Community of Practice KI*PerWeb*

Austausch zur Nutzung und Entwicklung KI-gestützter Webanwendungen

KIPERWEB

f-bb Forschungsinstitut Betriebliche Bildung

Agenda

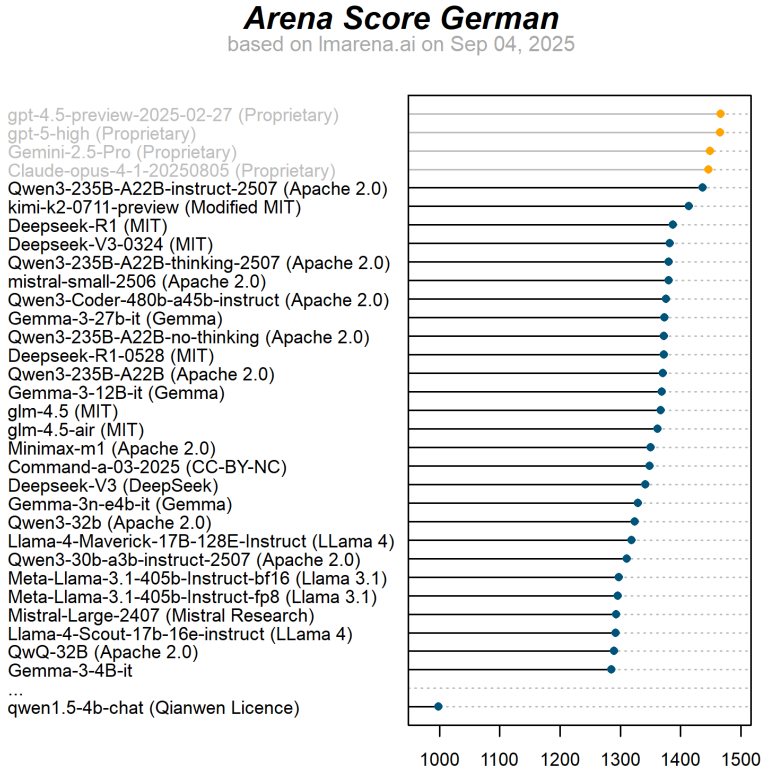
- Update
  - News & Leaderboard-Update
- Input
  - "KI & Web-Search-Augmented Generation"
- Diskussion

Leaderboard-Update (04.09.2025)

Rechts das aktuelle LMArena-Leaderboard für die Kategorie German mit Blick auf die besten open-weights-Modelle (sowie ausgewählte proprietäre Modelle).

Bemerkenswert:

- GPT-5-high wurde als Spitzenreiter abgelöst von GPT-4.5-preview-2025-02-27
- Qwen3-235B-A22B-instruct-2507 ist zum besten open-weights-Modell aufgestiegen
- Mistral-small-2506 (24B) und Gemma-3-27b-it liegen aktuell vor dem bislang besseren Gemma-3-12B-it.
- Fliegengewicht Gemma-3n-e4b-it liegt aktuell sogar noch vor Qwen3-32b
- OpenAI‘s gpt-oss sowie NVIDIA‘s Nemotron-Modelle aktuell für die Kategorie German nicht mehr auf dem Leaderboard
- Ausgewiesenes Schlusslicht ist nun qwen1.5-4b-chat (998) statt Chatglm2-6b (zuletzt 755) – das Niveau steigt



Datengrundlage: https://lmarena.ai/leaderboard/text/german

Fokusthema: KI & Web-Search-Augmented Generation

- Prompt:

  „KI & Web-Search-Augmented Generation“

  (rechts visualisiert von FLUX.1-schnell, seed 1775064495)

  ...

Quelle: https://huggingface.co/spaces/black-forest-labs/FLUX.1-schnell

Wikipedia-Search as a Tool (2023)

- LangChain Agents mit Wikipedia-Tool gibt es in der #CoP_KIPerWeb seit 17. April 2023: https://github.com/AndreasFischer1985/code-snippets/blob/890a3f24585bb7754d14b6f834984b4d9467cc88/py/LangChain_HuggingFace_examples.py#L137-L147

```
from langchain.agents import Tool, initialize_agent
from langchain.utilities import WikipediaAPIWrapper
tools=[Tool(
  name="Wikipedia",
  func=WikipediaAPIWrapper(top_k_results=2).run,
  description="A wrapper around Wikipedia."+
    " Useful for when you need to answer general questions about"+
    " people, places, companies, historical events, or other subjects."+
    " Input should be a search query.")]
agent = initialize_agent(tools, llm, agent="zero-shot-react-description", verbose=True)
agent("What is the meaning of life?")
```

Wikipedia-Agent Prompt (Credits to LangChain ´23)

```
# Wikipedia-agent-prompt:
"""
Answer the following questions as best you can. You have access to the following tools:

Wikipedia: A wrapper around Wikipedia. Useful for when you need to answer general questions about
people, places, companies, historical events, or other subjects. Input should be a search query.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [Search]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:
"""
```

Vgl. Template unter https://python.langchain.com/api_reference/_modules/langchain/agents/react/agent.html#create_react_agent
Sowie **Update** unter https://smith.langchain.com/hub/langchain-ai/react-agent-template (insb. ergänzt um Möglichkeit direkter Antworten)

Source: https://github.com/AndreasFischer1985/code-snippets/blob/8c7f8d5a34bb3176069e8b89b6c798c029d9e732/py/LangChain_HuggingFace_examples.py#L286-L307

WebSearch-Agent (LangChain 2025)

```python
# Import relevant functionality
from langchain.chat_models import init_chat_model
from langchain_tavily import TavilySearch
from langgraph.checkpoint.memory import MemorySaver
from langgraph.prebuilt import create_react_agent

# Create the agent
memory = MemorySaver()
model = init_chat_model("anthropic:claude-3-5-sonnet-latest")
search = TavilySearch(max_results=2)
tools = [search]
agent_executor = create_react_agent(model, tools, checkpointer=memory)
```

Hinweis: Adapter für MCP-Tools: https://github.com/langchain-ai/langchain-mcp-adapters

Source: https://python.langchain.com/docs/tutorials/agents/

Exkurs: Web-Search via API im Eigenbau (curl)

Beispiel DuckDuckGo: Technisch bietet DuckDuckGo die Möglichkeit, Ergebnisse über eine API oder Web Scaping zu beziehen – über einfache GET-requests z.B. an:

- http://api.duckduckgo.com/?q=x&format=json (Instant Answer API)
- https://duckduckgo.com/html?q=x
- https://duckduckgo.com/lite?q=x

Es existieren auch deziderte Python-Packages (z.B. duckduckgo_search).

Die Robots.txt untersagt für duckduckgo.com jedoch das Scraping mit query-string (/*?) und für api.duckduckgo.com scheinbar sogar alles (/) – letzteres aber im Widerspruch zu früheren Äußerungen von Duckduckgo… 🤔

```python
import requests
def search_ddg(query):
  url = "https://api.duckduckgo.com/"
  params = {"q": query, "format": "json"}
  response = requests.get(url, params=params)
  return response.json()

results = search_ddg("Python programming")
print(results["RelatedTopics"])
```

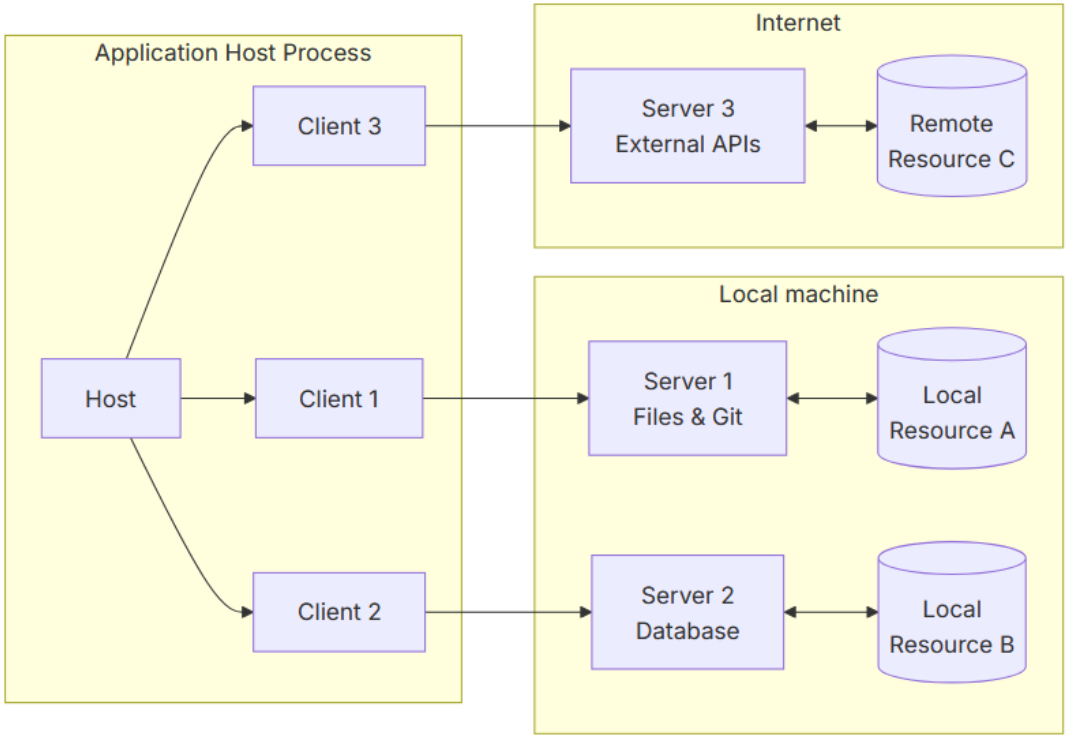
Cf. https://stackoverflow.com/questions/37012469/duckduckgo-api-getting-search-results
https://www.postman.com/api-evangelist/duckduckgo/documentation/i9r819s/duckduckgo-instant-answer-api

Web-Search heute: MCP Servers

- „Das **Model Context Protocol (MCP)** ist ein offener Standard und Open-Source-Framework, das vom US-Unternehmen Anthropic entwickelt wurde, um die Integration und den Datenaustausch zwischen künstlicher Intelligenz (KI), insbesondere großen Sprachmodellen (LLMs), und externen Tools, Systemen sowie Datenquellen zu standardisieren.“ (Wikipedia „MCP“)
- **MCP-Kernkomponenten** gemäß modelcontextprotocol.io (Version 2025-06.18):
  - MCP-Host (KI-System/-Applikation wie z.B. Claude Code, Claude Desktop oder ein Eigenbau)
  - MCP-Server (Programm, das *Tools, Resources* und/oder *Prompt*-Templates bereitstellt)
  - MCP-Client (Konnektoren für *Anfragen* von MCP-Hosts an Server, oder für *Sampling, Roots, Elicitation*)
- Zugriff auf MCP-Server erfolgt entweder lokal (stdio) oder remote (sse, http) möglich – speziell für WebSearch gibt es zahlreiche Angebote auf Github oder bei kommerziellen Anbietern

Cf. https://modelcontextprotocol.io/docs/getting-started/intro
https://modelcontextprotocol.io/specification/2025-06-18

Model Context Protocol (MCP) - Core Components



https://modelcontextprotocol.io/specification/2025-06-18/architecture

Beispiel: MCP-Server & -Client für Web Search

- Rechts ein Custom MCP-Server für WebSearch in Python (inkl. Client fürs Testing)
- Das Ergebnis hat folgende Form: *CallToolResult(content=[TextContent(type='text', text='{"Abstract":"", ...}', is_error=False)*
- Analog dazu lassen sich Clients in eigene Applikationen integrieren
- Über client.list_tools erhält man Einblick in die Tools, als Grundlage für den System Prompt eines MCP-Hosts - in unserem Fall z.B.
    - [Tool(name='websearch', title=None, description='Performs a web search ...', inputSchema={'properties':{'query':{'title': 'Query', 'type': 'string'}}, 'required': ['query'], 'type': 'object'}, outputSchema={'peroperties': {'result': {'title': 'Result', 'type': 'string'}}, 'required': ['result'], 'title': '_WrappedResult', 'type': 'object', 'x-fastmcp-wrap-result': True}, annotations=None, meta={'_fastmcp': {'tags': []}})]
- Auch sehr komfortabel sind MCP-Server via Gradio: https://www.gradio.app/guides/building-mcp-server-with-gradio

```python
from fastmcp import FastMCP, Client
from typing import Any
import httpx
import asyncio

mcp = FastMCP("DuckDuckGo")

@mcp.tool()
async def websearch(query: str) -> str:
    """Performs a web search using the DuckDuckGo Instant Answer API.

    Args:
        query: The search term.
    """
    url = f"https://api.duckduckgo.com/?q={query}&format=json"
    async with httpx.AsyncClient() as client:
        try:
            response = await client.get(url)
            response.raise_for_status()
            return response.text
        except Exception as e:
            return e

async def main():
  async with Client(mcp) as client:
    result = await client.call_tool("websearch",{"query":"Test"})
    print(result)

if __name__ == "__main__":
    asyncio.run(main())
```

Diskussion

- Fragen?
- Anregungen?
- Erfahrungen?