiedliche Ansätze hinsichtlich Architektur, Modularität und Bedienbarkeit.[13] LangChain zeichnet sich durch eine stark modulare Architektur aus, während AutoGen auf eine workfloworientierte Architektur setzt. LangChain bietet Entwicklern detaillierte Kontrolle über alle Komponenten eines Agentensystems, von der Datenvorverarbeitung über das Retrieval bis zur Tool-Nutzung. Es eignet es sich besonders gut für Use Cases, die die hohe Anpassungstiefe benötigen. Im Gegensatz dazu setzt AutoGen mit seiner workflow-orientiertem Architektur darauf, die Komplexität beim Erstellen von LLM-Pipelines zu minimieren. Es ermöglicht mit Low-Code-Konfiguration das schnelle Aufsetzen von mehrstufigen Aufgaben. Typische Use Cases sind z. B. Textklassifikation, Zusammenfassungen oder einfache Multi-Agenten-Kommunikation. Während LangChain sich vor allem an technisch versierte Entwickler richtet, die bereit sind, sich in eine umfangreiche Dokumentation einzuarbeiten, ist AutoGen mit niedrigen Einstiegshürde, vorkonfigurierten Templates und einer flacheren Lernkurve einsteigerfreundlicher. In Bezug auf technische Einstiegshürden ist LangChain durch seine hohe Konfigurierbarkeit mächtiger, aber auch komplexer in der Einarbeitung. AutoGen hingegen ermöglicht einen deutlich schnelleren Start für einfache bis mittelkomplexe Workflows und eignet sich gut für Teams mit weniger Programmiererfahrung.

[13] Vgl hierzu und im Folgenden Kargwal 2025b; Vgl. ebenfalls LangChain o. J.; Vgl. ebenfalls Tutorials By David 2025

Literaturverzeichnis

Barkai, Atai 2023. *RAG vs. context window in GPT-4*. https://webflow.copilotkit.ai/blog/rag-vs-context-window-in-gpt-4?.com (Zugriff vom 31.07.2025).

Eruysaliz, Ismail 2024. *Top 7 Herausforderungen bei der Retrieval-Augmented Generation.* https://www.valprovia.com/de/blog/top-7-herausforderungen-bei-der-retrieval-augmented-generation (Zugriff vom 31.07.2025).

Kargwal, Aryan 2025a. *Ultimate Guide to AI Agent Routing (2025)*. https://botpress.com/blog/ai-agent-routing (Zugriff vom 31.07.2025).

Kargwal, Aryan 2025b. *Choosing the Right LLM Agent Framework in 2025*. https://botpress.com/blog/llm-agent-framework (Zugriff vom 31.07.2025).

Kronsbein, Janera 2025. *What architecture is behind secure and reliable RAG systems.* https://www.x1f.one/en/blogbeitrag/what-architecture-is-behind-secure-and-reliable-rag-systems/ (Zugriff vom 30.07.2025).

LangChain o. J.. *Architecture*. https://python.langchain.com/docs/concepts/architecture/ (Zugriff vom 31.07.2025).

Lewis, Patrick; Perez, Ethan; Piktus, Aleksandra; Petroni, Fabio; Karpukhin, Vladimir; Goyal, Naman; Küttler, Heinrich; Lewis, Mike; Yih, Wen-tau; Rocktäschel, Tim; Riedel, Sebastian; Kiela, Douwe 2021. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* arXiv.org: o. Verl..

Prompt Engineering Guide 2025. *LLM Agents*. https://www.promptingguide.ai/research/llm-agents (Zugriff vom 31.07.2025).

Schwaber-Cohen, Roie 2023. *What is a Vector Database & How Does it Work? Use Cases + Examples.* https://www.pinecone.io/learn/vector-database/ (Zugriff vom 30.07.2025).

Simpson, Kody 2025. *Retrieval Augmented Generation (RAG) with Langchain: A Complete Tutorial*. https://www.youtube.com/watch?v=YLPNA1j7kmQ (Zugriff vom 31.07.2025).

TrueFoundry 2025. *What are LLM Agents?*. https://www.truefoundry.com/blog/llm-agents?.com (Zugriff vom 31.07.2025).

Tutorials By David 2025. *New! CrewAI vs Autogen vs LangGraph vs Langchain – 2025 Comparison.* https://www.youtube.com/watch?v=2bXU0jJK444 (Zugriff vom 31.07.2025).

Wikipedia o. J.. *Retrieval-Augmented Generation.* https://de.wikipedia.org/wiki/Retrieval-Augmented_Generation *(Zugriff vom 31.07.2025)*.

Winland, Vanna; Noble, Joshua 2024. What is LLM Orchestration?. https://www.ibm.com/think/topics/llm-orchestration (Zugriff vom 31.07.2025).

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Anwendungsaspekte des Machine Learning* selbstständig verfasst habe.

Uhingen, 11.08.2025

(Ort, Datum) (Unterschrift)