

# Vorabfragen

- Stellen Sie sich vor, Sie möchten einen Chatbot entwickeln, der Fragen zu PDF-Dokumenten beantwortet. Welche Teilprobleme müssten Sie Ihrer Meinung nach lösen?



# Deep Learning, Machine Learning und Künstliche Intelligenz – SS 26

Gelesen von Prof. Dr. Daniel Gaida

02.05.2026

Prof. Dr. Daniel Gaida

Professor für Cyber-Physische Systeme

Fakultät für Informatik und Ingenieurwissenschaften - Institut für Informatik

Seite 2

**Technology**  
**Arts Sciences**  
**TH Köln**

# Lernziele von heute und Fragen zur Überprüfung der Lernziele

Die Studierenden können sprachbasierte User Interfaces in Python entwickeln, indem sie

- für eine gegebene Anwendung ein geeignetes vortrainiertes Deep Learning Modell (LLM, Text2Speech, Speech2Text) auswählen und herunterladen,
- mit Gradio eine App unter Einbindung des Deep Learning Modells in Python programmieren,
- die App testen,

um später sprachbasierte User Interfaces unter Nutzung von Deep Learning Modellen umsetzen zu können.

Überprüfung des Lernziels: S. fortlaufende Übung heute.

# Lernraum II: Deep Learning

- Inhalt
  - Entwickeln eines Chatbots mit Spracherkennung und -synthese
  - Text2Image, Text2Video, Image2Text, Video2Text, ...

# Chatbot Entwicklung

- Chatbot mit dem man beliebig formulierte Fragen zu einem oder mehreren Dokumenten stellen kann.

## PDF RAG Chatbot

Ask questions about the content of your PDF documents.

Chatbot

What is this PDF about?

Summarize the second section.

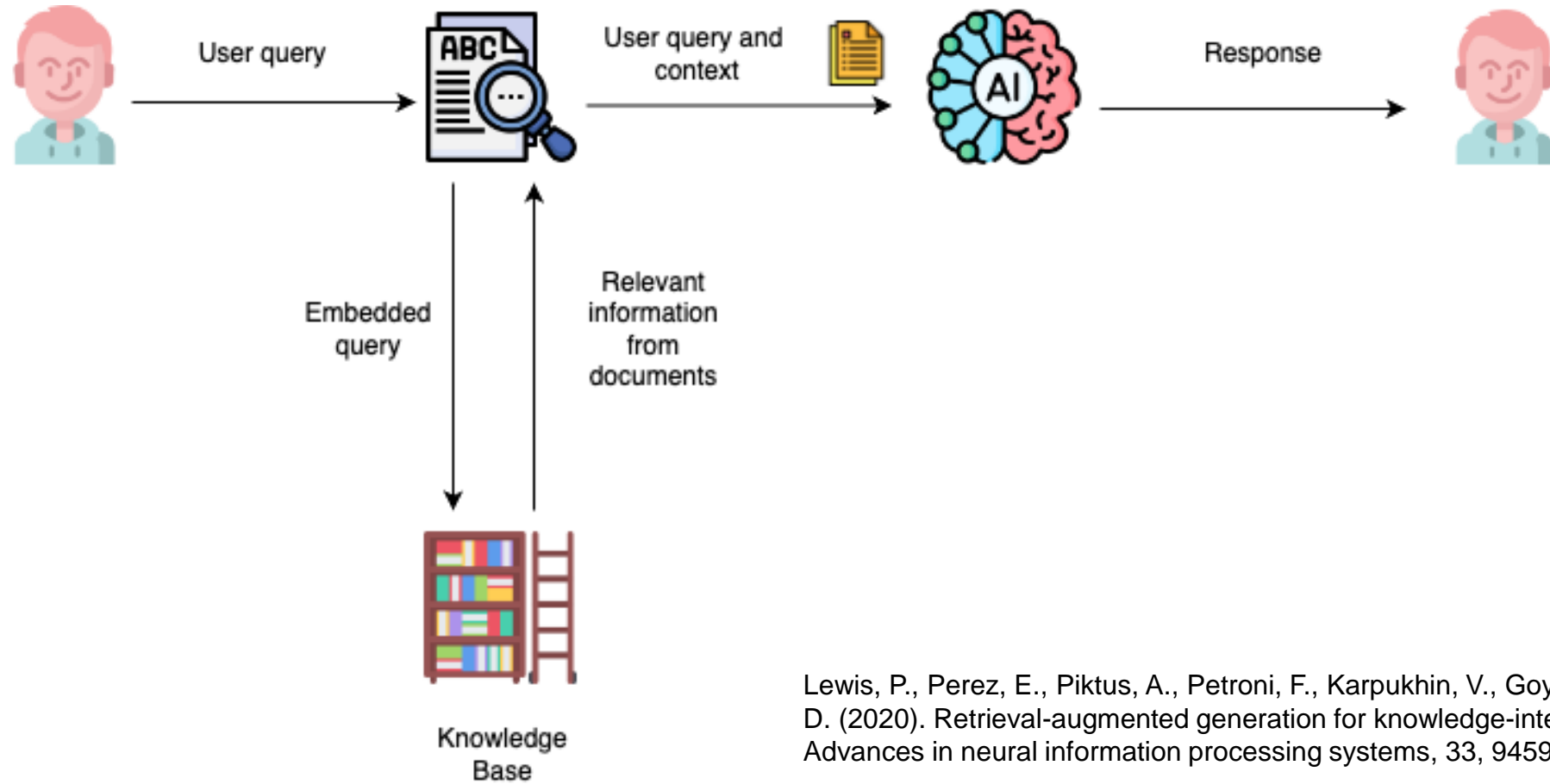
02.05.2026

Seite 5

Type a message...



# Retrieval Augmented Generation (RAG)



Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33, 9459-9474.

# Retrieval Augmented Generation

1. Erstellung der Wissensdatenbank
2. Relevante Inhalte aus Wissensdatenbank zurück liefern und an LLM übergeben
3. Antwort von LLM generieren lassen

# Erstellung der Wissensdatenbank

- Wissensdatenbank kann erstellt werden aus:
  - PDFs, Markdown, txt, docx, pptx, html, ...
- Dokument Parser wird benötigt
  - Es gibt viele verschiedene; Hier: Unstructured.io genutzt
- Dokument muss in kleine Abschnitte geteilt werden (bspw. Sätze, Absätze, ...)
  - Damit zur Suchanfrage relevante Abschnitte gefunden werden können
- Alle Dokumentabschnitte werden in einer Datenbank gespeichert
- → Wissensdatenbank

# Dokument Parser – Unstructured.io

- <https://github.com/Unstructured-IO/unstructured-api>

## General Pre-Processing Pipeline for Documents

This repo implements a pre-processing pipeline for the following documents. Currently, the pipeline is capable of recognizing the file type and choosing the relevant partition function to process the file.

| Category  | Document Types  |
|-----------|---|
| Plaintext | <code>.txt</code> , <code>.eml</code> , <code>.msg</code> , <code>.xml</code> , <code>.html</code> , <code>.md</code> , <code>.rst</code> , <code>.json</code> , <code>.rtf</code>                        |
| Images    | <code>.jpeg</code> , <code>.png</code>  |
| Documents | <code>.doc</code> , <code>.docx</code> , <code>.ppt</code> , <code>.pptx</code> , <code>.pdf</code> , <code>.odt</code> , <code>.epub</code> , <code>.csv</code> , <code>.tsv</code> , <code>.xlsx</code> |
| Zipped    | <code>.gz</code>  |

# Dokument Parser

```
# --- Step 1: Set up PDF file path ---  
# Put your PDF files in a folder named "pdfs" in the current directory  
PDF_DIR = "pdfs"  
  
# --- Step 2: Load and parse PDFs with uMiner (via UnstructuredReader) ---  
all_documents = utils.read_pdf_files_with_unstructured_reader(PDF_DIR)  
  
print(f"Loaded {len(all_documents)} documents")  
for doc in all_documents[:3]:  
    print(doc.text[:300]) # Zeige ersten Ausschnitt
```

# Erste Schritte mit dem Notebook

Gehen Sie zu: [https://dgaida.github.io/llm\\_client/dev/tutorials/rag-chatbot/](https://dgaida.github.io/llm_client/dev/tutorials/rag-chatbot/)

Klicken Sie auf: [https://github.com/dgaida/llm\\_client/blob/master/notebooks/RAGChatbot\\_groq\\_API.ipynb](https://github.com/dgaida/llm_client/blob/master/notebooks/RAGChatbot_groq_API.ipynb)  
und öffnen Sie das Notebook in Google Colab

# Erste Schritte mit dem Notebook

- Notebook „RAGChatbot\_groq\_API“ in Google Colab öffnen
- Ordner „pdfs“ erstellen und mindestens ein PDF Dokument einfügen
- Erste Zelle des Notebooks ausführen (installiert alle Packages)
  - Es erscheint eine Meldung, dass Sitzung neu gestartet werden muss
  - Auf Button „Restart Session“ klicken

```
Uninstalling nvidia-cusolver-cu12-11.6.3.83:  
Successfully uninstalled nvidia-cusolver-cu12-  
Successfully installed XlsxWriter-3.2.3 aiofiles-24.  
WARNING: The following packages were previously imported in this runtime:  
[google,importlib_metadata]  
You must restart the runtime in order to use newly installed versions.
```

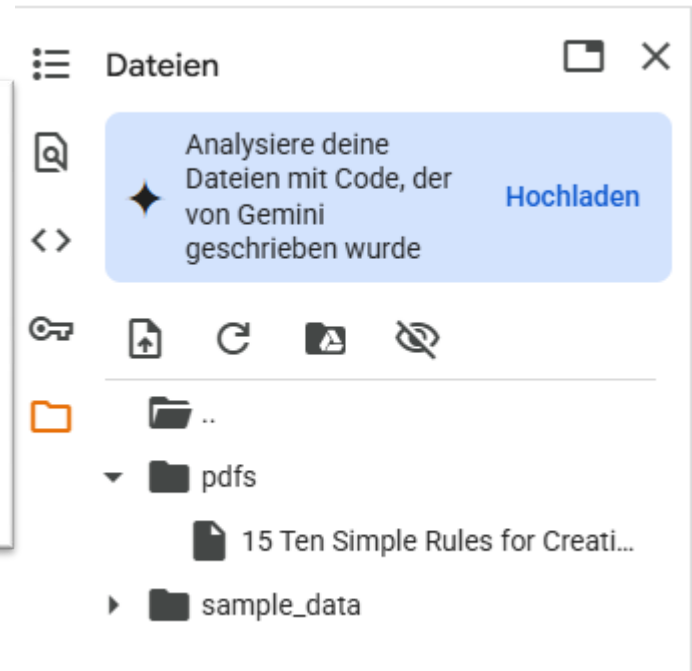
RESTART SESSION

### Sitzung neu starten

WARNING: The following packages were previously imported in this runtime:  
[google,importlib\_metadata]  
You must restart the runtime in order to use newly installed versions.

Restarting will lose all runtime state, including local variables.

Abbrechen [Sitzung neu starten](#)



# Erste Schritte mit dem Notebook

- Drei weitere Zellen ausführen bis einschließlich:
  - Installation von `llm_client`, alle imports, Dokument(e) einlesen

```
▶ # --- Step 2: Load and parse PDFs with uMiner (via UnstructuredReader) ---  
all_documents = utils.read_pdf_files_with_unstructured_reader(PDF_DIR)  
  
print(f"Loaded {len(all_documents)} documents")  
for doc in all_documents[:3]:  
    print(doc.text[:300]) # Zeige ersten Ausschnitt
```

# LlamaIndex

- UnstructuredReader wird von LlamaIndex importiert
- <https://www.llamaindex.ai/>

## Build AI Knowledge Assistants over your enterprise data

Build production agents that can find information, synthesize insights, generate reports, and take actions over the most complex enterprise data.

GET STARTED

CONTACT SALES

# Dokument in Abschnitte teilen

```
node_parser = SentenceSplitter(chunk_size=256, chunk_overlap=0)
```

Was bedeutet `chunk_size` und `chunk_overlap`?

- `chunk_size`: Größe eines Textabschnitts gemessen in Anzahl Tokens (hier 256 Tokens)
- `chunk_overlap`: Überlapp zwischen zwei aufeinander folgenden Textabschnitten (hier 0 Token)

# Dokument in Abschnitte teilen

## Chunk Overlap

The image shows a web-based interface for splitting a document into chunks. At the top right, there is an "Upload .txt" button. The main control area includes a dropdown menu set to "Character Splitter", a "Chunk Size" input field with a value of 256 and a slider, and a "Chunk Overlap" input field with a value of 0 and a slider. Below these controls, a document text is displayed with several segments highlighted in different colors (blue, yellow, green, pink). To the right of the text, summary statistics are shown: "Total Characters: 3258", "Number of chunks: 13", and "Average chunk size: 250.6". A second "Upload .txt" button is located on the far right.

# Dokument in Abschnitte teilen

## Chunk Overlap

Ohne chunk\_overlap:

Hans geht nach Hause. Hans freut sich auf den Abend. Der Grund ist, dass er sich noch mit Freunden treffen wird.

Warum freut sich Hans auf den Abend?

Mit chunk\_overlap:

Hans geht nach Hause. Hans freut sich auf den Abend. Der Grund ist, dass er sich noch mit Freunden treffen wird.

# Erste Schritte mit dem Notebook

- Eine weitere Zellen ausführen bis einschließlich:
  - Dokument(e) in Chunks einteilen

```
▶ # --- Step 2: Load and parse PDFs with uMiner (via UnstructuredReader) ---  
all_documents = utils.read_pdf_files_with_unstructured_reader(PDF_DIR)  
  
print(f"Loaded {len(all_documents)} documents")  
for doc in all_documents[:3]:  
    print(doc.text[:300]) # Zeige ersten Ausschnitt
```

```
# split documents into smaller chunks for better retrieval  
node_parser = SentenceSplitter(chunk_size=256, chunk_overlap=0) # 512, 50
```

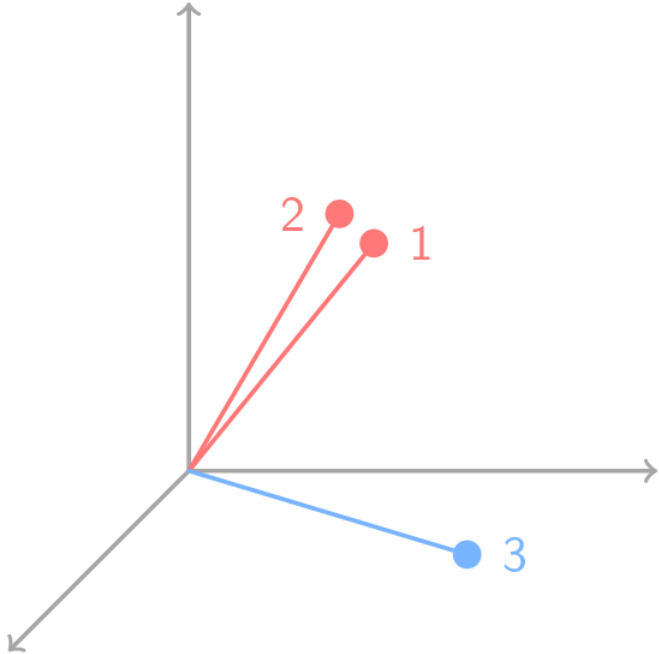
# Erstellung der Wissensdatenbank

- Ziel: Wir wollen in der Wissensdatenbank nach Chunks suchen, die inhaltlich zu der Frage des Nutzers passen. D.h.:
  - Eine **semantische Suche** soll durchgeführt werden!
- Vorteil einer semantischen Suche vs. Keyword-basierte Suche?
  - Semantische Suche interpretiert die Bedeutung einer Anfrage, statt nur exakte Schlagwörter zu matchen  
(z. B. "Wie groß ist ein Elefant?" → findet auch "Durchschnittliche Höhe eines Rüsseltiers").
  - Flexiblere Formulierungen: Liefert auch bei umformulierten oder synonymen Anfragen relevante Ergebnisse – Keyword-basierte Suche scheitert oft daran.
  - Höhere Relevanz der Treffer und weniger irrelevante Ergebnisse.

# Wie misst man die semantische Ähnlichkeit von Chunks?

## Sentence Embedding Example

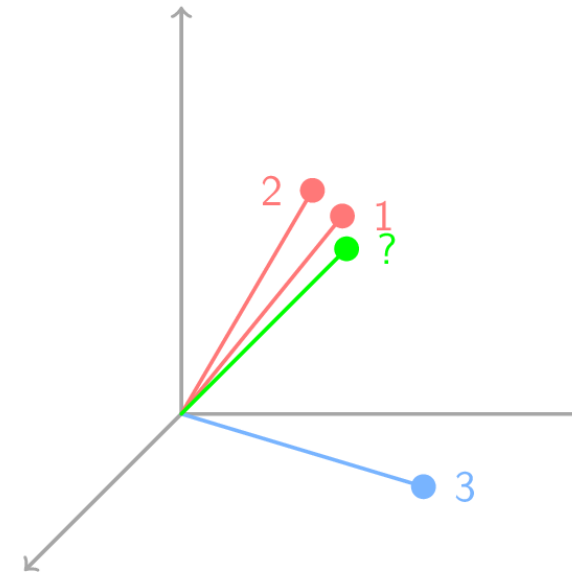
- 1. The cat slept on the warm couch.
- 2. The kitten rested beside the fireplace.
- 3. Bayern Munich won the Champions League.



# Chunks finden, die semantisch ähnlich zur Nutzerfrage sind

## Sentence Embedding Example

- 1. The cat slept on the warm couch.
- 2. The kitten rested beside the fireplace.
- 3. Bayern Munich won the Champions League.
- What did the cat do?



# Erstellung der Wissensdatenbank

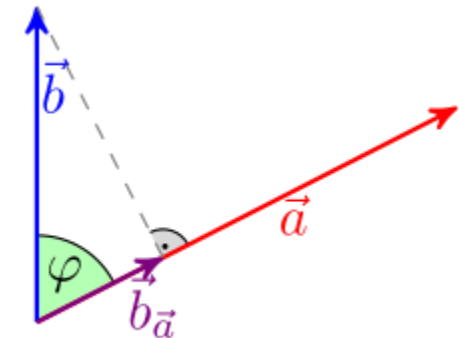
- Alle Dokumentabschnitte (Chunks) werden in einer Vektordatenbank gespeichert
  - Ermöglicht eine semantische Suche
- Vektordatenbank:
  - Schlüssel ist der Vektor
  - Wert ist der Chunk

Anmerkung: sehr vereinfachte Darstellung einer Vektordatenbank

| Key as Vector         | Value                                   |
|-----------------------|---|
| (0.1, 0.23, 0.5, ...) | The cat slept on the warm couch         |
| (0.8, 0.1, 0.25, ...) | Bayern Munich won the Champions League. |

# Erstellung der Wissensdatenbank

- Alle Dokumentabschnitte (Chunks) werden in einer Vektordatenbank gespeichert
  - Ermöglicht eine semantische Suche
    - Chunks müssen als Vektoren gespeichert werden
    - Ähnlichkeit zwischen Vektor der Suchanfrage und Vektoren in Datenbank
    - Chunks zu Vektoren, die ähnlich zu dem Vektor der Suchanfrage sind, werden zurück geliefert
- Warum wandelt man einen Satz in einen Vektor um?
  - Vektoren kann man besser miteinander vergleichen als Sätze
  - Bspw. über das Skalarprodukt (Cosine similarity)
- Wie wandelt man einen Satz in einen Vektor um?



# Von Dokumentabschnitten (Chunks) zu Vektoren

Im Bereich der Textverarbeitung/-generierung wird ...

- Text in Wörter oder Teilwörter unterteilt (Tokens, Tokenization)
  - Tokens sind das Vokabular
- jedes Token durch ein Deep Learning Modell in einen Vektor umgewandelt
  - Deep Learning Modell heißt **Embedding Modell** und
  - Vektoren heißen **Embedding Vektoren**
- Man kann auch einen ganzen Satz (bzw. Chunk) mit einem Embedding Modell in einen Vektor umwandeln
  - Vektor repräsentiert die Semantik (Bedeutung) des Satzes/Chunks

# Tokenization

- Demo: <https://tiktokenizer.vercel.app/>

## Tiktokenizer

```
Hello World.
```

cl100k\_base

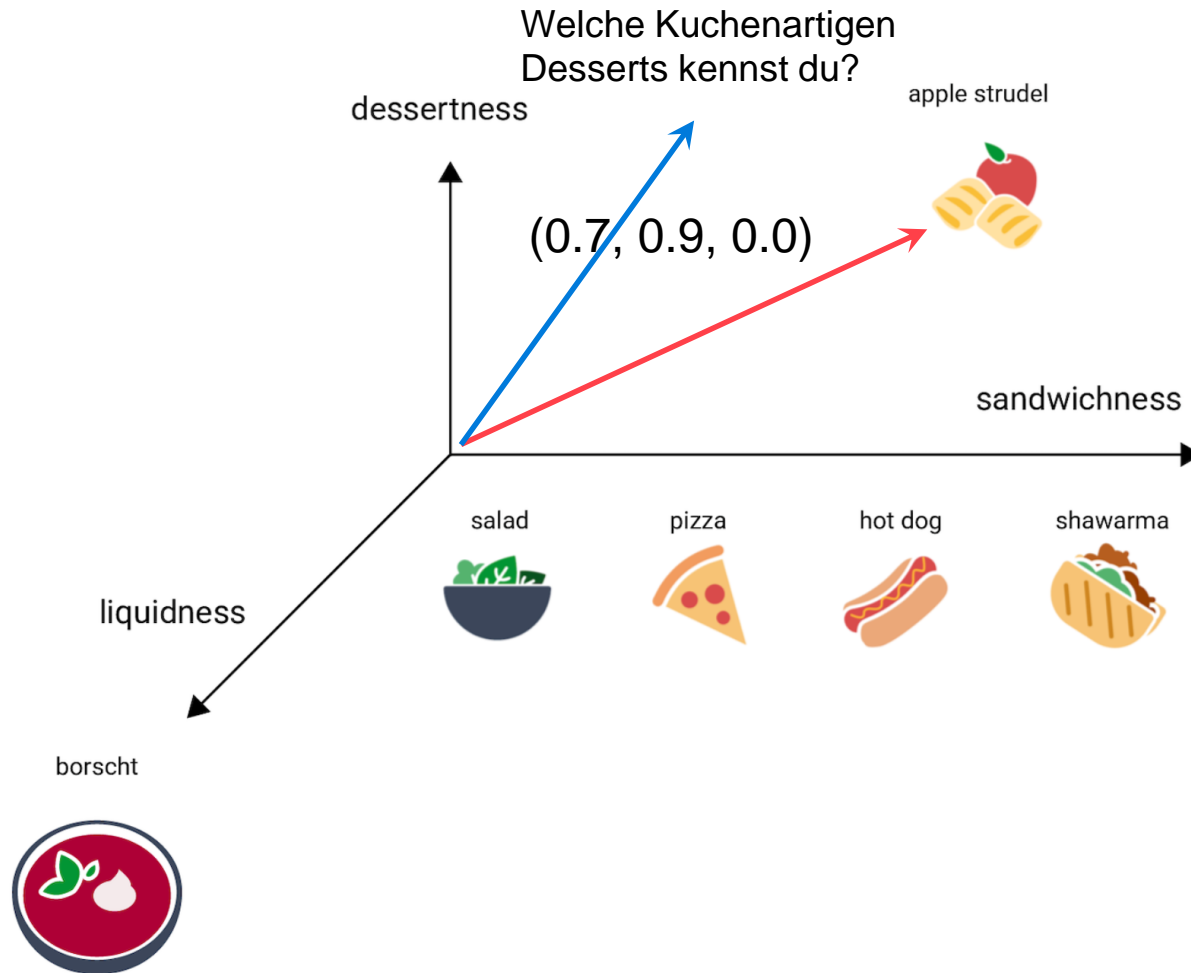
Token count  
3

```
Hello World.
```

9906, 4435, 627

Show whitespace

# From Tokens 2 Embeddings (Embedding Vektoren)

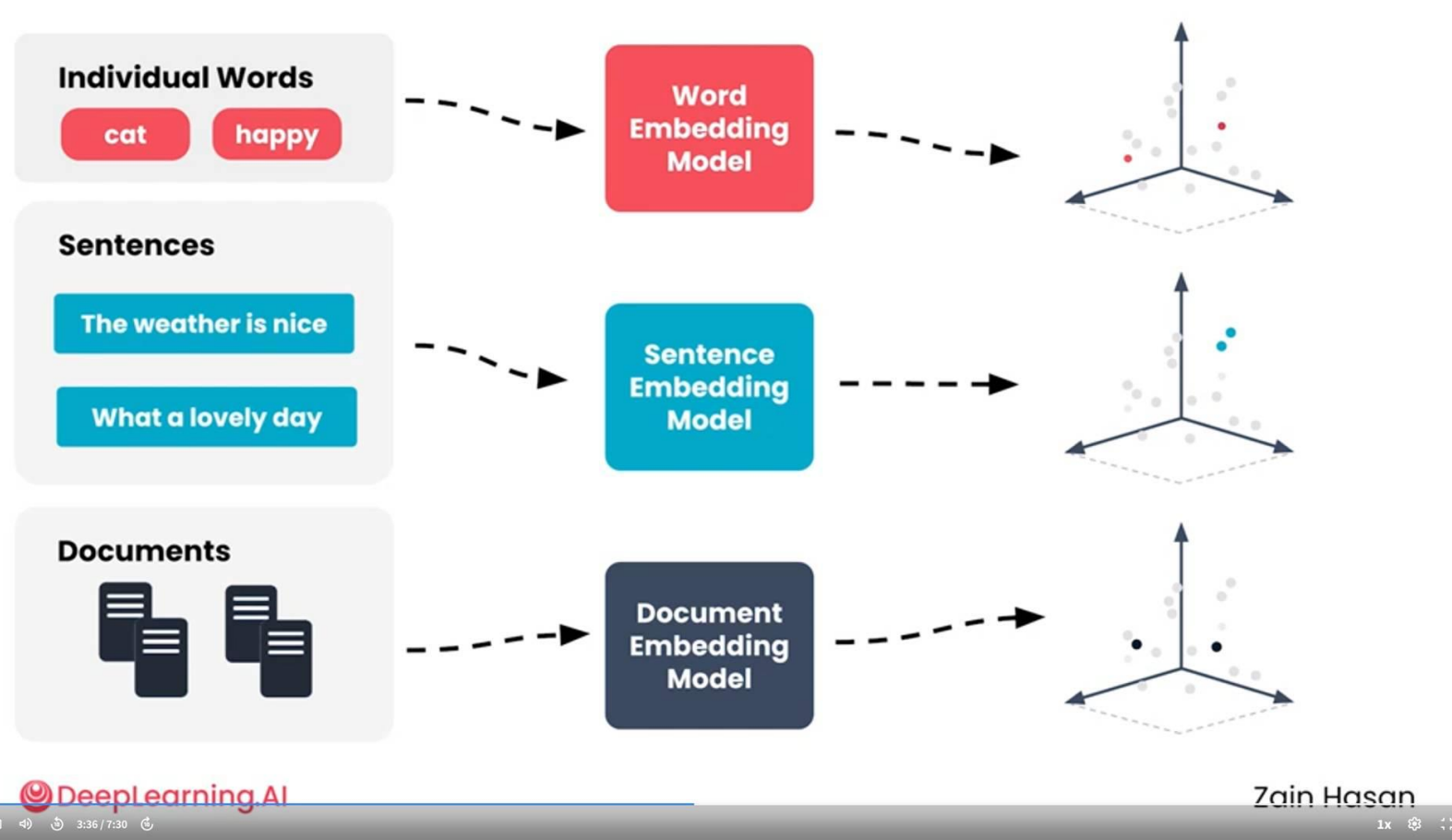


- Vektoren leben in einem hochdimensionalen Vektorraum
  - Bspw. 1024, unsere in 384 Dimensionen
- Hier Annahme:
  - 3-dimensionaler Vektorraum

Figure 5. Foods plotted by "sandwichness," "dessertness," and "liquidness."

# Embedding Modell

## Wandelt Token in Embedding Vektor um



# Embedding Modell angeben

## Wandelt Token in Embedding Vektor um

```
embed_model = HuggingFaceEmbedding("sentence-transformers/all-MiniLM-L6-v2")
```

“Our model is intended to be used as a sentence and short paragraph encoder. Given an input text, it outputs a vector which captures the semantic information. The sentence vector may be used for information retrieval, clustering or sentence similarity tasks.” <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

```
embed_model = HuggingFaceEmbedding(model_name = "intfloat/e5-small-v2")
```

„This model only works for English texts. Long texts will be truncated to at most 512 tokens.“

<https://huggingface.co/intfloat/e5-small-v2>

Embedding Modelle:

- <https://platform.openai.com/docs/guides/embeddings>

# Einrichtung eines Accounts bei Hugging Face (1/2)

- <https://huggingface.co/> und Sign Up
- Gehen Sie dann zu Ihrem Profil, klicken auf Settings und dann auf „Access Tokens“
- Erstellen Sie ein neues „Write“ Token mit einem sinnvollen Namen

## < Create new Access Token

### Token type

Fine-grained Read **Write**

ⓘ This cannot be changed after token creation.

### Token name

This token has read and write access to all your and your orgs resources and can make calls to Inference Providers on your behalf.

Create token

Profile

Account

Authentication

Organizations

Billing

**Access Tokens**

SSH and GPG Keys


Inference Providers

Quelle: <https://huggingface.co/>

# Einrichtung eines Accounts bei Hugging Face (2/2)

- Gehen Sie zu Google Colab und erstellen Sie dort ein neues „Secret“ Namens „HF\_TOKEN“ und kopieren Sie das erstellte Token hinein (beginnt mit „hf\_...“)

 Secrets  

 Konfigurieren Sie Ihren Code, indem Sie Umgebungsvariablen, Dateipfade oder Schlüssel speichern. Die hier gespeicherten Werte sind privat und nur für Sie und auf von Ihnen ausgewählten Notebooks sichtbar.

 Der geheime Name darf keine Leerzeichen enthalten.

[https://github.com/dgaida/llm\\_client/tree/master/notebooks#%EF%B8%8F-api-keys-als-secrets-in-google-colab-hinterlegen](https://github.com/dgaida/llm_client/tree/master/notebooks#%EF%B8%8F-api-keys-als-secrets-in-google-colab-hinterlegen)

| Notebook-Zugriff  | Name     | Wert  | Aktionen  |
|---|----------|-------|---|
|  | HF_TOKEN | ..... |    |

# Erstellung der Wissensdatenbank

- Alle Dokumentabschnitte werden in einer Vektordatenbank gespeichert
- Es gibt viele Vektordatenbanken
- Hier: Chroma genutzt

Chroma is the open-source AI application database. Batteries included.

Embeddings, vector search, document storage, full-text search, metadata filtering, and multi-modal. All in one place. Retrieval that just works. As it should be.

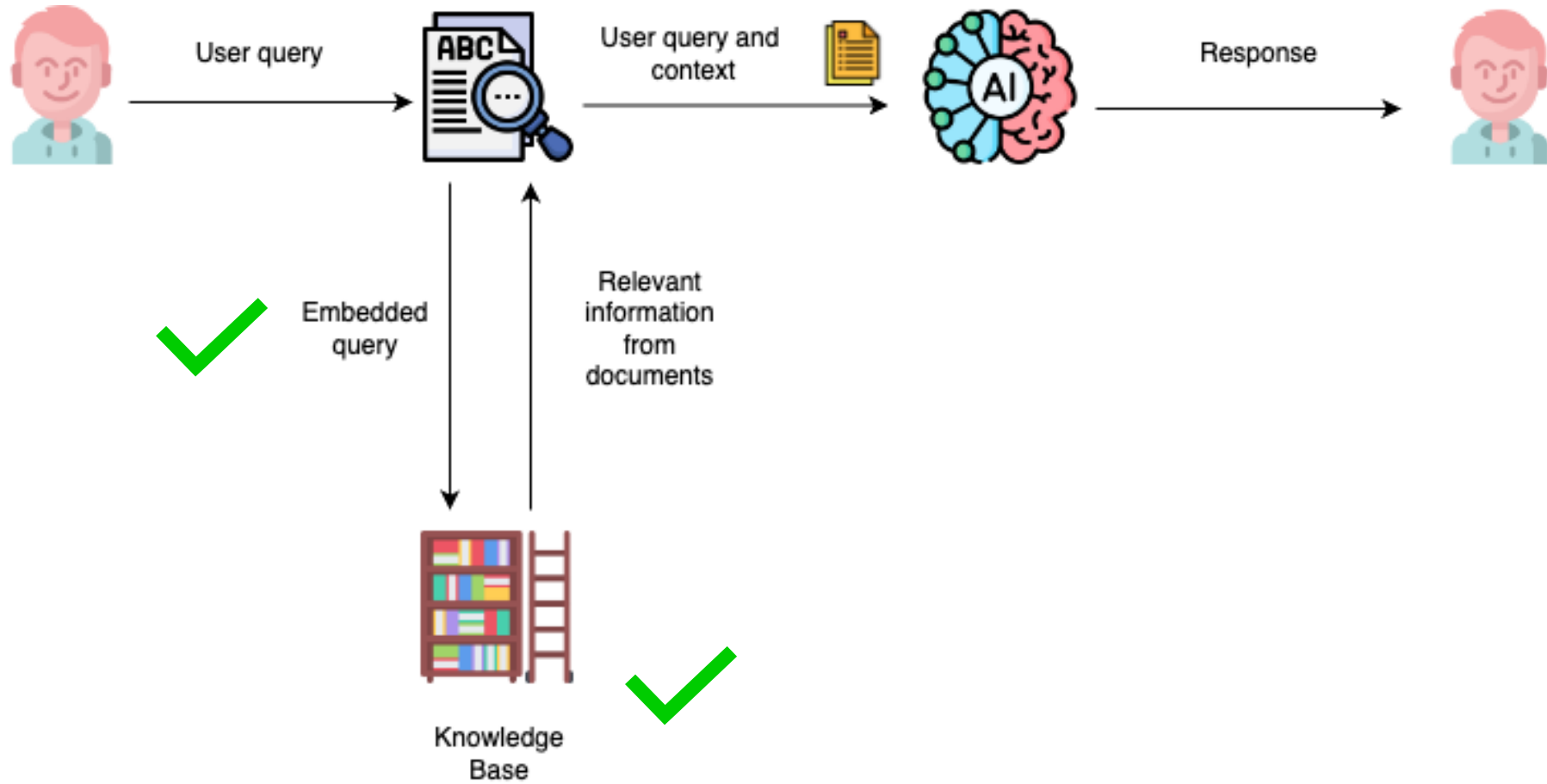
# Dokumentabschnitte in Vektordatenbank speichern - Chroma

# --- Step 4: Create vector store and index ---

```
index = utils.create_chromadb_vector_store_and_index(all_documents)
```

| Key                   | Value   |
|-----------------------|---|
| (0.1, 0.23, 0.5, ...) | The project aims to train sentence embedding models ... |
| (0.8, 0.1, 0.25, ...) | We fine-tune the model using a contrastive objective.   |

# Retrieval Augmented Generation (RAG)



# Wo läuft das Large Language Model (LLM)?

- LLM läuft in der Cloud, bereitgestellt über die Groq API
- Einige Suchanfragen pro Tag kostenlos
- Alle möglichen modernen Open Source LLMs nutzbar
- Benötigt einen kostenfreien Account bei Groq:
  - S.: [https://dgaida.github.io/llm\\_client/dev/tutorials/rag-chatbot/#groq-api-key-erstellen](https://dgaida.github.io/llm_client/dev/tutorials/rag-chatbot/#groq-api-key-erstellen)
- Verbindung zu Groq definiert in Klasse „LLMClient“
- Einfach änderbar zu:
  - OpenAI API (api\_choice = „openai“)
  - Google API (api\_choice = „gemini“)

# Large Language Modell (LLM) angeben

```
# client that uses openAI or Groq API, here groq API
```

```
client = LLMClient(api_choice = "groq")
```

```
# Adapter erzeugen
```

```
adapter = LLMClientAdapter(client=client)
```

```
from llama_index.core import Settings
```

```
Settings.llm = adapter
```

```
Settings.embed_model = embed_model
```

```
Settings.node_parser = node_parser # SentenceSplitter
```

# Relevante Chunks zurück liefern und Antwort generieren

# Analogie: index ist die Datenbank – `.as_query_engine()` macht daraus eine Suchmaschine.

`query_engine = index.as_query_engine()` # index ist ein `VectorStoreIndex` von `LlamaIndex`

```
def chat_with_pdf(query, history=None):
```

```
    return utils.safe_query_engine_call(query_engine, query)
```

`safe_query_engine_call()` ruft intern `query_engine.query(query)` auf ...

- wandelt query in Embedding Vektor um,
- führt semantische Suche in Vektordatenbank durch
- liefert relevante Chunks zurück und sendet diese inkl. query an das LLM
- liefert Antwort des LLMs zurück

# Entwicklung eines Chatbots

## Gradio App

# This creates a simple web UI for asking questions

```
chat_ui = gr.ChatInterface(  
    fn=chat_with_pdf,  
    title="PDF RAG Chatbot",  
    description="Ask questions about the content of your PDF documents.",  
    theme="default",  
    examples=["What is this PDF about?", "Summarize the second section."],  
)  
  
chat_ui.launch()
```

# Entwicklung eines Chatbots

## Gradio App

Testen Sie den Chatbot

### PDF RAG Chatbot

Ask questions about the content of your PDF documents.

Chatbot

What is this PDF about?

Summarize the second section.

02.05.2026

Seite 40

Type a message...



# Fragen - Pause

- Gibt es Fragen zu diesem Notebook?

# Entwicklung eines Chatbots

## Ollama.com

- Aktuelle LLMs können heruntergeladen und lokal ausgeführt werden
  - Was bedeutet 7b, 14b, ...?
    - Anzahl Modellparameter in Billions (Milliarden)
- Python und JavaScript API
  - <https://github.com/ollama/ollama-python>

```
client = LLMClient(api_choice = "ollama")
```

Neu:

- Ollama Cloud
- <https://docs.ollama.com/cloud>



Get up and running with large language models.

Run [Llama 3.3](#), [DeepSeek-R1](#), [Phi-4](#), [Mistral](#), [Gemma 2](#), and other models, locally.

Download ↓

Available for macOS, Linux, and Windows

### deepseek-r1

DeepSeek's first-generation of reasoning models with comparable performance to OpenAI-o1, including six dense models distilled from DeepSeek-R1 based on Llama and Qwen.

[1.5b](#) [7b](#) [8b](#) [14b](#) [32b](#) [70b](#) [671b](#)

↓ 19.7M Pulls    🔖 29 Tags    ⌚ Updated 2 weeks ago

### llama3.3

New state of the art 70B model. Llama 3.3 70B offers similar performance compared to the Llama 3.1 405B model.

[tools](#) [70b](#)

↓ 1.4M Pulls    🔖 14 Tags    ⌚ Updated 2 months ago

# Entwicklung eines Chatbots

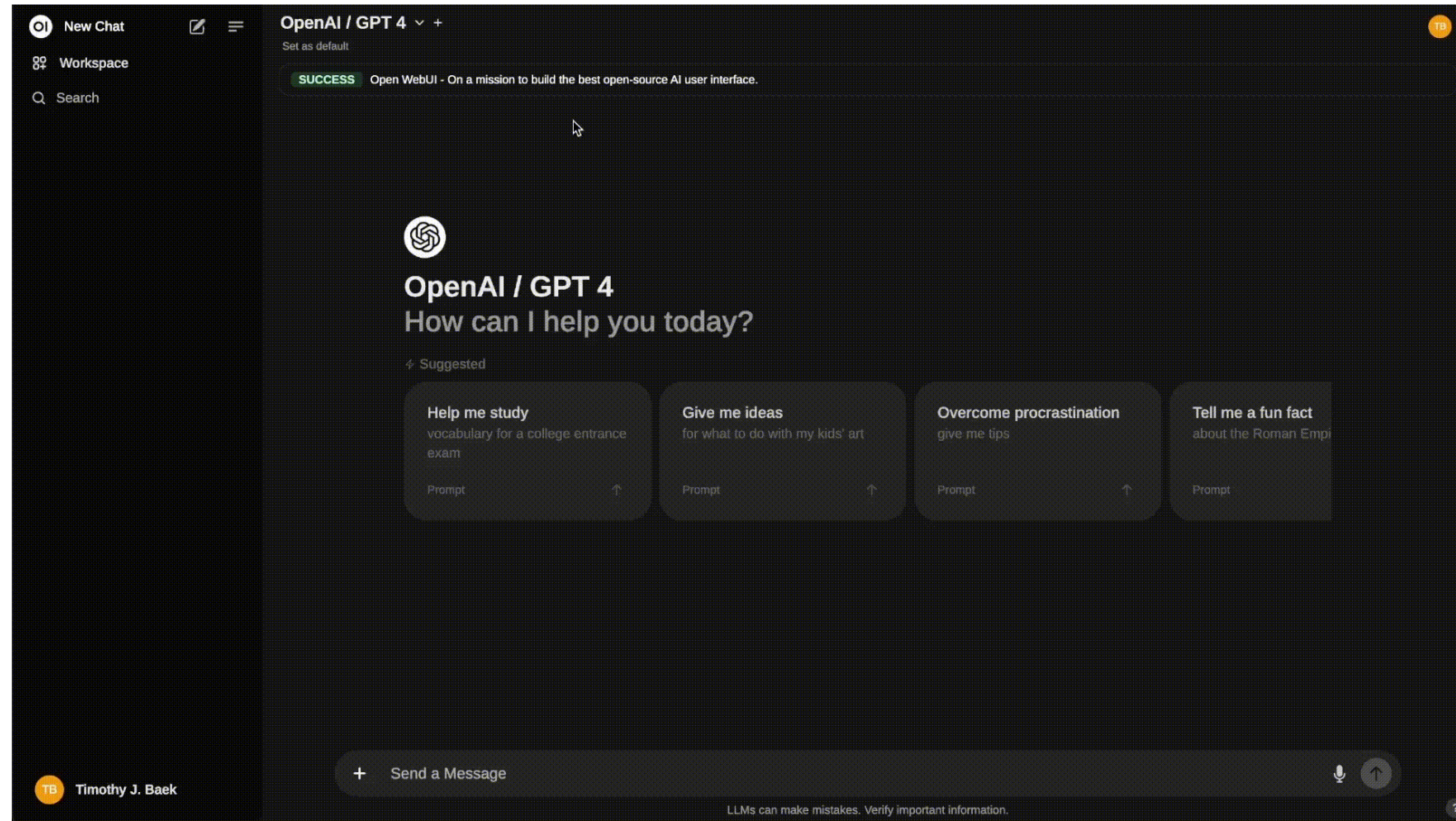
## OpenAI API

- LLM in der Cloud
- Vorteile:
  - Schnelle Inferenz, sehr große Modelle von OpenAI nutzbar
- Nachteile:
  - Kosten, Privatsphäre?

```
client = LLMClient(api_choice = "openai")
```

# Entwicklung eines Chatbots

## Open WebUI



# Entwicklung eines Chatbots

## Speech2Text

- Speech2Text
  - Deep Learning Modell, das gesprochene Sprache in Text umwandelt
  - OpenAI Whisper, Open Source, Multi-Lingual
  - “it enables transcription in multiple languages, as well as translation from those languages into English.”
  - <https://openai.com/index/whisper/>
  - Über Groq API Whisper Modelle nutzbar: <https://console.groq.com/docs/rate-limits>
  - Über OpenAI API
    - gpt-4o-mini-transcribe
    - <https://platform.openai.com/docs/guides/speech-to-text>
- Da Google Colab kein Zugriff auf Mikrofon hat, hier leider nicht testbar
- Weitere Infos: <https://github.com/dgaida/speech2text>

# Entwicklung eines Chatbots

## Text2Speech

- Text2Speech
  - Deep Learning Modell, das aus Text gesprochene Sprache synthetisiert
  - <https://kokorotts.net/>
    - Multilingual, mehrere Stimmen wählbar, <https://kokorotts.net/models/Kokoro/Kokoro-ai>
    - „Kokoro is an open-weight TTS model with 82 million parameters.“, Apache Lizenz, kostenlos
    - Quelle: <https://github.com/hexgrad/kokoro>
  - <https://elevenlabs.io/de>

## Die realistischste KI-Sprachplattform

KI-Sprachmodelle und Produkte, die Millionen von Entwicklern, Kreativen und Unternehmen unterstützen. Von latenzarmen Conversational AI agents bis hin zum führenden KI-Sprachgenerator für Voiceovers und Hörbücher.

KOSTENLOS STARTEN

VERTRIEB KONTAKTIEREN

02.05.2026

Seite 47



TEXT TO SPEECH



SPEECH TO TEXT



KONVERSATIONELLE KI



SYNCHRONISATION



VOICE CLONING



ELEVENREADER

# Entwicklung eines Chatbots

## Text2Speech

Neues Notebook:

- RAGChatbot\_groq\_API\_t2s.ipynb auf [github.com](https://github.com)

```
from kokoro import KPipeline  
tts = KPipeline(lang_code='a')
```

```
def chatbot_with_tts(user_message, history, tts_enabled):  
    response = utils.safe_query_engine_call(query_engine, user_message)  
    if tts_enabled:  
        audio_path = utils.generate_tts_audio(tts, response)  
    return response, audio_path
```

...

# Entwicklung eines Chatbots

## Text2Speech

- Audio (Waveform) könnte normalerweise über Python Package Sounddevice abgespielt werden
- Da Google Colab keinen Zugriff auf den Lautsprecher hat, müssen wir generiertes Audio als temporäre Audiodatei abspeichern.
- Die Audiodatei kann über Gradio App abgespielt werden.

# Entwicklung eines Chatbots

## Text2Speech

- Testen Sie das Notebook RAGChatbot\_groq\_API\_t2s.ipynb

The screenshot shows a chatbot interface. At the top, there is a large empty text area for the user's question. Below it is a text input field containing the word "Hallo". To the right of the input field is a checkbox labeled "Text-to-Speech aktivieren" which is checked. Further right is a grey button labeled "Absenden". Below the input field is an audio player. The audio player has a play button, a progress bar, and a volume icon. The audio player shows a waveform and a duration of 0:03. The text "Antwort zum Anhören" is visible above the audio player.

# Weitere Kurse zu RAG

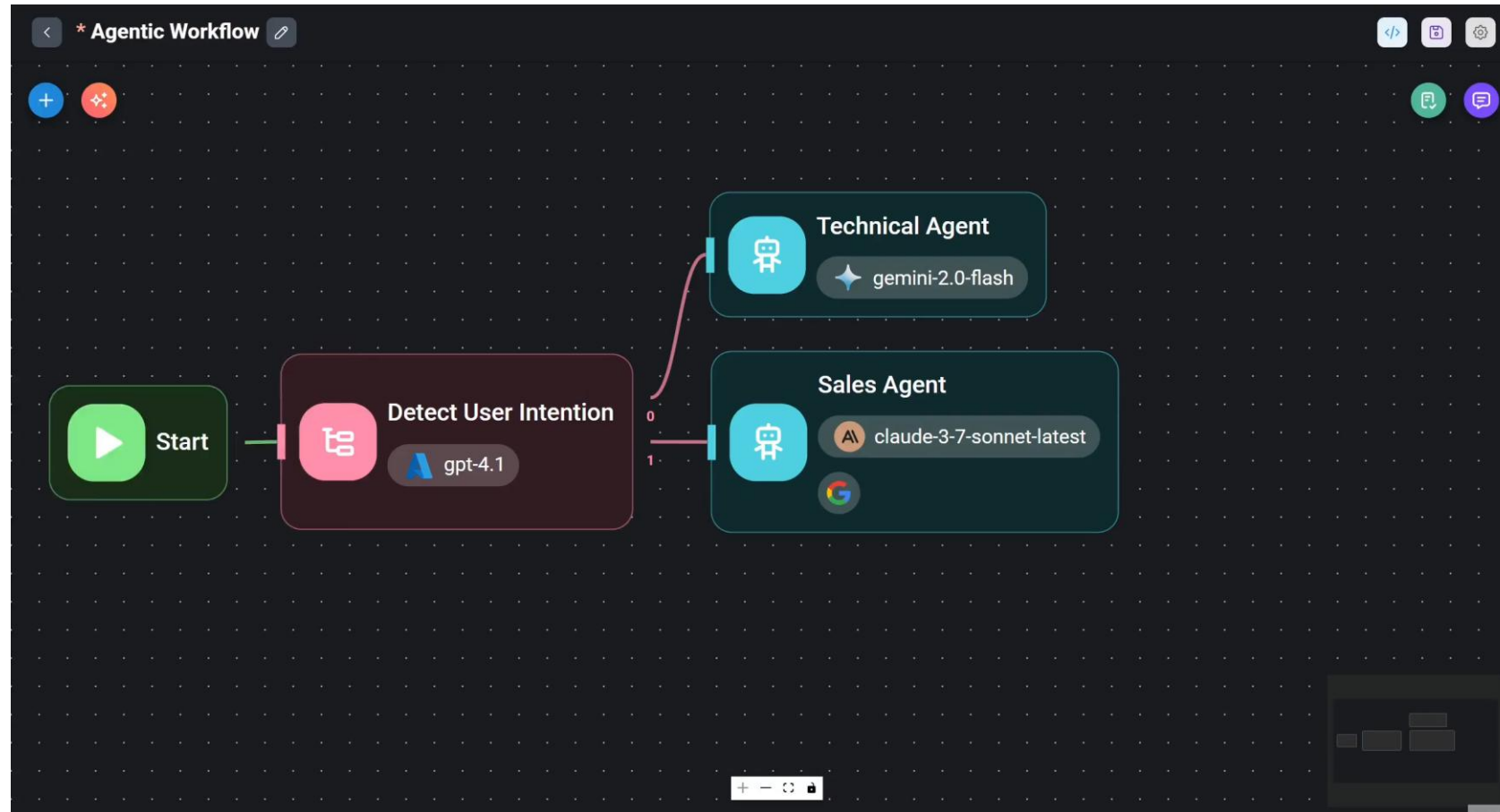
- <https://www.coursera.org/learn/retrieval-augmented-generation-rag>
- <https://www.deeplearning.ai/courses/>
- <https://www.deeplearning.ai/short-courses/building-agentic-rag-with-llamaindex/>
- <https://www.deeplearning.ai/short-courses/building-multimodal-search-and-rag/>
- <https://www.deeplearning.ai/short-courses/multimodal-rag-chat-with-videos/>
- <https://www.deeplearning.ai/short-courses/building-evaluating-advanced-rag/>

## What you'll learn

- ✓ Learn methods like sentence-window retrieval and auto-merging retrieval, improving your RAG pipeline's performance beyond the baseline.
- ✓ Learn evaluation best practices to streamline your process, and iteratively build a robust system.
- ✓ Dive into the RAG triad for evaluating the relevance and truthfulness of an LLM's response: Context Relevance, Groundedness, and Answer Relevance.

# RAG Agenten Visuell programmieren

<https://flowiseai.com/>



# Text2Image, Text2Video

- Text2Image
  - Bild aus Text generieren
  - Google API: <https://ai.google.dev/gemini-api/docs/image-generation?hl=de>
  - HuggingFace (lokale Ausführung): [https://huggingface.co/models?pipeline\\_tag=text-to-image&sort=trending](https://huggingface.co/models?pipeline_tag=text-to-image&sort=trending)
  
- Text2Video
  - Video aus Bild generieren
  - Google API: <https://ai.google.dev/gemini-api/docs/video?hl=de&example=dialogue>
  - HuggingFace (lokal): [https://huggingface.co/models?pipeline\\_tag=text-to-video&sort=trending](https://huggingface.co/models?pipeline_tag=text-to-video&sort=trending)

# Image2Text, Video2Text

- Image2Text
  - Text aus Bild generieren (Bild interpretieren)
  - Groq API: <https://console.groq.com/docs/vision>
    - Bilder müssen < 30k Tokens groß sein für free plan
  - Google API: <https://ai.google.dev/gemini-api/docs/image-understanding?hl=de>
  - HuggingFace (lokal): [https://huggingface.co/models?pipeline\\_tag=image-to-text&sort=trending](https://huggingface.co/models?pipeline_tag=image-to-text&sort=trending)
  - Ollama lokal/Cloud: <https://docs.ollama.com/capabilities/vision#python>
- Video2Text
  - Text aus Video generieren (Video interpretieren)
  - Google API: <https://ai.google.dev/gemini-api/docs/video-understanding?hl=de>
  - HuggingFace (lokal): bisher nicht vorhanden

# Nächste Vorlesung und Fragen

- Erklärbare KI
- Fragen?

# Optionale Übung: Embedding Modell angeben und ausprobieren

- Cosine similarity liefert Werte zwischen -1 und 1 zurück (üblicherweise aber eher zwischen 0 und 1)

```
# this cell is optional - not needed for the notebook to run
# you can play with this if you want to test how close two embedding vector are
embedding_1 = embed_model.get_text_embedding("Hello, my name is Daniel")
embedding_2 = embed_model.get_text_embedding("How are you?")

# Compute dot product similarity
similarity = sum(a * b for a, b in zip(embedding_1, embedding_2))
print(similarity)
```

```
0.8202128970159881
```

```
# this cell is optional - not needed for the notebook to run
# you can play with this if you want to test how close two embedding vector are
embedding_1 = embed_model.get_text_embedding("Hello, my name is Daniel")
embedding_2 = embed_model.get_text_embedding("Hello, my name is not Daniel")

# Compute dot product similarity
similarity = sum(a * b for a, b in zip(embedding_1, embedding_2))
print(similarity)
```

```
0.9513346940599776
```

# Optionale Übung: Embedding Modell angeben und ausprobieren

```
import numpy as np
```

KING - MAN + WOMAN = ~~QUEEN~~  
≈ KING ?

```
# Convert lists to NumPy arrays before performing arithmetic
```

```
embedding_1_np = np.array(embed_model.get_text_embedding("King")) -  
np.array(embed_model.get_text_embedding("Man")) +  
np.array(embed_model.get_text_embedding("Woman"))
```

```
embedding_2_np = np.array(embed_model.get_text_embedding("Queen"))
```

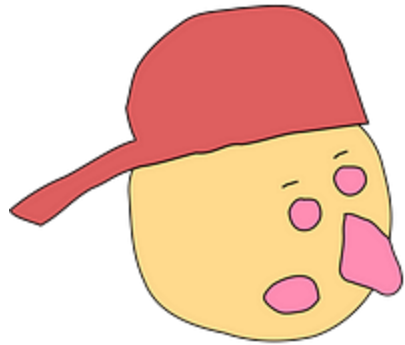
```
# Compute dot product similarity using NumPy's dot function
```

```
similarity = np.dot(embedding_1_np, embedding_2_np)
```

```
print(similarity)
```

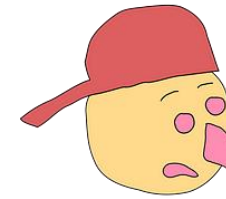
# Optionale Übung: Embedding Modell angeben und ausprobieren

What does it give you for:  
Martini - Gin + Whiskey?



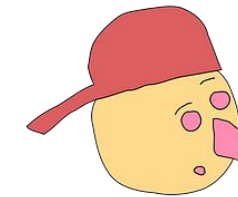
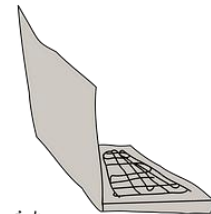
See how stupid AI is?

"Cocktail"\*



Yeah. I can't believe it.  
You trained it on the entire  
Wikipedia, and it still doesn't  
know how to mix a Manhattan!

Yeah! Everybody knows that  
that's a Manhattan!



Guess I need to train it  
on some cocktail books.



\*I briefly tested it. With a Continuous Skip-Gram algorithm trained on Google News (100 billion tokens) I got for *gin is to Martini what whiskey is to X*:  
1. signature\_cocktails, 2. Cocktails, 3. Cocktail, 4. Vodka\_Martini, 5. cocktails  
CC BY 4.0 Florian Huber

# Optionale Übung: Jupyter Notebook zur Übung

- Öffnen Sie das folgende Notebook in Google Colab:  
[https://github.com/dgaida/llm\\_client/blob/master/notebooks/HF\\_EmbeddingModel\\_test.ipynb](https://github.com/dgaida/llm_client/blob/master/notebooks/HF_EmbeddingModel_test.ipynb)
- Probieren Sie das Embedding Modell aus
- Schauen Sie sich bei Bedarf <https://projector.tensorflow.org/> an

Bei Bedarf:

- Schauen Sie sich die Doku zu Ihrem Embedding Modell an:  
<https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>
- Testen Sie weitere Embedding Modelle, bspw. für die deutsche Sprache:
  - <https://huggingface.co/spaces/mteb/leaderboard>