



Community of Practice KIPerWeb

Austausch zur Nutzung und Entwicklung KI-gestützter Webanwendungen



KIPerWEB



**Forschungsinstitut
Betriebliche Bildung**

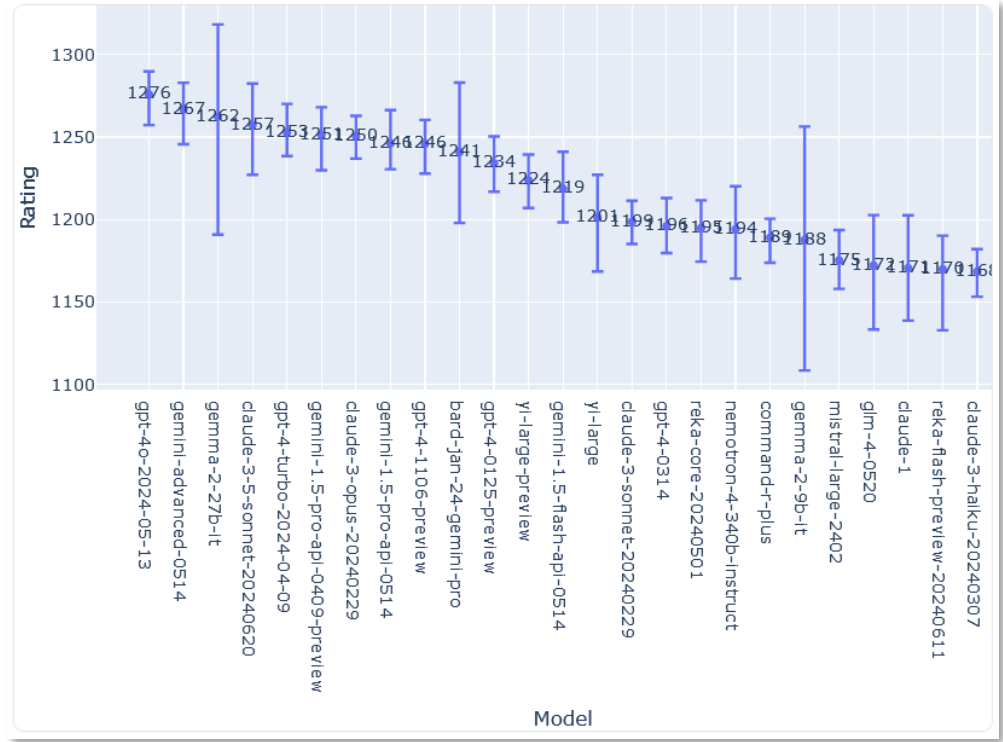
- **Update**
 - News & Leaderboard-Update
- **Input**
 - „Chatbots mit Large Language Models realisieren - Varianten & Best Practice“
- **Diskussion**

News & Update (26.06.2024)



- Gemma-2-27b-it liegt für deutsche Dialoge auf Platz 3, Gemma-2-9b-it knapp hinter Command-R-Plus!
- Claude 3.5 Sonnet liegt für deutsche Dialoge auf Platz 4 (generell auf Platz 2 und für Coding sogar auf Platz 1 noch vor GPT-4o)
 - allerdings unter proprietärer Lizenz 🗨️
- Nemotron-4-340b-Instruct (NVIDIA Open Model Licence) liegt noch vor Command-R-Plus (CC-BY-NC-4.0)!
- Mixtral-8x22b-Instruct-v0.1 bleibt das beste Modell unter Apache 2.0

Confidence Intervals on model strength (Arena Elo, German)



Ausgewählte historische Meilensteine

- **1966: ELIZA** — a computer program for the study of natural language communication between man and machine (Weizenbaum, 1966)
 - Antworten auf Basis von regex/pattern-matching und simplen Transformationen (z.B. mein→dein)
 - vgl. <https://andreasfischer1985.github.io/code-snippets/html/eliza32.html>
- **1995: AIML** (Wallace, 1995)
 - AIML = „Artificial Intelligence Markup Language“
 - mehr Kontext, Variablen und komplexere Funktionen
- **2017: Transformers** (Vaswani, 2017)
 - Chatbots auf Basis von Transformer-Modellen bzw. „Large Language Models“ im Stile von ChatGPT & HuggingChat
 - vgl. <https://huggingface.co/spaces/AFischer1985/AI-RAG-Interface-to-Hub>



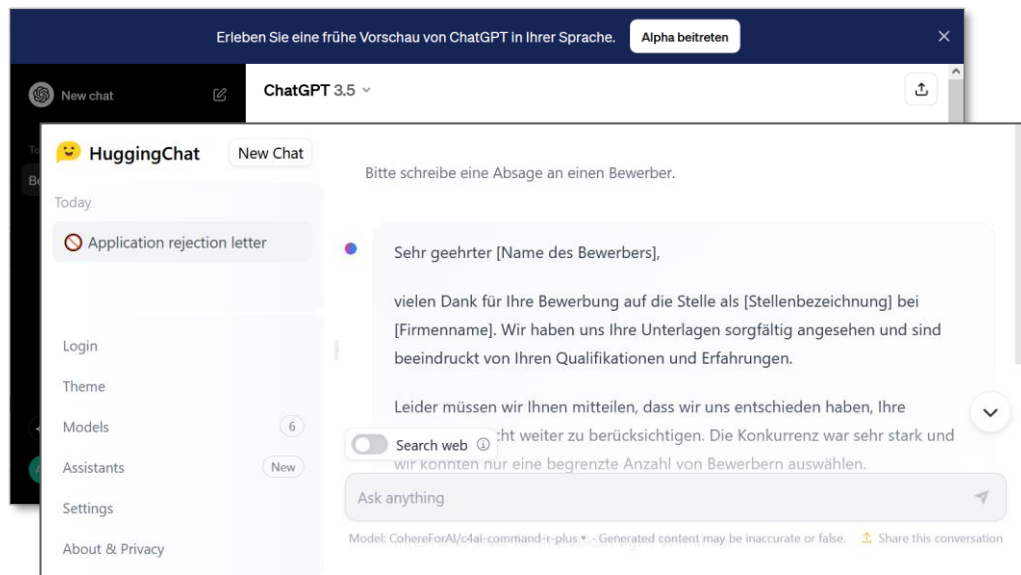
Quelle https://de.wikipedia.org/wiki/Joseph_Weizenbaum

```
<category>
  <pattern>WHAT IS YOUR NAME</pattern>
  <template><![CDATA[My name is <bot name="name"/>]]></template>
</category>
<category>
  <pattern>WHAT ARE YOU CALLED</pattern>
  <template>
    <srai>what is your name</srai>
  </template>
</category>
```

Chatbot-Paradebeispiele ChatGPT & Co.




- ChatGPT seit 30.11.2022, HuggingChat seit 25.04.2023
 - Chat-Interfaces zu großen Sprachmodellen (LLMs)
- Grundidee:
 - User stellt Anfrage („Prompt“), KI-Modell generiert plausible Antwort
 - KI-Modell generiert plausible Fortsetzung Token für Token auf Basis eines komplexen statistischen Modells (sog. Transformer Model)
- Eigene No-Code-Chatbots?
 - Custom GPTs / HuggingChat Assistants – erlauben aktuell das Zusammenklicken eigener Chatbots, sind aber
 - nur begrenzt modifizierbar,
 - nicht „on premises“ nutzbar,
 - datenschutztechnisch bedenklich,
 - in ständiger Entwicklung &
 - pot. nicht langfristig verfügbar



<https://huggingface.co/chat>

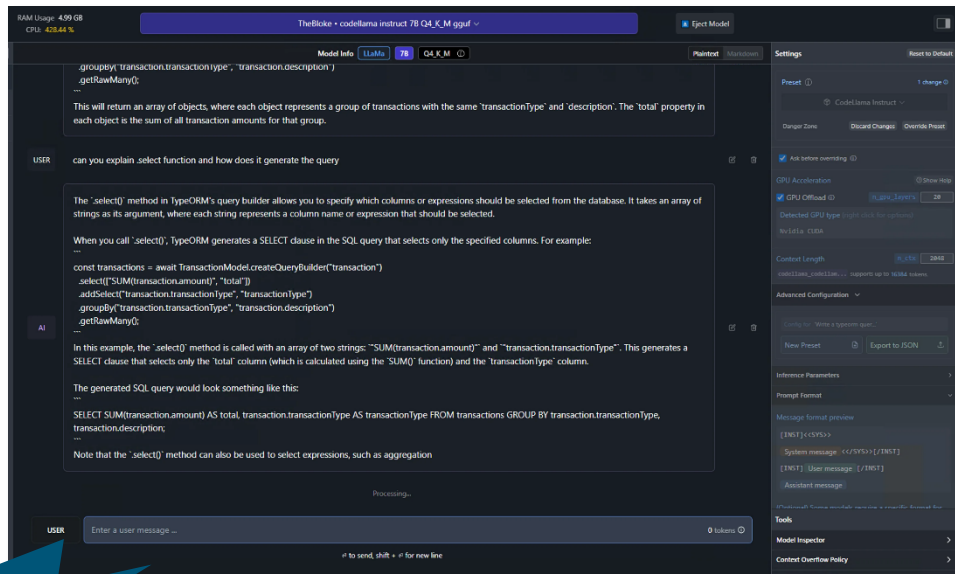
LMStudio - die digital souveräne ChatGPT-Alternative



- **LMStudio** ist das Rundum-sorglos-Paket für den digital souveränen Einsatz von LLMs
 - Bietet Interfaces für AI-Chat, (multi-model) Playground & Local Server
 - Auch für multimodale Chats (LlaVA) 
 - basiert auf llama.cpp (inkl. opt-out für „Keep entire model in RAM“)
 - Zugriff auf alle gguf-Modelle (auch die bereits vorliegenden 😊)
 - Intuitiv und informativ zugleich
 - Übersichtlich aufbereitet: Tooltips, weiterführende Links, Code-Examples, etc.
- Alternativen?
 - Ollama + Web-UI
 - GPT4all
 - Huggingface/Chat-ui
 - Uvm.

Alles auf
llama.cpp-Basis!

Quelle: <https://www.reddit.com/r/LMStudio/>



(llama.cpp server-ui, janhq/jan, SillyTavern, Msty, Chatbox, NextChat, Librechat, AnythingLLM, ggozad/oterm, dustinblackman/oatmeal, Big AGI, ellama, LobeChat, OpenLocalUI, oobabooga/text-generation-webui, Chatbot-ui, ...)

Pro-Tipp: eigene Llama-cpp-Server



- Für den lokalen Zugriff auf ein LLM lässt sich ein Llama.cpp-Server auch direkt starten (z.B. in Python als subprocess)
- Zugriff erfolgt dann z.B. via UI im Browser (localhost:2600), via Programmiersprache oder von einem anderen Rechner im Netzwerk

```
import subprocess
command = ["python3", "-m", "llama_cpp.server", "--model",
           "/home/af/gguf/models/c4ai-command-r-v01-Q4_0.gguf",
           "--host", "0.0.0.0", "--port", "2600", "--n_threads", "8",
           "--n_gpu_layers", "10"]
subprocess.Popen(command)
```

Quelle:
https://www.reddit.com/r/LocalLLaMA/comments/18534f1/i_have_given_llamacpp_server_ui_a_facelift/?rdt=64238

Beispiel: Zugriff via POST-request (basic)

```
import requests
url="http://0.0.0.0:2600/v1/completions"
prompt="<BOS_TOKEN><|START_OF_TURN_TOKEN|><|SYSTEM_TOKEN|>Du bist ein  
Chatbot.<|END_OF_TURN_TOKEN|><|START_OF_TURN_TOKEN|><|USER_TOKEN|  
>Hallo!<|END_OF_TURN_TOKEN|><|START_OF_TURN_TOKEN|><|CHATBOT_TOKEN|>"
body={"prompt":prompt, "max_tokens":None, "echo":"False",  
      "stream":"False"}
reply=requests.post(url, json=body)
reply.text
```

<https://huggingface.co/AFischer1985>

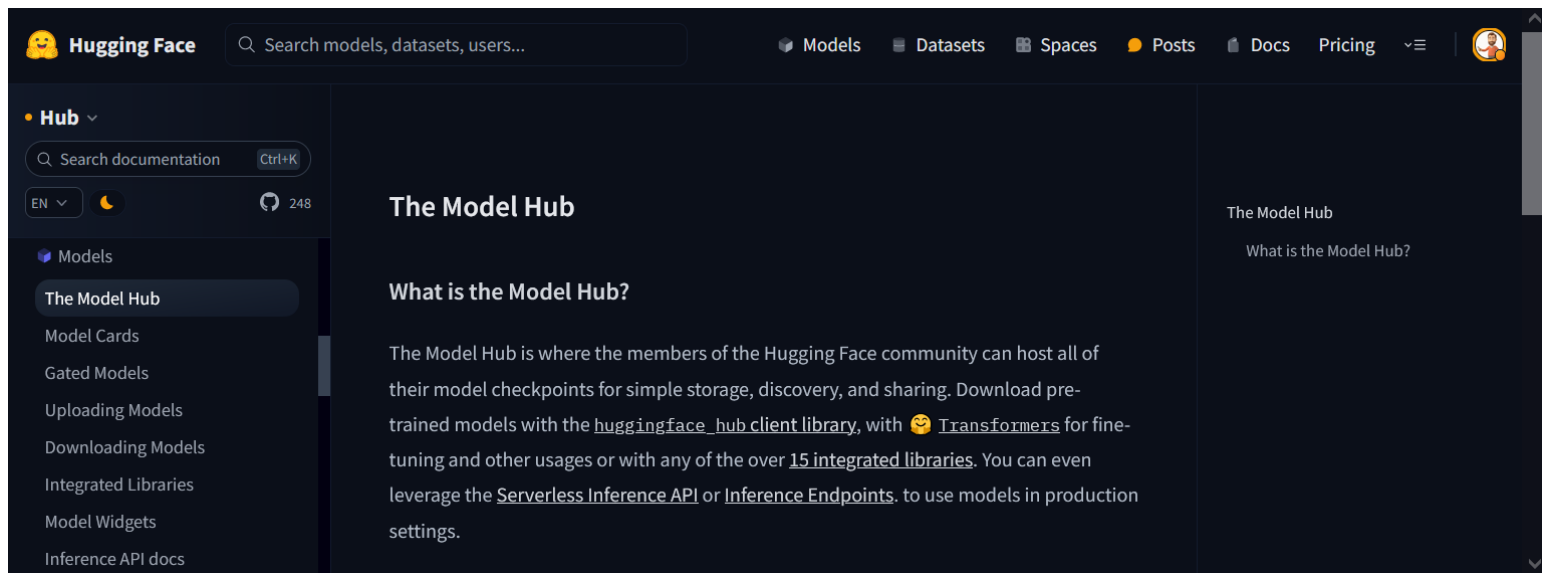
Beispiel: Zugriff via POST mit Token-Streaming



```
import requests
import json
url="http://0.0.0.0:2600/v1/completions"
prompt="<BOS_TOKEN><|START_OF_TURN_TOKEN|><|SYSTEM_TOKEN|>Du bist ein Chatbot.<|END_OF_TURN_TOKEN|><|START_OF_TURN_TOKEN|><|USER_TOKEN|>Hallo!<|END_OF_TURN_TOKEN|><|START_OF_TURN_TOKEN|><|CHATBOT_TOKEN|>"
body={"prompt":prompt, "max_tokens":None, "echo":"False", "stream":"True"}
response=""
buffer=""
for text in requests.post(url, json=body, stream=True):
    if buffer is None: buffer=""
    buffer=buffer+str(text)
    text=text.decode('utf-8')
    if((text.startswith(": ping -")==False) & (len(text.strip("\n\r"))>0)): buffer=buffer+str(text)
    buffer=buffer.split('finish_reason': null)}}')
    if(len(buffer)==1):
        buffer=buffer+str(text)
        pass
    if(len(buffer)==2):
        part=buffer[0]+'finish_reason': null}}'
        if(part.lstrip('\n\r').startswith("data: ")): part=part.lstrip('\n\r').replace("data: ", "")
        try:
            part = str(json.loads(part)["choices"][0]["text"])
            response=response+part
            buffer="" # reset buffer
        except Exception as e:
            print("Exception:"+str(e))
            pass
    yield response
```

vgl. <https://huggingface.co/spaces/AFischer1985/Schreibassistenz/raw/main/run.py>

- Huggingface bietet kostenfreien Zugriff auf diverse KI-Modelle (Quellcode/Gewichte und Inference Endpoints): <https://huggingface.co/models?sort=likes>



Quelle: <https://huggingface.co/docs/hub/models-the-hub>

Beispiel: Zugriff via InferenceClient



```
# Connect to Model on the Huggingface Hub
#-----
from huggingface_hub import InferenceClient
myModel="mistralai/Mixtral-8x7B-Instruct-v0.1"
client = InferenceClient(
    model=myModel,
    token=myToken #token="hf_..."
)

# Generate Response
#-----
prompt="<s> [INST] Hallo Welt! [/INST]"
generate_kwargs = dict(temperature=float(0.9), max_new_tokens=1000, top_p=float(0.95),
    repetition_penalty=1.0, do_sample=True, seed=42)
stream = client.text_generation(prompt, **generate_kwargs, stream=True, details=True,
    return_full_text=False)
response = ""
for text in stream: # stream response token by token
    part=text.token.text
    response += part
    print(part,end="", flush=True)
    yield response
```

Gradio ChatInterfaces to HF Hub and to Llama.cpp



Gradio Interface to Huggingface Hub

<https://huggingface.co/spaces/AFischer1985/GradioChatbotDemo/blob/main/run.py>

The interface is titled "Gradio Chatbot Demo". It features a chat window with a message input field containing the text "Wie lautet die Definition von komplexem Problemlösen?". Below the input field is a large text box displaying the response in German. The response defines complex problem solving as a multi-stage process and lists steps for problem identification and analysis. At the bottom of the chat window are buttons for "Retry", "Undo", and "Clear". Below the chat window is a "Type a message..." input field with a "Submit" button. At the bottom, there is an "Additional Inputs" section with two fields: "System Prompt" containing "Du bist ein hilfsbereiter Chatbot und antwortest bevorzugt in deutscher Sprache." and "HF_token" containing a long alphanumeric string.

Gradio Interface to local Llama.cpp-Server

<https://huggingface.co/spaces/AFischer1985/GGUF-Interface>

The interface is titled "AI-Interface". It features a chat window with a message input field containing the text "What is the definition of complex problem solving according to Fischer, Greiff & Funke (2012)?". Below the input field is a large text box displaying the response in English. The response defines complex problem solving and describes it as involving uncertainty, ambiguity, and conflicting information. At the bottom of the chat window are buttons for "Retry", "Undo", and "Clear". Below the chat window is a "Type a message..." input field with a "Submit" button.

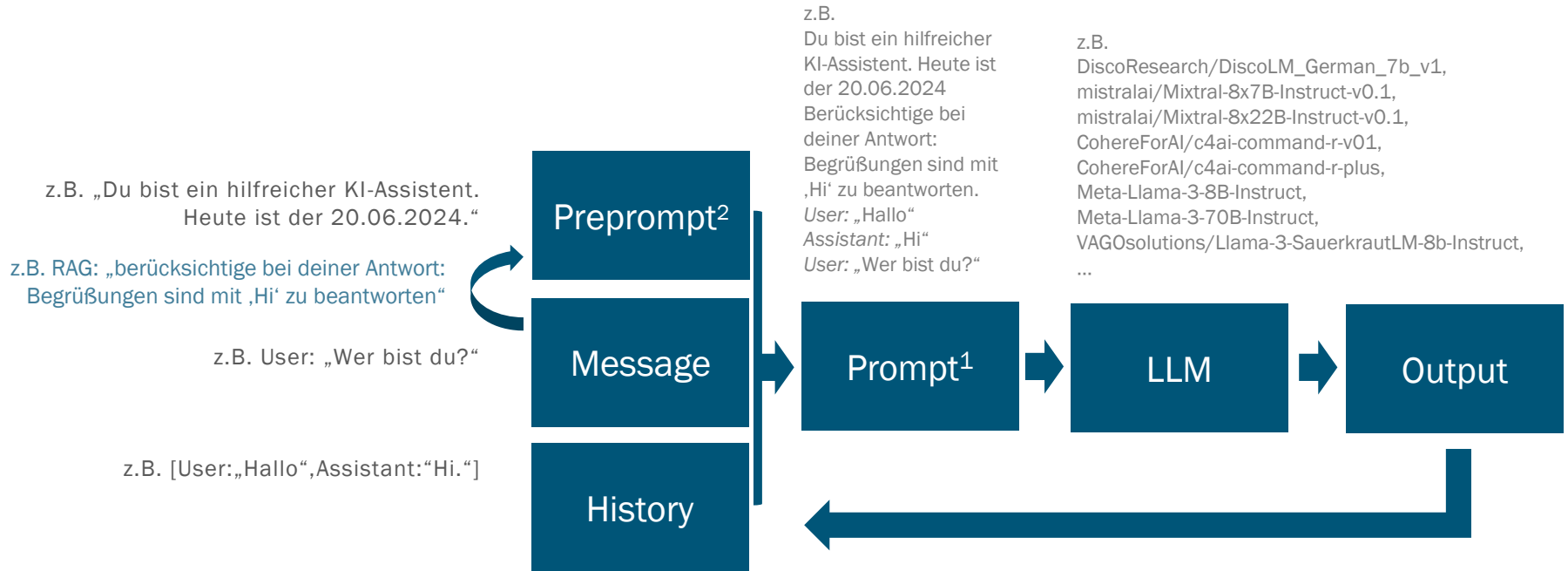
Was braucht es für dig. souveräne Chatbotentwicklung?



Man nehme...

- LLM-Server (z.B. Huggingface Hub oder Llama.cpp-server)
- Web-Server für's User-Interface (z.B. Gradio ChatInterface oder Steamlit)
- Algorithmus des Antwortverhaltens (z.B. LLM-response auf Nutzer-Anfrage)
 - Ggf. nach Aufbereitung der Nutzeranfrage (z.B. Formatierung iSv. Prompt-Template)
 - Ggf. nach Ergänzung des Dialogverlaufs (sog. Chat-History)
 - Ggf. nach Ergänzung um weitere Anweisungen (sog. System-prompt oder Preprompt)
 - Ggf. nach Ergänzung um weitere Informationen (z.B. Retrieval-Augmented Generation, RAG)

LLM-Chatbot Components



- 1) Achtung: Formatierung des Prompts je nach Prompt-Template des LLM
- 2) ggf. auch dynamisch ergänzt um weitere Kontext-Informationen
(*aktuell verfügbare Tools, zur Message passende Fakten, etc.*)

Codebeispiel

- Der Code...
 - ...definiert eine Funktion *format_prompt*, um message uvm. zu Prompts zu formatieren
 - ...definiert eine Funktion *response*, die festlegt, wie zu einer message eine Antwort formuliert wird
 - ...legt einen sog. *InferenceClient* an, mit dem ein LLM auf dem Huggingface Hub kontaktiert werden kann
 - ...startet ein *Gradio ChatInterface* mit zahlreichen Parametern

Quelle:

<https://huggingface.co/spaces/AFischer1985/GradioChatbotDemo/blob/main/run.py>

Dr. Andreas Fischer | 28.06.2024 | 16

```
#####
# Title:  Gradio Chatbot Demo
# Author: Andreas Fischer
# Date:   June 22nd, 2024
# Last update: June 22nd, 2024
#####

myToken=None

# Specify Prompt Formatting
#-----

import re
def format_prompt(message="", history=None, system=None, RAGAddon=None, system2=None,
zeichenlimit=None, historylimit=4, removeHTML=True):
    if zeichenlimit is None: zeichenlimit=1000000000 # :-)
    startOfString="<s>"
    template0=" [INST] {system} [/INST] </s>"
    template1=" [INST] {message} [/INST]"
    template2=" {response}</s>"
    prompt = ""
    if RAGAddon is not None:
        system += RAGAddon
    if system is not None:
        prompt += template0.format(system=system)
    message=message.replace("[INST]", "")
    message=message.replace("/[/INST]", "")
    message=message.replace("</s>", "")
    message=re.sub("<[|](im_start|im_end|end_of_turn)[|]>", '', message)
    if history is not None:
        for user_message, bot_response in history[-historylimit:]:
            if user_message is None: user_message = ""
            if bot_response is None: bot_response = ""
            if removeHTML==True: bot_response = re.sub("<(.*?)>","\n", bot_response)
            if user_message is not None: prompt += template1.format(message=user_message[:zeichenlimit])
            if bot_response is not None: prompt += template2.format(response=bot_response[:zeichenlimit])
    if message is not None: prompt += template1.format(message=message[:zeichenlimit])
    if system2 is not None:
        prompt += system2
    return startOfString+prompt

# Specify Chatbot Response
```