

# KI Anwendungen im Unternehmen

mit Java und Opensource LLMs sicher umsetzen

David Beisert - Basalone 2025

# Workshop Überblick

**Ziel:** Implementierung von AI-Anwendungen mit Java und Open-Source LLMs

**Zielgruppe:** Java-Entwickler

**Technische Voraussetzungen:** Java 21, Docker, IDE (IntelliJ)

**Agenda:** Von Grundlagen bis zur produktiven Implementierung

# Vorbereitungen

LM Studio installieren <https://lmstudio.ai/>

- Download text-embedding-nomic-embed-text-v2
- Download openai/gpt-oss-20b oder llama3

Git

Java 21, Maven

Node > 20, NPM

IntelliJ

Docker desktop oder podman

git clone <https://github.com/beisdog/baselone-ai-workshop.git>

# Vorstellungsrunde

Wer bin ich und  
was habe ich mit  
LLMs zu tun?

# Privat

Name: David Beisert

Alter: 49 Jahre 🤖

Kinder: 3 🤖

Hobbies: Keine 🤖

## Instructions

Go to

**[www.menti.com](https://www.menti.com)**

Enter the code

**1618 7452**



Or use QR code



# kurze Geschichte von LLMs



# Was gab es denn vor ChatGPT?

- RNN und LSTM

- Probleme:

- Sequentielle Verarbeitung der Wörter -> Langsam
    - Lange Abhängigkeiten: Informationen aus früheren Sätzen gingen verloren, wurden vergessen

# Der Durchbruch 2017

- Ein Paper von Google Research 2017
  - "Attention is all you need"
- Parallele Verarbeitung
  - Jedes Wort steht mit anderen Worten in Beziehung
  - Pro Satz wird für jedes Wort der Self Attention Score zu den anderen Worten berechnet
  - Transformer Architektur: Encoder, Decoder

# Und dann kam schon ChatGPT ...

- **2018: BERT (Google)** - Ein bidirektionaler Encoder Transformer für NLP-Aufgaben.
- **2020: GPT-3 (OpenAI)** - Ein autoregressives Transformer-Modell mit 175 Milliarden Parametern.
- **30.11. 2022:** Öffentlicher Launch von Chatgpt mit **GPT-3.5**
- **14.03.2023: GPT 4**
- **2023+: Gemini, LLaMA, Mistral, Deepseek ...**

# Warum ist das so mächtig?

- **Berücksichtigt globale Zusammenhänge:** Jedes Wort kann mit allen anderen Wörtern interagieren.
- **Erkennt Abhängigkeiten unabhängig von der Wortreihenfolge.**
- **Ermöglicht parallele Verarbeitung:** Alle Wörter können gleichzeitig verarbeitet werden, im Gegensatz zu sequentiellen Modellen wie RNNs.

Self-Attention ist der Grund, warum **Transformer-Modelle wie GPT und BERT so leistungsfähig sind!**

Was sind die  
aktuellen  
Bedenken beim  
Einsatz von KI  
im Unternehmen?

# Bedenken beim Einsatz von KI im Unternehmen

Datenschutz und Compliance: GDPR, BDSG, EU AI Act Anforderungen

Datensouveränität: Kontrolle über sensible Unternehmensdaten 

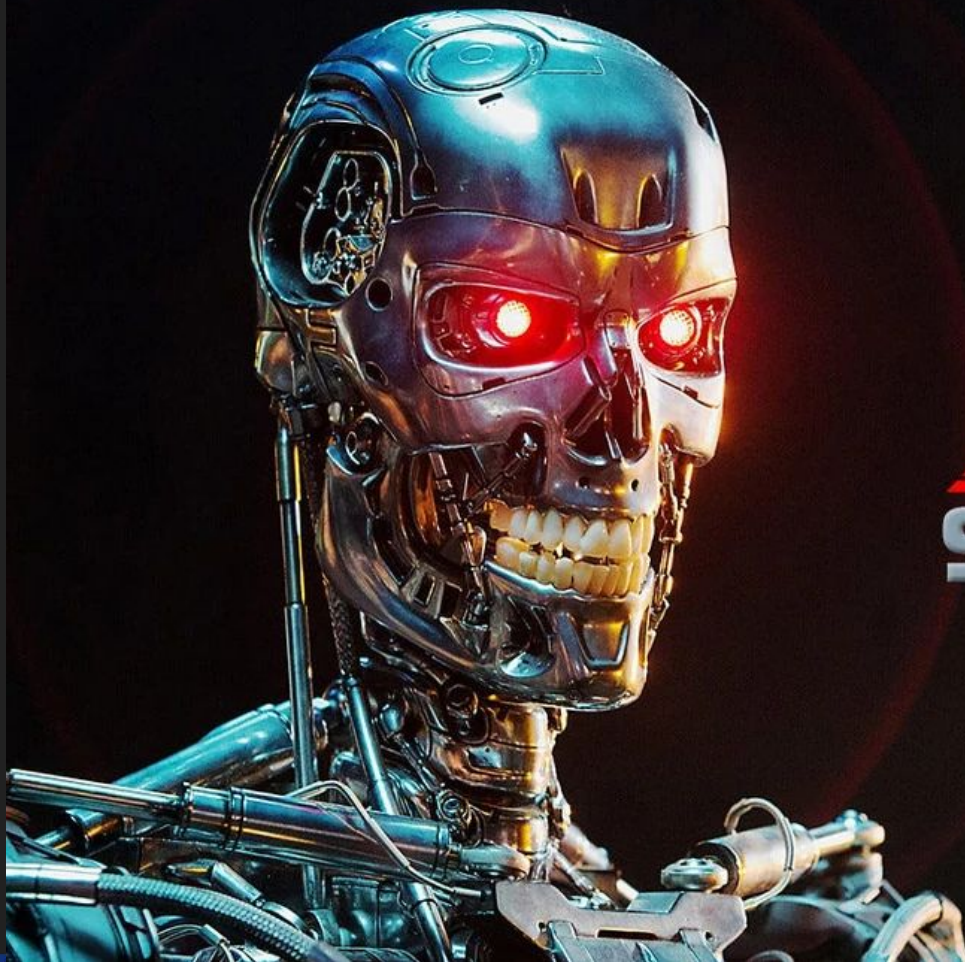


Kosten: Unvorhersehbare API-Kosten bei Cloud-Anbietern

Performance: Latenz und Verfügbarkeit kritischer Anwendungen

Integration: Einbindung in bestehende Enterprise-Architekturen

Sicherheit: Schutz vor Prompt Injection, Data Leakage



Wie kann ich  
Opensource LLMs  
einsetzen?



# Fragen zu Opensource LLMs

Sind die so gut wie kommerzielle? Sind die gut genug?

Welche soll ich nehmen?

Wie teste ich die LLMs?

Können die deutsch?

Welche Features brauchen wir? Multimodal, Werkzeuge ...

Wie teuer ist die Hardware?

Welche HW brauche ich für x Benutzer?

# Hardware und Kosten

Es muss halt in den  
passen ... VRAM

## 2 Faktoren bestimmen VRAM

Arch ▲	Params	Publisher	Model	Quant	Size
<a href="#">gpt_oss</a>		openai	openai/gpt-oss-20b <a href="#">↗</a>	<a href="#">8bit</a>	22.26 GB
<a href="#">gpt-oss</a>	<a href="#">120B</a>	openai	openai/gpt-oss-120b <a href="#">↗</a>	<a href="#">MXFP4</a>	63.39 GB

## 2 Faktoren bestimmen VRAM

Arch ▲	Params	Publisher	Model	Quant	Size
<a href="#">gpt_oss</a>		openai	<a href="#">openai/gpt-oss-20b</a>	<a href="#">8bit</a>	22.26 GB
<a href="#">gpt-oss</a>	<a href="#">120B</a>	openai	<a href="#">openai/gpt-oss-120b</a>	<a href="#">MXFP4</a>	63.39 GB

## 2 Faktoren bestimmen VRAM

Arch ▲	Params	Publisher	Model	Quant	Size
<a href="#">gpt_oss</a>		openai	<a href="#">openai/gpt-oss-20b</a>	<a href="#">8bit</a>	22.26 GB
<a href="#">gpt-oss</a>	<a href="#">120B</a>	openai	<a href="#">openai/gpt-oss-120b</a>	<a href="#">MXFP4</a>	63.39 GB

Faustformel:

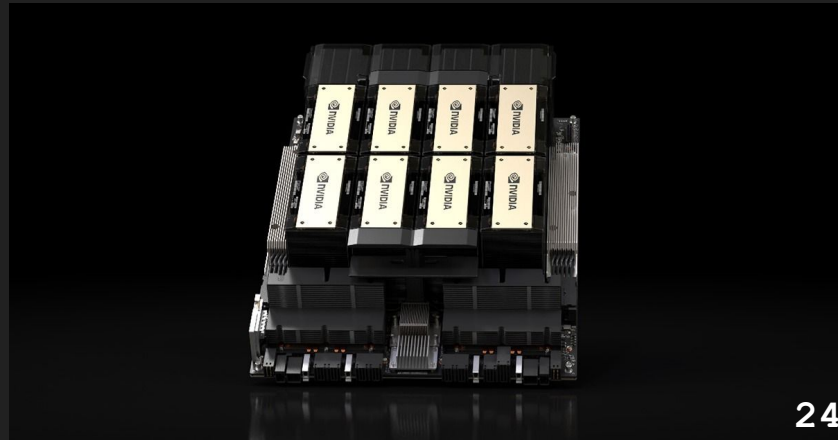
$\text{Parameter (in Billion)} * \text{Quant.Bit}/8 * 1.2 (\text{Puffer}) = \text{VRAM}$

$$\text{Parameter (in Billion)} * \text{Bit}/8 * 1.2 = \text{VRAM}$$

Arch ▲	Params	Publisher	Model	Quant	Size
<a href="#">gpt_oss</a>		openai	<a href="#">openai/gpt-oss-20b</a>	<a href="#">8bit</a>	22.26 GB
<a href="#">gpt-oss</a>	<a href="#">120B</a>	openai	<a href="#">openai/gpt-oss-120b</a>	<a href="#">MXFP4</a>	63.39 GB

Bsp gpt-oss-120b

$$120B * 4/8 (\text{Quant}) * 1.2 (\text{Puffer}) \\ = 72 \text{ GB VRAM}$$





# Kosten für NVIDIA GPUs für 80GB VRAM

GPU Modell	Anzahl Karten x VRAM	Monatlicher Stromverbrauch	Preis pro Karte	Gesamt
RTX 4080	5 x 16GB	~806 kWh	~1.200 CHF	~6'000 CHF
RTX 6000 ADA	2 x 48GB	~302 kWh	~7.000 CHF	~14'000 CHF
A100	1 x 80GB	~151 kWh	~14.000 CHF	~14.000 CHF
H100	1 x 80GB	~353 kWh	~23.000 CHF	~23.000 CHF
H200	1 x 141GB	~353 kWh	~32.000 CHF	~32.000 CHF

Achtung: Es ist  
einfacher wenn  
ein Modell auf  
eine GPU passt



NVIDIA / TensorRT-LLM

# Praxisteil

# Übung 1: LM Studio benutzen

## Voraussetzungen:

- LM Studio: [lmstudio.ai](https://lmstudio.ai)

## Übung:

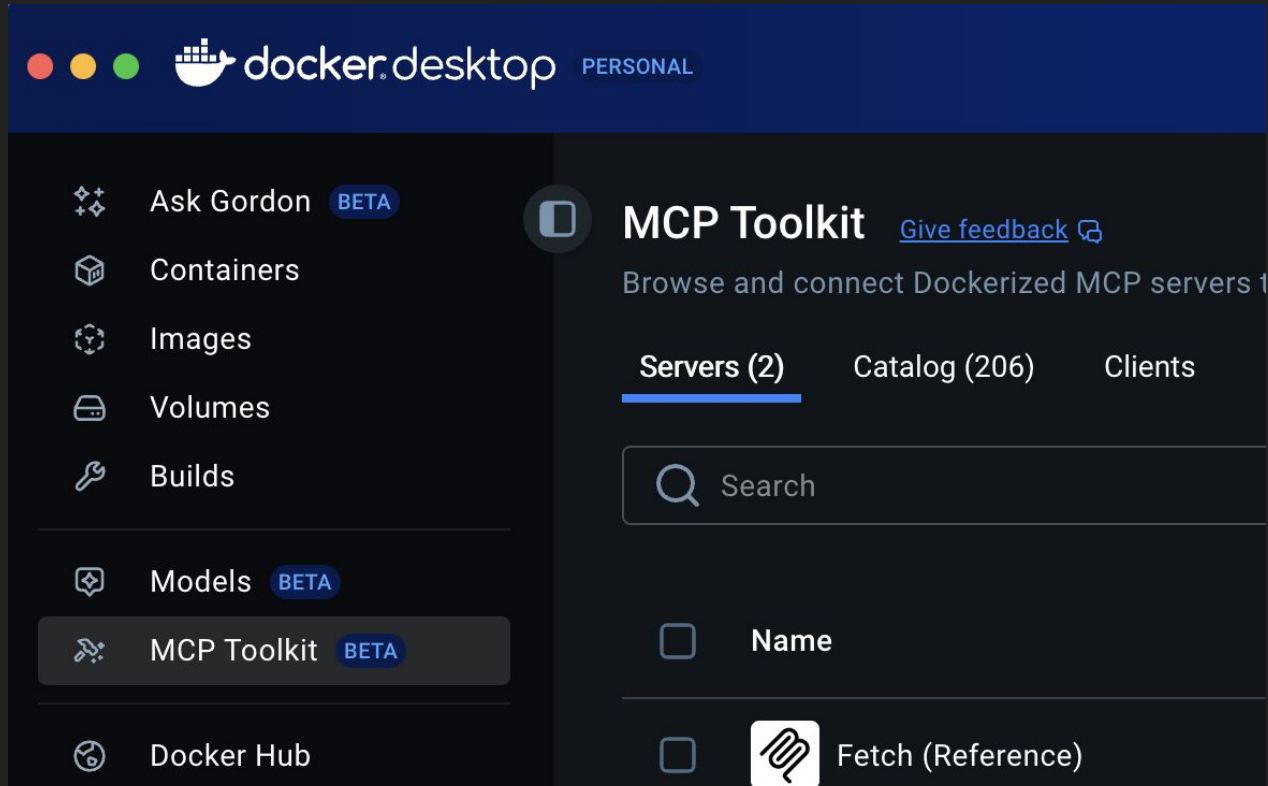
- Modell laden
- Chatten in der UI



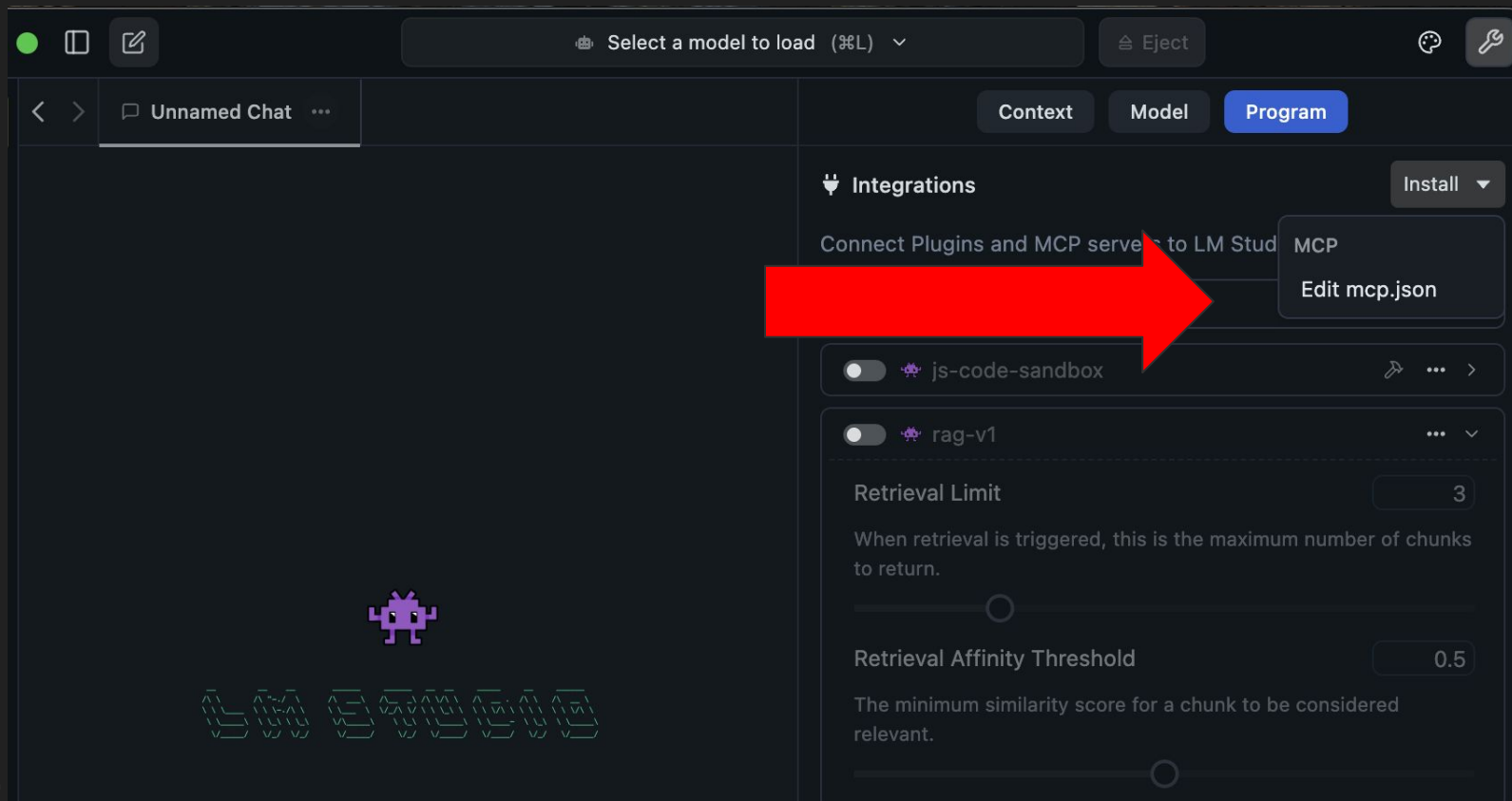
## Übung 2: MCP Tool einbinden

Ziel: DuckGoGo Suche in den LM Studio Chat integrieren

# 1. Docker desktop und MCP Toolkit



## 2. MCP in LM Studio eintragen



# LM Studio - mcp.json

```
{
  "mcpServers": {
    "duckduckgo": {
      "command": "docker",
      "args": [
        "run",
        "-i",
        "--rm",
        "mcp/duckduckgo"
      ]
    }
  }
}
```



# Ausprobieren

openai/gpt-oss-120b

> Thought for a brief moment

🔗 `search({"query":"Spiegel topnachrichten heute","max_resul..."})`

mcp/duckduckgo >

Hier sind die heutigen Top-Nachrichten von **Spiegel.de** (Stand 13. September 2025). Die wichtigsten Artikel finden Sie direkt über die angegebenen Links.

Rubrik	Titel (Kurz)	Kernaussage	Link
Politik	<i>Bundesregierung plant neues Klimaschutz-Paket</i>	Das Kabinett will bis 2030 den CO <sub>2</sub> -Ausstoß um 55 % senken, u. a. durch höhere Strompreise und Förderungen für Wasserstoff.	<a href="https://www.spiegel.de/politik/deutschland/klimaschutz-paket-regierung-2025-a-1234567.html">https://www.spiegel.de/politik/deutschland/klimaschutz-paket-regierung-2025-a-1234567.html</a>
Wirtschaft	<i>Deutsche Wirtschaft wächst im zweiten Quartal um 1,2 %</i>	Trotz Lieferkettenproblemen verzeichnet das Bruttoinlandsprodukt ein solides Wachstum; Exportüberschuss steigt.	<a href="https://www.spiegel.de/wirtschaft/deutschland/bip-wachstum-q2-2025-a-2345678.html">https://www.spiegel.de/wirtschaft/deutschland/bip-wachstum-q2-2025-a-2345678.html</a>
International	<i>Ukraine-Krise: Friedensgespräche in Genf beginnen</i>	Vertreter der Ukraine und Russlands treffen sich erstmals seit 2024 in Genf, um über ein Waffenstillstandsabkommen zu verhandeln.	<a href="https://www.spiegel.de/ausland/ukraine-friedensverhandlungen-genf-2025-a-3456789.html">https://www.spiegel.de/ausland/ukraine-friedensverhandlungen-genf-2025-a-3456789.html</a>
Science	<i>KI-System entdeckt neues Exoplaneten-System</i>	Das KI-Programm „StellarNet“ identifizierte 12 potenziell bewohnbare Planeten im Sternbild Lyra.	<a href="https://www.spiegel.de/wissenschaft/astronomie/ki-entdeckt-exoplaneten-lyra-a-4567890.html">https://www.spiegel.de/wissenschaft/astronomie/ki-entdeckt-exoplaneten-lyra-a-4567890.html</a>



# LangChain4j



<https://docs.langchain4j.dev/>

Supercharge your Java application with the power of LLMs

**Introduction**

# Übung: LLM aufrufen über Langchain4j- 10min

Ziel: LLM Aufruf mit Langchain4j ausführen

Vorbereitung:

`git clone`

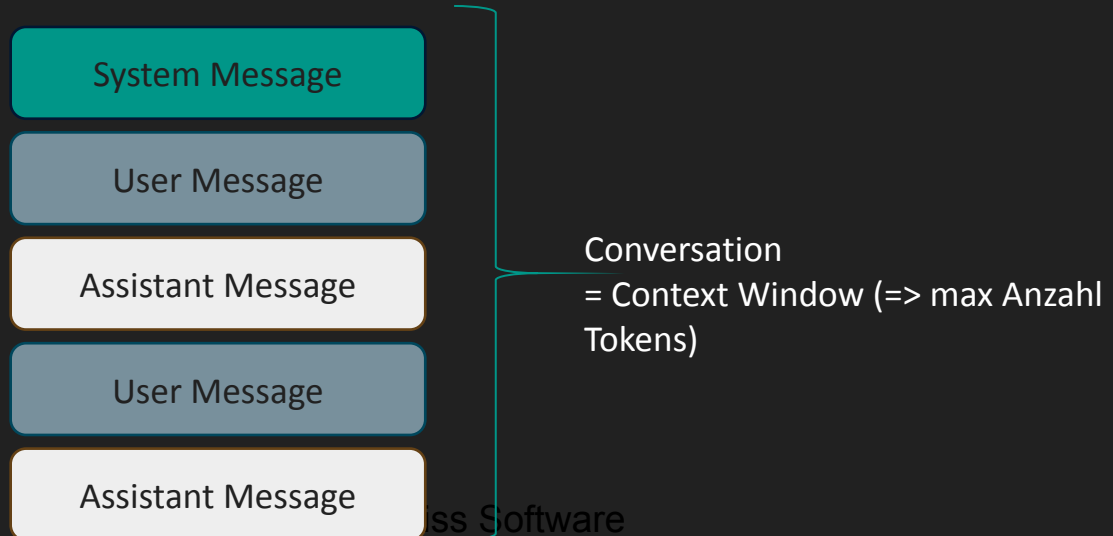
<https://github.com/beisdog/baselone-ai-workshop.git>

Gehe zu:

`sessions/basicsession/ ../basic/Lesson_01BasicChat`

# Aufruf vom LLM via Java mit Historie

Chat Models sind Stateless, die ganze Konversation muss mitgegeben werden



# System Message

- Es gibt nur 1 System Message
- Instruiert das LLM über:
  - Allgemeines zur Aufgabenstellung worum es geht
  - Die Rolle die das LLM einnehmen soll
  - Den Ton (lustig, flirtend, als Pirat, die Sprache) den das LLM benutzen soll

≡ cv\_rag\_system\_prompt.txt ×

```
1 Du bist ein Assistent der Fragen zu einem Lebenslauf eines Mitarbeiters beantwortet.  
2 Bitte gib wenn möglich Bulletpoints aus.
```

# User Message

- Die Benutzer Frage.
- Der dynamische Teil der Konversation



Wer ist Donald Trump?

# Assistant Message

## • Die Antwort des LLM

Donald Trump ist ein amerikanischer Unternehmer, Immobilienentwickler, Fernsehproduzent und Politiker. Er wurde am 14. Juni 1946 in Queens, New York City, geboren. Trump ist besonders bekannt als der 45. Präsident der Vereinigten Staaten, ein Amt, das er von Januar 2017 bis Januar 2021 ausübte. Vor seiner politischen Karriere war er vor allem für sein Immobiliengeschäft und als Moderator der Reality-Show „The Apprentice“ bekannt.

Trump ist eine umstrittene Figur, die polarisierende Meinungen hervorruft. Seine Präsidentschaft war geprägt von einer Vielzahl von politischen Entscheidungen und Kontroversen, einschließlich seiner Einwanderungspolitik, seiner Haltung zu internationalen Beziehungen sowie seiner Reaktion auf soziale und wirtschaftliche Themen.

Nach seiner Amtszeit blieb Trump in der politischen Arena aktiv und deutete an, dass er eine erneute Kandidatur für die Präsidentschaft anstreben könnte.

# Übung: LLM mit History- 10min

Ziel: LLM Aufruf mit Langchain4j ausführen

Vorbereitung:

<https://github.com/beisdog/baselone-ai-workshop.git>

Gehe zu:

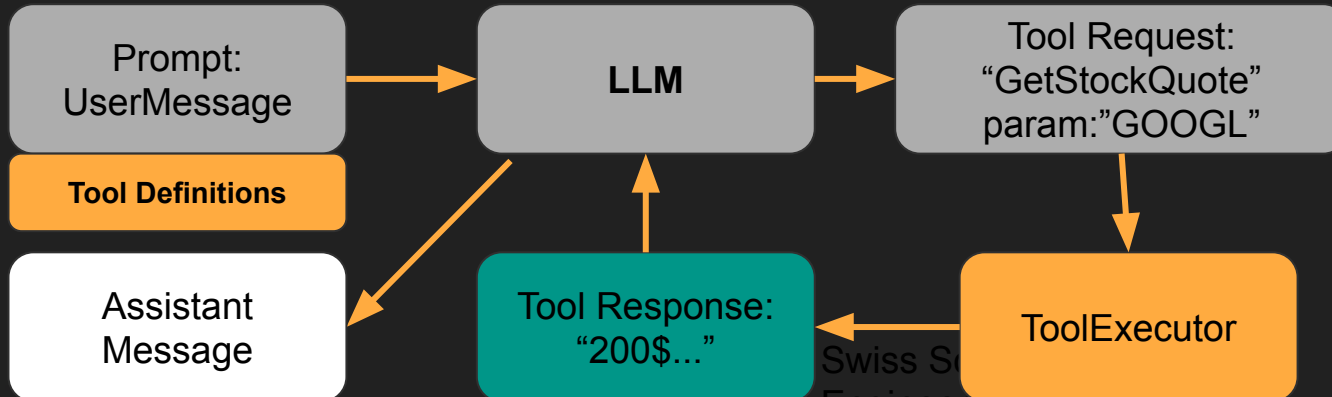
sessions/basicsession/../../basic/Lesson\_02ChatWithMemory



# Werkzeuge

# Tool Request/Response Message

- LLM bekommt Prompt und Definition von Funktionen
- Das LLM kann mit einem Tool Request antworten
- Wir müssen das Tool dann für das LLM aufrufen und geben im das Ergebnis als Tool Response mit



# Übung: LLM mit Toolcall - 10min

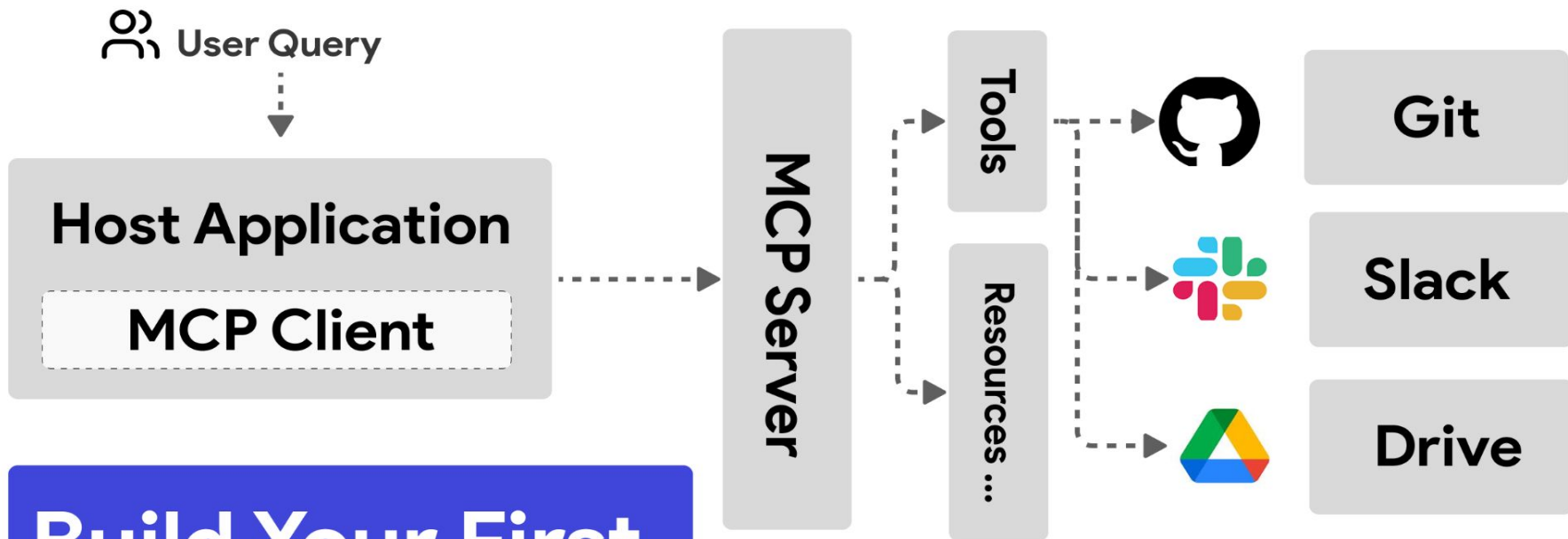
Ziel: LLM mit Toolcall

<https://github.com/beisdog/baselone-ai-workshop.git>

Gehe zu:

sessions/basicsession/../../basic/Lesson\_03ChatMarketTool

# MCP For Software Engineers - Pt 1 (Server, Client, Host App)



**Build Your First  
MCP Server**



Learn more!  
[newsletter.victordibia.com](https://newsletter.victordibia.com)

# Übung: LLM mit Toolcall mit MCP - 10min

Ziel: LLM mit Toolcall von duckgogo Suche von

Vorbereitung:

git clone

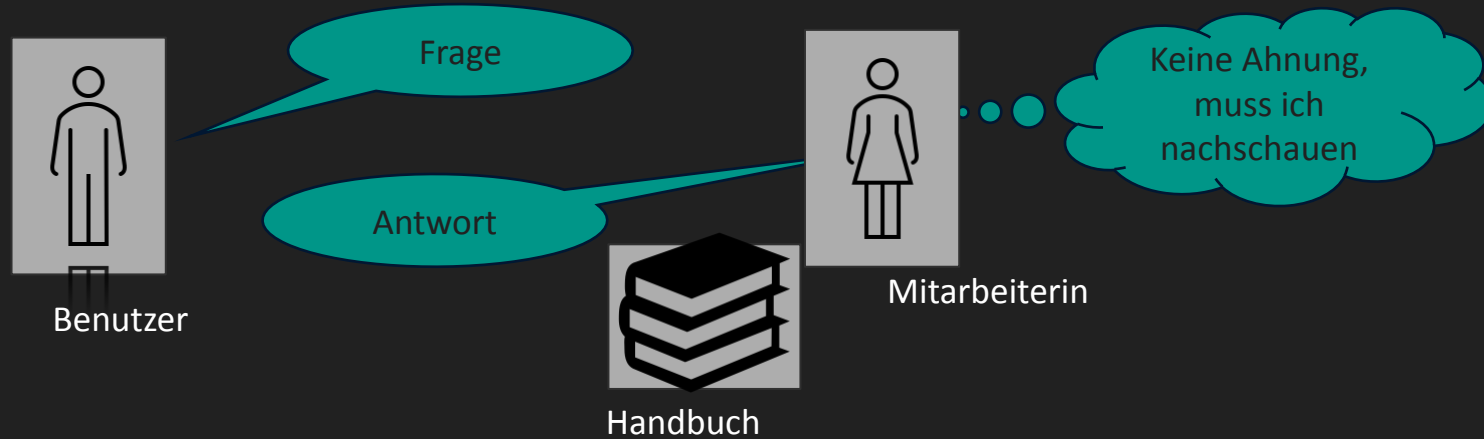
<https://github.com/beisdog/baselone-ai-workshop.git>

Siehe : sessions/basicsession/../../basic/ChatMarketTool aber schreibe eine neue Klasse, die statt des Markettools ein MCP Tool von DuckGogo einfügt.

<https://docs.langchain4j.dev/tutorials/mcp#using-the-github-mcp-server-through-docker>

# Retrieval Augment Generation

# Retrieval Augment Generation

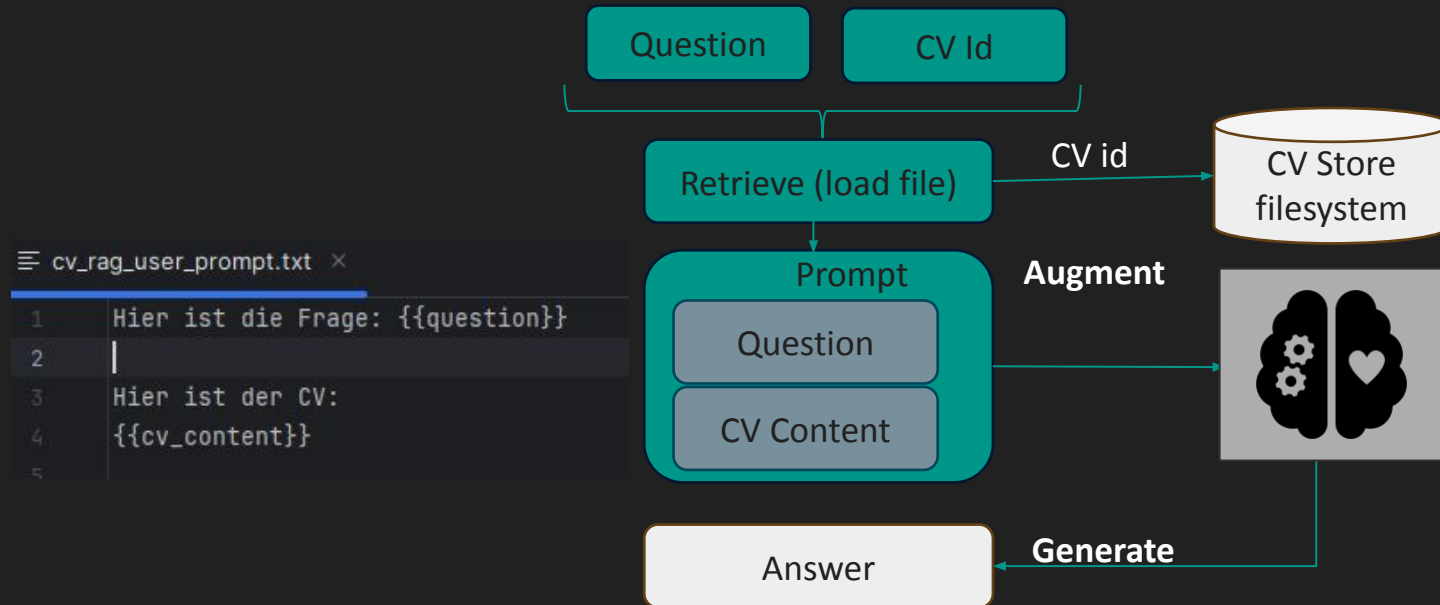


# Was wollen wir bauen?

- Einen Chatbot der uns Fragen zu CVs beantworten kann
- Wir binden LLMs über LM Studio ein
- Wir benutzen das Filesystem und Postgresql Vector Store für die Wissenssuche



# Erster einfacher RAG Aufruf



# Übung: RAG Simple RAG - 10min

Gehe zu:

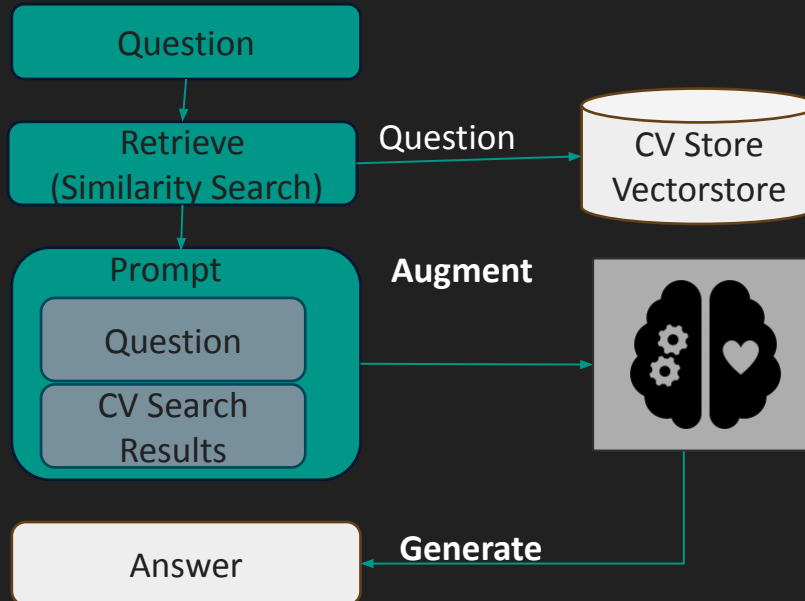
`sessions/basic_session/.../rag/`

`Lesson_00ChatWithRag.java`

# RAG mit einem Vectorstore

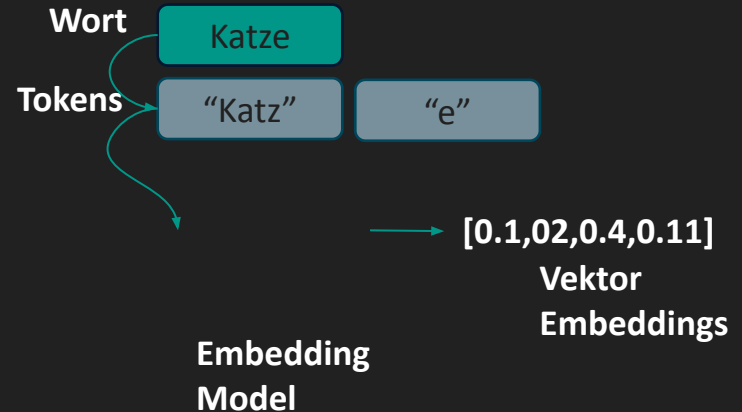
≡ cv\_rag\_vs\_user\_prompt.txt ×

```
1 Hier ist die Liste der CV:
2 {{cv_list}}
3
4 Hier die Frage: {{question}}
```



# Tokens

- Wortteile: ca 0.75 eines Wortes
- Sind die Einheit in der LLMs abrechnen
- Jedes LLM hat einen Maximal Wert von Tokens die es lesen und generieren kann.
- Es gibt Input, Output und Reasoning Tokens

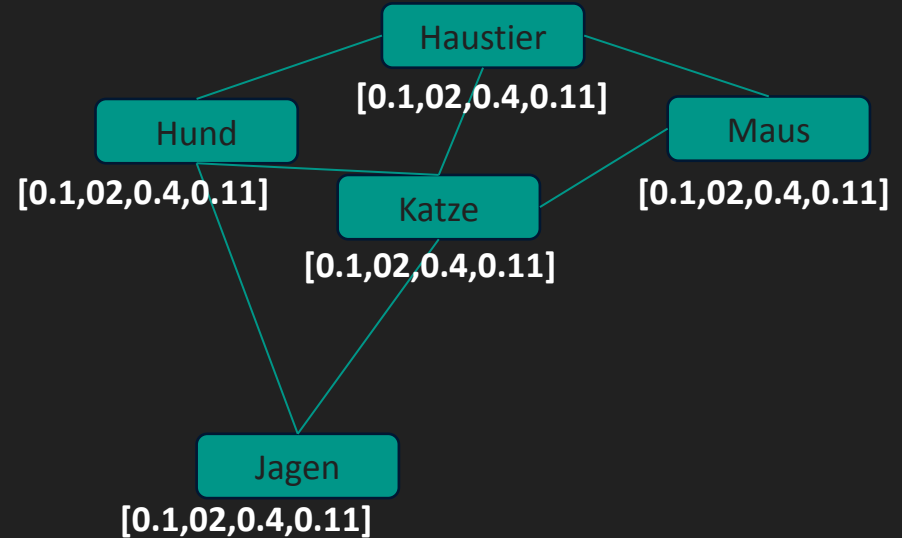


# Context Window

- Beschreibt die Anzahl Tokens die man einem Aufruf an das LLM mitgeben darf
- **Die Grösse des Context Window ist das zentrale Problem in der Arbeit mit LLMs**
  - Grösse bestimmt Informationsmenge, Kosten und Genauigkeit
- Massnahmen:
  - Konversationen verkleinern, Nur notwendige Daten mitgeben

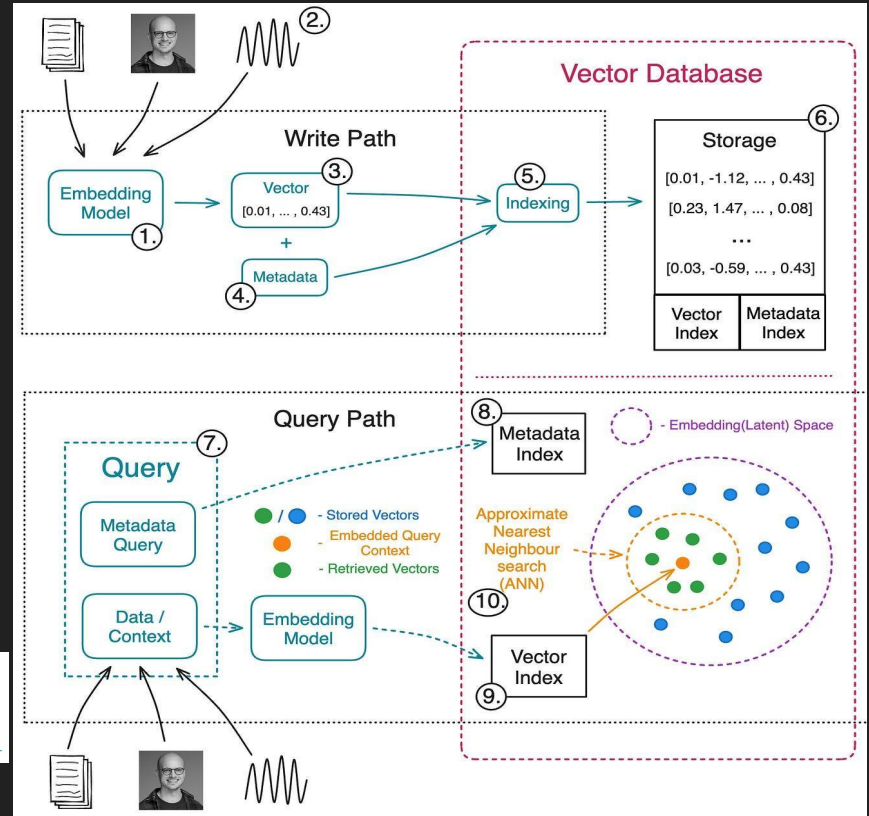
# Embeddings

- Embeddings (Konvertieren Tokens in Vektoren):
- Embedding Modelle stelle die Beziehungen und Gewichtungen zwischen Tokens/Worten dar.
  - Bsp Model: Word2Vec, Open AI: Text ADA 002



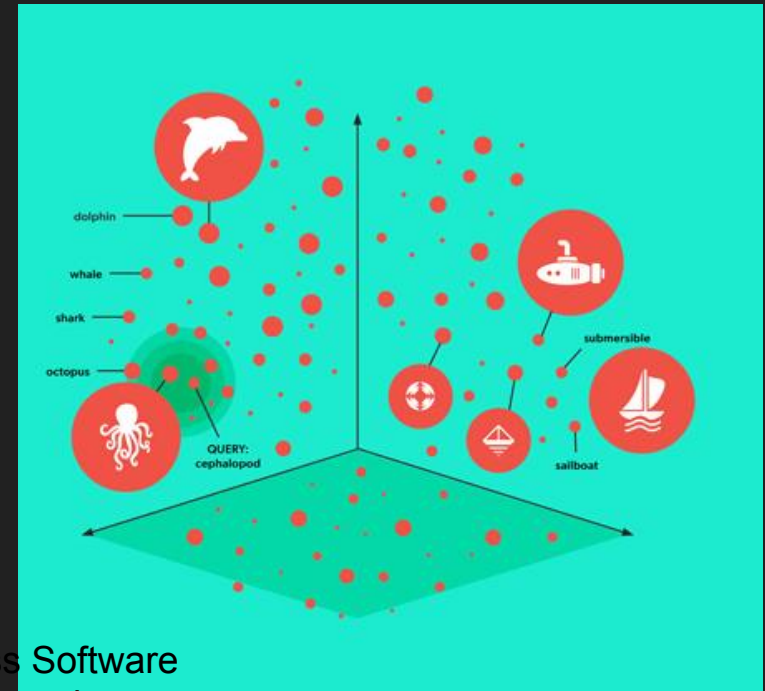
# Was ist ein Vectorstore?

- Datenbank die Vektoren abspeichern
- Ist optimiert für Vektorensuche
  - Z.B. Nearest Neighbor etc
- Vektoren werden durch ein Embedding Model vektorisiert



# Warum ein Vectorstore?

- **Semantische Suche:**
  - Versteht Bedeutung statt nur Volltextsuche
- **Effizientes Retrieval:**
  - Holt relevante Infos für LLMs
- **Skalierbar & Schnell:**
  - Optimiert für Millionen von Einträgen

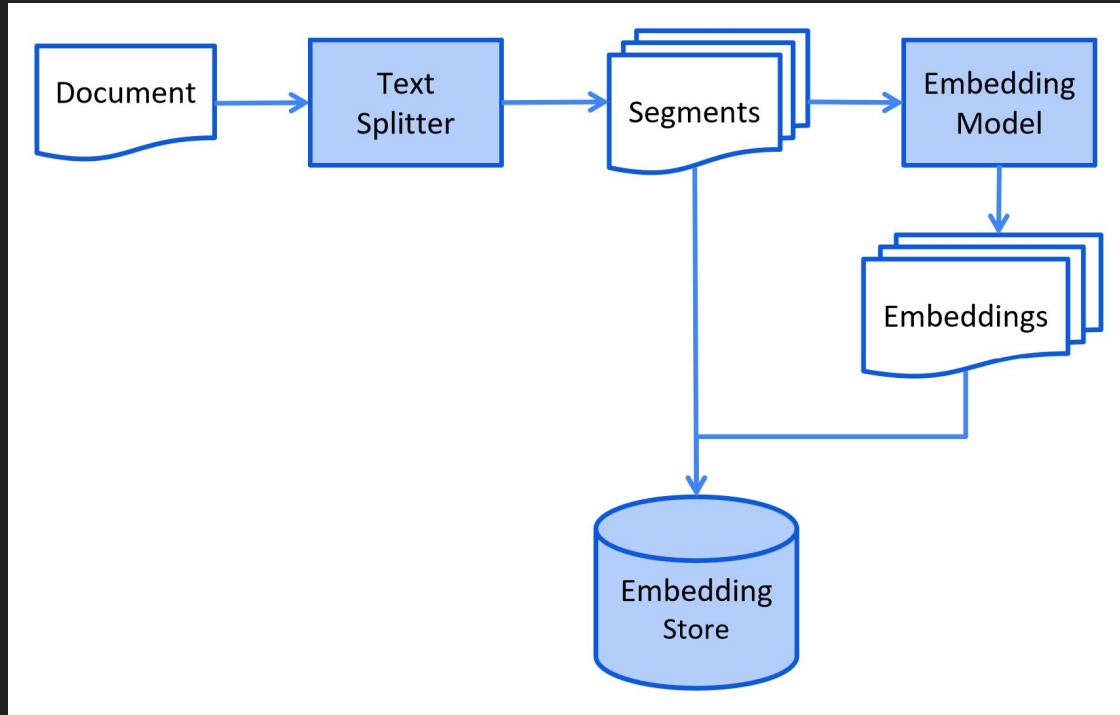




# Nachteile von Vectorstores

- Die Grösse eines Dokumentes ist begrenzt durch Tokenlimit vom Embeddingmodel
  - Man muss Dokumente splitten
  - Beim splitten geht der Kontext verloren -> Overlapping
- Ungenaue Ergebnisse
- Auch nicht die letzte Lösung man braucht noch weitere Tricks um gute Chatbots zu bauen (Sie RAG Strategien)
- Müssen synchron zur Quelle gehalten werden

## 4.1 Vectorstore befüllen



# Übung: RAG Grundlagen 20min

<https://docs.langchain4j.dev/tutorials/rag>

1. Embedding Text
2. Ingesting into Vectorstore
3. Suchen

Gehe zu:

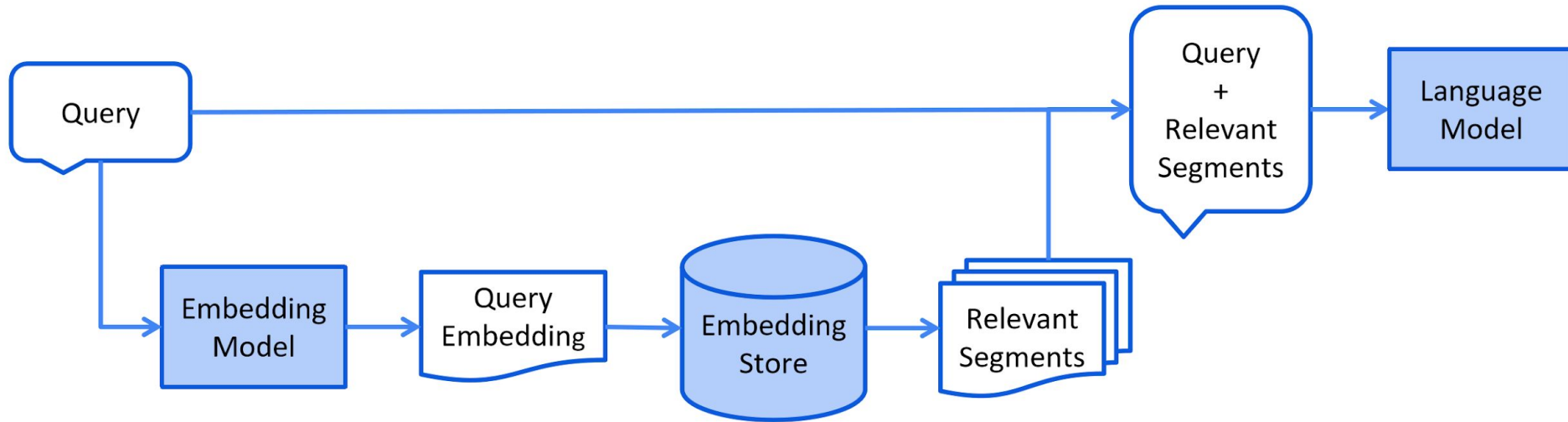
[sessions/basic\\_session/.../rag/](#)

Löse Lesson\_01..03...java

# Splitting: The secret ingredient

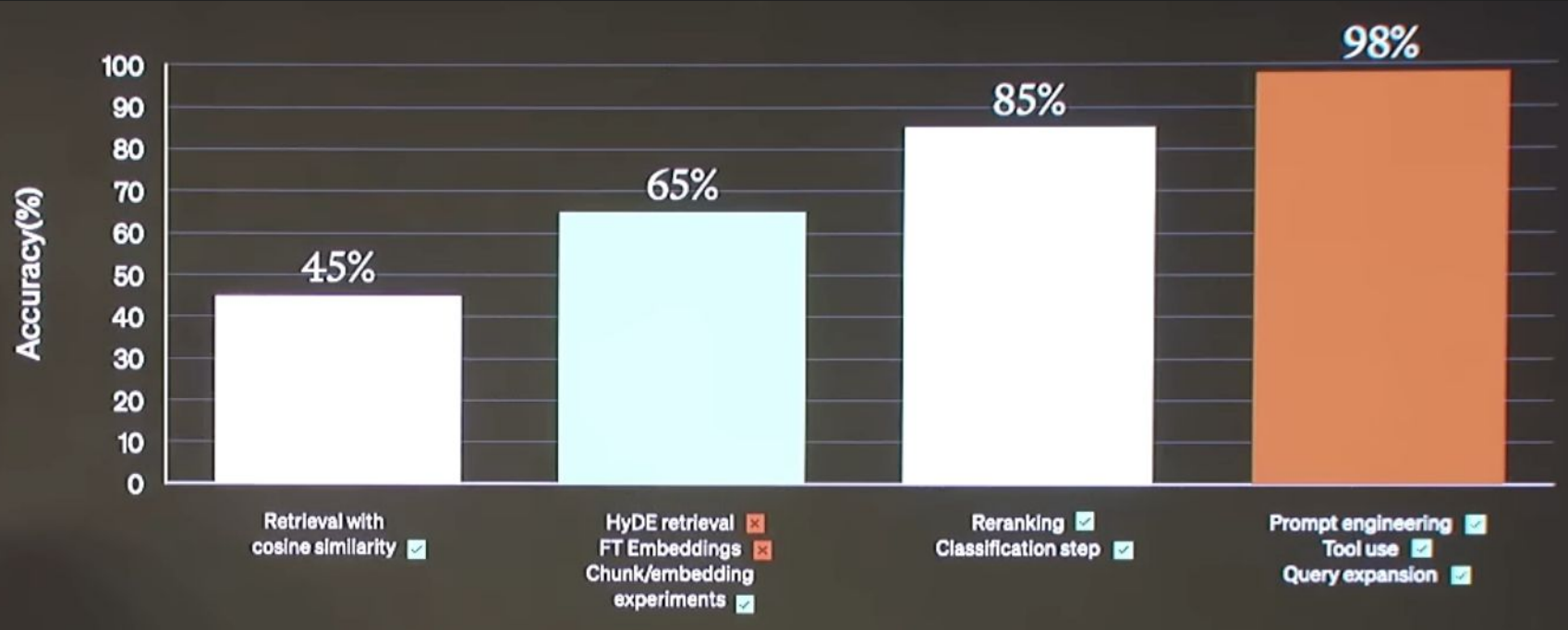
1. Ein Text muss in das Contextwindow passen!!!
2. Ist der Text ein sinnvolles Suchergebnis?
  - a. Schätzen der Tokens
  - b. Semantisches: Splitten nach Satz Ende, Paragraph, Überschrift
  - c. Brauche ich einen Overlap vom vorherigen Text Segment?
  - d. Wie gehe ich mit Bildern und Tabellen um?

## 4.2 In Vectorstore suchen

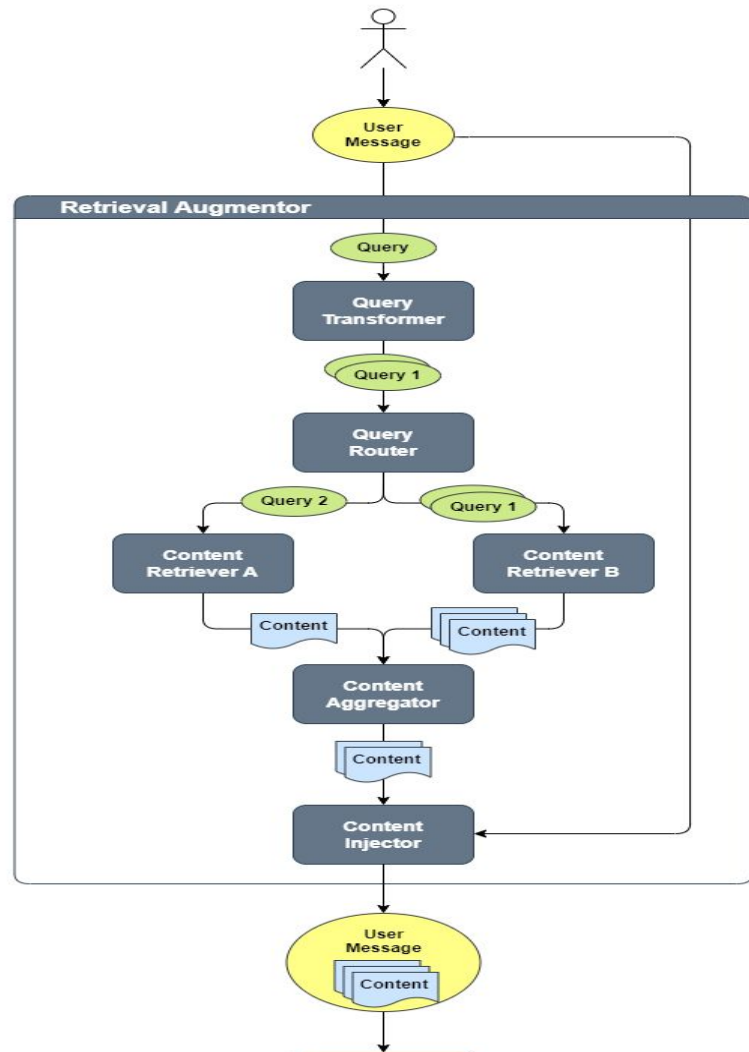


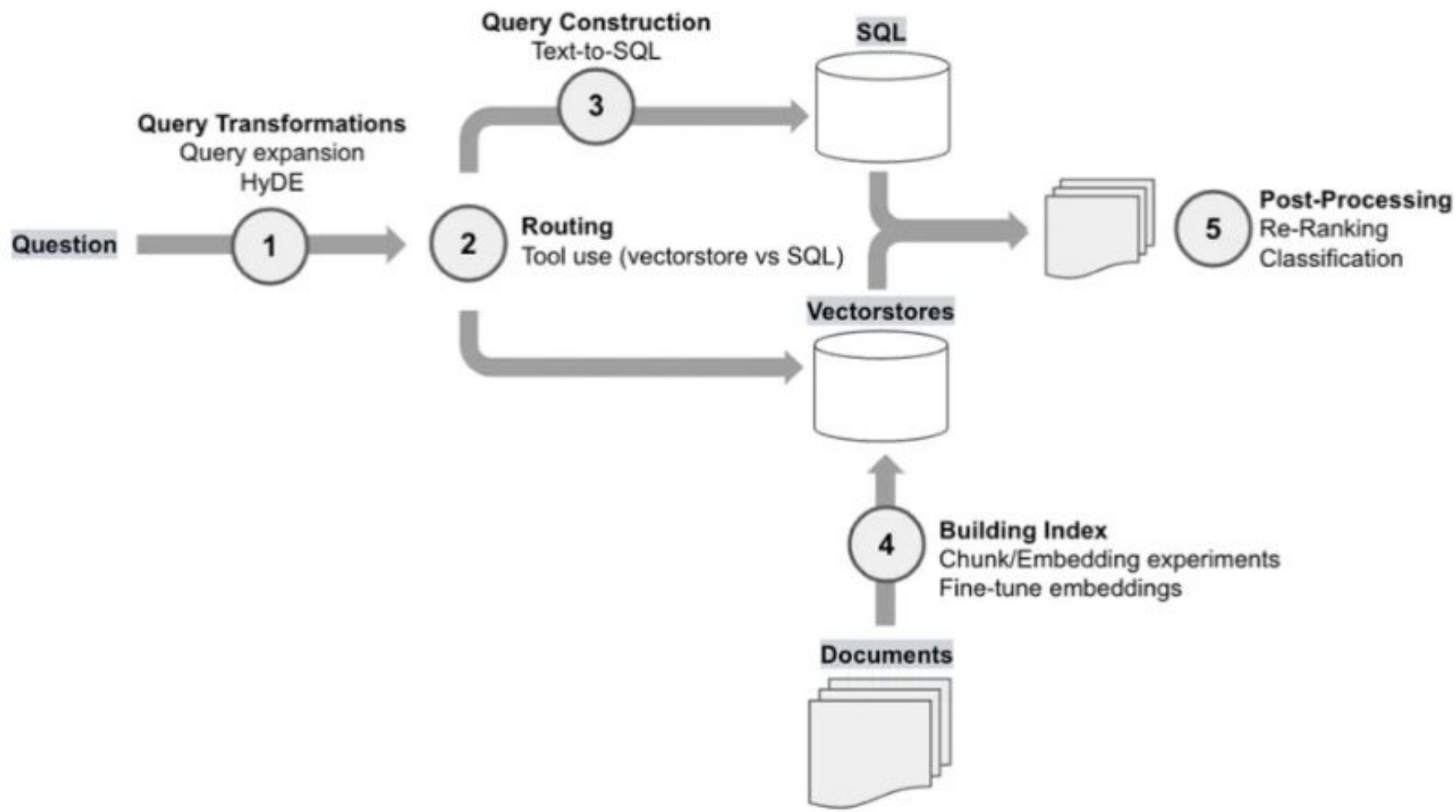
# Open AI Untersuchung zu RAG Strategien

<https://blog.langchain.dev/applying-openai-rag/>



# Erweitertes RAG

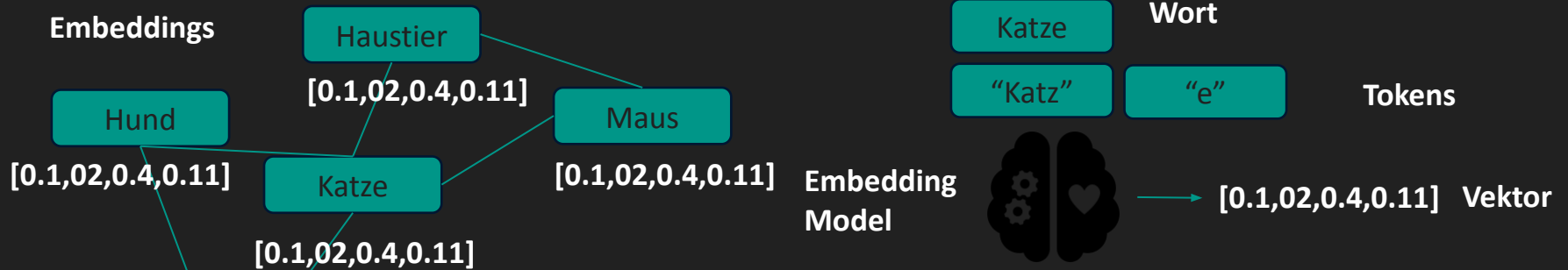






**ENDE**

# Jedes Wort ist in Beziehung

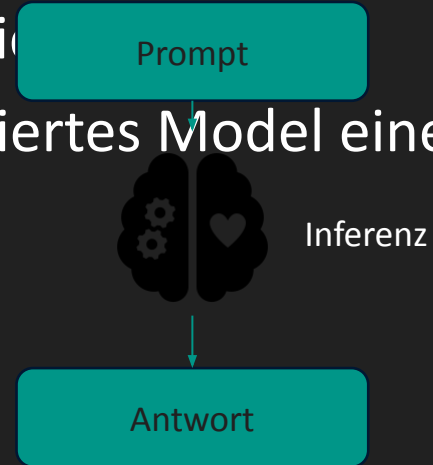


LLM berechnet aus Embedding den Self-Attentionscore

Wort	die	Katze	jagt	die	Maus
jagt	0.1	0.8	1.0	0.2	0.9

# *Prompt & Inferenz*

- Prompt: Eingabe die man an das LLM schickt
- Inferenz ist der Prozess aus dem ein trainiertes Model eine Antwort generiert



# 1 Aufruf vom LLM via Java

- Benutzen des ChatLanguageModel
- Wird automatisch über spring-boot starter injiziert
- Wird über application.yml konfiguriert mit API key etc.

```
@PostMapping("/ask/simple") @beisdog *  
public Message ask(@RequestBody AskInput input) {  
    var response = this.chatLanguageModel.chat(input.question);  
}
```

# *“Die Katze jagt die Maus”*

## *Tokenisierung und Vektorisierung*

- Satz wird in Tokens zerlegt
- ["Die", "Katze", "jagt", "die", "Maus"]
- Token -> Embedding Model -> Vector
- [0.23, -0.17, 0.45, ..., 0.12]
- Vector Dimension hängt vom Modell ab, Worte mit ähnlicher Bedeutung sind in ähnlichen Vektorräumen

# *“Die Katze jagt die Maus”*

## *2. Berechnung von Query, Key, Value*

- Jedes Token enthält 3 Vektoren:
  - Query: “Wonach suche ich?”
  - Key: “Welche Informationen biete ich an?”
  - Value: “Welche Informationen gebe ich weiter?”

Berechnung durch Matrix-Multiplikation mit den Embeddings und ursprünglichen Wortvektor

# *“Die Katze jagt die Maus”*


## *3. Berechnung Attention Score*

Wort	die	Katze	jagt	die	Maus
jagt	0.1	0.8	1.0	0.2	0.9

- Jagt steht in grosser Beziehung zu Katze und Maus

# *“Die Katze jagt die Maus”*

## *4. Berechnung Wort-Repräsentation*

- Jedes Wort bekommt nun eine gewichtete Summe der Value-Vektoren (V) der anderen Wörter, basierend auf den berechneten Attention-Scores.
-  Beispiel für "jagt":
- Da "Katze" (0.8) und "Maus" (0.9) eine hohe Relevanz haben, wird der neue Repräsentationsvektor von "jagt" stark von den Value-Vektoren von "Katze" und "Maus" beeinflusst.
- Dadurch kann das Modell verstehen, dass "jagt" eine Handlung zwischen diesen beiden Wörtern beschreibt.



# *“Die Katze jagt die Maus”*

## *5. Multi Head Attention*

- Anstatt nur eine einzelne Self-Attention-Berechnung durchzuführen, nutzt der Transformer mehrere Attention-Köpfe, die den Satz aus verschiedenen Perspektiven betrachten.
  - Ein Kopf könnte sich auf die grammatikalische Struktur fokussieren. Ein anderer Kopf könnte semantische Bedeutungen erfassen.
  - Diese unterschiedlichen "Sichten" werden dann kombiniert und an die nächste Schicht des Modells weitergegeben.