



# Community of Practice KIPerWeb

Austausch zur Nutzung und Entwicklung KI-gestützter Webanwendungen



**KIPerWEB**



**Forschungsinstitut  
Betriebliche Bildung**

# Agenda

---



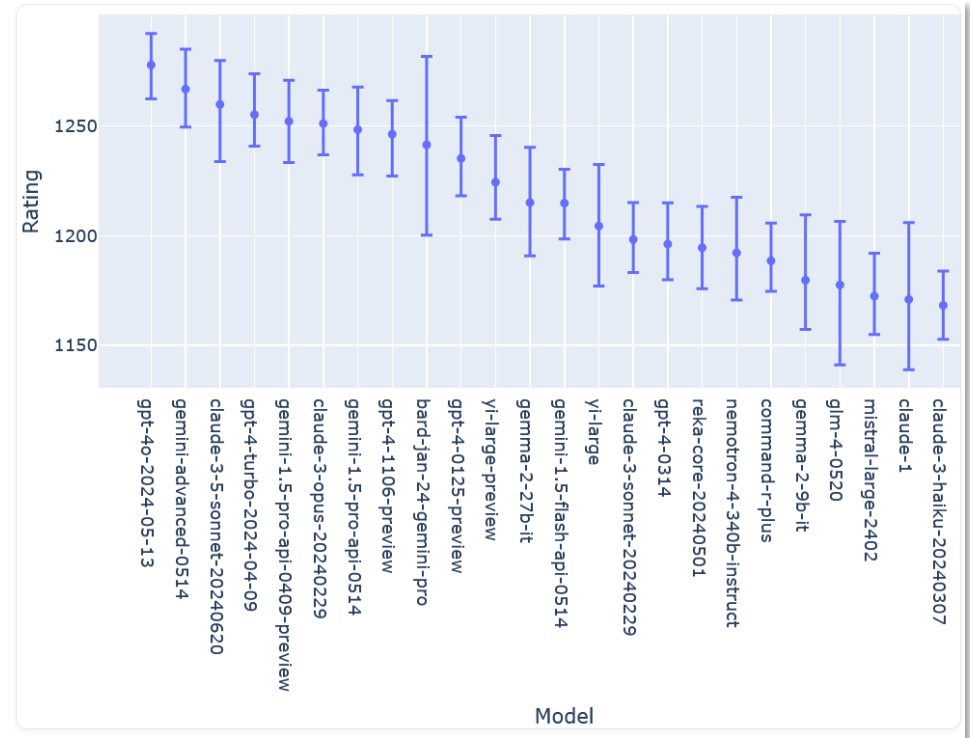
- **Update**
  - News & Leaderboard-Update
- **Input**
  - „Knowledge Graphs & Retrieval-Augmented Generation “
- **Diskussion**

# News & Update (11.07.2024)



- Gemma-2-27b-it liegt – ebenso wie *Nemotron-4-340b-Instruct* (NVIDIA Open Model Licence) – noch vor Command-R-Plus (CC-BY-NC-4.0), knapp Gemma-2-9b-it (Gemma Licence)!
- *Mixtral-8x22b-Instruct-v0.1* – nicht mehr im Bild aber trotzdem hervorragend 😊 – bleibt vor *Mixtral-8x7b-Instruct-v0.1* das beste Modell unter Apache 2.0

Confidence Intervals on model strength (Arena Elo, German)



# Knowledge Graph (KG)



- Wissensgraphen repräsentieren Wissen als Knoten (Entitäten) und Kanten (Beziehungen)
  - Ggf. jeweils mit Labels & Attributen nach Wahl
- Lassen sich z.B.
  - Für Trainindatensynthese verwenden,
  - mit LLMs extrahieren,
  - von LLMs befragen (i.S.v. Cypher-Query)
  - an LLMs senden (i.S.v. RAG)
  - ...

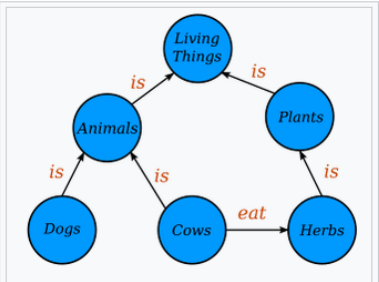
## Knowledge graph

Article [Talk](#) Read [Edit](#) [View history](#) [Tools](#)

From Wikipedia, the free encyclopedia

*For other uses, see [Knowledge graph \(disambiguation\)](#).*

In [knowledge representation and reasoning](#), a **knowledge graph** is a [knowledge base](#) that uses a [graph-structured data model](#) or [topology](#) to represent and operate on [data](#). Knowledge graphs are often used to store interlinked descriptions of [entities](#) – objects, events, situations or abstract concepts – while also encoding the free-form [semantics](#) or relationships underlying these entities.<sup>[1][2]</sup>



Example conceptual diagram

The diagram illustrates a knowledge graph with six entities represented as blue circles: 'Living Things', 'Animals', 'Plants', 'Dogs', 'Cows', and 'Herbs'. Relationships are shown as orange arrows with labels: 'is' connects 'Dogs' to 'Animals', 'Animals' to 'Living Things', and 'Plants' to 'Living Things'; 'is' also connects 'Cows' to 'Animals' and 'Herbs' to 'Plants'; an 'eat' relationship connects 'Cows' to 'Herbs'.

Quelle: [https://en.wikipedia.org/wiki/Knowledge\\_graph](https://en.wikipedia.org/wiki/Knowledge_graph)

Cf. Agarwal et al., (2010) Synthetic Corpus Generation; Xu et al. (2024) RAG with KG; He et al. (2024) G-Retriever; Hu et al. (2024) GRAG; Traditional GraphRAG: Baek et al., (2023); Kang et al. (2023); Sen et al (2023); Sun et al. (2023)

# Retrieval-Augmented Generation (RAG)



- RAG zielt darauf ab, die User-Message im Prompt, um passende Informationen aus einer Datenbank anzureichern.
  - Informationen/Texte werden über eine sog. „embedding function“ in Vektoren überführt (sog. Sentence Embeddings)
  - Texte werden nebst Sentence Embeddings idR. in einer Vektordatenbank hinterlegt (z.B. ChromaDB)
  - User-Message wird zur Laufzeit ebenfalls in ein Sentence Embedding überführt und die Texte mit den mathematisch ähnlichsten Vektoren in der Datenbank werden abgerufen und in den Prompt für's LLM integriert
- Rechts ein Beispiel für Embedding- & Retrieval-Prozess
  - z.B. Ergänzung im Preprompt:  
*„Beantworte Fragen und berücksichtige dabei ggf. die folgenden Informationen: {Texte}“*

Quelle: <https://huggingface.co/spaces/AFischer1985/AI-RAG-Interface-to-Hub>

```
# Chroma-DB
#-----
import os
import chromadb
path="/home/user/app/db"
client = chromadb.PersistentClient(path=path)
from chromadb.utils import embedding_functions
default_ef = embedding_functions.DefaultEmbeddingFunction()
instructor_ef = embedding_functions.InstructorEmbeddingFunction(
    model_name="hkunlp/instructor-large" #; device="cuda")
print(str(client.list_collections()))
global collection
if("name=ChromaDB1" in str(client.list_collections())):
    print("ChromaDB1 found!")
    collection = client.get_collection(name="ChromaDB1", embedding_function=instructor_ef)
else:
    print("ChromaDB1 created!")
    collection = client.create_collection(
        "ChromaDB1",
        embedding_function=sentence_transformer_ef,
        metadata={"hnsw:space": "cosine"})
    collection.add(
        documents=[
            "Text generating AI model mistralai/Mistral-7B-Instruct-v0.1",
            "Image generating AI model stabilityai/sdxl-turbo",
            "Audio transcribing AI model openai/whisper-large-v3",
            "Speech synthesizing AI model coqui/XTTS-v2",
            "Code generating AI model deepseek-ai/deepseek-coder-6.7b-instruct",
            "Translation AI model Helsinki-NLP/opus-mt",
            "Search result-integrating AI model phind/phind-v9-models"
        ],
        metadatas=[{"source": "AF"}, {"source": "AF"}, {"source": "AF"},
                    {"source": "AF"}, {"source": "AF"}, {"source": "AF"},
                    {"source": "AF"}],
        ids=["ai1", "ai2", "ai3", "ai4", "ai5", "ai6", "ai7"],
    )

print("Database ready!")
print(collection.count())
message="Dies ist ein Test!"
results=collection.query(
    query_texts=[message],
    n_results=2)
```

## GraphRAG:

- a) use LLM to identify key entities and their relationships from texts to build a knowledge graph, (Indexing) and/or
- b) use knowledge graph form Retrieval-Augmented Generation (Querying)

e.g.

- LlamaIndex GraphRAG (Dec, 2023)
  - [https://docs.llamaindex.ai/en/stable/examples/query\\_engine/knowledge\\_graph\\_rag\\_query\\_engine/](https://docs.llamaindex.ai/en/stable/examples/query_engine/knowledge_graph_rag_query_engine/)
  - <https://medium.aiplanet.com/implement-rag-with-knowledge-graph-and-llama-index-6a3370e93cdd>
- Microsoft GraphRAG (Feb, 2024; open-sourced Jul, 2024)
  - <https://github.com/microsoft/graphrag>
  - <https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/>
- Neo4j GraphRAG (May, 2024)
  - <https://medium.com/neo4j/from-zero-to-graphrag-in-5-minutes-4ffcfcb4ebc2>
  - <https://neo4j.com/developer-blog/going-meta-knowledge-graph-rag-vector/>
- Diffbot GraphRAG (Jul, 2024)
  - [https://www.linkedin.com/posts/leann-chen\\_neo4j-diffbot-graphrag-ugcPost-7211592479052087297-04nK?utm\\_source=share&utm\\_medium=member\\_desktop](https://www.linkedin.com/posts/leann-chen_neo4j-diffbot-graphrag-ugcPost-7211592479052087297-04nK?utm_source=share&utm_medium=member_desktop)

- Basic idea: RAG based on (a) sentence embeddings of Nodes/Entities and (b) sub-graphs of these nodes (2-hops)
- KnowledgeGraphRAGRetriever:
  - Search related Entities of the question/task
  - Get SubGraph of those Entities (default 2-depth) from the KG
  - Build Context based on the SubGraph



Quellen: [https://docs.llamaindex.ai/en/stable/examples/query\\_engine/knowledge\\_graph\\_rag\\_query\\_engine/](https://docs.llamaindex.ai/en/stable/examples/query_engine/knowledge_graph_rag_query_engine/) ;  
[https://github.com/run-llama/llama\\_index/blob/main/llama-index-core/llama\\_index/core/indices/knowledge\\_graph/retrievers.py](https://github.com/run-llama/llama_index/blob/main/llama-index-core/llama_index/core/indices/knowledge_graph/retrievers.py)

# Microsoft GraphRAG



## Element Instances:

- nodes/entities,
- Edges/relationships
- Ggf. weitere Attribute

Summary of instance-level descriptions for each element

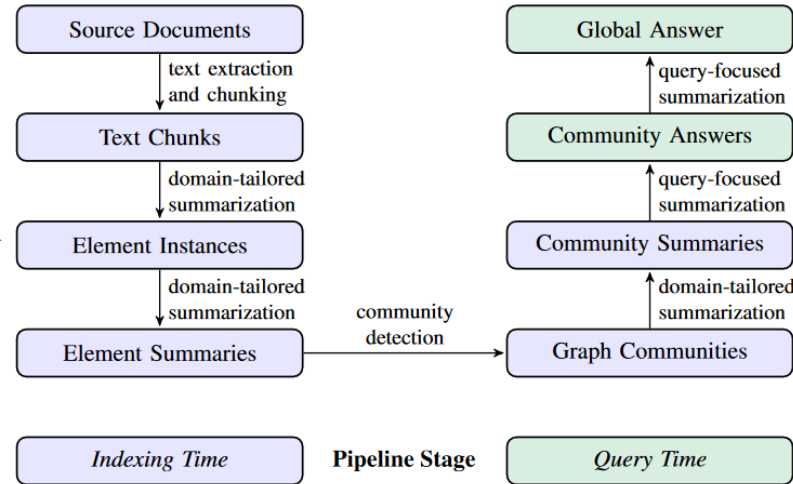


Figure 1: Graph RAG pipeline using an LLM-derived graph index of source document text. This index spans nodes (e.g., entities), edges (e.g., relationships), and covariates (e.g., claims) that have been detected, extracted, and summarized by LLM prompts tailored to the domain of the dataset. Community detection (e.g., Leiden, Traag et al., 2019) is used to partition the graph index into groups of elements (nodes, edges, covariates) that the LLM can summarize in parallel at both indexing time and query time. The “global answer” to a given query is produced using a final round of query-focused summarization over all community summaries reporting relevance to that query.

## Global Search:

- Map: Answer based on each community
- Reduce: summarize partial answers

## Local Search:

- Answer based on nodes and their nearest neighbours

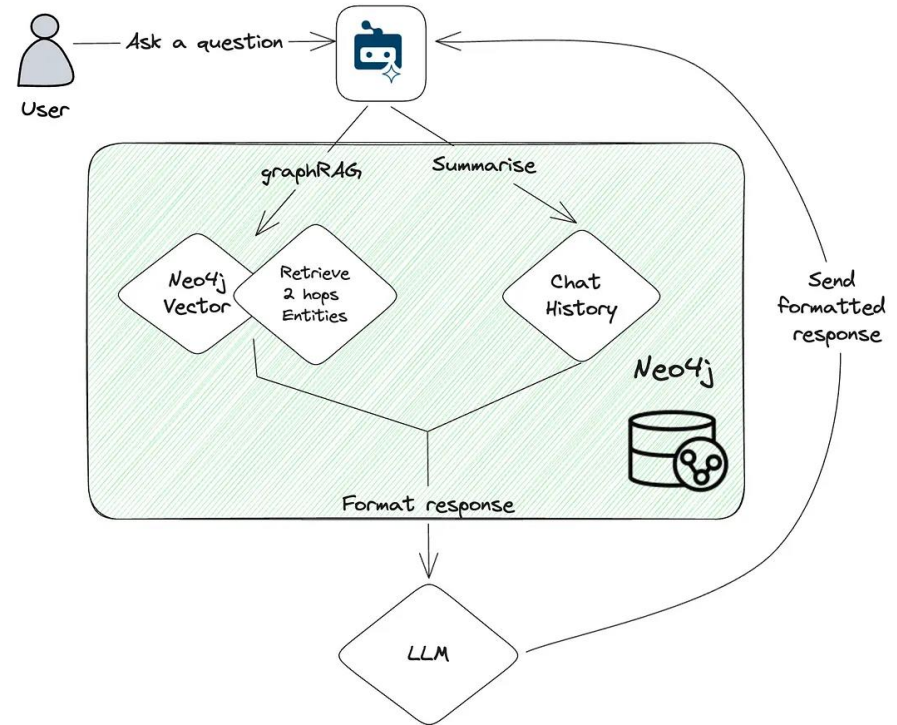
cf. Jonathan Larson in <https://www.youtube.com/watch?v=r09tJfON6kE>



# Neo4j GraphRAG-Retrieval

- „When the user asks a question, we use the Neo4j vector index with a retrieval query to find the most relevant chunks for the question and their connected entities up to a depth of 2 hops. We also summarize the chat history and use it as an element to enrich the context.”
- Zur Veranschaulichung: ein 2-hop subgraph von Knoten vom Typ A zum Knoten mit ID 123 lässt sich in Neo4j z.B. via Cypher folgendermaßen abfragen:

```
MATCH (c:A{ID:'123'})-[*1..2]->(d)
WHERE NONE(rel in r WHERE startNode(rel) = endNode(rel))
RETURN Distinct(d)
```



# Neo4j GraphRAG-Retrieval Retrieval Prompt



```
RETRIEVAL_QUERY = ""
WITH node as chunk, score
MATCH (chunk)-[:PART_OF]->(d:Document)
CALL { WITH chunk
MATCH (chunk)-[:HAS_ENTITY]->(e)
MATCH path=(e)((-)[rels:!HAS_ENTITY&!PART_OF]-()){0,3}{(!Chunk&!Document)}
UNWIND rels as r
RETURN collect(distinct r) as rels
}
WITH d, collect(distinct chunk) as chunks, avg(score) as score, apoc.coll.toSet(apoc.coll.flatten(collect(rels))) as rels
WITH d, score,
[c in chunks | c.text] as texts,
[r in rels | coalesce(apoc.coll.removeAll(labels(startNode(r)),['__Entity__'])[0],"") + ":" + startNode(r).id + " " + type(r) + " " +
coalesce(apoc.coll.removeAll(labels(endNode(r)),['__Entity__'])[0],"") + ":" + endNode(r).id] as entities
WITH d, score,
apoc.text.join(texts,"\n----\n") +
apoc.text.join(entities,"\n")
as text, entities
RETURN text, score, {source: COALESCE(CASE WHEN d.url CONTAINS "None" THEN d.fileName ELSE d.url END, d.fileName),
entities:entities} as metadata
""
```

# Neo4j GraphRAG-Retrieval Final Prompt



```
FINAL_PROMPT = """
```

You are an AI-powered question-answering agent tasked with providing accurate and direct responses to user queries. Utilize information from the chat history, current user input, and Relevant Information effectively.

Response Requirements:

- Deliver concise and direct answers to the user's query without headers unless requested.
- Acknowledge and utilize relevant previous interactions based on the chat history summary.
- Respond to initial greetings appropriately, but avoid including a greeting in subsequent responses unless the chat is restarted or significantly paused.
- For non-general questions, strive to provide answers using chat history and Relevant Information ONLY do not Hallucinate.
- Clearly state if an answer is unknown; avoid speculating.

Instructions:

- Prioritize directly answering the User Input: {question}.
  - Use the Chat History Summary: {chat\_summary} to provide context-aware responses.
  - Refer to Relevant Information: {vector\_result} only if it directly relates to the query.
  - Cite sources clearly when using Relevant Information in your response [Sources: {sources}] without fail. The Source must be printed only at the last in the format [Source: source1,source2] . Duplicate sources should be removed.
- Ensure that answers are straightforward and context-aware, focusing on being relevant and concise.

```
"""
```

# Diffbot GraphRAG: Vector-only vs. Vector+KG



- Q: „List out nvidia's partners who are also its competitors at the same time.“

Introduction

Import article

Natural

Enhance

Dashboard

Network graph

Chat agent

Context Data

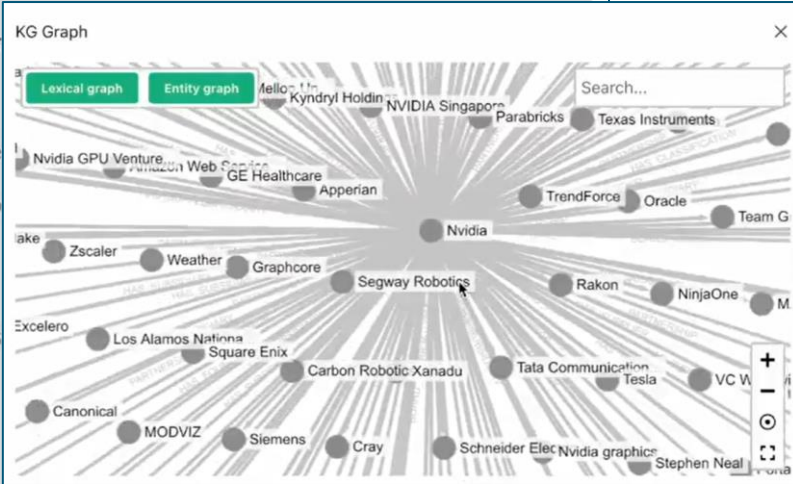
is case, the model chosen was  
ch itself was trained by Nvi  
ced. Nvidia pushed Megatron  
rful of the so-called "tran  
ia's "Ampere" A100 GPU acce  
Megatron LLM chosen by the  
DeepSpeed library for the PyTor  
accelerate its own GPT implemen  
that created DeepSpeed said tha  
There are a lot of tricks for i  
here.)  
Fujitsu and RIKEN tweaked the M  
called Tofu D, employed in the  
Fujitsu and which has 7.3 milli  
petaflops at FP64 precision. (M  
throughput.) Presumably the Dee  
the Megatron model down to FP16  
of Technology worked on the sev  
the performance of the Tofu D c  
framework that supports Megatro  
the Fugaku-LLM.  
The source code for Fugaku-LLM  
there. Quick to seize any oppor  
Samba-1 collection of models fo

## Context Data

### Structured data:

Nvidia - HAS\_SUBSIDIARY -> NVIDIA  
Nvidia - PARTNERSHIP -> CRWD  
Nvidia - PARTNERSHIP -> Deutsche Bank  
Nvidia - PARTNERSHIP -> Square Enix  
Nvidia - PARTNERSHIP -> YTL Power  
Nvidia - PARTNERSHIP -> Tobii  
Nvidia - PARTNERSHIP -> Getty Images  
Nvidia - PARTNERSHIP -> Georgia Institute of T  
Nvidia - PARTNERSHIP -> Harvard University  
Nvidia - PARTNERSHIP -> Canonical  
Nvidia - PARTNERSHIP -> Johnson & Johnson  
Nvidia - PARTNERSHIP -> Mozilla's Common Voice  
Nvidia - PARTNERSHIP -> Amazon  
Nvidia - PARTNERSHIP -> Hewlett Packard Enterp  
Nvidia - PARTNERSHIP -> YTL  
Nvidia - PARTNERSHIP -> Amazon Web Services  
Nvidia - PARTNERSHIP -> Mod DB  
Nvidia - PARTNERSHIP -> Segway Robotics  
Nvidia - PARTNERSHIP -> United States Postal S  
Nvidia - PARTNERSHIP -> University of Arizona  
Nvidia - PARTNERSHIP -> Microsoft  
Nvidia - PARTNERSHIP -> Zscaler  
Nvidia - PARTNERSHIP -> Dell  
Nvidia - PARTNERSHIP -> Volvo


- Relations u.a.
- PARTNERSHIP
  - HAS\_FOUNDER
  - HAS\_COMPETITOR
  - HAS\_SUPPLIER
  - ...



Quelle: [https://www.linkedin.com/posts/leann-chen\\_neo4j-diffbot-graphrag-ugcPost-7211592479052087297-04nK?utm\\_source=share&utm\\_medium=member\\_desktop](https://www.linkedin.com/posts/leann-chen_neo4j-diffbot-graphrag-ugcPost-7211592479052087297-04nK?utm_source=share&utm_medium=member_desktop)

# GraphRAG ohne Datenbank? NetworkX



- Für Wissensgraphen in Python reichen auch open-source Bibliotheken wie NetworkX  vgl.  
[https://github.com/AndreasFischer1985/code-snippets/blob/master/py/BERUFENET\\_KnowledgeGraphExample.py](https://github.com/AndreasFischer1985/code-snippets/blob/master/py/BERUFENET_KnowledgeGraphExample.py)

- Das Beispiel rechts
  - Erstellt einen „directed graph“
  - Bestimmt zu einem Knoten (1) alle Knoten, die nicht weiter als 2 Relationen entfernt sind
  - Extrahiert einen Subgraph mit den o.g. Knoten und all ihren Verbindungen

```
import networkx as nx

DG=nx.DiGraph()
DG.add_nodes_from([1,2,3,4,5])
DG.add_edges_from([(1,2),(2,3),(3,4),(2,3),(2,5),(3,5)])

path_lengths=nx.single_source_shortest_path_length(DG, source=1, cutoff=2)
#{1: 0, 2: 1, 3: 2, 5: 2}
nodes_within_2_hops=[node for node, length in path_lengths.items() if length <=2]
#[1, 2, 3, 5]

connected_nodes=nodes_within_2_hops
connected_nodes
subgraph=nx.DiGraph()
for node in connected_nodes:
    subgraph.add_node(node)

for node in connected_nodes:
    for neighbor in connected_nodes:
        if node != neighbor and DG.has_edge(node,neighbor):
            subgraph.add_edge(node,neighbor)
```

Quelle: [https://github.com/AndreasFischer1985/code-snippets/blob/master/py/networkx\\_KnowledgeGraphs%26Subgraphs.py](https://github.com/AndreasFischer1985/code-snippets/blob/master/py/networkx_KnowledgeGraphs%26Subgraphs.py)

- Wo macht KG-RAG Sinn, wo reicht ggf. klassisches RAG?
- Welche KG-/GraphRAG-Variante ist besonders zukunftsweisend?
- Welche weiteren Varianten und Komponenten sollte man kennen/prüfen?
  - z.B. RAG von Entities, Relations & Triplets in einem hierarchischen RAG-Prozess?
  - z.B. empfehlenswerte Sentence-/KG-Embeddings?