



**HOCHSCHULE LANDSHUT**

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN

FAKULTÄT INFORMATIK

# **Studienarbeit**

ERSTELLUNG EINES LLM-BASIERTEN CHATBOTS

Florian Tobias Paul

Lena Anna Klosik

Florian Freiburger

Betreuer:

Prof.Dr. Christian Osendorfer

# *Abstract*

*In diesem Projekt wird die Entwicklung eines innovativen Suchmaschinen-Prototyps für YouTube-Inhalte vorgestellt, der mithilfe von Großen Sprachmodellen (LLMs) und fortschrittlichen Technologien realisiert wurde. Es kombiniert Spracherkennung, Inhaltsindexierung und eine interaktive Chatbot-Schnittstelle, um eine präzise Suche und Zugriff auf Informationen in Videoinhalten zu ermöglichen. Durch die Transkription und Indexierung von Videomaterial wird eine effektive und schnelle Informationsrecherche ermöglicht, die den Zugang zu Wissen in digitalen Medien erheblich verbessert und traditionelle Suchmethoden übertrifft.*

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Grundlagen</b>	<b>3</b>
2.1	LLM . . . . .	3
2.2	RAG . . . . .	3
2.2.1	Funktionsweise von RAG . . . . .	3
2.2.2	Anwendungsfälle . . . . .	4
2.3	Vektorstore . . . . .	4
2.3.1	Funktionsweise in RAG Systemen . . . . .	5
<b>3</b>	<b>Architektur</b>	<b>6</b>
3.1	Verwendete Tools . . . . .	6
3.1.1	Hugging Face . . . . .	6
3.1.2	Llama Index . . . . .	6
3.1.3	Gradio . . . . .	7
3.1.4	Verwendete Hilfstools . . . . .	8
3.2	Pipeline und Entrypoints . . . . .	8
3.2.1	Youtube-Download . . . . .	9
3.2.2	Video/Audio Converter . . . . .	10
3.2.3	Transcription . . . . .	10
3.2.4	Erstellen der Vektor Datenbank . . . . .	11
3.3	Chatbot . . . . .	12
3.4	Interface . . . . .	14
<b>4</b>	<b>Evaluierung und Vergleich</b>	<b>17</b>
4.1	Modelauswahl . . . . .	17
4.1.1	Schritte der Model-Evaluierung . . . . .	18
4.2	Chat Engine und Query Engine . . . . .	22
4.2.1	Query Engine . . . . .	22
4.2.2	Chat Engine . . . . .	23

<i>Inhaltsverzeichnis</i>	2
4.2.3 Bewertung der Chat und Query Engines . . . . .	25
<b>5 Verwendung</b>	<b>32</b>
5.1 Setup . . . . .	32
5.1.1 Projekt einrichten . . . . .	32
5.1.2 Anwendung starten . . . . .	33
5.2 Verwendung - UI . . . . .	33
5.2.1 Archiv-Auswahl . . . . .	34
5.2.2 Einlesen neuer Daten . . . . .	35
5.2.3 Chatbot . . . . .	36
<b>6 Fazit</b>	<b>38</b>
<b>Abbildungsverzeichnis</b>	<b>42</b>
<b>Code Listings</b>	<b>43</b>

# 1 Einleitung

Es gibt inzwischen mehr als 3,9 Milliarden Videos auf YouTube, damit stellt es eine enorme Informationsquelle dar, die sowohl für Bildungszwecke als auch für Unterhaltung genutzt werden kann. Die Herausforderung liegt jedoch in der effizienten Selektion relevanter Inhalte bei dieser Informationsflut. Aktuelle Suchmechanismen beschränken sich größtenteils auf Titel, Beschreibungen und Metadaten. Hierdurch können z.B. relevante Videos, Playlists oder Kanäle gefunden werden. Allerdings wäre es aber nun erforderlich, alle gefundenen Videos entweder komplett durchzuschauen oder zumindest nach relevanten Informationen zu durchforsten. Sucht man beispielsweise nach "Machine Learning" erhält eine Playlist des "Stanford Online" YouTube Kanals als Treffer, welche allerdings über 40 Stunden Videomaterial enthält. Um nun die Antworten zu bestimmten Fragen zu bekommen, müsste man einen großen Teil der Videos durchstöbern. Die gezielte Suche nach spezifischen Informationen zu bestimmten Themen stellt eine große Herausforderung dar, da die Informationen nicht zwingend auf den ersten Blick ersichtlich sind.

Das Ziel dieses Projekts ist die Entwicklung einer modernen Suchmaschine, die über traditionelle Methoden hinausgeht und die Inhalte von angegebenen YouTube-Videos, Playlists oder Kanälen direkt analysiert und indiziert. Diese Suchmaschine basiert auf einem Large Language Model, sie versteht also Suchanfragen und antwortet direkt darauf. Das entwickelte System soll in der Lage sein gesprochenes Wort in Videos zu verarbeiten und in Suchanfragen mit zu berücksichtigen. Daraus soll ein Suchsystem entstehen, das es Nutzern ermöglicht, präzise nach spezifischen Inhalten innerhalb der Videos zu suchen – seien es bestimmte Aussagen, Personen, Objekte oder sogar spezifische Handlungen.

Das Hauptziel dieses Projekts ist die Entwicklung einer Prototyp-Suchmaschine, die eine effiziente und genaue Suche innerhalb der Videoinhalte auf YouTube ermöglicht. Die Transkription der Videoinhalte erfolgt über das Speech-To-Text-Modell Whisper3. Anschließend indiziert Llama-Index den transkribierten Text, wodurch die erkannten Informationen strukturiert und für ein vortrainiertes LLM zugänglich gemacht werden. Dieses wird dem Nutzer in Form eines Chatbots zur Verfügung gestellt, welcher

die Möglichkeit bietet, Fragen zu angegebenen Videos zu stellen und diese zusammen mit der Angabe der Quelle beantwortet. Abschließend wird das Ergebnis evaluiert.

Das Endergebnis soll ein innovatives Tool darstellen, das die Art und Weise, wie wir Informationen in Videomaterial finden und darauf zugreifen, verbessert und erweitert.

## 2 Grundlagen

### 2.1 LLM

Große Sprachmodelle (Large Language Models, LLMs) sind fortschrittliche KI-Systeme, die darauf trainiert sind, menschliche Sprache zu verstehen, zu generieren und mit ihr zu interagieren. Diese Modelle werden mit riesigen Mengen an Textdaten aus dem Internet trainiert. LLMs werden in einer Vielzahl von Anwendungen eingesetzt, darunter Textgenerierung, Übersetzung, Zusammenfassung, Frage-Antwort-Systeme und mehr. Ein Problem bei LLMs besteht jedoch darin, dass sie zu einem bestimmten Zeitpunkt mit begrenzten Datenbasis trainiert werden, hierdurch veraltet das Wissen zum einen und zum anderen fehlt dem Modell anwendungsspezifisches Wissen.

### 2.2 RAG

Retriever-Augmented Generation (RAG) ist ein fortschrittliches Konzept der künstlichen Intelligenz, das speziell darauf ausgerichtet ist, die Genauigkeit und Relevanz von Antworten in Frage-Antwort-Systemen zu verbessern. RAG-Modelle kombinieren die Leistungsfähigkeit von Large Language Models (LLMs) mit einem Informations-Retrieval-Mechanismus [Mic23]. Dieser Ansatz ermöglicht es dem System, nicht nur auf intern gelerntes Wissen zu antworten, sondern auch externe Datenquellen dynamisch in der Generierung zu berücksichtigen, um präzisere, aktuellere und kontext-bezogene Antworten zu generieren. Dadurch kann die Wissensbasis eines Modells erweitert werden.

#### 2.2.1 Funktionsweise von RAG

Zwei-Stufen-Ansatz: RAG-Modelle arbeiten in zwei Schritten. Zuerst verwendet der „Retriever“ eine Suchanfrage, um relevante Informationen aus einer großen Datenbank oder einem Dokumentenkörper zu extrahieren. Anschließend nutzt der „Generator“, oft ein LLM, diese Informationen, um eine präzise und fundierte Antwort zu

generieren.

**Dynamische Informationssuche:** Im Gegensatz zu traditionellen LLMs, die ausschließlich auf vorher trainiertem Wissen basieren, ermöglicht RAG das Abrufen und Integrieren aktueller, spezifischer Informationen aus externen Quellen in Echtzeit.

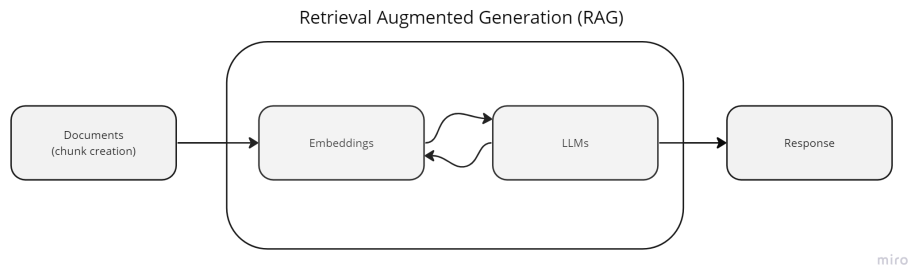


Abb. 2.1: RAG Diagramm

## 2.2.2 Anwendungsfälle

**Wissensbasierte Frage-Antwort-Systeme:** RAG ist besonders nützlich für detaillierte und spezifische Anfragen, bei denen die Antwort möglicherweise nicht direkt im Trainingsdatensatz des Modells enthalten ist. Für Aufgaben, die aktuelle oder sehr spezifische Informationen erfordern, wie z.B. die neuesten Forschungsergebnisse oder Nachrichten, bietet RAG eine effektive Lösung.

**Verbesserte generative Fähigkeiten:** RAG kann auch in der Generierung von Inhalten eingesetzt werden, insbesondere wenn es darum geht, präzise und gut recherchierte Artikel, Berichte oder Zusammenfassungen zu erstellen.

## 2.3 Vektorstore

Ein Vektorstore, auch bekannt als Vektor-Datenbank oder Vektor-Speicher, ist ein spezialisierter Datenbanktyp, der entwickelt wurde, um hochdimensionale Vektoren effizient zu speichern, zu indizieren und abzufragen [aws23]. Diese Vektoren repräsentieren typischerweise eingebettete Darstellungen (Embeddings) von verschiedenen Datentypen wie Texten, Bildern oder Videos, die von künstlichen Intelligenzmodellen erzeugt werden. Vektorstorages spielen eine zentrale Rolle in Systemen, die auf Retrieval-Augmented Generation (RAG) basieren, da sie eine schnelle und präzise Suche nach den relevantesten Informationen aus umfangreichen Datenmengen ermöglichen.



### 2.3.1 Funktionsweise in RAG Systemen

In Retrieval-Augmented Generation (RAG)-Systemen kommen Vektorstorages zur Speicherung von Embeddings von Dokumenten zum Einsatz. Nutzeranfragen werden in Vektoren umgewandelt und im Vektorstorage nach ähnlichen Embeddings gesucht, die relevante Informationen repräsentieren. Diese werden vom RAG-System für präzise Antworten genutzt. Die Funktionsweise von Vektor-Speichern in RAG-Systemen wird durch Abbildung 2.2 veranschaulicht. Eine User Query wird in einen semantischen Vektor wie  $[0.1 \ 0.4 \ 0.2]$  umgewandelt. Die Vector Store Representation speichert die Vektoren von Informationsfragmenten in einer Datenbank mit eindeutigen IDs. Nach der Umwandlung der Anfrage sucht das System im Vektor-Speicher nach ähnlichen Embeddings, um relevante Kontexte wie „LLMs are awesome!“ zu identifizieren. Das LLM nutzt diesen Kontext, um präzise Antworten zu generieren, basierend auf den relevanten Informationen.

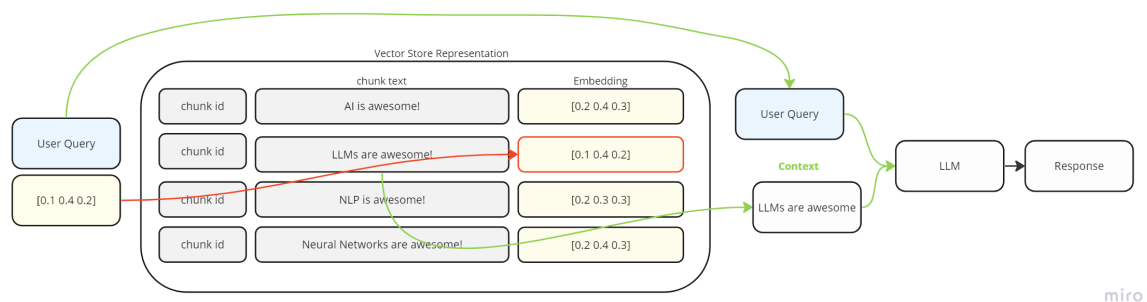


Abb. 2.2: Vector Store Diagram

## 3 Architektur

Im folgenden Kapitel wird ein Überblick über die Architektur unserer Anwendung gegeben. Anfangs werden kurz die wichtigsten verwendeten Tools und Bibliotheken erklärt. Im Anschluss werden die einzelnen Bausteine der Architektur genauer beschrieben.

### 3.1 Verwendete Tools

#### 3.1.1 Hugging Face

Hugging Face [Hug d] ist eine zentrale Plattform für vortrainierte Machine Learning-Modelle und Tools, sowie Datensätze und Projekte. Sie bietet einfachen Zugang zu fortschrittlichen ML-Modellen durch die Transformers-Bibliothek. Durch diese Bibliothek kann wie in Listing 3.1 beispielhaft aufgezeigt, durch Verwendung der `AutoModelForCausalLM` und `AutoTokenizer` Klassen sehr simpel ein Modell und ein zugehöriger Tokenizer erzeugt werden.

```
1 model_id = "mistralai/Mistral-7B-Instruct-v0.2"
2 model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=torch.↵
    ↳ float16).to("cuda")
3 tokenizer = AutoTokenizer.from_pretrained(model_id)
```

Code Listing 3.1: Verwendung der Transformers-Bibliothek

#### 3.1.2 Llama Index

LlamaIndex [Lla23h] ist ein Framework zur Erstellung und Verwaltung von Vektor-Datenbanken, das für die effiziente Durchführung semantischer Suchen in großen Datenmengen konzipiert wurde. Es ermöglicht die Indexierung von Daten in einer Form, die schnelle und präzise Suchergebnisse liefert, indem es die Inhalte in hochdimensionale Vektoren umwandelt. Diese Vektoren repräsentieren die semantische Bedeutung der Daten, was eine Suche basierend auf dem Inhalt und nicht nur auf Schlüsselwörtern ermöglicht.

Eines der Hauptmerkmale von LlamaIndex ist seine Fähigkeit, komplexe Datenstrukturen effizient zu verarbeiten und zu indizieren. Dadurch eignet es sich besonders für Anwendungen in den Bereichen Natural Language Processing (NLP), Bildererkennung und anderen ML-basierten Domänen. LlamaIndex unterstützt Entwickler und Forscher dabei, leistungsfähige Such- und Analysewerkzeuge zu erstellen, die in der Lage sind, die wachsenden Anforderungen moderner Datenverarbeitung zu erfüllen.

In unserem System spielt LlamaIndex eine zentrale Rolle bei der Erstellung einer leistungsfähigen Vektor-Datenbank aus transkribierten Texten. Diese Datenbank ist das Fundament unseres Suchsystems und erlaubt es, schnell relevante Informationen aus zuvor verarbeiteten Multimedia-Inhalten zu extrahieren. Durch die Nutzung von LlamaIndex ist es möglich die semantischen Repräsentationen der transkribierten Texte zu indizieren, hierdurch können relevante Suchergebnisse schnell identifiziert und bereitgestellt werden.

### 3.1.3 Gradio

Gradio [Gra24] ist eine Python-Bibliothek, die die Entwicklung interaktiver Web-Applikationen für Machine Learning (ML) vereinfacht. Sie ermöglicht es, mit wenig Programmieraufwand Benutzeroberflächen zu erstellen, die Nutzern direkte Interaktionen mit ML-Modellen im Webbrowser bieten. Benutzer können unterschiedliche Daten, wie Bilder, Texte oder Audiodateien, eingeben und erhalten prompte Ergebnisse.

Diese Bibliothek erleichtert die Entwicklung von Prototypen und die Verbreitung von Modellen, indem sie Entwicklern und Forschern ermöglicht, ihre ML-Modelle auch ohne tiefgreifende Webentwicklungs- oder Designkenntnisse einem breiten Publikum zugänglich zu machen. Gradio zeichnet sich durch die Unterstützung einer breiten Palette von Ein- und Ausgabeformaten aus, was es für eine Vielzahl von Anwendungsfällen geeignet macht, darunter Bildklassifikation, Textanalyse und Spracherkennung.

In unserem System wird Gradio genutzt, um eine benutzerfreundliche Schnittstelle bereitzustellen. Dies ermöglicht Endnutzern, effektiv Suchanfragen zu stellen, mit dem Chatbot zu interagieren, Archive für spezifisches Wissen auszuwählen und neue Daten in das System einzuspeisen, um daraus neue Archive zu generieren. Durch die Implementierung von Gradio wird die Interaktivität und Zugänglichkeit unseres Systems verbessert, was die Nutzung und das Verständnis von komplexen ML-Modellen für ein breites Spektrum von Anwendern vereinfacht.

### 3.1.4 Verwendete Hilfstools

#### Pytube

Pytube [Pyt23] ist eine Python-Bibliothek zum Herunterladen von Videos von YouTube. Sie ermöglicht den Download von Videos in verschiedenen Auflösungen und Formaten. Pytube bildet die Grundlage für den Youtube-Download-Prozess in unserer Architektur, indem es den schnellen und zuverlässigen Download von Videoinhalten ermöglicht.

#### shutil

Shutil [Pyt24] ist eine Python-Bibliothek, die eine Reihe von hochleveligen Operationen auf Dateien und Sammlungen von Dateien bietet. In unserem System wird shutil hauptsächlich für das Kopieren und Verschieben von Dateien verwendet. Dies ist besonders nützlich beim Verarbeiten der hochgeladenen Inhalte und deren Vorbereitung für die Konvertierung und Transkription.

#### FFMpeg

FFMpeg [FFm23] ist ein vielseitiges Tool zur Verarbeitung von Multimedia-Daten, das Konvertierung, Aufnahme, Umwandlung und Streaming von Audio und Video unterstützt. In unserer Architektur wird FFMpeg verwendet, um Videos in ein standardisiertes Format zu konvertieren und Audiospuren aus Videos zu extrahieren. Diese Funktionalität ist entscheidend für die nachfolgenden Schritte der Transkription und Analyse.

#### Whisper3

Whisper [Ope23] ist ein universell einsetzbares Spracherkennungsmodell, das genaue Transkriptionen von Audioinhalten liefert. Es wurde auf einem großen Datensatz mit unterschiedlichen Audiodaten trainiert und ist ein Multitasking-Modell, das mehrsprachige Spracherkennung, Sprachübersetzung und Sprachidentifikation durchführen kann. Es wird in unserer Pipeline eingesetzt, um die extrahierten Audiodaten in Text umzuwandeln, was eine wesentliche Voraussetzung für die Textanalyse und -indizierung ist.

## 3.2 Pipeline und Entrypoints

In diesem Kapitel wird die Datenverarbeitungspipeline, dargestellt in Abbildung 3.1, beschrieben. Diese Pipeline bildet das Kernstück des Systems und zielt darauf ab, multimediale Inhalte effizient zu verarbeiten und zu analysieren. Sie setzt sich aus

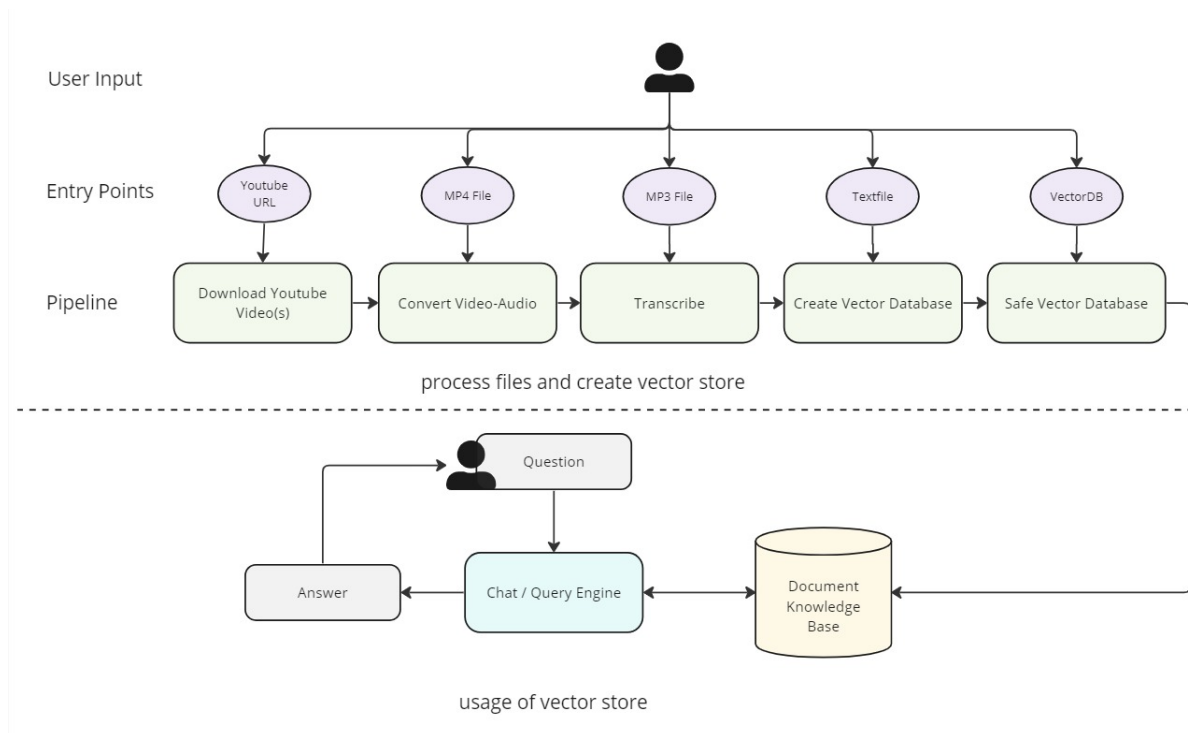


Abb. 3.1: Architektur-Schema

folgenden Schlüsselkomponenten zusammen: Download von YouTube-Videos, Umwandlung von Videos in Audio-Dateien, Transkription der Audios und Erstellung einer Vektor-Datenbank aus den Textdaten.

Ein wesentliches Merkmal der Pipeline ist die Flexibilität ihrer Entrypoints. Nutzer können den Prozess an einem beliebigen Punkt starten, je nach vorliegenden Daten und spezifischen Bedürfnissen. Dies ermöglicht eine direkte Nutzung der Konvertierungs-, Transkriptions- oder Datenbankerstellungsfunktionen, ohne den vorherigen Prozessschritt durchlaufen zu müssen.

Diese Modularität erhöht die Benutzerfreundlichkeit und ermöglicht eine breite Anwendung der Pipeline für verschiedene Projekte. Die folgenden Abschnitte erläutern die Funktionen der einzelnen Komponenten.

### 3.2.1 Youtube-Download

Im Rahmen unserer Datenverarbeitungspipeline ist der erste Schritt der Download von YouTube-Inhalten. Hierfür wird die in Abschnitt 3.1.3 detailliert beschriebene Bibliothek *Pytube* eingesetzt. Der Prozess beginnt mit der Eingabe durch den Nutzer, die in Form von einem oder mehreren YouTube-Links erfolgt. Anschließend wird überprüft, ob es sich bei der Eingabe um eine YouTube-Playlist handelt. Falls erforder-

derlich, wird die *Playlist*-Klasse von *Pytube* verwendet, um die URLs aller Videos in der Playlist zu extrahieren. Die URLs werden zunächst in einer Textdatei namens *Video-Urls* gespeichert. Diese Datei wird dann Zeile für Zeile ausgelesen, und jedes Video wird mittels der *YouTube*-Klasse von *Pytube* heruntergeladen. Die heruntergeladenen Videos werden in einem spezifischen Verzeichnis unseres File-Storages unter dem Pfad *llm\_studienprojekt/file\_storage/mp4s/archive\_name\_time\_stamp* abgelegt. Dieser Vorgang wird fortgesetzt, bis alle in der Textdatei aufgeführten URLs verarbeitet sind.

### 3.2.2 Video/Audio Converter

Im zweiten Schritt unserer Datenverarbeitungspipeline erfolgt die Konvertierung der im ersten Schritt heruntergeladenen Videos in Audio-Dateien mit Hilfe der *FFmpeg*-Bibliothek, wie bereits in Abschnitt 3.1.3 beschrieben.

Dieser Prozess beginnt mit der Definition der Quell- und Zielverzeichnisse für die Umwandlung, wobei die Video-Dateien im Verzeichnis *llm\_studienprojekt/file\_storage/mp4s/archive\_name\_time\_stamp* liegen. Für jede dieser Dateien wird ein Eingabestrom mittels *ffmpeg.input(source\_path)* erstellt, um die Videos zu verarbeiten. Anschließend wird ein Ausgabestrom mit *ffmpeg.output(stream, destination\_path)* generiert, der die Konvertierung in das Audioformat vornimmt. Durch den Einsatz von *ffmpeg.run(stream)* wird dieser Vorgang ausgeführt, womit die Videos effektiv in MP3-Dateien umgewandelt und im Zielverzeichnis *llm\_studienprojekt/file\_storage/mp3s/{archive\_name\_time\_stamp}* abgelegt werden. Dieser Vorgang wird für alle im Quellverzeichnis gefundenen MP4-Dateien wiederholt, bis jede in eine MP3-Datei konvertiert und entsprechend gespeichert wurde.

### 3.2.3 Transcription

Im dritten Schritt unserer Datenverarbeitungspipeline erfolgt die Transkription der Audio-Dateien, die aus den zuvor konvertierten Videos stammen. Für diese Aufgabe wird das in Abschnitt 3.1.3 beschriebene Spracherkennungsmodell *Whisper*, speziell die Version *Whisper-large-v3*, eingesetzt. Dieses Modell bietet dank seiner fortschrittlichen Algorithmen eine hohe Genauigkeit bei der Umwandlung von Sprache zu Text.

Zu Beginn dieses Schrittes wird das Gerät (CPU oder GPU) und der Datentyp (float16 oder float32) definiert, basierend darauf, ob eine GPU verfügbar ist, um eine optimale Leistungsfähigkeit während der Transkription zu gewährleisten. Anschließend wird das *Whisper*-Modell geladen und für die Verarbeitung vorbereitet. Eine Pipeline für automatische Spracherkennung, die speziell für diese Aufgabe konfiguriert

ist, wird genutzt, um die Audio-Dateien effektiv zu verarbeiten. Diese Konfiguration beinhaltet unter anderem die Festlegung der maximalen Token-Anzahl, die Länge der Audio-Abschnitte und die Batch-Größe, um eine effiziente und genaue Transkription zu ermöglichen.

Die Transkription erfolgt in Batches, wobei jede MP3-Datei aus dem festgelegten Verzeichnis *llm\_studienprojekt/file\_storage/mp3s/{archive\_name\_time\_stamp}* gelesen und verarbeitet wird. Für jede Datei extrahiert die Pipeline den Text aus den Audioinhalten und speichert die resultierenden Transkripte in einer separaten Textdatei. Diese Textdateien werden im Verzeichnis *llm\_studienprojekt/file\_storage/document\_collections/archive\_name\_time\_stamp* abgelegt, wodurch eine strukturierte Sammlung von Dokumenten für die spätere Analyse und Verwendung entsteht.

Nachdem alle Audio-Dateien transkribiert wurden, wird die Pipeline bereinigt. Dies umfasst das Löschen des Modells und der Prozessoren, um Speicher freizugeben, sowie die Leerung des GPU-Caches und die explizite Anforderung der Garbage Collection, um sicherzustellen, dass das System für nachfolgende Aufgaben optimiert bleibt.

### 3.2.4 Erstellen der Vektor Datenbank

Der vierte und abschließende Schritt unserer Datenverarbeitungspipeline befasst sich mit dem Erstellen einer Vektor-Datenbank aus den transkribierten Texten, die im vorherigen Schritt generiert wurden. Dieser Prozess ist entscheidend für die Ermöglichung effizienter Such- und Analysefunktionen innerhalb unseres Systems.

Zunächst wird der Service-Kontext mit dem gewählten Sprachmodell initialisiert, wobei die Verfügbarkeit einer GPU geprüft wird und dementsprechend das Gerät sowie den Datentyp (float16 oder float32) für die Verarbeitung festlegt wird. Dies gewährleistet eine optimale Nutzung der Hardware-Ressourcen während des Indexierungsprozesses.

Anschließend wird versucht, eine bestehende Vektor-Datenbank aus dem Speicher zu laden. Sollte dies nicht möglich sein, etwa weil noch keine Indexierung für das aktuelle Archiv durchgeführt wurde, beginnt der Prozess der Erstellung einer neuen Vektor-Datenbank. Dazu werden die transkribierten Texte aus dem entsprechenden Verzeichnis *llm\_studienprojekt/file\_storage/document\_collections/archive\_name\_time\_stamp* gelesen. Diese Texte dienen als Grundlage für die Erstellung der Vektor-Datenbank.

Mit Hilfe der *SimpleDirectoryReader*-Komponente werden die Daten geladen und dann von der *VectorStoreIndex*-Komponente genutzt, um aus den Dokumenten eine

Vektor-Datenbank zu erstellen. Dieser Schritt beinhaltet die Transformation der Texte in hochdimensionale Vektoren, die dann im festgelegten Verzeichnis *llm\_studienprojekt/file\_storage/vector\_stores/archive\_name\_time\_stamp* gespeichert werden. Die Vektorisierung ermöglicht es uns, Inhalte schnell durchsuchbar zu machen und relevante Informationen effizient abzurufen.

Während des Indexierungsprozesses wird der Fortschritt angezeigt, um den Nutzern Feedback über den aktuellen Status zu geben. Nach Abschluss der Indexierung wird die Vektor-Datenbank persistent gespeichert, um eine Wiederverwendung zu ermöglichen.

Dieser systematische Ansatz zur Erstellung der Vektor-Datenbank bildet einen wesentlichen Bestandteil unserer Pipeline und ermöglicht es, die in den vorherigen Schritten verarbeiteten Inhalte für zukünftige Anfragen und Analysen zugänglich zu machen.

### 3.3 Chatbot

In diesem Abschnitt wird sich mit der Entwicklung einer Chatbot-Engine, die speziell darauf ausgelegt ist, Kontextinformationen aus Dokumentensammlungen mittels Vektorindizes abzurufen und zu nutzen, beschäftigt. Der Entwicklungsprozess umfasst mehrere Schlüsselschritte, die den Einsatz diverser Technologien und Bibliotheken beinhalten.

Zunächst beginnt der Aufbau mit der Initialisierung des Service-Kontexts für die Chatbot-Engine. In diesem frühen Stadium wird ein Sprachmodell, insbesondere ein Transformer-basiertes Modell von Hugging Face, geladen und für den Einsatz konfiguriert. Abhängig von der Verfügbarkeit wird für die Nutzung die CPU oder GPU als Recheneinheit ausgewählt und der entsprechende Datentyp (float32 oder float16) festgelegt. Zusätzlich wird ein Tokenizer geladen, und es entsteht ein HuggingFaceLLM-Objekt, das die Schnittstelle zum generativen Modell bildet 3.2.

```

1 def init_service_context(model_id):
2     device = "cuda" if torch.cuda.is_available() else "cpu"
3     torch_dtype = torch.float16 if torch.cuda.is_available() else torch.↵
        ↪ float32
4
5     model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=↵
        ↪ torch_dtype).to(device)
6     if model.config.pad_token_id is None:
7         model.config.pad_token_id = model.config.eos_token_id
8     tokenizer = AutoTokenizer.from_pretrained(model_id)
9     hf_llm = HuggingFaceLLM(
10         model=model,

```



```

11     model_name=model_id,
12     tokenizer=tokenizer,
13     max_new_tokens=MAX_NEW_TOKENS,
14     generate_kwargs=GENERATE_KWARGS,
15     device_map="auto",
16     model_kwargs={"torch_dtype": torch_dtype},
17     is_chat_model=True,
18 )
19 service_context = ServiceContext.from_defaults(
20     llm=hf_llm,
21     embed_model="local:BAAI/bge-base-en-v1.5",
22 )
23 return service_context, model

```

Code Listing 3.2: Initialisieren des Service-Kontexts

Nach der Initialisierung des Service-Kontexts folgt das Laden oder Erstellen eines Vektorindexes für eine Dokumentensammlung. Ist ein solcher Index bereits vorhanden, wird dieser verwendet. Andernfalls werden Dokumente aus einem festgelegten Ordner entnommen, um einen neuen Vektorindex zu erstellen, der anschließend für zukünftige Anwendungen gespeichert wird 3.3.

```

1 def get_vector_index(archive_name, service_context):
2     try:
3         st_cont = StorageContext.from_defaults(persist_dir=str(os.path.join(
4             ↪ VECTOR_STORES_DIR, archive_name)))
5         vector_index = load_index_from_storage(st_cont)
6     except:
7         archive_path = os.path.join(DOCUMENT_COLLECTIONS_DIR, archive_name)
8         if not os.path.exists(archive_path):
9             os.makedirs(archive_path)
10        documents = SimpleDirectoryReader(str(archive_path)).load_data()
11        vector_index = VectorStoreIndex.from_documents(documents, ↪
12            ↪ service_context=service_context, show_progress=True)
13        vector_index.storage_context.persist(persist_dir=str(os.path.join(
14            ↪ VECTOR_STORES_DIR, archive_name)))
15    return vector_index

```

Code Listing 3.3: Initialisierung der Chat-Engine

Mit dem vorbereiteten Service-Kontext und einem Informations-Retrieval-Mechanismus, hier als Retriever bezeichnet, wird eine Chat-Engine instanziiert. Diese Engine ist dafür zuständig, Benutzeranfragen zu verarbeiten und dabei den Kontext aus den verfügbaren Dokumentensammlungen zu nutzen 3.4.

```

1 def init_chat_engine(service_context, retriever):
2     return CustomCondensePlusContextChatEngine.from_defaults(
3         service_context=service_context,
4         skip_condense=True,
5         retriever=retriever,)

```

Code Listing 3.4: Erstellen des Vektorindex

Darüber hinaus werden verschiedene Parameter für die Textgenerierung festgelegt, einschließlich der Temperatur für die Stichprobenauswahl, Entscheidungen über die Stichprobenziehung, die Wiederholungsstrafe und die Anzahl der zurückgegebenen Sequenzen. Ebenfalls wird die maximale Anzahl von Tokens, die das Modell in einer Anfrage generieren kann, definiert.

Der gesamte Entwicklungsprozess unserer Chatbot-Engine veranschaulicht, wie durch den Einsatz von fortschrittlichen Techniken und Methoden ein System realisiert wird, das in der Lage ist, aus umfangreichen Dokumentensammlungen zu lernen und diese Informationen nahtlos in Echtzeitkonversationen zu integrieren. Die Anwendung von Vektorindizierung, Transformer-basierten Sprachmodellen sowie die Integration von Natural Language Processing, Informations-Retrieval und Deep Learning ermöglichen eine effiziente und reaktionsschnelle Chatbot-Engine.

Die gezielte Auswahl des Sprachmodells und der spezifisch genutzten Chat- bzw. Query-Engine bildet dabei einen Kernaspekt unserer Architektur, wobei Effizienz und Eignung für unsere spezifischen Anforderungen im Vordergrund stehen. In Kapitel 4 wird detailliert auf die Evaluation verschiedener Modelle und Engines eingegangen, indem ihre Performance verglichen wird.

## 3.4 Interface

Unser System nutzt Gradio, eine Python-Bibliothek, die es ermöglicht, schnell und einfach interaktive Web-Applikationen für Machine Learning-Modelle zu entwickeln. Durch Gradio wird eine benutzerfreundliche Oberfläche bereitgestellt, die direkte Interaktionen mit unserem System ohne umfangreiches technisches Wissen ermöglicht.

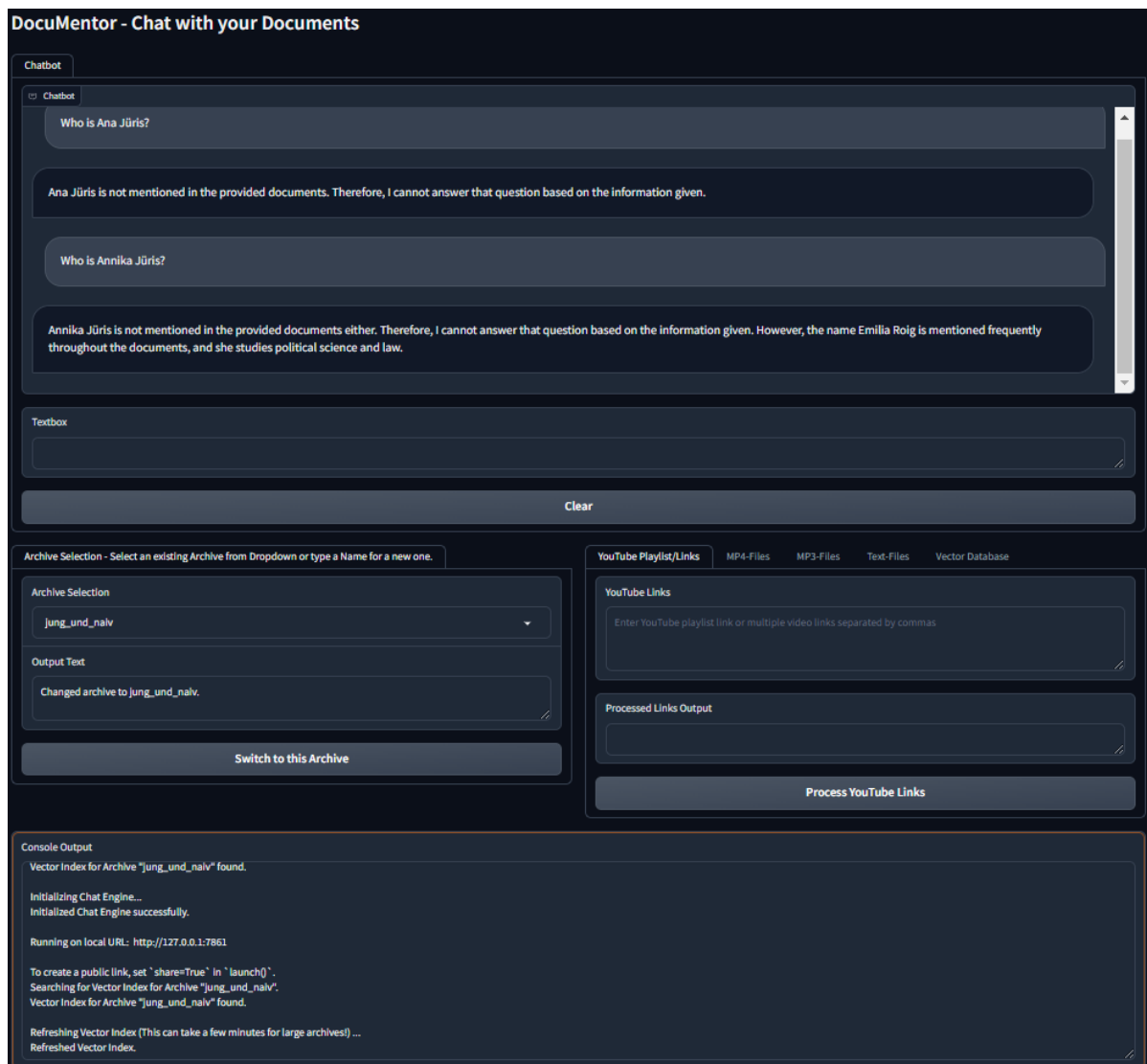


Abb. 3.2: Userinterface der LLM-basierten Suchmaschine

Die Benutzeroberfläche umfasst verschiedene Komponenten (siehe Abbildung 3.2), die zentrale Funktionen unseres Systems unterstützen und den Nutzern eine effiziente Interaktion ermöglichen:

### Chatbot-Tab

Ermöglicht Benutzern, Anfragen zu stellen und Antworten vom Chatbot in Echtzeit zu erhalten.

### Archive Selection

Benutzer können aus verschiedenen archivierten Datenquellen auswählen, um die Antworten des Chatbots zu spezifizieren.

**YouTube Playlist/Link**

Benutzer haben die Möglichkeit, YouTube-Links oder Playlists hochzuladen, die dann heruntergeladen, konvertiert und transkribiert werden, um die Datenbasis zu erweitern.

**MP4- und MP3-Files**

Direktes Hochladen von Audio- und Video-Dateien zur Transkription und anschließenden Indexierung.

**Text-Files**

Ermöglicht das Hochladen von Textdateien, die direkt in die Vektor-Datenbank aufgenommen werden.

**Vector Database**

Benutzer können bestehende Vektor-Datenbanken hochladen, um die Informationsbasis des Systems zu erweitern.

**Konsolen Ausgabe**

Das System teilt dem Nutzer Informationen über die Prozesse mit, welche in der Anwendung ausgeführt werden.

Durch die Integration dieser Komponenten in das Interface wird eine umfassende Plattform geschaffen, die nicht nur das Abfragen von Informationen ermöglicht, sondern auch die aktive Erweiterung der Wissensbasis durch die Nutzer unterstützt wird.

Die Implementierung von Gradio stellt somit eine essenzielle Verbindung zwischen der Datenverarbeitungspipeline und dem Chatbot dar, indem sie eine einheitliche Schnittstelle für die Nutzerinteraktion bietet. Dies gewährleistet, dass unser System sowohl zugänglich als auch effektiv ist und macht es zu einem wertvollen Werkzeug für die Suche und Analyse von multimedialem Inhalt.

## 4 Evaluierung und Vergleich

### 4.1 Modelauswahl

Um die Auswahl eines LLMs für Verwendung als Chatbot zu begründen, wurden einige Modelle von Hugging Face getestet. Die Vorauswahl dieser Modelle beruht auf Popularität auf Hugging Face und der angegebenen Qualität im Vergleich zu anderen Modellen. Außerdem musste auch die Modellgröße berücksichtigt werden, um die Modelle auf der gegebenen Grafikkarte laufen lassen zu können. Die vorausgewählten Modelle sind:

- Intel/neural-chat-7b-v3-1 [Hug23a]
- mistralai/Mistral-7B-Instruct-v0.1 [Hug23c]
- mistralai/Mistral-7B-Instruct-v0.2 [Hug23d]
- meta-llama/Llama-2-7b-chat-hf [Hug23b]

Wie zu erkennen, sind alle Modelle für Chat-Anwendungen spezialisiert.

Anschließend musste ein Weg gefunden werden, diese Modelle vergleichen, um eine Auswahl zu treffen. Neben der Findung einer geeigneten Metrik war auch die Verfügbarkeit von Datensätzen ein Problem. Bezüglich der Metrik wurde beschlossen, nur den für uns relevanten Anwendungsfall zu testen. Das bedeutet also, dass getestet werden sollte, ob das Modell nach Erweiterung der Wissensbasis durch Llama-Index auch in der Lage ist, Informationen aus den gegebenen Dokumenten verlässlich wiederzugeben. Es wurde gefunden, dass Llama-Index eine Klasse `RagDatasetGenerator` [Lla23a] bereitstellt. Diese bietet die Möglichkeit, von einem LLM einen Katalog von Fragen zusammen mit den Kontexten, in welchen die Antworten gegeben wären erstellen zu lassen. Anschließend können diese Fragen an die zu testenden Modelle gestellt werden und abschließend deren Antworten evaluiert werden. Im Folgenden werden diese Schritte genauer besprochen.

### 4.1.1 Schritte der Model-Evaluierung

#### Generierung des Fragen-Datensatzes (dataset\_generation.py)

Der Datensatz wurde mit Hilfe der LlamaIndex Klasse `RagDatasetGenerator` erstellt (s. Listing 4.1). Diese erhält eine Kollektion von Dokumenten, aus welchen die Fragen generiert werden. Es wurden zwei Datensätze erstellt, eines mit 24 zufällig ausgewählten Dokumenten der „Jung und Naiv“ Kollektion und ein weiterer mit 12 Textdokumenten, welche aktuelle Wikipedia-Artikel enthalten. Hier wurden nur kürzlich veröffentlichte Artikel verwendet, um sicherzustellen dass die Modelle die relevanten Informationen nicht der eigenen Wissensbasis entnehmen können. Weiterhin wird durch Angabe von `num_questions_per_chunk` die Anzahl der zu generierenden Fragen pro Node bestimmt. Die Funktionsweise des Generators lässt sich durch die angegebenen Parameter `service_context` sowie `text_question_template` und `question_gen_query` erkennen. Als Grundlage des Service Contexts wurde `mistralai/Mistral-7B-Instruct-v0.2` gewählt. Dieses Modell erhält für die Generierung einer Frage den Text des `text_question_templates`, wobei `context_str` mit dem Text des aktuellen Nodes ersetzt wird.

```

42 dataset_generator_rag = RagDatasetGenerator.from_documents (
43     documents=sampled_documents,
44     num_questions_per_chunk=1,
45     service_context=service_context,
46     show_progress=True,
47     text_question_template=Prompt (
48         "A sample from an interview by the YouTube Channel \"Jung und Naiv\"
→ \" is given below.\n"
49         "-----\n"
50         "{context_str}\n"
51         "-----\n"
52         "Using the interview sample, carefully follow the instructions ↗
→ below:\n"
53         "{query_str}"
54     ),
55     question_gen_query=(
56         "You are an evaluator for a search pipeline. Your task is to write ↗
→ a single question "
57         "using the provided documentation sample above to test the search ↗
→ pipeline. The question could "
58         "reference specific names, facts, information or events. Restrict ↗
→ the question to the context information "
59         "provided. When referring to individuals or specific entities or ↗
→ events, use their full names or a detailed "
60         "designation rather than pronouns or general titles. It should be ↗
→ understandable who or what is referred to "
61         "even without the direct context. Keep the question concise.\n"
62         "Question: "
63     ),

```

64 )

## Code Listing 4.1: Generierung des Fragen-Datensatzes

**Generierung der Antworten durch ausgewählte Modelle  
(answers\_generation.py)**

Nachdem der Fragen-Datensatz generiert wurde, können nun diese Fragen von den Modellen beantwortet werden. Es werden hiervon nicht nur die Antworten gespeichert, sondern auch die gestellten Fragen und die von diesem Modell angegebenen Quellen-Kontexte. Das Ergebnis dieses Schrittes ist also eine Json-Datei für jedes Modell, welches für jede Frage die Werte `query`, `reponse` und `contexts` enthält.

**Evaluierung der Antworten (evaluation.py)**

Die tatsächliche Evaluierung der Antworten wird mit zwei Techniken und daraus resultierenden Metriken durchgeführt. LlamaIndex stellt die Klassen `CorrectnessEvaluator` [Lla23e] und `FaithfulnessEvaluator` [Lla23f] bereit. Auch hier basiert die Evaluierung ähnlich zur Generierung des Fragen-Datensatzes auf einem Service-Kontext, welchem ein Modell zugrunde liegt. Auch hier wurde `mistralai/Mistral-7B-Instruct-v0.2` für beide Evaluatoren gewählt.

Innerhalb des `CorrectnessEvaluator` erhält auch hier das Modell eine Anweisung in Form eines System-Strings<sup>4.2</sup>. Dieser instruiert das Modell, eine Antwort gegeben einer Frage in Bezug auf die Korrektheit zu bewerten. Das Ergebnis soll ein Score zwischen 1.0 und 5.0 sein. Es kann zusätzlich eine `reference_answer` gegeben werden. Diese wäre auch Teil des generierten RAG-Datensatzes, da sie allerdings vom einem Modell generiert wurde, welches auch bewertet wird, wurde diese in diesem Fall ausgelassen. Ideal wäre es, für die Generierung des RAG-Datensatzes und zur Bewertung der Antworten ein eindeutig besseres Modell wie zum Beispiel GPT-4 zu verwenden, was aufgrund der damit verbundenen Kosten hier nicht getan wurde.

```
17 DEFAULT_SYSTEM_TEMPLATE = """
18 You are an expert evaluation system for a question answering chatbot.
19
20 You are given the following information:
21 - a user query, and
22 - a generated answer
23
24 You may also be given a reference answer to use for reference in your ↗
   → evaluation.
25
26 Your job is to judge the relevance and correctness of the generated answer.
```

```

27 Output a single score that represents a holistic evaluation.
28 You must return your response in a line with only the score.
29 Do not return answers in any other format.
30 On a separate line provide your reasoning for the score as well.
31
32 Follow these guidelines for scoring:
33 - Your score has to be between 1 and 5, where 1 is the worst and 5 is the ↗
   ↪ best.
34 - If the generated answer is not relevant to the user query, \
35 you should give a score of 1.
36 - If the generated answer is relevant but contains mistakes, \
37 you should give a score between 2 and 3.
38 - If the generated answer is relevant and fully correct, \
39 you should give a score between 4 and 5.
40
41 Example Response:
42 4.0
43 The generated answer has the exact same metrics as the reference answer, \
44 but it is not as concise.
45
46 """
47
48 DEFAULT_USER_TEMPLATE = """
49 ## User Query
50 {query}
51
52 ## Reference Answer
53 {reference_answer}
54
55 ## Generated Answer
56 {generated_answer}
57 """

```

Code Listing 4.2: CorrectnessEvaluator: Internes System Template (llama\_index\evaluation\textbackslash correctness.py)

Der FaithfulnessEvaluator bewertet, ob die generierte Antwort der angegebenen Quelle treu ist. Auch hier wird ein Evaluation Template gegeben 4.3. Dieses enthält die Anweisung, entweder „YES“ oder „NO“ auszugeben, je nachdem ob die Antwort durch den angegebenen Kontext unterstützt wird oder nicht. Falls die generierte Antwort des Evaluierungs-Modells anschließend „YES“ enthält, wird der Score 1 zugewiesen, bei „NO“ 0.

```

77 faith_eval_template = PromptTemplate(
78     "Please tell if a given piece of information "
79     "is supported by the context.\n"
80     "You need to answer with either YES or NO.\n"
81     "Answer YES and only YES if any of the context supports the information ↗
   ↪ , even "
82     "if most of the context is unrelated. "
83     "Answer NO and only NO if none of the context supports the information. ↗
   ↪ "
84     "Your answer can only be YES or NO, without further text."

```



```

85     "Some examples are provided below. \n\n"
86     "Information: Apple pie is generally double-cruste.\n"
87     "Context: An apple pie is a fruit pie in which the principal filling "
88     "ingredient is apples. \n"
89     "Apple pie is often served with whipped cream, ice cream "
90     "('apple pie a la mode'), custard or cheddar cheese.\n"
91     "It is generally double-cruste, with pastry both above "
92     "and below the filling; the upper crust may be solid or "
93     "latticed (woven of crosswise strips).\n"
94     "Answer: YES\n"
95     "Information: Apple pies tastes bad.\n"
96     "Context: An apple pie is a fruit pie in which the principal filling "
97     "ingredient is apples. \n"
98     "Apple pie is often served with whipped cream, ice cream "
99     "('apple pie a la mode'), custard or cheddar cheese.\n"
100    "It is generally double-cruste, with pastry both above "
101    "and below the filling; the upper crust may be solid or "
102    "latticed (woven of crosswise strips).\n"
103    "Answer: NO\n"
104    "Information: {query_str}\n"
105    "Context: {context_str}\n"
106    "Answer: "
107 )

```

Code Listing 4.3: FaithfulnessEvaluator: Prompt Template (llama\_index\evaluation\textbackslash evaluation.py)

## Ergebnisse der Evaluierung

Zur Auswertung der Ergebnisse wurde für jedes Modell der Durchschnittswert der Correctness und Faithfulness berechnet. Die Ergebnisse werden in den Tabellen 4.1 und 4.2 aufgezeigt.

	Mist.-Inst.-v01	Mist.-Inst.-v02	Llama2 Chat 7b	Neural Chat
Correctness	4.07	4.15	4.26	4.17
Faithfulness	0.55	0.56	0.55	0.54

Tab. 4.1: Ergebnisse auf Jung und Naiv-Datensatz

	Mist.-Inst.-v01	Mist.-Inst.-v02	Llama2 Chat 7b	Neural Chat
Correctness	4.04	4.32	4.28	4.24
Faithfulness	0.66	0.72	0.72	0.74

Tab. 4.2: Ergebnisse auf Wikipedia-Datensatz

Wie aus den Tabellen zu entnehmen ist, liefert Mistral Instruct v02 in jeder Hinsicht bessere Ergebnisse auf beiden Datensätzen als sein Vorgänger. Dies wird als

Bestätigung der Relevanz dieser Ergebnisse aufgenommen. Die Faithfulness ist bei allen Modellen sehr ähnlich, mit Ausnahme von Mistral Instruct v01 auf dem Wikipedia-Datensatz. Mistral Instruct v02 und Llama Chat 7b erzielten insgesamt die besten Ergebnisse. Da Mistral Instruct v02 auf dem besser kontrollierten Wikipedia-Datensatz ein besseres Ergebnis erzielen konnte, wurde dieses Modell für den Chatbot ausgewählt.

## 4.2 Chat Engine und Query Engine

Im nachfolgenden Abschnitt wird das Ziel verfolgt, die Qualität der Ausgaben des LLMs zu steigern, indem der Wechsel von der Query-Engine zur Chat-Engine angestrebt wird. Diese Transition ist von entscheidender Bedeutung, um den Kontext vorheriger Nachrichten zu berücksichtigen und somit eine nahtlose, kontextsensitive Kommunikation zu ermöglichen.

### 4.2.1 Query Engine

Die Query-Engine ist darauf ausgerichtet, gezielte Anfragen zu verarbeiten und darauf basierend präzise und klare Informationen oder Antworten bereitzustellen. Ihre Funktionalität konzentriert sich darauf, spezifische Datenabfragen durchzuführen, um exakte Antworten zu erhalten. Typische Anwendungsfälle finden sich in Systemen, in denen der Schwerpunkt auf der schnellen und präzisen Bereitstellung klarer Informationen liegt, ohne die Notwendigkeit für kontextbezogene oder dynamische Konversationen. Die Query-Engine greift ausschließlich auf die Wissensdatenbank (Vektor Store) des RAG-Systems zu und liefert Antworten auf gestellte Fragen, wie in Abbildung 4.1 veranschaulicht. [Lla23g]

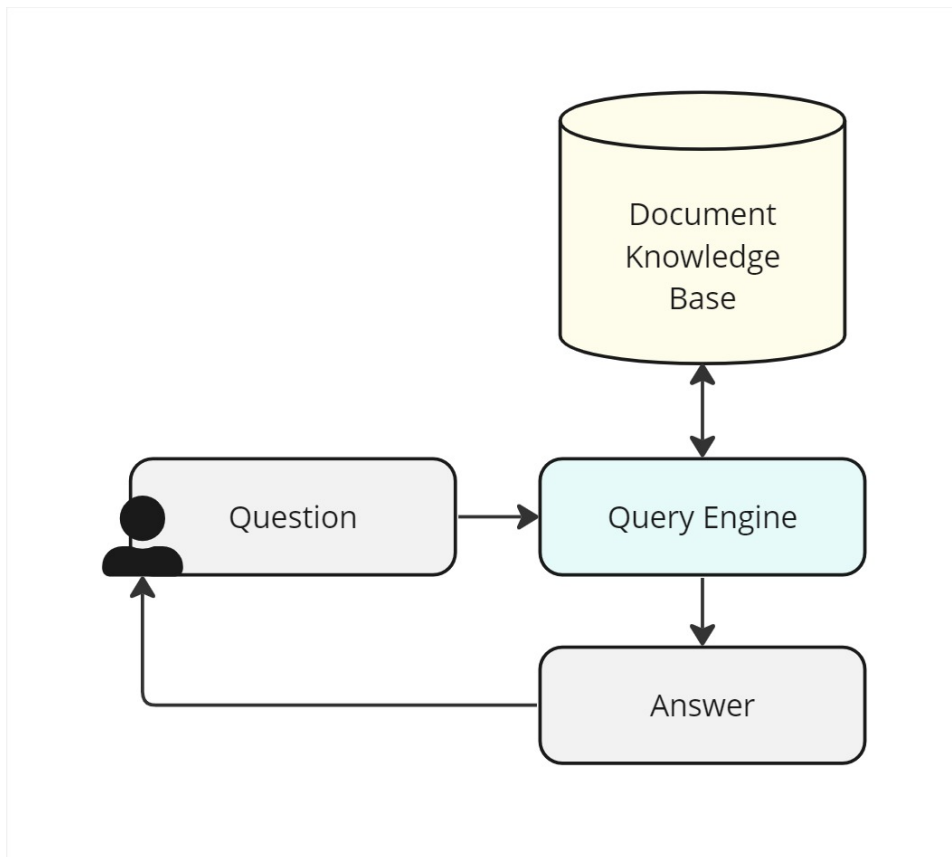


Abb. 4.1: Funktionsweise einer Query Engine

Das folgende Beispiel illustriert Frage-Antwort-Paare, die von einer Query-Engine behandelt wurden. Es wird deutlich, dass bereits bei der zweiten impliziten Erwähnung von Sarah Wagenknecht kein Bezug mehr zu ihr hergestellt werden kann. Daher ist die zweite Antwort nicht korrekt.

**Q:** Wer ist Sarah Wagenknecht?

**A:** Sarah Wagenknecht ist eine Politikerin und war Mitglied der Linkspartei.

**Q:** Welcher Partei gehört sie an?

**A:** Sie gehört der Partei DIE PARTEI an.

### 4.2.2 Chat Engine

Das Ziel besteht darin, präzise Antworten auf Basis unserer Wissensdatenbank bereitzustellen und gleichzeitig eine natürliche Sprachkommunikation mit dem Chatbot zu ermöglichen. Hierbei zielt eine Chat-Engine darauf ab, natürliche und dynamische Gespräche mit Benutzern zu führen. Sie ist in der Lage, eine Vielzahl von Benutzeranfragen zu verarbeiten, kontextsensitive Interaktionen zu ermöglichen und

darauf abzielen, menschenähnliche Konversationen zu führen. Ihre Hauptanwendung liegt in interaktiven und dialogbasierten Benutzererfahrungen bei Chatbots. Ähnlich wie die Query Engine greift auch eine Chat-Engine auf die Daten aus der Wissensdatenbank zu, verwendet jedoch zusätzlich Informationen aus dem Kontext der Chat-Historie, um entsprechende Antworten zu formulieren (siehe Abbildung 4.2). [Lla23d]

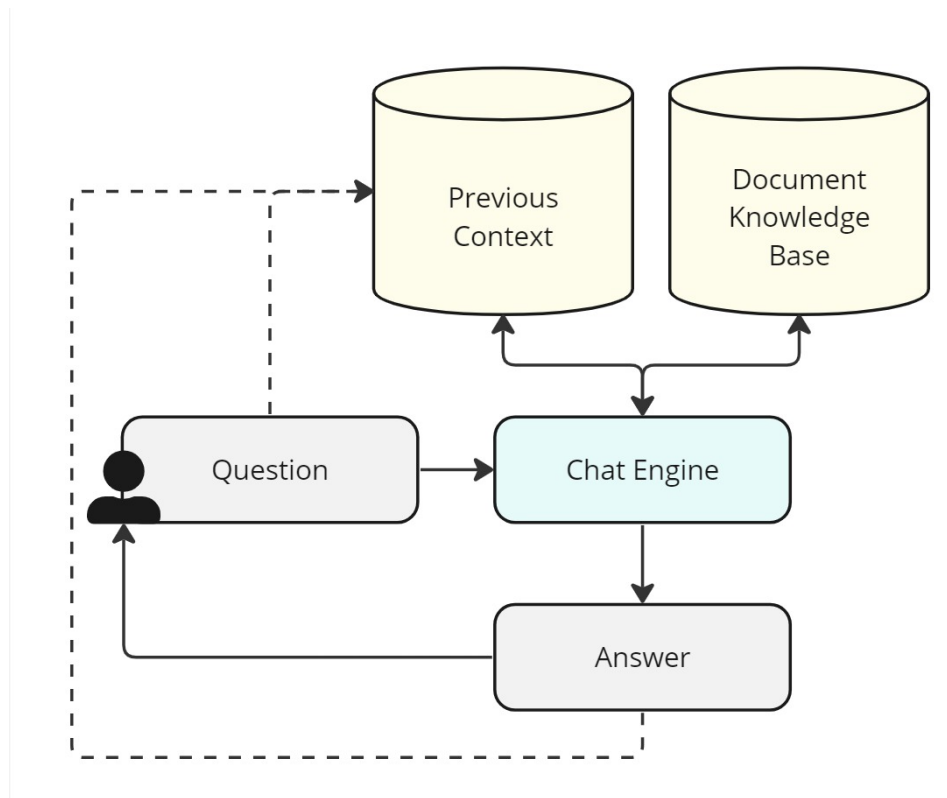


Abb. 4.2: Funktionsweise einer Chat Engine

Das Beispiel zeigt dieselben Frage-Antwort-Paare wie bei der Query Engine, diesmal jedoch beantwortet einer Chat-Engine.

**Q:** Wer ist Sarah Wagenknecht?

**A:** Sarah Wagenknecht ist eine Politikerin und war Mitglied der Linkspartei.

**Q:** Welcher Partei gehört sie an?

**A:** Früher gehörte Sarah Wagenknecht der Linkspartei an, inzwischen ist sie aus der Partei ausgetreten.

In diesem Fall wird deutlich, dass die implizite Nennung von Sarah Wagenknecht mit dem Wort *ſie* eindeutig erkannt wird, was zu einer korrekten Beantwortung der zweiten Frage führt.

## LLamaIndex Chat und Query Engines

Im folgenden Abschnitt werden verschiedene Chat-Engines von LLamaIndex verglichen, um die für diesen Anwendungsfall am besten geeignete Option auszuwählen. Dabei werden insbesondere drei Optionen näher untersucht:

- Condense Mode
- Condense Plus Context
- Query

Der Condense-Modus nutzt eine Abfrage-Engine, die darauf abzielt, bei jeder Chat-Interaktion eine eigenständige Frage aus dem Kontext des Gesprächs und der letzten Nachricht zu generieren. Im Anschluss wird diese verdichtete Frage an die Abfrage-Engine gestellt, um eine Antwort zu erhalten. Diese Methode ist einfach und eignet sich besonders für Fragen, die direkt mit der vorhandenen Wissensbasis verknüpft sind. Allerdings kann sie Schwierigkeiten haben, Metafragen wie „Was habe ich dich zuvor gefragt?“ zu beantworten [Lla23c]. Im Rahmen der Konfiguration des Condense-Modus können verschiedene Parameter angepasst werden, um die Leistung zu optimieren. Dazu gehören beispielsweise die Größe der generierten Fragen, die maximale Anzahl von Tokens sowie die Wahrscheinlichkeit der Verwendung bestimmter Tools innerhalb der Abfrage-Engine. Durch das Testen verschiedener Parameterkombinationen lassen sich potenzielle Verbesserungen in der Leistung des Modus erzielen, wobei auch eine Bewertung der Ausgangsleistung durch einen Test ohne Parameter durchgeführt wird.

Im Gegensatz dazu basiert der Condense-Plus-Context-Modus auf einem Retrieval-System über die vorhandenen Daten. Bei jeder Chat-Interaktion werden sowohl das laufende Gespräch als auch die neueste Benutzernachricht zu einer eigenständigen Frage zusammengefasst. Anschließend wird ein Kontext für diese eigenständige Frage aus dem Retrieval-System erstellt und zusammen mit der Aufforderung und der Benutzernachricht an das LLM übergeben, um eine Antwort zu generieren [Lla23b]. Diese Methode ist ebenfalls einfach und eignet sich besonders für Fragen, die direkt mit der vorhandenen Wissensbasis und allgemeinen Interaktionen zusammenhängen.

### 4.2.3 Bewertung der Chat und Query Engines

Im folgenden Abschnitt werden die ausgewählten Chat- und Query-Engines anhand verschiedener Aspekte bewertet.

## Erstellung von Frage-Antwort-Paaren

Angeichts der Notwendigkeit, den Kontext bei der Evaluation von Chat-Engines zu berücksichtigen, gestaltet sich die automatische Generierung von Testfällen als herausfordernd. Daher wird ein manueller Ansatz verfolgt, bei dem Fragen zu einem spezifischen Dokument formuliert und die erwarteten Antworten manuell anhand des Dokuments festgelegt werden. Diese Vorgehensweise ermöglicht die Erstellung von Fragen, die sich auf den vorherigen Kontext beziehen, und gewährleistet somit eine realistischere Bewertung der Chat-Engines. Insgesamt wurden 14 Frage-Antwort-Paare für die Evaluation erstellt.

## Chat-Engine Einstellungen und Parameter

Die Evaluierung umfasst die Analyse von drei Betriebsmodi: den Condense-Modus, den Condense Plus Context-Modus und die Query-Engine als Referenz für Vergleichszwecke. Im Rahmen des Condense-Modus werden unterschiedliche Variationen von Parametern untersucht, um ihre Einflüsse auf die Leistung zu beurteilen. Die Parameter `condense_factors`, `max_tokens_values`, `split_condense_threshold_values` und `query_engine_tool_probabilities` werden jeweils mit verschiedenen Werten variiert, was zusammen 16 verschiedene Konfigurationen der Chat-Engine ergibt. Zusätzlich zu den anderen Chat-Engines werden insgesamt 19 Testfälle durchgeführt, die auf die 14 vordefinierten Fragen angewandt werden, was zu 224 Frage-Antwort-Paaren führt.

### **`condense_factors = [0.3, 0.9 ]`**

Diese Liste enthält verschiedene Werte für den `condense_factor`, der angibt, wie stark die Generierung der eigenständigen Frage im Vergleich zur Gesamtlänge des Gesprächs sein soll.

### **`max_tokens_values = [150, 200 ]`**

Diese Liste enthält verschiedene Werte für `max_tokens`, die maximale Anzahl von Tokens, die für die generierte Frage verwendet werden sollen.

### **`split_condense_threshold_values = [100, 150 ]`**

Diese Liste enthält verschiedene Werte für `split_condense_threshold`, einen Schwellenwert, der angibt, ab welcher Länge das Gespräch in mehrere Teile aufgeteilt wird, bevor die Frage generiert wird.

### **`query_engine_tool_probabilities = [0.3, 0.9 ]`**

Diese Liste enthält verschiedene Wahrscheinlichkeiten für `query_engine_tool_probability`,

die angibt, mit welcher Wahrscheinlichkeit bestimmte Werkzeuge oder Funktionen innerhalb der Abfrage-Engine verwendet werden sollen

### **Qualitätsbewertung**

Um die Qualität einer Antwort angemessen zu bewerten, bedienen wir uns einer Vielzahl von Qualitätsbewertungsmaßen, die es ermöglichen, verschiedene Aspekte der Antwort umfassend zu analysieren und zu bewerten.

#### **BERT**

Es wird ein Algorithmus verwendet, der Textdaten mit dem BERT-Modell [Hug23e] in numerische Vektoren umwandelt. Dies ermöglicht uns, die semantische Ähnlichkeit zwischen Texten zu berechnen. Der Algorithmus verwendet die Cosinus-Ähnlichkeit, um die Ähnlichkeit zwischen den eingebetteten Vektoren zu messen. Die Cosinus-Ähnlichkeit misst den Winkel zwischen Vektoren im mehrdimensionalen Raum und liefert einen Wert zwischen -1 und 1, wobei 1 eine perfekte Ähnlichkeit und -1 eine perfekte Unähnlichkeit darstellt. Hierbei werden jeweils die erwarteten Antworten mit den von der Chat-Engine gegebenen Antworten verglichen.

#### **Manuelles Testen**

Zusätzlich zu automatisierten Bewertungsmethoden werden auch manuelle Tests durchgeführt, da Abweichungen von den erwarteten Antworten dennoch korrekt sein können. Dabei wird die Sinnhaftigkeit der Antworten anhand von drei verschiedenen Aspekten beurteilt:

1. Die Korrektheit im Bezug auf den verwendeten Kontext der Antwort: Hier wird bewertet, ob die Antwort im Kontext der gestellten Frage und der spezifischen Situation oder des Themas, auf das sich die Frage bezieht, korrekt ist. Dies bedeutet, dass beispielsweise eine Antwort auf eine Frage über das Wetter in einer bestimmten Stadt den Kontext dieser Stadt berücksichtigen muss.
2. Die Korrektheit des Inhalts unabhängig von der Frage: In diesem Aspekt wird die inhaltliche Richtigkeit der Antwort an sich betrachtet, ohne direkten Bezug zur gestellten Frage. Es wird darauf geachtet, ob die Informationen oder Aussagen in der Antwort sachlich korrekt sind und einem bekannten Wissensstand entsprechen.
3. Die Korrektheit in Bezug auf die gestellte Frage selbst: Dies bezieht sich darauf, ob die Antwort angemessen auf die tatsächlich gestellte Frage eingeht. Selbst

wenn die Antwort inhaltlich korrekt ist, kann sie dennoch als unangemessen betrachtet werden, wenn sie nicht direkt auf die Frage eingeht oder diese umgeht.

Für jedes Frage-Antwort-Paar erfolgt eine separate Bewertung jeder Metrik unter Verwendung einer Skala von 1 für schlecht, 2 für akzeptabel und 3 für gut.

### **Ergebnisse der Evaluierung**

Die Evaluierung der verschiedenen Chat-Engines ermöglicht einen tiefen Einblick in ihre Leistungsfähigkeit und ihre Fähigkeit, Fragen angemessen zu beantworten. Die Abbildung 4.3 veranschaulicht die durchschnittlichen Korrektheitswerte sowie die durchschnittlichen Ähnlichkeiten pro Chat-Engine. Hierbei sind auf der x-Achse die verschiedenen Engines aufgelistet, wobei die folgende Abkürzung kurz erläutert werden sollen. Die x-Achse listet die verschiedenen Engines auf, wobei „Con“ für Condense ohne Parameter, „Con()“ für Condense mit Parameter, „ConPlusCont“ für CondensePlusContext ohne Parameter und „Query“ für die Query Engine ohne Parameter stehen. Insgesamt zeigt sich, dass der Condense-Modus im Vergleich zum Condense Plus Context-Modus tendenziell schlechter abschneidet. Allerdings kann durch geschickte Anpassung der Parameter der Condense-Modus durchaus aufholen und eine vergleichbare oder sogar bessere Leistung erzielen.



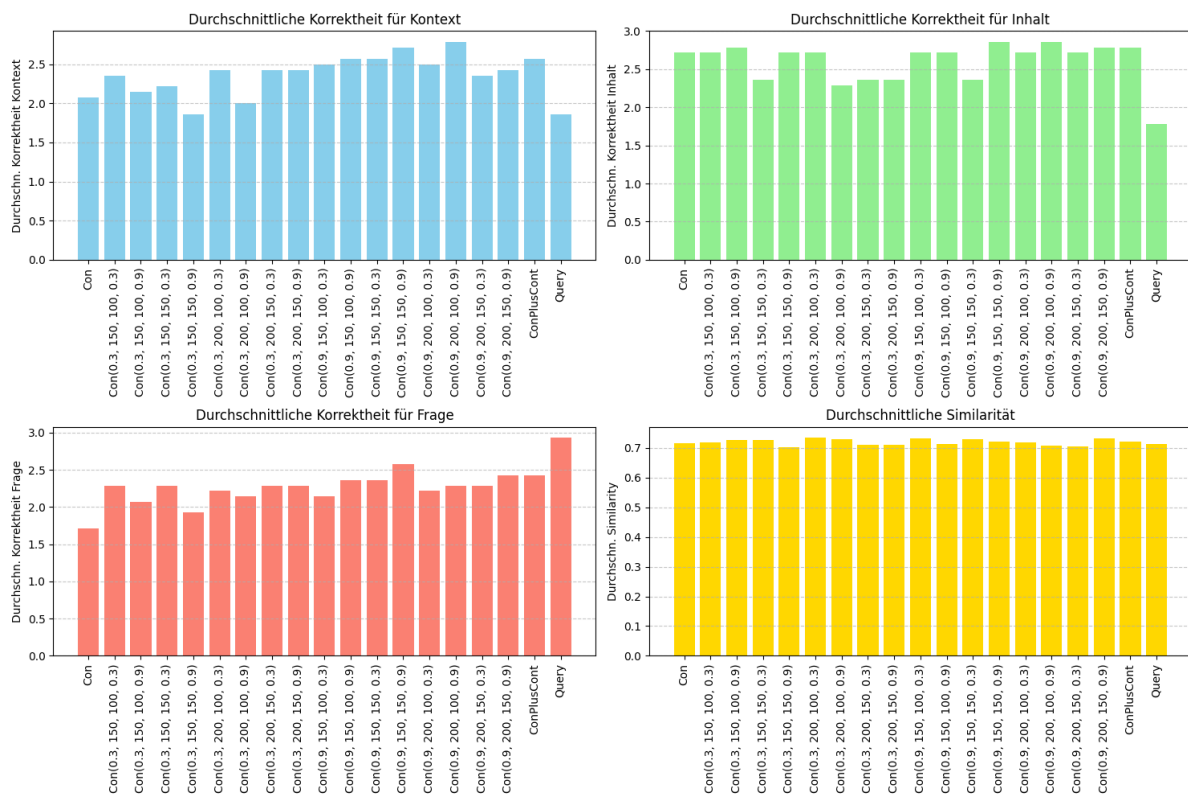


Abb. 4.3: Korrektheit für jede Engine

Auffällig ist insbesondere, dass die Query-Engine in den Kategorien „Korrektheit für Kontext“ und „Korrektheit für Inhalt“ schlechte Ergebnisse erzielt. Dies lässt sich teilweise darauf zurückführen, dass die Engine Schwierigkeiten hat, den Zusammenhang zwischen verschiedenen Fragen herzustellen, was zu inkonsistenten Antworten führt. Trotzdem zeigt sich, dass die Query-Engine in Bezug auf die korrekte Formulierung der Fragestellung gut abschneidet, da sie spezifisch auf die gestellten Fragen eingeht.

Die Analyse der verschiedenen Parametereinstellungen für den Condense-Modus offenbart, dass die Konfiguration „0.9, 150, 150, 0.9“ im Durchschnitt die beste Leistung erzielt. Diese Einstellung ermöglicht eine präzise Generierung von Fragen, die stark auf den Gesprächskontext abgestimmt sind und gleichzeitig relevante Informationen erfassen. Durch einen hohen Condense-Faktor, eine angemessene Frage-Länge durch max\_Tokens, eine geeignete Aufteilung des Gesprächs und die häufige Nutzung von Abfrage-Engine-Tools wird eine ausgewogene Balance zwischen Kontextbezug, Relevanz der Fragen und Nutzung der Abfrage-Engine erreicht, was zu einer verbesserten Leistung der Chat-Engine führt. Des Weiteren fällt auf, dass die durchschnittliche Ähnlichkeit zwischen den Antworten aller Chat-Engines relativ nahe beieinander

liegt. Dies deutet darauf hin, dass die Länge und Genauigkeit der Antworten weitgehend konsistent sind, jedoch das Maß der Ähnlichkeit nicht so präzise ist.

Des Weiteren ist zu bemerken, dass in manchen Szenarien ein Übermaß an Kontext berücksichtigt wird, was potenziell dazu führen kann, dass relevante Informationen in den Antworten untergehen. Beobachtet wird zudem, dass die erste Frage häufig inkorrekt beantwortet wird, was möglicherweise auf Schwächen im Retrieval-Mechanismus hinweist. Sollte eine vorherige Frage fehlerhaft beantwortet worden sein, tendieren nachfolgende Antworten dazu, sich darauf zu beziehen, was trotz angemessenem Kontext möglicherweise nicht zur adäquaten Beantwortung der tatsächlichen Fragestellung führt.

In Abbildung 4.4 werden die Qualitätsbewertungen pro Frage-Antwort-Paar dargestellt. Hierbei stellt die x-Achse die 14 verschiedenen Fragen dar und die y-Achse die durchschnittlichen Evaluationswerte. In der Abbildung wird deutlich, dass die Korrektheit der Antworten je nach Kontext und Fragestellung variiert. Während einige Fragen korrekt beantwortet werden können, fehlt bei anderen Fragen der erforderliche Kontext. Dennoch ist es ermutigend festzustellen, dass die Fakten, die in den Antworten präsentiert werden, in den meisten Fällen korrekt sind.

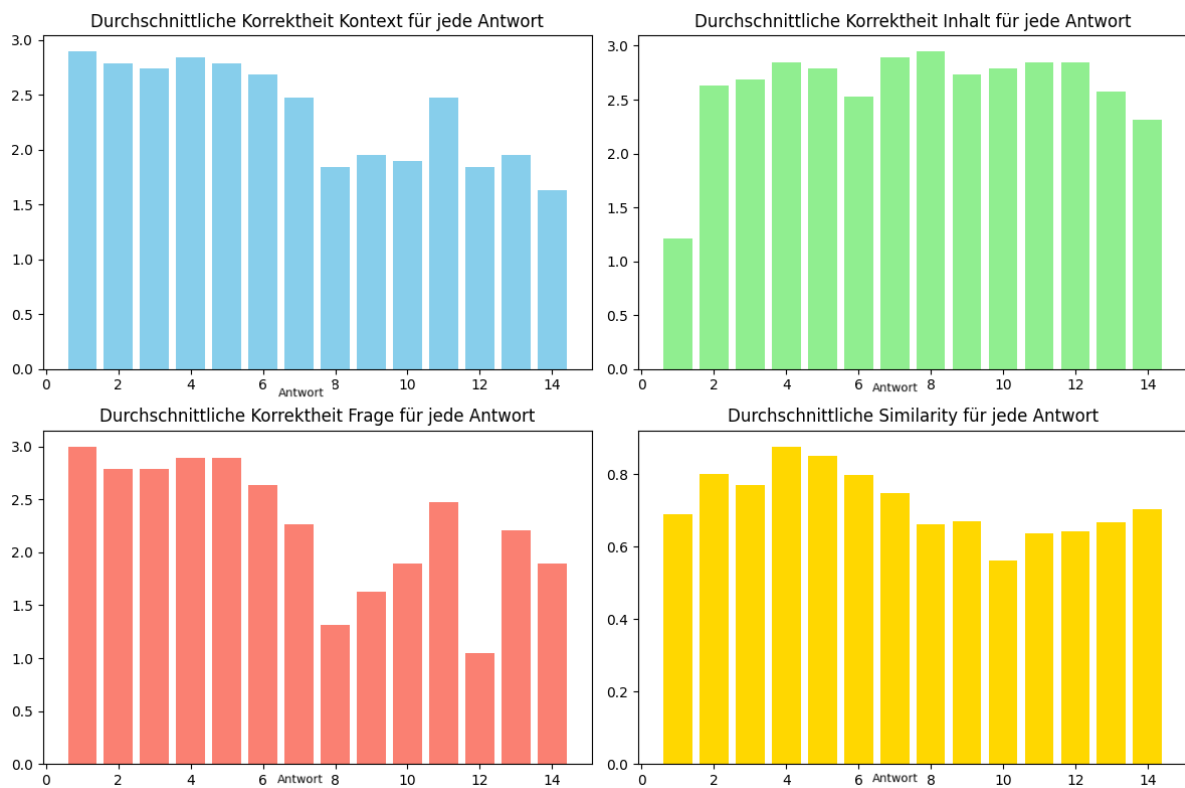


Abb. 4.4: Korrektheit für jede Antwort

Insgesamt lässt sich feststellen, dass die Integration von Chat-Engines im Vergleich zu Query-Engines eine deutliche Verbesserung der Antwortqualität und -leistung bietet. Insbesondere der Condense Plus Context-Modus erweist sich als effektive Methode zur Beantwortung von Fragen, während auch der normale Condense-Modus effizient ist, insbesondere unter optimalen Parametereinstellungen. Diese Erkenntnisse bieten wichtige Einblicke in die Leistungsfähigkeit verschiedener Chat-Engines und liefern Ansätze für deren Optimierung und Weiterentwicklung.

Für zukünftige Tests empfiehlt es sich, die spezifischen Parameter weiter zu verfeinern und auch den Condense Plus-Modus genauer zu untersuchen. Zusätzliche Einstellungen wie das sinnvolle Zurücksetzen der Chat-Engine könnten ebenfalls die Leistung und Benutzerfreundlichkeit verbessern. Die Anpassung von Parametern für verschiedene Modelle könnte die Leistungsfähigkeit weiter steigern. Obwohl im aktuellen Test nur das Neural Chat Language Model untersucht wurde, könnten zukünftige Studien verschiedene LLMs einbeziehen, um deren Reaktion auf unterschiedliche Parametereinstellungen zu evaluieren.

Es ist anzumerken, dass aufgrund der zeitaufwendigen manuellen Testphase nicht alle potenziellen Parameterkombinationen umfassend bewertet werden konnten. Für eine verbesserte Effizienz des Evaluierungsprozesses könnten zukünftige Ansätze die Einführung besserer automatisierter Metriken in Betracht ziehen. Darüber hinaus könnte die Erstellung einer erweiterten Menge von Testfragen die Genauigkeit der Ergebnisse erhöhen. Jedoch stieß dieser Ansatz aufgrund der manuellen Testprozedur und der langen Antwortgenerierungszeiten an seine Grenzen.

# 5 Verwendung

In diesem Kapitel wird die Einrichtung und Nutzung der Anwendung beschrieben. Es dient dazu, neuen Benutzern den Einstieg zu erleichtern und die verschiedenen Funktionen des Systems zu erklären.

## 5.1 Setup

### 5.1.1 Projekt einrichten

#### Projekt klonen

Um die Anwendung zu nutzen, muss zunächst das Repository geklont werden. Verwenden Sie dafür den folgenden Befehl im gewünschten Verzeichnis auf Ihrem lokalen Rechner:

```
$ git clone https://github.com/example/repo.git
```

#### Python Umgebung erstellen

Nachdem das Projekt geklont wurde, erstellen Sie eine virtuelle Python-Umgebung mit Conda, um Konflikte mit anderen Projekten oder der globalen Python-Installation zu vermeiden. Navigieren Sie zunächst in das erstellte Verzeichnis und geben sie den folgenden Befehl ein:

```
$ conda env create -f llm_studienprojekt_environment.yml
```

Hierdurch wird eine Python-Umgebung mit dem Namen *llm\_env* und der Python-Version 3.11.5 erzeugt und alle im *llm\_studienprojekt\_environment.yml*-File gelisteten Packages installiert. Die Umgebung wird mit dem folgendem Befehl aktiviert:

```
$ conda activate llm_env
```

## 5.1.2 Anwendung starten

### Visual Studio

Wollen Sie die Anwendung in Visual Studio ausführen, so öffnen Sie das Projekt in IDE. Navigieren sie hierfür über *File* und *Open Folder* zu dem entsprechenden Pfad, unter welchem Sie das Projekt abgespeichert haben.

Wählen Sie anschließend den zuvor erstellten Python-Interpreter aus. Hierfür müssen Sie über *View* und *Command Palette* zur Suche navigieren und nach *Python: Select Interpreter* suchen und *llm\_env* auswählen.

Abschließend kann das Projekt über die *main.py* ausgeführt werden.

### Terminal

Aktivieren Sie die Conda-Umgebung, navigieren Sie zum Verzeichnis *llm\_studienprojekt* und starten Sie die Anwendung mit den folgenden Befehlen:

```
$ cd llm_studienprojekt
```

```
$ python main.py
```

Stellen Sie zudem sicher, dass die richtige Python-Umgebung ausgewählt ist. Diese wird in der Kommandozeile angezeigt.

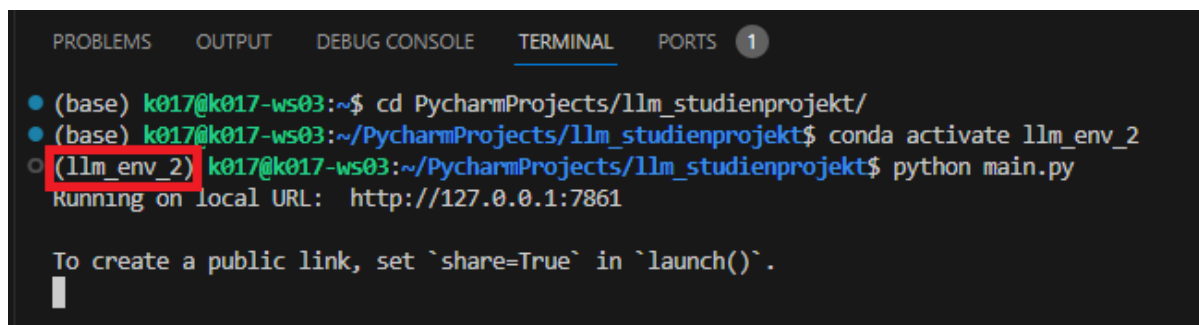
A screenshot of a terminal window within an IDE. The terminal shows a sequence of commands and their outputs. The first command is 'cd PycharmProjects/llm\_studienprojekt/'. The second command is 'conda activate llm\_env\_2', and the prompt changes to '(llm\_env\_2)'. The third command is 'python main.py', which outputs 'Running on local URL: http://127.0.0.1:7861'. Below this, there is a message: 'To create a public link, set `share=True` in `launch()`'. The terminal tabs at the top are 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is active), and 'PORTS'.

Abb. 5.1: Application Start via Terminal

## 5.2 Verwendung - UI

In Abbildung 5.2 ist die Benutzeroberfläche der LLM-basierten Suchmaschine zu sehen. Diese setzt sich aus drei Hauptkomponenten zusammen, dem Chatbot im oberen Teil des Bildes, der Archiv-Auswahl und Möglichkeit zum Einlesen neuer Daten im unteren Teil des Bildes.

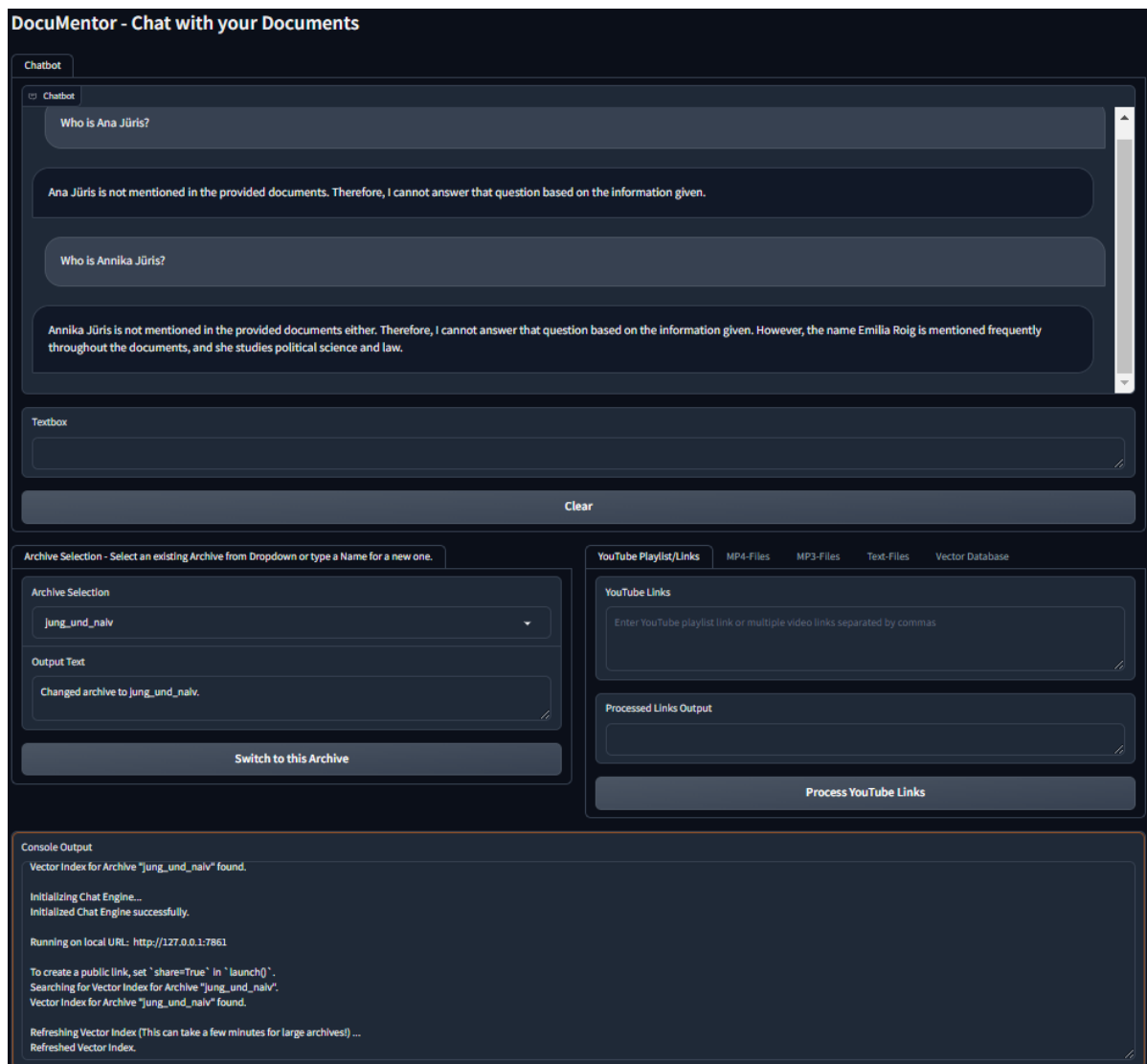


Abb. 5.2: Userinterface der LLM-basierten Suchmaschine

### 5.2.1 Archiv-Auswahl

Um die Anwendung verwenden zu können, muss vom Nutzer ein Archiv ausgewählt bzw. ein neues Archiv erstellt werden. Der Nutzer kann aus verschiedenen Archiven oder Datenquellen wählen, um die Antworten des Chatbots zu spezifizieren. Der Begriff „Archiv“ bezeichnet typischerweise eine Sammlung von Daten oder Dokumenten, die als Grundlage für die Erzeugung von Antworten oder die Durchführung spezifischer Aufgaben dient. Diese Daten werden in einer strukturierten Form gespeichert. Die Auswahl eines Archivs erfolgt in der Benutzeroberfläche über das Dropdown *Archive Selection* wie in Abbildung 5.3 zu sehen ist. Das Erstellen eines neuen Archivs erfolgt ebenfalls über das Dropdown, indem man in dieses einfach den

gewünschten Namen eingibt. Ist das gewünschte Archiv ausgewählt muss der „Switch to this Archive“-Button ausgewählt werden. Ist das Laden des Archives erfolgreich abgeschlossen, erhält man die Ausgabe „Changed archive to archive\_name“.

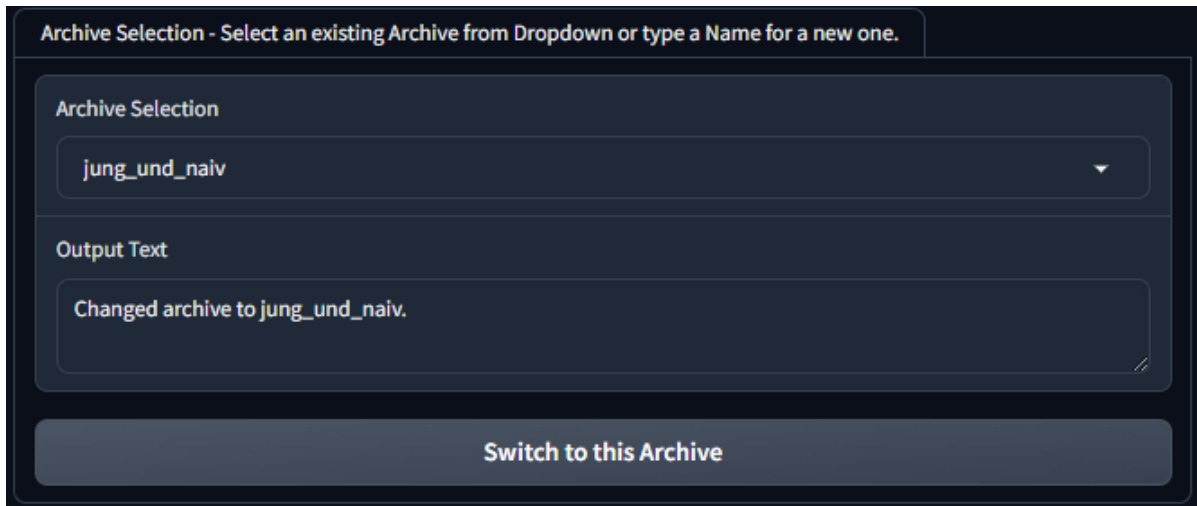


Abb. 5.3: Archiv Auswahl

### 5.2.2 Einlesen neuer Daten

Zudem besteht die Möglichkeit neue Daten in das System einzulesen, um das Wissen des Chatbots zu erweitern. Diese Informationen können entweder zu einem bestehenden Archiv hinzugefügt oder in einem neuen Archiv angelegt werden. Hierfür sind Entry-Points an jeder Stelle der Pipeline implementiert und in der Benutzeroberfläche abgebildet. Mögliche Dateiformate sind ein Link einer Youtube-Playlist, die Links eines oder mehrerer Youtube-Videos, MP4-Dateien, MP3-Dateien, Text-Files oder schon verarbeitete Daten, die in Form einer Vektor-Datenbank bereitgestellt werden. Sollen neue Informationen in die Anwendung eingelesen werden, wählt der Nutzer zuvor das gewünschte Archiv aus und tätigt anschließend die Eingabe in Form von Links/-Files. Anschließend wird der Prozess über den *Process*-Button gestartet.

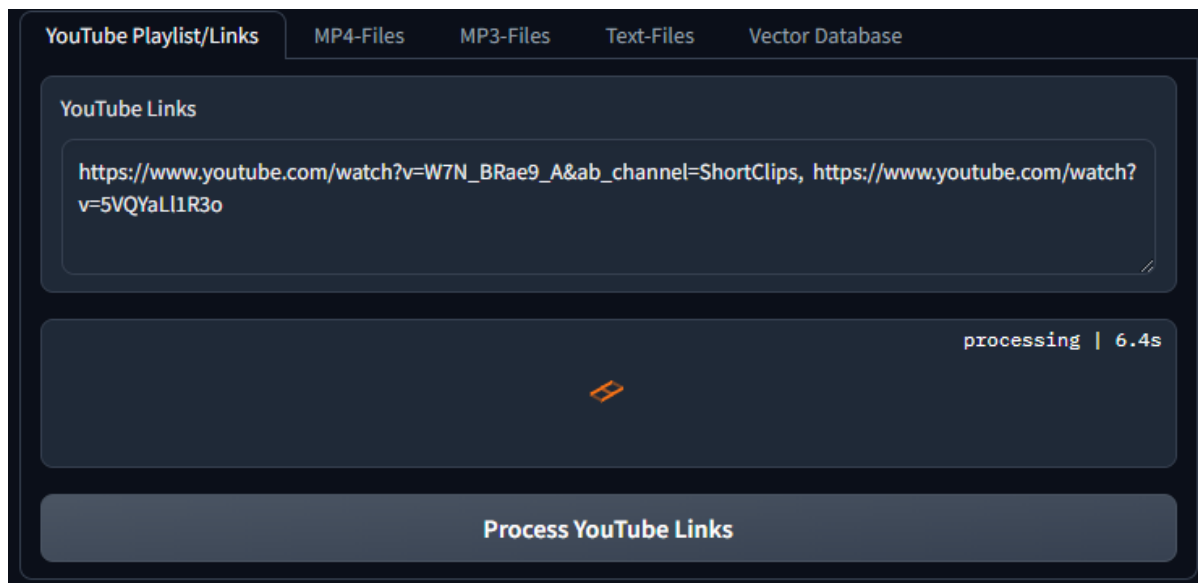


Abb. 5.4: Daten einlesen

### 5.2.3 Chatbot

Die Hauptkomponente der Anwendung ist der Chatbot, der in Abbildung 5.5 zu sehen ist. Nach dem Start der Anwendung können Nutzer Fragen stellen, auf die der Chatbot basierend auf dem trainierten Modell und den verfügbaren Daten antwortet. Zudem wird der Kontext der vorherigen Nachrichten berücksichtigt. Wird eine Frage gestellt, so versucht der Bot diese mit dem spezifischen Wissen, welches ihm über die Archive zur Verfügung gestellt wird, zu beantworten. Finden sich in diesen Daten keine Informationen, versucht er die Frage mit dem Weltwissen des LLMs zu beantworten. Die Nutzung des Bots erfolgt über die Eingabe der Frage und das Absenden über die *Enter*-Taste. Der *Clear*-Button dient dazu die Chat-Historie zu löschen und somit das LLM vom vorherigen Kontext zu bereinigen. Stellen Sie vor der Verwendung des Chatbots sicher, dass Sie zuvor ein Archiv ausgewählt haben (Kapitel 5.2.2) bzw. Daten eingelesen haben, um ein neues Archiv anzulegen (Kapitel 5.2.1).



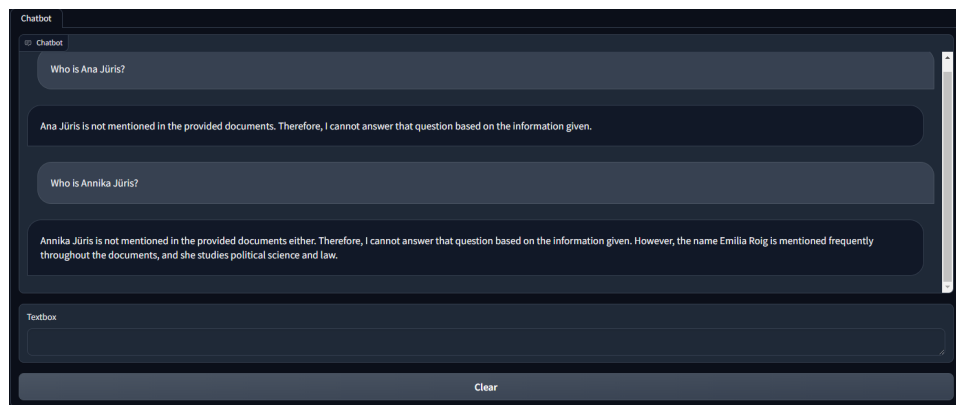


Abb. 5.5: Chatbot

## 6 Fazit

Das Projekt demonstriert, wie durch die Nutzung von Großen Sprachmodellen und fortschrittlichen Technologien ein innovativer Suchmaschinen-Prototyp für YouTube-Inhalte entwickelt wurde. Es hebt hervor, wie die Kombination aus Spracherkennung, Inhaltsindexierung und einer nutzerfreundlichen Chatbot-Schnittstelle das Extrahieren spezifischer Informationen aus Videoinhalten ermöglicht. Durch die effektive Transkription und Indexierung von Videomaterial wird eine präzise und schnelle Suche ermöglicht, die traditionelle Suchmethoden übertrifft und den Zugang zu Wissen erleichtert. Dieses Projekt stellt eine Verbesserung in der Art und Weise dar, wie Informationen in digitalen Medien gesucht werden können und mit diesen interagiert werden kann.

Das Projekt wurde nicht nur erfolgreich abgeschlossen, sondern bot auch eine wertvolle Erfahrung für das Team, indem es die Möglichkeit eröffnete, sich mit neuen Technologien vertraut zu machen. Der Einsatz von Tools wie Hugging Face ermöglichte eine Vertiefung des Fachwissens in Künstlicher Intelligenz und Maschinellem Lernen. Das Projekt forderte dazu heraus, kreative Lösungen für komplexe Herausforderungen zu entwickeln und erlaubte es, theoretisches Wissen aus Vorlesungen praktisch anzuwenden. Die Teamarbeit brachte jedoch auch Herausforderungen mit sich, insbesondere aufgrund der begrenzten Ressourcen, da nur ein PC mit einer GPU zur Verfügung stand.

Für zukünftige Verbesserungen könnte eine systematische Anpassung und Feinabstimmung der Parameter in ServiceContext, Chat Engines und anderen relevanten Systemkomponenten vorgenommen werden, um diese optimal für den spezifischen Anwendungsfall zu konfigurieren. Dies könnte nicht nur die Effizienz und Genauigkeit des Suchmaschinen-Prototyps weiter steigern, sondern auch die Benutzererfahrung verbessern.

# Literaturverzeichnis

- [aws23] aws. Was ist eine Vektordatenbank?, zuletzt zugegriffen: 07.02.2024. <https://aws.amazon.com/de/what-is/vector-databases/>, 2023.
- [FFm23] FFmpeg. FFmpeg Security, zuletzt zugegriffen: 08.02.2024. <https://ffmpeg.org/>, 2023.
- [Gra24] Gradio. Gradio, zuletzt zugegriffen: 08.02.2024. <https://www.gradio.app/>, 2024.
- [Hug23a] Hugging Face. Intel\_neural-chat-7b-v3-1 · Hugging Face, zuletzt zugegriffen: 07.02.2024. <https://huggingface.co/Intel/neural-chat-7b-v3-1/>, 2023.
- [Hug23b] Hugging Face. meta-llama.Llama-2-7b-chat-hf · Hugging Face, zuletzt zugegriffen: 07.02.2024. <https://huggingface.co/meta-llama/Llama-2-7b-chat-hf>, 2023.
- [Hug23c] Hugging Face. Mistral-7B-Instruct-v0.1 · Hugging Face, zuletzt zugegriffen: 07.02.2024. <https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1>, 2023.
- [Hug23d] Hugging Face. Mistral-7B-Instruct-v0.2 · Hugging Face, zuletzt zugegriffen: 07.02.2024. <https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>, 2023.
- [Hug23e] HuggingFace. BERT, zuletzt zugegriffen: 07.02.2024. [https://huggingface.co/docs/transformers/model\\_doc/bert](https://huggingface.co/docs/transformers/model_doc/bert), 2023.
- [Hug d] Hugging Face. Hugging Face - The AI community building the future., zuletzt zugegriffen: 07.02.2024. <https://huggingface.co/>, o. d.

- [Lla23a] LlamaIndex. Building A LabelledRagDataset, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/stable/module\\_guides/evaluating/evaluating\\_with\\_llamadatasets.html#building-a-labelledragdataset](https://docs.llamaindex.ai/en/stable/module_guides/evaluating/evaluating_with_llamadatasets.html#building-a-labelledragdataset), 2023.
- [Lla23b] LlamaIndex. Chat Engine - Condense Plus Context Mode, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/latest/examples/chat\\_engine/chat\\_engine\\_condense\\_plus\\_context.html](https://docs.llamaindex.ai/en/latest/examples/chat_engine/chat_engine_condense_plus_context.html), 2023.
- [Lla23c] LlamaIndex. Chat Engine - Condense Question Mode, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/latest/examples/chat\\_engine/chat\\_engine\\_condense\\_question.html#chat-engine-condense-question-mode](https://docs.llamaindex.ai/en/latest/examples/chat_engine/chat_engine_condense_question.html#chat-engine-condense-question-mode), 2023.
- [Lla23d] LlamaIndex. Chat Engine, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/latest/module\\_guides/deploying/chat\\_engines/root.html](https://docs.llamaindex.ai/en/latest/module_guides/deploying/chat_engines/root.html), 2023.
- [Lla23e] LlamaIndex. Correctness Evaluator, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/stable/examples/evaluation/correctness\\_eval.html](https://docs.llamaindex.ai/en/stable/examples/evaluation/correctness_eval.html), 2023.
- [Lla23f] LlamaIndex. Faithfulness Evaluator, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/stable/examples/evaluation/faithfulness\\_eval.html](https://docs.llamaindex.ai/en/stable/examples/evaluation/faithfulness_eval.html), 2023.
- [Lla23g] LlamaIndex. Query Engine Engine, zuletzt zugegriffen: 07.02.2024. [https://docs.llamaindex.ai/en/latest/module\\_guides/deploying/query\\_engine/root.html](https://docs.llamaindex.ai/en/latest/module_guides/deploying/query_engine/root.html), 2023.
- [Lla23h] LlamaIndex Inc. LlamaIndex - Data Framework for LLM Applications, zuletzt zugegriffen: 08.02.2024. <https://www.llamaindex.ai/>, 2023.
- [Mic23] Microsoft. Retrieval Augmented Generation (RAG) in Azure AI Search, zuletzt zugegriffen: 07.02.2024. <https://learn.microsoft.com/en-us/azure/search/retrieval-augmented-generation-overview>, 2023.
- [Ope23] OpenAI. Introducing Whisper, zuletzt zugegriffen: 08.02.2024. <https://openai.com/research/whisper>, 2023.

- [Pyt23] Pytube. pytube – pytube 15.0.0 documentation, zuletzt zugegriffen: 08.02.2024. <https://pytube.io/en/latest/>, 2023.
- [Pyt24] Python Software Foundation. shutil — High-level file operations – Python 3.12.2 documentenation, zuletzt zugegriffen: 08.02.2024. <https://docs.python.org/3/library/shutil.html>, 2024.

# Abbildungsverzeichnis

2.1	RAG Diagramm . . . . .	4
2.2	Vector Store Diagram . . . . .	5
3.1	Architektur-Schema . . . . .	9
3.2	Userinterface der LLM-basierten Suchmaschine . . . . .	15
4.1	Funktionsweise einer Query Engine . . . . .	23
4.2	Funktionsweise einer Chat Engine . . . . .	24
4.3	Korrektheit für jede Engine . . . . .	29
4.4	Korrektheit für jede Antwort . . . . .	30
5.1	Application Start via Terminal . . . . .	33
5.2	Userinterface der LLM-basierten Suchmaschine . . . . .	34
5.3	Archiv Auswahl . . . . .	35
5.4	Daten einlesen . . . . .	36
5.5	Chatbot . . . . .	37

## Code Listings

3.1	Verwendung der Transformers-Bibliothek . . . . .	6
3.2	Initialisieren des Service-Kontexts . . . . .	12
3.3	Initialisierung der Chat-Engine . . . . .	13
3.4	Erstellen des Vektorindex . . . . .	13
4.1	Generierung des Fragen-Datensatzes . . . . .	18
4.2	CorrectnessEvaluator: Internes System Template (llama_index\evaluation textbackslash correctness.py) . . . . .	19
4.3	FaithfulnessEvaluator: Prompt Template (llama_index\evaluation text- backslash evaluation.py) . . . . .	20