Retrieval-Augmented Generation für den Mittelstand
KI-Innovationswettbewerb – Generative KI für den Mittelstand

Projekt ID: 01MK250104 **Projektstart:** 01.02.2025 **Laufzeit:** 36 Monate

Ergebnis 3.1: Implementierung einer RAG-Pipeline für die Basisversion

Publikationslevel	Öffentlich
Zieldatum	Monat 9, 31.10.2025
Abschlussdatum	Monat B, DD.MM.YYYY
Arbeitspaket	AP3 – RAG Pipeline
Ergebnis	E3.1
Typ	Report
Status	DRAFT Final
Version	0.1

Kurzzusammenfassung: In diesem Deliverable wird die Implementierung einer RAG-Pipeline für die Basisversion der Learn2RAG Plattform beschrieben.

Gefördert durch:

Bundesministerium
für Forschung, Technologie
und Raumfahrt

Dieses Projekt wurde finanziert vom Bundesministerium für Wirtschaft und Klimaschutz (01MK250104).

History

Version	Datum	Änderung	Author
0.0	29.08.2025	Struktur erstellt	Michael Röder
0.1	29.09.2025	Struktur überarbeitet	Carolin Walter
0.2	03.10.2025	Inhalte erstellt	Carolin Walter
0.3	20.10.2025	Review	Alex Rudolph
0.4			
1.0	DD.MM.YYYY	Finale Version abgeschlossen und veröffentlicht	

Zusammenfassung

In diesem Deliverable wird die Implementierung der Basisversion für die RAG-Pipeline beschrieben. Eine Analyse der Anforderungen liefert erste Grundvoraussetzungen für die Implementierung. Die RAG-Pipeline wird mit Hilfe einer Systemübersicht im Projektkontext dargestellt. Anschließend wird die Wahl der Vector Store getroffen. Es folgte eine Beschreibung der Schritte der Indexierung und des Retrievals. Danach wird auf die Anbindung und das Prompttemplate des Sprachmodells eingegangen und schlussendlich folgt die erste Evaluation der RAG-Pipeline.

List of Abbreviations

AP	Arbeitspaket
LLM	Large Language Model
RAG	Retrieval Augmented Generation
UPB	University of Paderborn

Table of Contents

Zusammenfassung 3
1. Einleitung 6
2. Analyse der Anforderungen 6
3. Systemübersicht 7
4. Vector Store 8
5. Indexierung 9
6. Retrieval 9
7. Sprachmodell 10
8. Evaluation 11

1. Einleitung

Mit der Basisversion wurde eine erste lauffähige Anwendung für die Learn2RAG Plattform geschaffen. Diese dient als Basis für die kommenden Weiterentwicklungen hin zur selbstadaptierenden RAG-Pipeline und als erste Demo für die Workshops, sodass interessierte KMUs sehr früh im Projekt einen ersten Prototypen ausprobieren können.

Für die Entwicklung wurden die in AP1 erhobenen Anforderungen und die in AP2 entwickelten Schnittstellen genutzt. AP4 liefert das User Konfigurationsfile, während das hier beschriebene AP3 das Backend für die in AP4 entwickelte Plattformlösung darstellt.

Für die Basisversion der RAG-Pipeline wurde der State-of-the-Art recherchiert, es wurden Entscheidungen über Komponenten getroffen und eine pragmatische Lösung implementiert. Insbesondere in Bezug auf Parameter wurden Vorerfahrungen genutzt, da diese in den folgenden Pipeline Versionen ohnehin automatisiert bestimmt werden sollen.

2. Analyse der Anforderungen

In Deliverable E1.1 wurden die Anforderungen an die Basisversion erhoben. Hier werden die im Hinblick auf AP3 relevanten Anforderungen analysiert und Implikationen abgeleitet.

GESCHÄFTLICHE UND STRATEGISCHE ANFORDERUNGEN
Vendor-Lock-in vermeiden (MUST):
Es werden nur Open Source Lösungen für die Komponenten in Betracht gezogen. Der Vector Store wird selbst gehosted. Ebenso werden das Embeddingmodell und das LLM lokal betrieben.
Wachstumsfähigkeit und skalierbares Datenhandling (COULD):
Wir haben die Komponenten so gewählt, dass eine Skalierung möglich ist. Anforderungen an Laufzeit und Kosten, werden im Laufe des Projekts behandelt, sobald konkretere Anforderungen vorliegen.

FUNKTIONALE ANFORDERUNGEN
Die Basisversion muss eine RAG-Pipeline-basierte Lösung implementieren (MUST):
Es wurde eine RAG-Pipeline implementiert.
Nachverfolgbarkeit (COULD):
Die Basisversion kann bereits eine Nachverfolgbarkeit unterstützen, indem das LLM die Herkunft der für die Antwortgenerierung genutzten Informationen zurückgibt.
Modell-Flexibilität (COULD):
Umschaltbar zwischen lokalen Open-Weight-Modellen und externen API-Modellen, kann in Zukunft umgesetzt werden, wenn auch die Parameter auf das jeweilige Modell automatisiert werden.
Mehrsprachige Suche und Antwortgenerierung (SHOULD):
Es sollen mindestens Deutsch und Englisch umgesetzt werden. Dies ist in der Implementierung für die Basisversion der Fall. Das LLM antwortet in der Sprache der Fragenstellung, soweit das Sprachmodell mit dieser Sprache trainiert wurde.
Kontextspezifisches Retrieval (COULD):
Die bisherigen Architekturentscheidungen wurden so getroffen, dass ein Filtern generell möglich ist.

QUALITÄT, WARTBARKEIT UND ERWEITERBARKEIT
Modularer Aufbau (COULD):
Der Quellcode wurde modular aufgebaut. Es existieren REST-Endpunkte für Retrieval und Generation.

Konfiguration-als-Code (COULD):
Es gibt zum einen die `user_config`, in der die Pipelinekonfigurationen des Anwendenden gespeichert werden und zum anderen die `opt_config`, in der die Parameter der Pipeline gesetzt werden. Beide sind als JSON strukturiert.
Dokumentation (MUST):
Bei der Basisversion wurde die README regelmäßig aktualisiert und gepflegt. Als Dokumentation der Basisversion der Pipeline kann dieses Deliverable herangezogen werden.

3. Systemübersicht

In Abbildung 1 ist ein Überblick über die Learn2RAG Plattform gegeben.
Ein User nutzt die in AP4 entwickelte UI, um die Konfiguration der Pipeline vorzunehmen, welche in einem JSON-File gespeichert wird. Außerdem kann der User ein LLM auswählen und starten, welches dann dem RAG-Service aus AP3 zur Verfügung steht. Durch die konfigurierten Pfade werden die entsprechenden Datenquellen angebunden, hierfür werden die Schnittstellen und die Importer aus AP2 genutzt. Sobald alle Dokumente in der Vector Store indiziert sind, steht der RAG-Service dem User bereit und kann über die Demo UI angesprochen werden.

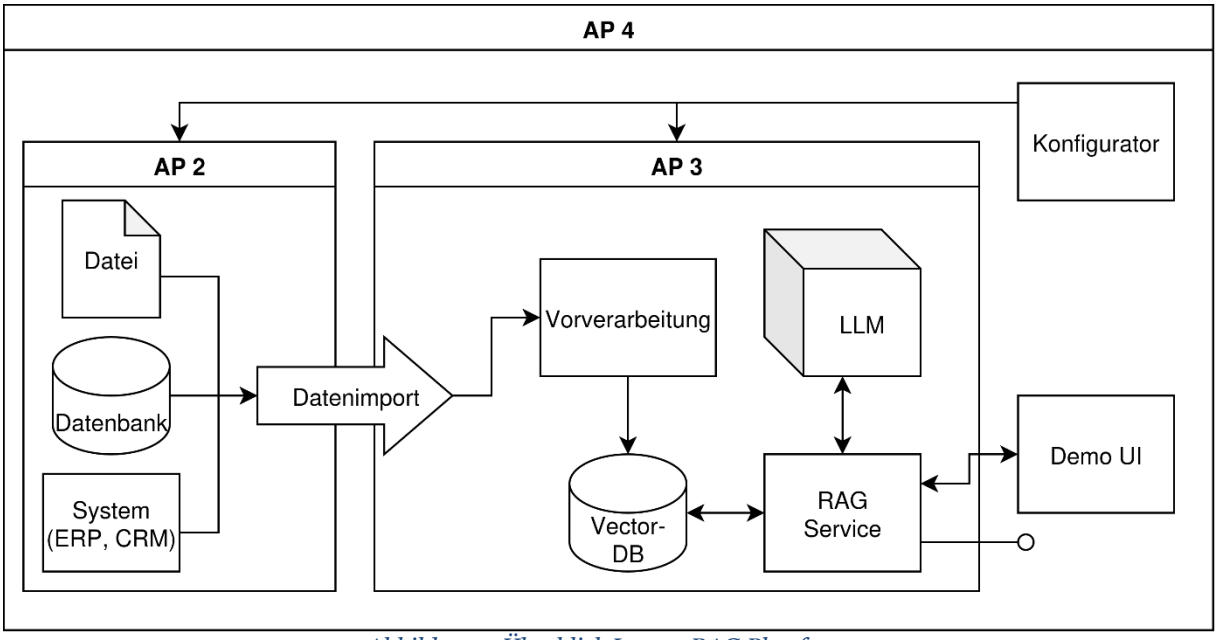

Abbildung 1 Überblick Learn2RAG Plattform

Die RAG-Pipeline der Basisversion ist in Abbildung 2 detailliert dargestellt. Die aufbereiteten Dokumente werden durch ein Embeddingmodell in Vektoren überführt und in einer Vector Store gespeichert. Anwendende können dann ihre Fragen stellen. Dabei wird die Frage durch dasselbe Embeddingmodell vektorisiert, sodass die Ähnlichkeit zwischen der Frage und den Dokumenten berechnet werden kann. Die Dokumente der ähnlichsten Vektoren werden dann gemeinsam mit der Frage in einen Prompt gegeben und an ein LLM geschickt. Dieses generiert auf Basis der mitgelieferten Dokumente eine Antwort auf die Frage.

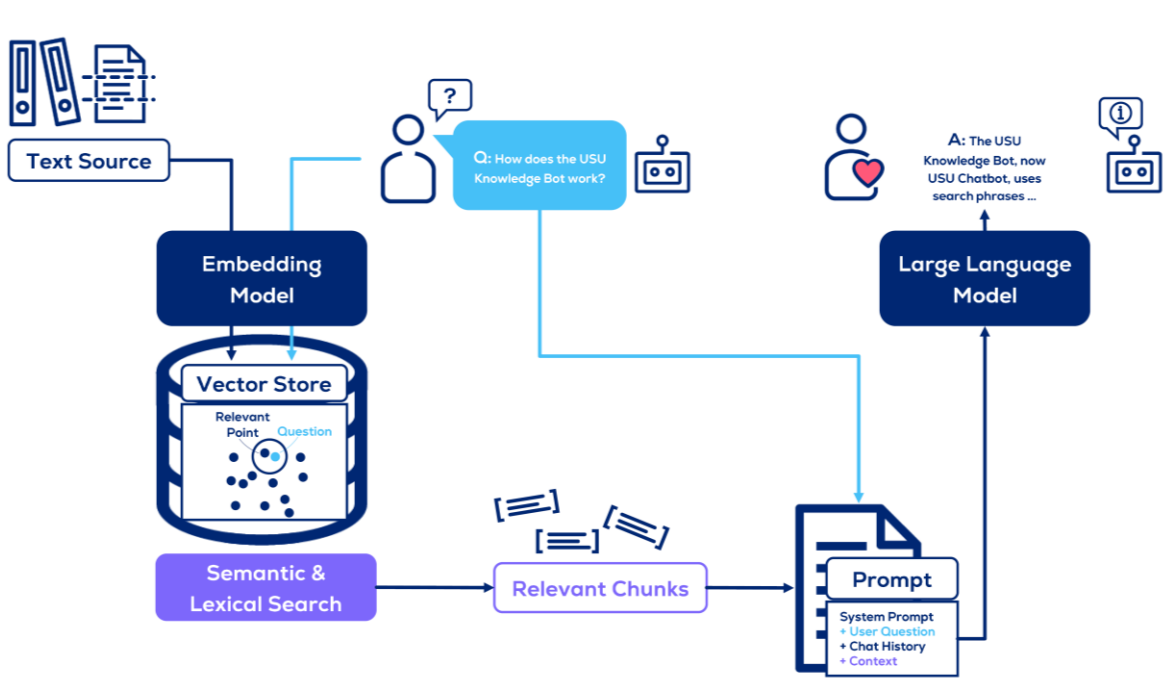

Abbildung 2 Überblick RAG-Pipeline der Basisversion

4. Vector Store

Die Vector Store stellt im Rahmen von RAG eine zentrale Komponente dar. Sie dient dazu, unstrukturierte Inhalte in Form von Vektorrepräsentationen zu speichern und bei einer Anwendendenanfrage relevante Inhalte effizient zu finden.

Dabei sollte die Vector Store, insbesondere bei einer großen Anzahl an Vektoren, eine performante semantische Suche bieten. Die semantische Suche sollte durch zusätzliche lexikale Vektoren eine hybride Suche bieten, da diese häufig noch bessere Suchtreffer ermöglich. Die Vector Store muss mit relevanten Embeddingmodellen kompatibel sein. Die Anwendung sollte unter Windows und Linux laufen und containerisierbar sein. Außerdem soll sie skalierbar sein und Open Source. Unter https://superlinked.com/vector-db-comparison ist eine umfassende Liste an Vector Store zu finden. Dort werden auch die meisten der obengenannten Anforderungen für die jeweiligen Hersteller gelistet. Der ANN-Benchmark (https://ann-benchmarks.com/index.html) liefert einen Vergleich der Hersteller auf Basis der approximate nearest neighbour Algorithmen. Hier bekommt man einen Eindruck, wie gut die Performance ist, wie groß der benötigte Speicherplatz, die Build Time und vieles mehr.

Aus Vorerfahrungen und der Arbeiten innerhalb des Projekts waren dem Konsortium pgvector, Vespa, Redis und Qdrant bekannt. Schlussendlich fiel die Wahl auf Qdrant (https://qdrant.tech/) als Vector Store für die Learn2RAG-Plattform, da hier die Anforderungen am besten erfüllt wurden. Zudem hat Qdrant eine sehr aktive Community, eine gute Dokumentation und bietet viele Möglichkeiten wie die Modellierung von Metadaten mit Facets und ihrer Filterung.

5. Indexierung

Die Indexierung (`ingestion`) ist der erste Schritt in der RAG-Pipeline. Die Inputdaten werden als JSON aus AP2 übermittelt. Dabei besteht ein Dokument initial aus dem `content` und mehreren Feldern unter `metadata`.

Für die Indexierung wird jedes einzelne Dokument zunächst in kürzere Abschnitte unterteilt, das sogenannte Chunking. Durch die kürzeren Textabschnitte gewinnen die Emdeddings an semantischer Bedeutung. Die Wahl des Chunkingverfahrens und die Parameter für die Chunklänge und den Overlap der einzelnen Chunks ist abhängig vom Datensatz. Unterschiedliche Dokumentarten und vor allem unterschiedliche Sprachen haben ganz unterschiedliche ideale Parameter. Für die Basisversion wurden Erfahrungswerte verwendet. Diese sind eine `chunk_size=2000`, `chunk_overlap=200` und als Chunkingverfahren wurde der `RecursiveCharacterTextSplitter` gewählt. Dabei wird ein langer Text rekursiv in kleinere Segmente unterteilt, bis die Segmente die gewünschte Chunklänge haben. Begonnen wird dabei mit groben Kriterien, wie Absätzen und schließlich feiner auf Sätzen oder Wörtern. Auf diese Weiße können Kontextinformationen weitestgehend erhalten werden. In Zukunft sollen diese Parameter auf Basis der Dokumente des Anwendungsfalls automatisiert optimiert werden.

Die einzelnen Chunks werden nun von einem Embeddingmodell in eine Vektorrepräsentation überführt. Für die Basisversion wurde das multilinguale `BGE-M3` (https://huggingface.co/BAAI/bge-m3) Modell gewählt. Als Output kann dieses Dense-, Sparse- und Colbert-Vektoren liefern. Da das Embeddingmodell ein sehr entscheidender Punkt für die Güte des Retrievals ist, weil es die Semantik des Inputs bestmöglichst verstehen muss, soll auch die Wahl des Embeddingmodells in Zukunft automatisiert optimiert werden. In der Basisversion werden zunächst lediglich die Dense-Embeddings genutzt.

Die Vektoren werden schließlich in den Vektor Store geschrieben. Neben dem Dense-Vektor nutzen wir in der Basisversion zusätzlich die Payload `content` für den Inhalt in natürlicher Sprache, den `path` um die Quelle des Kontextes nachverfolgen zu können und einen `content_hash`, mit dem wir eine Doppelung des Inhalts vermeiden können.

6. Retrieval

Für das Retrieval (`search` Endpoint) wird zunächst die Frage des Anwendenden, mit demselben Embeddingmodell wie bei der Indexierung, embedded. Anschließend wird dieser Punkt im Vektorraum genutzt um über die cosine-similarity nach den `k` ähnlichsten Vektoren zu suchen. `top_k` ist dabei die Anzahl der Chunks die im nächsten Schritt als Kontext dem LLM mitgegeben werden sollen. Dieser Wert wurde in der Basisversion zunächst auf `top_k=4` gesetzt und kann in Zukunft ebenfalls Teil der Optimierung sein. Desweitern können zusätzliche Nachbarchunks hinzugefügt werden, wenn der Anwendungsfall dies erfordert.
Die Implementation des Retrievals für die Basisversion ist so erfolgt, dass einfach von der semantischen Suche auf die hybride Suche umgestellt werden kann. Diese kombiniert dense und sparse Vektoren über eine Fusion. Sodass das Optimum aus semantischer und lexikaler Suche erzielt werden kann. Dies ist für Anwendungsfälle entscheidend, bei denen es auf exakte Wortübereinstimmungen ankommt, zum Beispiel bei Daten oder Versionsnummern.

7. Sprachmodell

In der Basisversion der RAG-Pipeline wurde als LLM das Llama3.3:70b (https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/) und das Google Gemma 3 (https://ai.google.dev/gemma/docs/core/model_card_3) verwendet. Angesprochen wird das Modell über die Langchain `chain`-Methode. Der durch das Retrieval erschlossene Kontext wird konkateniert und das Prompttemplate damit befüllt. Anschließend wird diese Systemmessage gemeinsam mit der Usermessage, der Frage des Anwendenden an das LLM geschickt. Dazu stehen zwei Endpunkte zur Verfügung. Einer, der den Output des LLMs in einem ausgibt und ein weiterer Endpunkt, welcher Streaming unterstützt und so die Antwort tokenweise an den Anwendenden zurückgibt.

Prompttemplate in der Learn2RAG Basisversion:

```
# Role and Objective
You will act as a smart AI chatbot that answers questions only by using the
content from the provided information list.

# Instructions
Your rules for answering:
- You should respond in the language in which the Current Question has been
asked.
- You should answer clear and concise.
- IMPORTANT: Only use the information provided after 'Information' to
answer the user question.
- NEVER use your general knowledge.

## Sub-categories for detailed instructions
- You will be given excerpts of information that come from various sources
of information and have a 'Source' and 'Content'. The excerpts from
information sources are separated by lines of '-----'.

# Reasoning Steps
- Decide which excerpts of information are relevant to the question.
- Revise your information list and only keep excerpts of information that
contain parts of your answer.
- Always only use your pre-selected sources of information.
- If the provided information does not contain the answer: Let me know and
never answer the user question.

# Output Format
- Always use Markdown as the output format for your entire answer.
- If you refer to sources of information within your answer, please use the
Source.
- Sort the pre-selected information by importance. The most relevant
information should be at the top. List your pre-selected sources of
information in a bulleted list at the end of your answer, only using the
Source. If you can't answer to the user question, don't provide a list.

# Information:
{context}
```

Um den Anwendenden eine Auswahl an LLMs zur Verfügung stellen zu können, muss zunächst gesichert sein, dass der Prompt auch für andere LLMs in gleicher Weise geeignet ist oder es müssen weitere Prompttemplates hinterlegt werden. Auch hier wäre eine mögliche Option in Zukunft Prompttemplates basierend auf dem Sprachmodell zu optimieren.

8. Evaluation

Um die Güte der RAG-Pipeline sicherzustellen, wurde ein Open Source Evaluationsdatensatz herangezogen. Dabei ist es wichtig, dass der Datensatz nicht nur Fragen und Antworten beinhaltet, sondern Dokumente, Fragen zu diesen Dokumenten und Antworten. Solche Datensätze sind begrenzt und die Wahl fiel auf den RepLiQA-Datensatz (https://huggingface.co/datasets/ServiceNow/repliqa). Hier wurden fiktive Inhalte erstellt, speziell um Modelle darauf zu testen geeigneten Kontext zu finden und LLMs zu testen diesen Kontext korrekt zu verwenden. Der Datensatz wurde gestaffelt im Laufe von 2025 freigegeben, sodass LLMs noch nicht mit all diesen Daten trainiert wurden konnten.

Da der Prompt so designed wurde, dass das LLM nur die im Kontext zur Verfügung gestellten Information für die Antwortgenerierung nutzen soll, ist es entscheidend, dass hier die richtigen Informationen verfügbar sind. Daher wurde die Suche zunächst mittels Recall evaluiert. Mit dem oben beschriebenen Setting konnte ein Recall@4 von 0,79 erzielt werden. Das ist mehr als beeindrucken, da etwa 20% der Fragen im Datensatz nicht mittels der Dokumente beantwortet werden kann. Die Erfahrung zeigt, dass diese Zahlenwerte aus unterschiedlichen Gründen in Real-World Datensätzen kaum erreichbar sind. Dennoch zeigt diese Evaluation, das für die Basisversion sinnvolle Parameter gewählt wurden. Damit macht auch eine Parameteroptimierung erst für realistische Datensätze, wie sie mit den Use Cases aus AP5 erwartet werden, Sinn.

Schlussendlich muss die Antwort des LLMs evaluiert werden. Dazu wurde das Langfuse Framework (https://langfuse.com/) genutzt. Über diese Plattform kann neben Latenz und Kosten die inhaltliche Qualität der Anfragen beobachtet werden. Hierzu können Nutzeranfragen als Traces gespeichert oder direkt ein Evaluationsdatensatz erstellt werden. Für die Evaluation bietet Langfuse eigene Metriken und Metriken von Ragas an. Zudem lassen sich Custom Metriken erstellen, um spezielle Anforderungen zu evaluieren. Es gibt ein Python SDK und die Möglichkeit Experimente über die UI zu starten.
Im Rahmen der Implementation der Basisversion der RAG-Pipeline wurde ein Teil des RepLiQA-Datensatzes als Evaluationsdatensatz in Langfuse registriert und es wurden Metriken erstellt und ausgesucht mit denen sich die Antwortqualität evaluieren lässt. Langfuse soll in Zukunft noch stärker genutzt werden, um Auswertungen beim Prompt Engineering zu fahren und die Antwortqualität verschiedener LLM zueinander zu evaluieren. Alle Evaluationsergebnisse lassen sich in Langfuse speichern und lassen sich einzeln für jede Trace oder über Experimente hinweg auch visuell betrachten.