

# Retrieval Augmented Generation (RAG) und Agentensysteme

Theoretische Recherche und Einordnung

vorgelegt am 31. August 2025

Fakultät Wirtschaft und Gesundheit

Studiengang Wirtschaftsinformatik

Kurs WWI2022A

von

Mia Holzkamp

DHBW Stuttgart:

Ralf Höchenberger

**Inhaltsverzeichnis**

Abkürzungsverzeichnis .....III

Abbildungsverzeichnis..... IV

1 RAG-Systeme .....1

    1.1 Architektur .....1

    1.2 Generative Modelle und QA-Systeme .....1

    1.3 Embeddings und Vektordatenbanken.....2

    1.4 Prompting und Kontextkonstruktion .....2

    1.5 Modellwahl und Kostenaspekte.....2

    1.6 Herausforderungen bei der Umsetzung.....3

    1.7 Implementierung – Dokumentation.....3

2 Agentensysteme .....4

    2.1 Zentrale Fähigkeiten .....5

    2.2 Frameworks .....6

Literaturverzeichnis .....8

**Abkürzungsverzeichnis**

KI	=	Künstliche Intelligenz
LLM	=	Large Language Model
MAS	=	Multi-Agenten-System
QA	=	Question Answering
RAG	=	Retrieval-Augmented Generation

**Abbildungsverzeichnis**

Abbildung 1: RAG-Systemarchitektur .....	1
Abbildung 2: Architektur Implementation RAG.....	3
Abbildung 3: Visualisierung LLM-Agent .....	4

## 1 RAG-Systeme

Retrieval-Augmented Generation (RAG) beschreibt ein System, das Textgenerierung eines Large Language Models (LLM) mit einem speziellen und relevanten Informationskontext in Form von externen Quellen kombiniert. Somit kann die Qualität und Verlässlichkeit der Antwort gesteigert werden.<sup>1</sup>

### 1.1 Architektur

Ein typisches RAG folgt grundsätzlich demselben Aufbau. Wie der Name schon verdeutlicht, kann das RAG in drei Teile unterteilt werden. Der erste Teil beschreibt den Abruf der speziellen Informationen, die den Kontext für den Abruf darstellen (Retrieve Information). Anschließend wird das Sprachmodell, also das LLM, um den Kontext angereichert (Augment Data). Der letzte Teil umfasst die Generierung der Antwort, welche sowohl auf dem gelernten Wissen des LLMs als auch auf den Informationen des Kontextes basiert. Das Vorgehen innerhalb der RAG-Struktur ist in Abbildung 1 grafisch dargestellt:

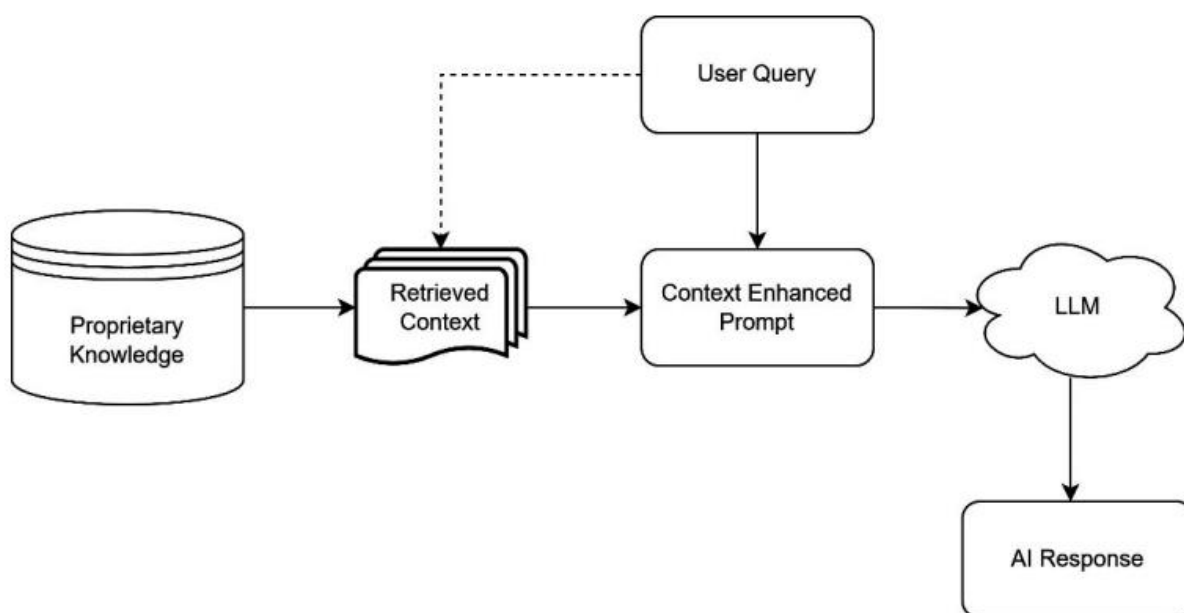


Abbildung 1: RAG-Systemarchitektur<sup>2</sup>

### 1.2 Generative Modelle und QA-Systeme

Ein RAG-System komponiert die Fähigkeiten eines rein generativen Modells und die eines klassischen QA-Systems (Question Answering). Rein generative Modelle generieren Texte auf Basis von vortrainiertem Wissen, ohne auf externe Datenquellen zuzugreifen. Klassische QA-Systeme geben präzise Antworten aus vordefinierten externen Datenbanken und Dokumenten

<sup>1</sup> Vgl. Galla et al. 2024, S. 392.

<sup>2</sup> Mit Änderungen übernommen aus: Gheorghiu 2024, S. 11.

aus. Ein RAG-System greift wie das QA-System auf die Informationen von externen Quellen zu und kann diese durch das LLM in generierter Textform ausgeben.<sup>3</sup>

### 1.3 Embeddings und Vektordatenbanken

Entscheidend für die Einbindung von externen Wissensquellen ist die Verarbeitung der Quelle, sodass sie als Kontext genutzt werden kann. Dafür werden die Inhalte der Quelle in kleinere Teile (Chunks) unterteilt. Anschließend werden die Chunks in eine mathematische Repräsentation überführt, die sogenannten Embeddings. Die Embeddings stellen die Bedeutung der Chunks in einem bestimmten Kontext und die Verbindung zu anderen Chunks dar. Die Embeddings der externen Quelle werden in einer Vektorendatenbank gespeichert.<sup>4</sup>

### 1.4 Prompting und Kontextkonstruktion

Bei einer User-Anfrage (Prompt) wird diese ebenfalls in eine Vektordarstellung konvertiert, sodass sie mit der Vektorendatenbank, die die Embeddings beinhaltet, abgeglichen werden kann (Retriever). Der Kontext aus der externen Quelle wird nun gemeinsam mit dem Prompt zum LLM weitergegeben, sodass die Anfrage beantwortet werden kann. Auf diese Art und Weise muss nicht das komplette LLM auf die neuen Informationen aktualisiert werden, sondern wird flexibel um die neuen Daten ergänzt.<sup>5</sup>

### 1.5 Modellwahl und Kostenaspekte

Mittlerweile gibt es viele verschiedene Open-Source-RAG-Frameworks, die Entwickler\*innen zur Verfügung stehen. Die Frameworks bieten die Grundlage, die die einzelnen Komponenten des RAG-Systems miteinander verbindet.<sup>6</sup>

Bei der Wahl eines RAG-Systems müssen verschiedene Faktoren berücksichtigt werden, darunter die Genauigkeit der Informationsrückgewinnung (Retrieval) und die Qualität der Textgenerierung (Generation). Ein einfacheres Modell könnte auf kleinere Datenquellen zugreifen, während komplexere Modelle auf große Datenbanken oder dynamische Webseiten zugreifen und so flexiblere, kontextualisierte Antworten generieren können. Die Modelwahl hängt auch davon ab, wie schnell und in welchem Umfang Informationen abgerufen werden sollen.

RAG-Modelle sind in der Regel kostenintensiver als rein generative Modelle, da sie zusätzliche Komponenten wie die Retrieval-Komponente, um Informationen abzurufen, und zugängliche

---

<sup>3</sup> Vgl. Lewis et al. 2020, 4f.

<sup>4</sup> Vgl. Microsoft 2025b.

<sup>5</sup> Vgl. AWS 2025.

<sup>6</sup> Vgl. Jeremiah 2025.

externe Datenquellen benötigen. Die Kosten ergeben sich aus der Implementierung und Wartung der Retrieval-Komponente sowie der Rechenleistung zur Textgenerierung und -verarbeitung.<sup>7</sup>

## 1.6 Herausforderungen bei der Umsetzung

Die Umsetzung des RAG-Systems bringt ebenfalls Herausforderungen mit sich. Die Qualität des Retrievals stellt einen wichtigen Faktor dar. Wenn relevante Dokumente nicht korrekt abgerufen oder falsch eingeordnet werden, kann das zu ungenauen oder fehlerhaften Antworten führen. Damit einhergehend, kann auch das richtige Kontextfenster eine Herausforderung darstellen. Die Integration von vielen Dokumenten in den Antwortprozess, kann zur Folge haben, dass entscheidende Informationen nicht betrachtet und somit im Kontext nicht berücksichtigt werden. Eine weitere Herausforderung wird in der Betrachtung der User-Experience sichtbar. Da eine komplexe Anfrage an ein RAG-System rechenintensiv sein kann, ist es entscheidend genug, aber aufgrund von Kosten nicht zu viel, Rechenleistung zur Verfügung zu stellen, damit es keine Verzögerungen im Abruf von Informationen und somit eine reibungslose Nutzer-Erfahrung gibt.<sup>8</sup>

## 1.7 Implementierung – Dokumentation

In Abbildung 2 ist die Architektur der Implementierung des RAG-Systems grafisch dargestellt. Zusätzlich sind detaillierte Erklärungen in Kommentaren im Code zu finden.

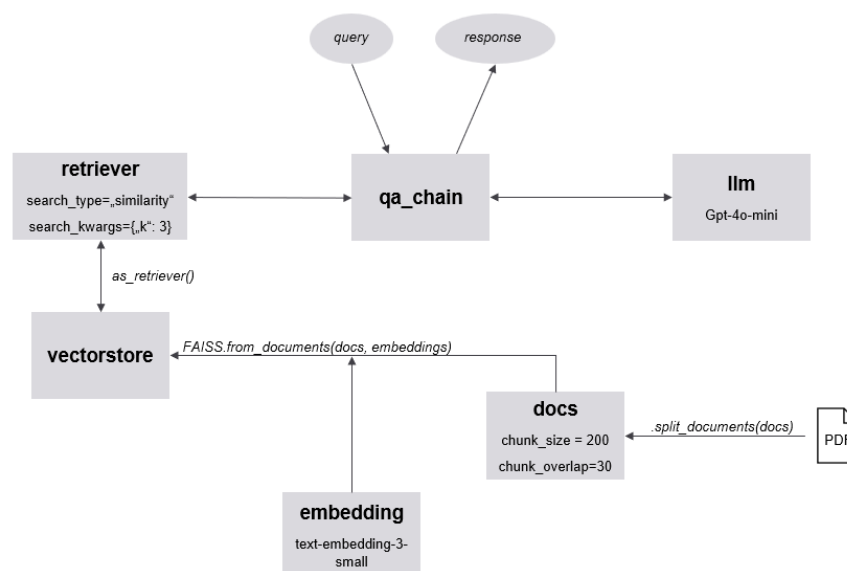


Abbildung 2: Architektur Implementation RAG

<sup>7</sup> Vgl. Honroth, Siebert, Kelbert 2024.

<sup>8</sup> Vgl. Barnett et al. 2024, S. 4.

## 2 Agentensysteme

Ein Agent beschreibt grundlegend alles, was über Sensoren die jeweilige Umwelt wahrnehmen und anschließend durch entsprechende Aktionen auf die Umwelt einwirkt.<sup>9</sup>

Intelligente Agenten im Bereich der künstlichen Intelligenz sind Systeme, die eigenständig Aufgaben erledigen, Daten verarbeiten, Entscheidungen treffen und kommunizieren aktiv mit ihrer Umgebung oder anderen Agenten, um komplexe Aufgaben zu bewältigen. Wenn ein System mehrere Agenten umfasst, spricht man von einem Multi-Agenten-System (MAS). Die Basis des Agentensystems stellt ein LLM dar, in welchem die gewohnte Textverarbeitung und -generierung stattfindet. Das System umfasst außerdem drei zusätzliche Module: Tools, Memory und Planning. In Abbildung 3 ist das System der Agenten grafisch dargestellt:

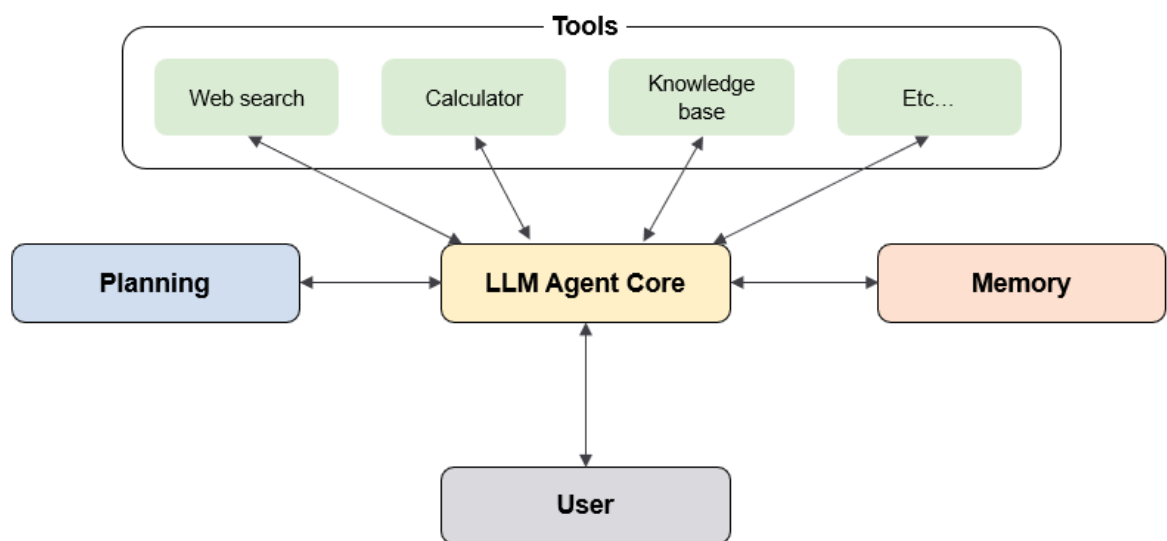


Abbildung 3: Visualisierung LLM-Agent<sup>10</sup>

Der LLM Agent Core, also das Kernmodul, nimmt die vom Nutzer gestellte Aufgabe, greift auf relevante Informationen aus der Memory zu, zerlegt die Aufgabe im Planungsmodul in kleinere, weniger komplexe Teilaufgaben und bearbeitet die Teilaufgaben mithilfe der verfügbaren Tools. Zuletzt werden die Teilergebnisse im LLM Agent Core zusammengeführt, um eine Antwort auf die ursprüngliche Aufgabe zu generieren, die dem Nutzer zurückgegeben wird.<sup>11</sup>

Im Gegensatz zu einer einfachen Prompt-Chain in einem LLM, kann ein Agentensystem komplexere Aufgaben lösen. Das System ist somit nicht auf das LLM und den dadurch trainierten Datensatz beschränkt, sondern kann auf durch die verschiedenen Module beispielsweise auf externe Quellen und vergangene Aufgaben zurückgreifen.

<sup>9</sup> Vgl. Russell, Norvig 2016, 34f.

<sup>10</sup> Mit Änderungen übernommen aus: Kamath et al. 2024, S. 432.

<sup>11</sup> Vgl. Kamath et al. 2024, S. 432.



## 2.1 Zentrale Fähigkeiten

Die zentralen Fähigkeiten eines Agentensystems ergeben sich aus den einzelnen Modulen wie in Abbildung 3 dargestellt. Die Fähigkeiten des LLMs als Kernmodul wird durch die zusätzlichen Module ergänzt.

Das Memory-Modul dient zur Sammlung und Speicherung von Informationen aus Interaktionen. Diese können bei Bedarf zur Erfüllung der aktuellen Aufgabe herangezogen werden. Meistens wird zwischen einem Kurz- und Langzeitspeicher unterschieden. Der Kurzzeitspeicher erfasst detaillierte Informationen der Interaktionen innerhalb der aktuellen Sitzung. Somit kann auch bei einer Sitzung, die mehrere Interaktionen umfasst, auf den genauen Kontext zurückgegriffen werden. Der Langzeitspeicher hingegen beinhaltet relevante Informationen aus verschiedenen vergangenen Sitzungen. Mithilfe des Langzeitspeichers können beispielsweise konsistente und personalisierte Antworten gegeben werden, da die vergangenen Interaktionen berücksichtigt werden. Zudem kann Zeit eingespart werden, da der Agent bei einer wiederkehrenden Aufgabe auf die alte Antwort zurückgreifen kann und nicht jedes Mal neu lernen muss.<sup>12</sup>

Das Planning-Modul unterteilt die Aufgabe des Users in Teilaufgaben, welche einfacher zu lösen sind, sodass die Aufgabe insgesamt sinnvoll beantwortet werden kann. Die Teilaufgaben werden anschließend einzeln gelöst. Beispiele für genutzte Techniken sind Chain of Thought (Single-Path Reasoning) oder Tree of Thoughts (Multi-Path Reasoning), welche die einzelnen Zwischenergebnisse ebenfalls ausgeben. Die Methoden nutzen initiale Pläne der Aufgaben. Da die erste Planung meistens nicht optimal ist oder sogar gar nicht zum Ergebnis führt, wird für die langfristige Planung von komplexen Problemen iteratives Feedback gegeben. Dadurch soll die Planung und die Ausführung verbessert werden. Das Feedback kann von Signalen der Umwelt oder direkt von Menschen gegeben werden. Das menschliche subjektive Feedback hat den Vorteil, dass das Ergebnis an menschliche Werte und Präferenzen angepasst wird. Neben den externen Einflussfaktoren kann durch vortrainierte Modelle auch internes Feedback gegeben werden.<sup>13</sup>

Tools werden genutzt, um die Fähigkeiten eines Agenten über die reine Textverarbeitung hinaus zu erweitern. Das LLM ist alleinstehend auf die trainierten Daten und Sprachgenerierung beschränkt, während die Einbindung von Tools die Integration von Umgebungen und externen Informationen ermöglichen, um komplexe Aufgaben auszuführen. Ein Tool kann beispielsweise eine RAG Pipeline nutzen, um spezielle Informationen aus einer externen Quelle abzufragen.<sup>14</sup>

---

<sup>12</sup> Vgl. Prompt Engineering Guide 2025.

<sup>13</sup> Vgl. Wang et al. 2024, 10ff.

<sup>14</sup> Vgl. Varshney 2023.

Da in Agentensystemen verschiedene Agenten der unterschiedlichen Module zusammenwirken, muss die Zusammenarbeit auch koordiniert werden. Das sogenannte Agent Routing beschreibt den Prozess, dass für eine spezifische Teilaufgabe der korrekte Agent ausgewählt wird, um das beste und effektivste Ergebnis zu erzielen. Somit kann sich der Routing Agent nur mit der korrekten Zuordnung befassen, während sich die anderen Agenten auf ihre individuellen Aufgaben konzentrieren.<sup>15</sup>

Bei komplexeren Aufgaben, die die Zusammenarbeit von mehreren Agenten benötigen, nutzt man die sogenannte Orchestrierung. Die Interaktionen der autonom agierenden Agenten wird dabei von dem Orchestrator verwaltet und koordiniert. Der Orchestrator sucht im Gegensatz zum Routing Agent nicht nur den richtigen Agenten aus, sondern koordiniert auch den richtigen Zeitpunkt für eine effiziente Synchronisation zwischen den Agenten. In der Praxis gibt es verschiedene Arten von Agenten Orchestrierung. Sie kann beispielsweise zentral von einem Agent ausgeführt werden oder auch dezentral, indem die Agenten direkt miteinander kommunizieren und zusammenarbeiten.<sup>16</sup>

## 2.2 Frameworks

Zur praktischen Umsetzung von Agentensystemen können Frameworks verwendet werden. Zwei vielverwendete Frameworks für Agentensysteme sind *LangChain Agents* von *LangChain* und *AutoGen* von *Microsoft*.

Im direkten Vergleich der beiden Frameworks zeigt sich, dass beide grundlegend dasselbe Ziel verfolgen. LLMs werden als zentrale Entscheidungsinstanz in agentischen Systemen nutzbar gemacht. *LangChain* baut auf einem modularen Planner-zu-Executor-Prinzip auf. Das Framework stellt Agenten, Tool-Adapter und verschiedene Plan-zu-Execute-Patterns bereit und legt dabei ein starkes Gewicht auf flexible Prompt-Architekturen und Tool-Integration.<sup>17</sup> *AutoGen* hingegen ist eher als Multi-Agenten-Framework konzipiert, das standardisierte Patterns für Agent-zu-Agent-Kommunikation und asynchrone Workflows bietet.<sup>18</sup>

*LangChain* ist sehr komponentenbasiert, was eine schnelle Austauschbarkeit der Komponenten ermöglicht. Dadurch ist das Framework in seiner Modularität flexibler, aber die Entwickler\*innen müssen die einzelnen Module auch selbst zusammenbauen.<sup>19</sup> Im Gegensatz dazu bietet *AutoGen* eher vorgefertigte Patterns, die gesamten Aufbau wie die Orchestrierung und

---

<sup>15</sup> Vgl. Patronus AI 2025.

<sup>16</sup> Vgl. Finio, Downie 2025.

<sup>17</sup> Vgl. Langchain 2024.

<sup>18</sup> Vgl. Microsoft 2025a.

<sup>19</sup> Vgl. LangGraph 2025.

die Konversation. Es ist somit einfacher komplexe Multi-Agenten-Flows zu bauen, aber für die Anwender\*innen weniger flexibel in den einzelnen Modulen.<sup>20</sup>

Grundsätzlich können die Frameworks als Äquivalente zueinander gesehen werden. Nichtsdestotrotz unterscheiden sich die typischen Use Cases der Frameworks. *LangChain* eignet sich aufgrund der flexiblen Modularität sehr gut für die Einbindung von RAG-Systemen in Agenten und für Agentensysteme, die viele Tools umfassen während *AutoGen* durch die vorgefertigten Patterns vorwiegend für interaktive Multi-Agenten-Workflows und die Zusammenarbeit von spezialisierten Agenten eingesetzt wird. Die technischen Einstiegshürden sind bei beiden Frameworks insgesamt gering, da sie als Open-Source-Frameworks zugänglich sind. Da *LangChain* aus zahlreichen Modulen besteht, die richtig kombiniert werden müssen, scheint der Einstieg eher komplex. Sobald das Ökosystem aus Modulen verstanden ist, lässt sich das Framework mit seiner hohen Flexibilität relativ einfach einsetzen.<sup>21</sup> *AutoGen* lässt sich dagegen leichter auf Anhieb verstehen, da das Framework direkte Patterns vorsieht, welche wiederum die Flexibilität erschweren.<sup>22</sup>

---

<sup>20</sup> Vgl. Microsoft 2025a.

<sup>21</sup> Vgl. LangGraph 2025.

<sup>22</sup> Vgl. Microsoft 2025a.

## Literaturverzeichnis

- AWS 2025. *Was ist Retrieval-Augmented Generation (RAG)?* <https://aws.amazon.com/de/what-is/retrieval-augmented-generation/> (Zugriff vom 28.08.2025).
- Barnett, Scott et al. 2024. *Seven Failure Points When Engineering a Retrieval Augmented Generation System*: arXiv.
- Finio, Matthew; Downie, Amanda 2025. *Was ist die Orchestrierung von KI-Agenten?* <https://www.ibm.com/de-de/think/topics/ai-agent-orchestration> (Zugriff vom 28.08.2025).
- Galla, Divyanshi et al. 2024. „CoURAGE: A Framework to Evaluate RAG Systems“, in *Natural Language Processing and Information Systems*, hrsg. v. Rapp, Amon et al., S. 392-407. Cham: Springer Nature Switzerland.
- Gheorghiu, Andrei 2024. *Building data-driven applications with LlamaIndex. A practical guide to retrieval-augmented generation (RAG) to enhance LLM applications*. Birmingham: packt.
- Honroth, Thorsten; Siebert, Julien; Kelbert, Patricia 2024. *Retrieval Augmented Generation (RAG). Chatten mit den eigenen Daten*. <https://www.iese.fraunhofer.de/blog/retrieval-augmented-generation-rag/> (Zugriff vom 28.08.2025).
- Jeremiah, Oluseye 2025. *RAG-Rahmenwerke, die du kennen solltest: Open-Source-Tools für intelligentere KI*. <https://www.datacamp.com/de/blog/rag-framework?> (Zugriff vom 28.08.2025).
- Kamath, Uday. et al. 2024. *Large Language Models: A Deep Dive. Bridging Theory and Practice*. Cham: Springer Nature Switzerland; Imprint: Springer.
- Langchain 2024. *Plan-and-Execute Agents*. <https://blog.langchain.com/planning-agents/> (Zugriff vom 28.08.2025).
- LangGraph 2025. *Agent architecture*. [https://langchain-ai.github.io/langgraph/concepts/agent\\_concepts/](https://langchain-ai.github.io/langgraph/concepts/agent_concepts/) (Zugriff vom 28.08.2025).
- Lewis, Patrick et al. 2020. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*: arXiv.
- Microsoft 2025a. *AutoGen. Getting Started*. <https://microsoft.github.io/autogen/0.2/docs/Getting-Started/> (Zugriff vom 28.08.2025).
- Microsoft 2025b. *RAG generate embeddings phase*. <https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings> (Zugriff vom 28.08.2025).

- Patronus AI 2025. *AI Routing. Tutorial & Best Practices*. <https://www.patronus.ai/ai-agent-development/ai-agent-routing> (Zugriff vom 28.08.2025).
- Prompt Engineering Guide 2025. *LLM Research Findings. LLM Agents*. <https://www.promptingguide.ai/research/llm-agents> (Zugriff vom 28.08.2025).
- Russell, Stuart; Norvig, Peter 2016. *Artificial intelligence. A modern approach*. Boston: Pearson.
- Varshney, Tanay 2023. *Introduction to LLM Agents*. <https://developer.nvidia.com/blog/introduction-to-llm-agents/> (Zugriff vom 28.08.2025).
- Wang, Lei et al. 2024. „A survey on large language model based autonomous agents“, in *Frontiers of Computer Science* 18, 6.

## Erklärung zur Verwendung generativer KI-Systeme

Bei der Erstellung der eingereichten Arbeit habe ich auf künstlicher Intelligenz (KI) basierte Systeme benutzt:

- ☒ ja  
☐ nein<sup>23</sup>

Falls ja: Die nachfolgend aufgeführten auf künstlicher Intelligenz (KI) basierten Systeme habe ich bei der Erstellung der eingereichten Arbeit benutzt:

1. ChatGPT
2. Microsoft Copilot

Ich erkläre, dass ich


- mich aktiv über die Leistungsfähigkeit und Beschränkungen der oben genannten KI-Systeme informiert habe,<sup>24</sup>
- die aus den oben angegebenen KI-Systemen direkt oder sinngemäß übernommenen Passagen gekennzeichnet habe,<sup>25</sup>
- überprüft habe, dass die mithilfe der oben genannten KI-Systeme generierten und von mir übernommenen Inhalte faktisch richtig sind,
- mir bewusst bin, dass ich als Autorin bzw. Autor dieser Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.

Die oben genannten KI-Systeme habe ich wie im Folgenden dargestellt eingesetzt:

Arbeitsschritt in der wissenschaftlichen Arbeit <sup>26</sup>	Eingesetzte(s) KI- System(e)	Beschreibung der Verwendungsweise
Textverfassung	ChatGPT, Microsoft Copilot	Text mit sprachlich geglättet.
Generierung des Programmcodes	ChatGPT	Programmcode optimiert und Fehler ausgearbeitet

28.08.2025, Stuttgart

(Ort, Datum)

  
 (Unterschrift)

<sup>23</sup> Die Erklärung ist in jedem Fall zu unterzeichnen, auch wenn Sie keine KI-Systeme genutzt haben und Ihr Kreuz bei „nein“ gesetzt haben.

<sup>24</sup> U.a. gilt es hierbei zu beachten, dass an KI weitergegebene Inhalte ggf. als Trainingsdaten genutzt und wiederverwendet werden. Dies ist insb. für betriebliche Aspekte als kritisch einzustufen.

<sup>25</sup> In der Fußnote Ihrer Arbeit geben Sie die KI als Quelle an, z.B.: Erzeugt durch Microsoft Copilot am dd.mm.yyyy. Oder: Entnommen aus einem Dialog mit Perplexity vom dd.mm.yyyy. Oder: Absatz 2.3 wurde durch ChatGPT sprachlich geglättet.

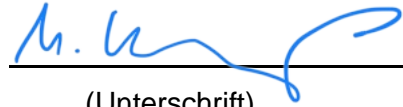
<sup>26</sup> Beispiele hierfür sind u.a. die folgenden Arbeitsschritte: Generierung von Ideen, Konzeption der Arbeit, Literatursuche, Literaturanalyse, Literaturverwaltung, Auswahl von Methoden, Datensammlung, Datenanalyse, Generierung von Programmcodes

**Erklärung**

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Retrieval Augmented Generation (RAG) und Agentensysteme* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

28.08.2025, Stuttgart

(Ort, Datum)

A handwritten signature in blue ink, consisting of a stylized 'M.' followed by a series of loops and a final flourish.

(Unterschrift)