Conversational AI. Dialogsysteme, Chatbots, Assistenten

Veranstalter: Christoph Ringlstetter

Sitzung 4: Langchain II & Prompting

Was machen wir denn heute.

- Orga. Referate, Zulassung, Termine.
- News of the Week
- Referat Prompting
- Besprechung Langchain Konzepte Buch Kap 2,3,4 fertig
- Besprechung Langchain 5, Building a Chatbot
- Besprechung Prompting, Jurafsky/Martin 12 was fehlt

Basis der folgenendn Ausführungen das Langchain Buch und die Tutorials

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

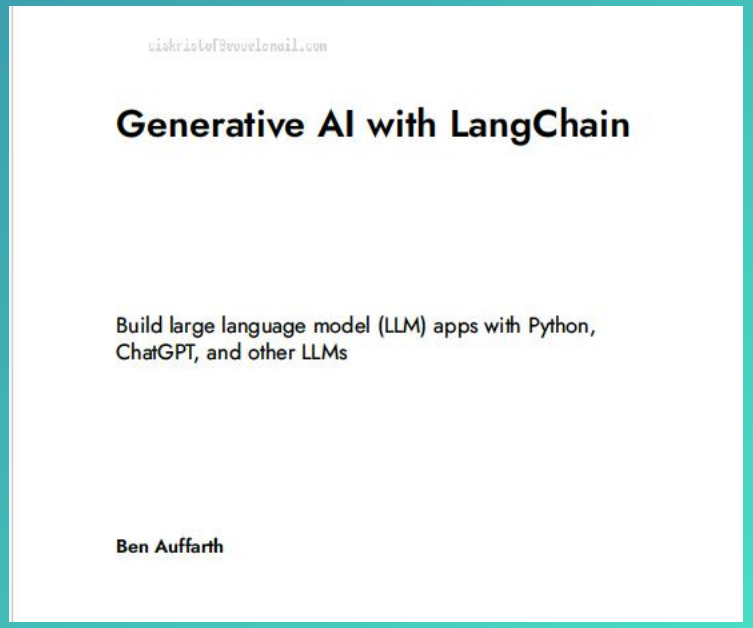
Basics

[Build a Simple LLM Application with LCEL]

[Build a Chatbot]

[Build vector stores and retrievers]

[Build an Agent]



Kap 5 im Buch

- Was ist ein Chatbot
- Retrieval mit Vektoren
- Load and Retrieve: Langchain Implementation
- Memory
- Moderation von Antworten

Domänenspezifische Q/A haben ein Wissensdefizit das mit dem RAG Ansatz: Generierung der Antwort mit externer Evidenz: korrekter und informativer.

- **Langchain – Was ist ein Chatbot**

  AI Programm das konversationelle Interaktion simuliert:

  Appointment Scheduling

  Information Retrieval

  Virtual Assistants

  Language Learning

  Mental Health Support

  Education

  HR + Recruietment

  Entertainment

  Law

  Medizin

Langchain – Was ist ein Chatbot

Start mit Informationsbots: Zugang zu Informationen

Dann Initialer Support/Assistenz

Reasoning und Analysefähigkeit für Fortgeschrittene Funktionen – “intelligentes“

LLM als Basis erforderlich

Respond vs Proaktive

Initieren von Konversation oder neuen logischen Turns: intentionale Chatbots

Action Bots die auch auf Systeme zugreifen können: Agents

Langchain – Retrieval und Vektoren

RAG vs RALMS

Semantische Suche per Vektorsuche: auf der Grundlage von Ähnlichkeit (e.g. Cos) zw Query und Documentembeddings werden „Dokumente“ gefunden.

OpenAI V_{dim} = 1536

Embeddings in Langchain: embed_query() Methode aus einer Embeddeeings Klasse e.g. OpenAIEmbeddings: gehen gegen installierte Services um die Embeddings zu erzeugen

Langchain – Retrieval und Vektoren

Code Schnipsel.

```
from langchain.embeddings.openai import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()
text = "This is a sample query."
query_result = embeddings.embed_query(text)
print(query_result)
print(len(query_result))
```

Langchain – Retrieval und Vektoren

Embeddings für mehrere „Dokumente“:

```
from langchain.embeddings.openai import OpenAIEmbeddings
words = ["cat", "dog", "computer", "animal"]
embeddings = OpenAIEmbeddings()
doc_vectors = embeddings.embed_documents(words)
```

Dokumentenbegriff später per Chunking.

Langchain – Retrieval und Vektoren

Rechnen mit Embeddings. LinAlgebra: numpy, pandas etc. meist Cos Distnaz.

Fake Embeddings: können benutzt werden um Pipelines ohne Aufruf gegen den Embedding-Algorithmus zu testen: Achtung Kosten für größere Dokumentensammlungen.

Langchain – Retrieval und Vektoren

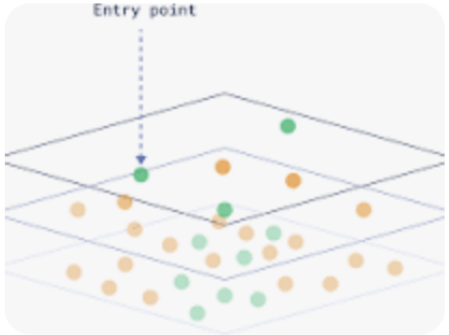
Vektor Speicherung (Storage)
Vergleich jeden Vektor im Vektorstore mit einer query um den ähnlichsten zu finden: Übliches IR Problem: quadratisch. Müssen einen Index haben.

1) Indexierung: organisiere die Vektoren um das Retrieval zu optimieren. K-D-Trees, ball-trees, Amog, Faiss Erstellen von Vektoren und Sortierung.
2) Vektor Bibliotheken: Funktionen für Vektor Operationen (dot-Produkt), Vektorindexierung. Manipulieren von Vektoren.
3) Vektor Datenbanken: z.B. Milvus, Piecore. Speichen, Managen und Retrieval von großen Vektormengen.

Langchain – Retrieval und Vektoren

Indexierung - Product Quantization.
Unterteilt den Vektorraum in kleine Subspaces quantiziert jeden Subspace separat. Dimensionalität wird reduziert. Lässt eine effiziente Speicherung und Suche zu: opfert Korrektheit. Z.B. K-D-Trees. Binäre Baumstruktur hält Daten entlang der Featurewerte. Effizient für niedrig dimensionale Daten. Ball Trees: HyperSphären: langsamer, geht für hochdim. Daten. ANNOY(oh yea) Local Sensitive Hashing.

Vector quantization is **a data compression technique used to reduce the size of high-dimensional data**. Compressing vectors reduces memory usage while maintaining nearly all of the essential information. This method allows for more efficient storage and faster search operations, particularly in large datasets. 25 Sept 2024



Langchain – Retrieval und Vektoren

Vector Libraries: Meta Faiss, Spotify Annoy. Für Suche: ANN Algorithmen.

FAISS: Facebook AI similarity search + Clustering von Dense Vektoren. CPU/GPU.

PQ, LSH Indexierer

Annoy: C++ ANN Suche. Effizient, Skalierbar

SPTAG: Microsoft ANN, K-D-Trees.

UMSLIB, HUSLIB: Amazon

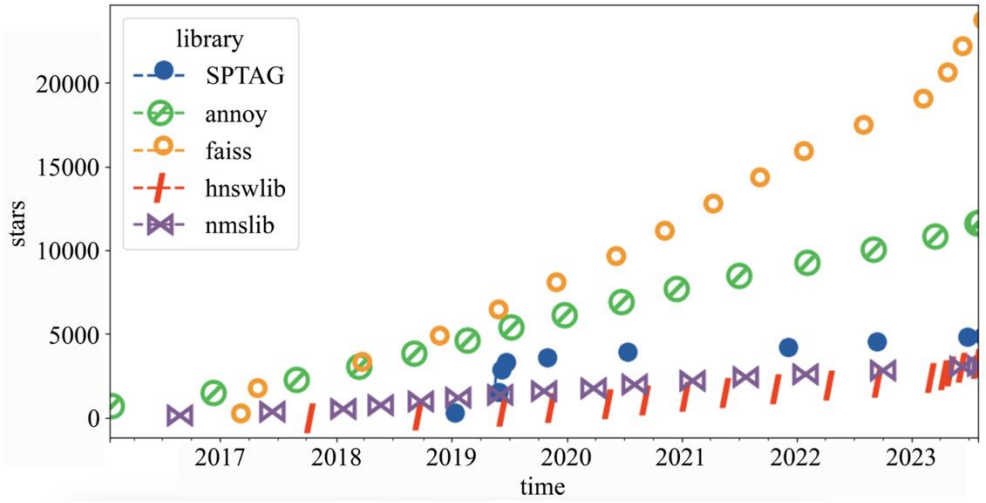


Figure 5.3: Star history for several popular open-source vector libraries

Langchain – Retrieval und Vektoren

Vektordatenbanken: Embeddings speichern, Suchen, Datenmanagement
Metadaten speichern, Filtern, Skalieren + Resilienz
Professionelle Lösungen auch für Multimodale Daten.

+ Anomalieerkennung
+ Personalisierung, Usermanagement
+ Standardimplementierungen NLP auch More Like This

z.B. Postgress im pg_embedding

Langchain – Retrieval und Vektoren

Name	Free tier	Queries Per Second	Self-Host	Managed in Cloud	SOC-2	HIPAA	Open Source	License	BM25	Aggregations	Size of vectors dimension	Metadata Filtering	Time Based Metadata Filtering	Time-Series Compression	Hybrid Search	Website URL
Qdrant	Self-hosted is free	300 by ANN (Around 350 by FastEmbed)	Yes	Yes	Can be (Depending on hosting)	Can be (Depending on hosting)	Yes	Apache License 2.0	No But Similar	No	Qdrant does not have any hard limits	Yes	Somewhat (Need to convert time to an integer)	No	Yes (Sparse-Dense Vectors	qdrant.tech
Weaviate	yes	518	Yes	Yes	Can be (Depending on hosting)	Can be (Depending on hosting)	Yes	Apache License 2.0	Yes	Yes	65535					
Pinecone	Yes	From Pinecone website (queries per second for 1M vectors of size 768; top_10): - s1 pod: 10 - p1 pod: 30 - p2 pod: 150	No	Yes	Yes	Yes	No	Commercial	Yes	No	20000	Yes	Somewhat (Need to convert date/time to integer in Unix time)	No	Yes (Sparse-Dense Vectors)	pinecone.io
Milvus	Yes	1,751	Yes	Yes	Yes	? (Depending on Hosting?)	Yes	Apache License 2.0	No	No	32768	Yes	Somewhat (Need to convert date/time to integer in Unix time)	No	No, they use the phrase "Hybrid Search", but it really means metadata filtering	milvus.io
ChromaDB	In memory of server	?	Yes	Not Yet	? (Depending on Hosting?)	? (Depending on Hosting?)	Yes	Apache License 2.0	No	No		Yes	Somewhat (Need to convert time to an integer)	No	query	chroma.com

Which Vector Database Should You Use? Choosing the Best One for Your Needs: Pablan Nayak

https://medium.com/the-ai-forum/which-vector-database-should-you-use-choosing-the-best-one-for-your-needs-5108ec7ba133

Langchain – Retrieval und Vektoren

Setup Chroma in Langchain: Storing and querying vectors using Chroma as a backend.

```
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
vectorstore = Chroma.from_documents(documents=docs,
embedding=OpenAIEmbeddings())
```

Müssen *pymupdf library* installiert haben

Langchain – Retrieval und Vektoren

```
from langchain.document_loaders import ArxivLoader
from langchain.text_splitter import
CharacterTextSplitter
loader = ArxivLoader(query="2310.06825")
documents = loader.load()
text_splitter = CharacterTextSplitter(chunk_size=1000,
chunk_overlap=0)
docs = text_splitter.split_documents(documents)
```

similar_vectors = vector_store.query(query_vector, k) für Query ev. noch ein embedding herrichten → notebook ausprobieren

Langchain – Retrieval und Vektoren

Loading and Retrieving in Langchain.

Implementiert ein Retrievalsystem über eine Toolchain. zB. Chatbot mit RAG.

dataloader, datatransformer, embedding model, vectorstore, retriever.

The relationship between them is illustrated in the diagram here (source: LangChain documentation):

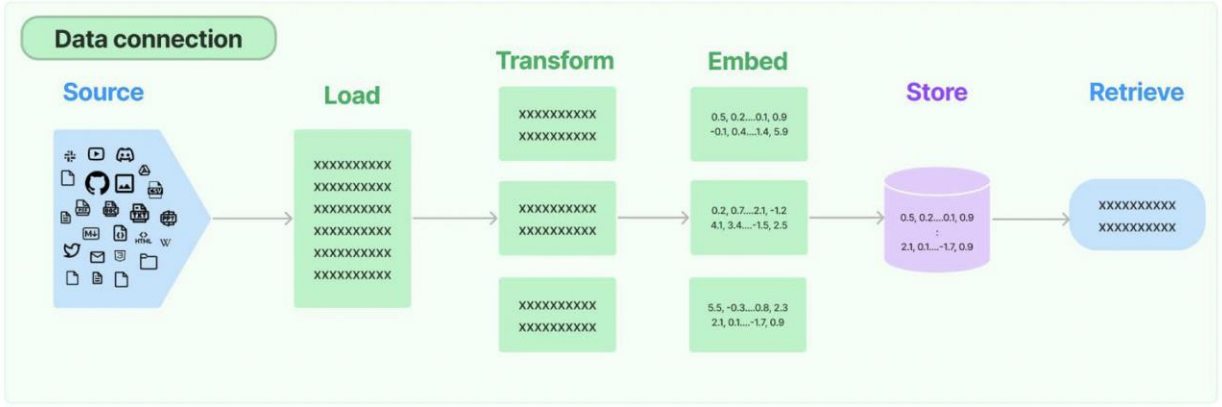


Figure 5.5: Vector stores and data loaders

Langchain – Retrieval und Vektoren

Document Loaders: laden Daten von einer Source als Document Object: text + Metadaten: TextLoader, ArxivLoader, YouTubeLoader + Integrationen für HTML, Images, PDF... Jeweils mit der load() Methode

```
loader = TextLoader(file_path= „pathtofile.txt“)
document = loader.load()
```

Wikipedia:

```
from langchain.document_loaders import WikipediaLoader
loader = WikipediaLoader("LangChain")
```

documents = loader.load() → Ausprobieren, funktioniert das so?

Langchain – Retrieval und Vektoren

Retrievers:

Suchen und Retrieval von Information aus einem Index → z.B. gespeichert in einem Chroma Vectorstore

BM25 Retriever --> Benutzt BM25 Altorithmus für das Ranking: tf + d_{length}

TFIDF → Docs symbolisch?

Dense Retrievers meistens COS

KNN Retriever

Langchain – Retrieval und Vektoren

KNN Beispiel. Symbolisch?

```
from langchain.retrievers import KNNRetriever
from langchain.embeddings import OpenAIEmbeddings
words = ["cat", "dog", "computer", "animal"]
retriever = KNNRetriever.from_texts(words, OpenAIEmbeddings())
result = retriever.get_relevant_documents("dog")
print(result)
```

Langchain – Retrieval und Vektoren

Custom Retriever

```
from langchain.schema import Document, BaseRetriever
class MyRetriever(BaseRetriever):
def get_relevant_documents(self, query: str, **kwargs) ->
list[Document]:
# Implement your retrieval logic here
# Retrieve and process documents based on the query
# Return a list of relevant documents
relevant_documents = []
# Your retrieval logic goes here…
return relevant_documents
```

Langchain – Retrieval und Vektoren

Building a chatbot now:

Documentloader

Store in VectorBase

Chatbot with Retrieval from Store.

=> made available via web interface streamlit

drop your document + ask questions

Langchain – Retrieval und Vektoren

Document Loader:

Für verschiedene Formate: importiere mehrere Loader von langchain.document_loaders import PyPDDLoader, TextLoader etc

über Extensions in einer Klasse Documentloader werden die loader Typen aufgerufen: dictionary mit Extensions. “.pdf“ : PyPDFLoader

Langchain – Retrieval und Vektoren

Vector Storage: setup Embedding Mechanismus, Vector Storage, Dokumenten Pipeline

1) Splitt der Dokumente in Chunks
2) Embedding Modell
3) InMemory Database: splits, embedings: speichern

Initialisieren der InMem Datenbank z.B. DocArray mit Abstandsmaß z.B. COS

Langchain – Retrieval und Vektoren

Retrieval: 2 Hauptoptionen

1) Similarity Search
2) Maximum Marginal Relevance (MMR)
diversity based reranking during retrieval: um verschiedene Perspektiven zu bekommen.
Verbesserung des Retrievals: contextual compression. Context der Query mit dem LLM wird benutzt um irrelevante Information auszuschließen.
Beispiel aus dem Buch Kap5 durchführen:
https://github.com/benman1/generative_ai_with_langchain/tree/softupdate/chapter5

Langchain – Memory

ermöglicht Chatbots informationen von früheren Interaktionen zu erhalten.
=> Kontinuität, konversationeller Kontext
andernfalls unverbunden, wenig zufriedenstellend
Memory ermöglicht Korrektheit auf der Seite des Systems durch Kontextverständnis während des gesamten Dialogs.
Holistische Perspektive.
Personalisierung. Faithful – sich selbst treu?

Langchain – Memory

Conversation Buffers.

```
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationChain
# Creating a conversation chain with memory
memory = ConversationBufferMemory()
llm = ChatOpenAI(
model_name="gpt-3.5-turbo", temperature=0, streaming=True
)
chain = ConversationChain(llm=llm, memory=memory)
```

Langchain – Memory

Conversation Buffers. → hier Frage stellen. Verändert sich der Dialog ohne M.

```
user_input = "Hi, how are you?"
# Processing the user input in the conversation chain
response = chain.predict(input=user_input)
print(response)
user_input = "What's the weather like today?"
response = chain.predict(input=user_input)
print(response)
# Printing the conversation history stored in memory
print(memory.chat_memory.messages)
```

Langchain – Memory
Conversation Buffers

Simplifiziert mit ConversationChain

```
conversation = ConversationChain(
llm=llm,
verbose=True,
memory=ConversationBufferMemory()
)
```

mit der save_context() Methode können dann in und outputs persistiert werden

Buffer Window:

```
from langchain.memory import ConversationBufferWindowMemory
memory = ConversationBufferWindowMemory(k=1)
```

Langchain – Memory

Customize:

```
template = """The following is a friendly conversation between a
Current conversation:
{history}
Human: {input}
AI Assistant:"""
PROMPT = PromptTemplate(input_variables=["history", "input"],
template=template)
conversation = ConversationChain( prompt=PROMPT, llm=llm, verbose=True,
memory=ConversationBufferMemory(ai_prefix="AI Assistant"),
)
```

Langchain – Memory

Remembering Conversation Summaries: ConversationSummaryMemory

Generieren ein Summary der Konversation während sie fortschreitet.

Instanz erzeugen, llm als Argument übergeben, save_context() Metohde

load_memory_variables um die kondensierte Konversationshistorie zu laden

```
from langchain.memory import ConversationSummaryMemory
from langchain.llms import OpenAI
# Initialize the summary memory and the language model
memory = ConversationSummaryMemory(llm=OpenAI(temperature=0))
# Save the context of an interaction
memory.save_context({"input": "hi"}, {"output": "whats up"})
# Load the summarized memory
memory.load_memory_variables({})
```

Langchain – Memory

Storing Memory in Knowledge Graphs

Aus Konversationen Fakten/Entitäten/Beziehungen extrahieren und in einen Graph speichern: Entities, Attributes, Relationships

memory = ConversationKGMemory (llm = llm)

Beispiel in der Langchain Doku.

https://python.langchain.com/v0.1/docs/modules/memory/types/kg/

Langchain – Memory

Combining several Memory Mechanisms

Combined_Memory Class instantieren einse LLM und mehrer Memory Typen

Beispiel im Buch.

Langchain Doku. Aber deprecated. Check!

https://python.langchain.com/v0.1/docs/modules/memory/multiple_memory/

Interessant damit herumzuspielen. Warmstart der Konversation…

Langchain – Memory
Longterm Persistance

z.B. ZEP Open Source Memory: id-gelabeltes Memory wird per Cloudservice gespeichert.
Recall, Understand, Extract Data from Chat Histories.
Personalisierte AI Erfahrung ermöglichen
Daten werden in einer Datenbank persistiert.

https://blog.getzep.com/announcing-zep-community-edition/

https://python.langchain.com/docs/integrations/vectorstores/zep/

Langchain – Moderation, Guardrails

appropriate, ethical, respectful

- Filtern von beleidigendem Content, abusiv
- Constitution um ethisch korrektes Verhalten abzusichern.
- Benutzer vor unpassenden Inhalten schützen
- Schädliches, beleidigendes Verhalten des Benutzers verhindern
- Brand Reputation: Bot antwortet CI gemäß, positiv, Topice
- Two Strikes Rule: nach zwei Versuchen zu beleidigen Abbruch
- Legal Compliance: verschieden je nach Land.

https://github.com/aws-samples/amazon-bedrock-workshop/blob/main/06_OpenSource_examples/03_NVIDIA_NeMo_Guardrails/01_NVIDIAs_NeMo_OpenSource_Guardrails_with_Amazon_Bedrock_LLM_Development.ipynb

Langchain – Moderation, Guardrails

Moderation Instanz der Chain hinzufügen oder ein Runable erzeugen.

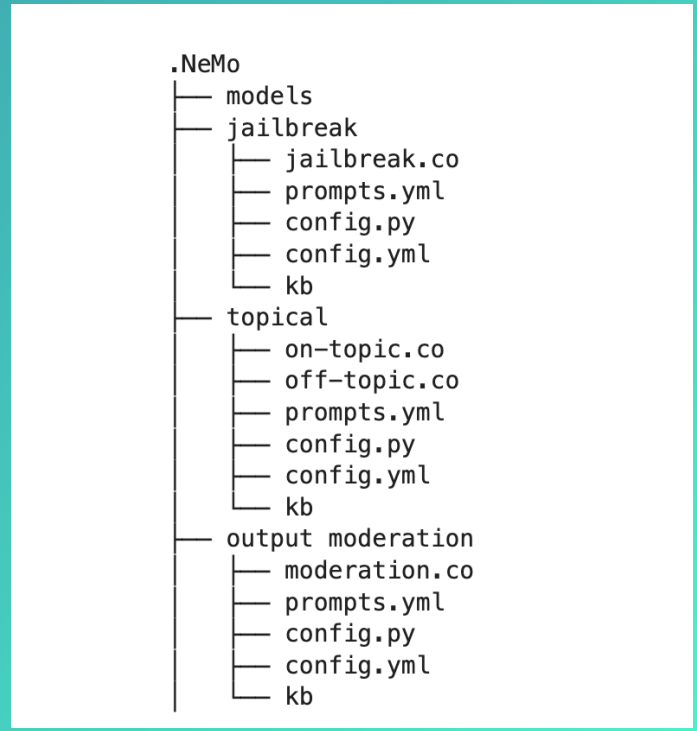
Handling: wirf einen Fehler und behandle ihn.

```
from langchain.chains import OpenAIModerationChain
moderation_chain = OpenAIModerationChain()
from langchain.prompts import PromptTemplate
from langchain.chat_models import ChatOpenAI
from langchain.schema import StrOutputParser
cot_prompt = PromptTemplate.from_template(
"{question} \nLet's think step by step!"
)
llm_chain = cot_prompt | ChatOpenAI() | StrOutputParser()
chain = llm_chain | moderation_chain
response = chain.invoke({"question": "What is the future of programming?"})
```

Langchain – Moderation, Guardrails

Programmierbare Constraints um den output des Modells zu kontrollieren

- Topics: „keine Politik“
- Vordefiniertes Dialogverhalten: flow, konsistente Antworten
- Sprachstil: Ton, Formalität
- Strukturierte Datenextraktion: Information, Aktionen



Jurafsky/Martin Kap 12: Model Alignment, Prompting, In-Context Learning

LLMs sollen Aufgaben durchführen die über natürliche Sprache vermittelt werden:

Menschen sagen was gemacht werden soll.

Übersetze, Outline für einen Talk, Draft für ein Email

Instruktion des LLM: Prompt

Antwort des Modells: Token W‘keit gg den Prompt $P(w_i|w_{<i})$

Kann auch Demonstrationen enthalten: Beispiele

Kann generative Aufgaben durchführen aber auch diskriminative (Klassifikation)

Jurafsky/Martin – Prompting

Prompting auch als In-Context Learning bezeichnet: ohne die Modellparameter zu ändern.

Funktioniert nur zufriedenstellend wenn das Modell ein Instruction Tuning durchlaufen hat.

Ausschluss von schädlichem Verhalten: Sicherheitstraining in die Instructionphase einbringen: RLHF oder DPO == Model Alignment.

Direct Preference Optimization: Your Language Model is Secretly a Reward Model

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, Chelsea Finn

Jurafsky/Martin – Prompting

Prompting Def.: String vom User an das Modell geschickt mit der Aufforderung etwas Nützliches zu machen.

Prompt Engineering: finde Task effektive Prompts.LLMs können aufgrund ihrer generellen Natur viele verschiedene Aufgaben getriggert durch einen String und ggf Beispiele durchführen

Summarization {input}: tl.dr;

Translation {input}: translate to French:

Sentiment {input}: over all it was:

Aspects {input}: What aspects were positive, what were negative:

Templates werden auf inputs angewandt und es entstehen Filled Prompts

Jurafsky/Martin – Prompting

Template

1) gegebene Aufgabe -> taskspezifisches Template: Freier Parameter für den Inputtext

2) Input + template → Prompt wird aufgefüllt es entsteht ein Filled Prompt der an das LLM geshickt wird

3) Token Outputs durch autoregressives Decoding

4) Finales Resultat für den User: Output des Models kann direkt verwendet werden -> natürlich generative Aufgaben oder muss aus dem Output extrahiert werden – diskriminative Aufgaben z.B. Klassifikation.

Jurafsky/Martin – Prompting

Beispiel für absichernden Promptstil (constrained)

Human: Do you think {input} has a negative or positive sentiment?

Choices:

(P) Positive

(N) Negative

Assistant: I believe the best anser is: ()

Jurafsky/Martin – Prompting

Lernen durch Demostration Few Shot Prompting

Einige gelabelte Beispiele: Godstandard Frage und Antwortpaare.

Da der Haupteffekt schon vom ersten Beispiel kommt reichen wenige.

Kontrast hierzu: Finetuning von speziellen Classifier Heads. Je mehr Daten desto besser.

Geht vor allem um die Form – Modell lehrt Vorgehensweise.

Vlg Min et al 2022 Incorrect Examples.

Min, S., X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? EMNLP.

Jurafsky/Martin – Prompting

Chain of Thought Prompting

Verbessern der Performanz von LLMs für schwierige Aufgaben – Reasoning erforderlich.

=> Intuition. Menschen lösen Aufgaben die Schließen, Nachdenken erfordern über schrittweises Vorgehen.

=> über den Prompt sicher stellen dass LLMs diese Vorgehensweise nachahmen

=> Demonstrationen im Pormpt werden mit Text angereichert der die Reasoning Steps darstellt => der schrittweises Output/Input Prozess verursacht die korrekte Antwort in den Beispielen.

== Evidenz: LLMs geben die korrekte Antwort mit höherer Wahrscheinlichkeit.

Wei, J., X.Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. 2022. Chain-of-thought promptingelicits reasoning in large language models. NeurIPS, volume 35.

Jurafsky/Martin – Prompting

Beispiel aus Wei et al. Mathematisches Schließen

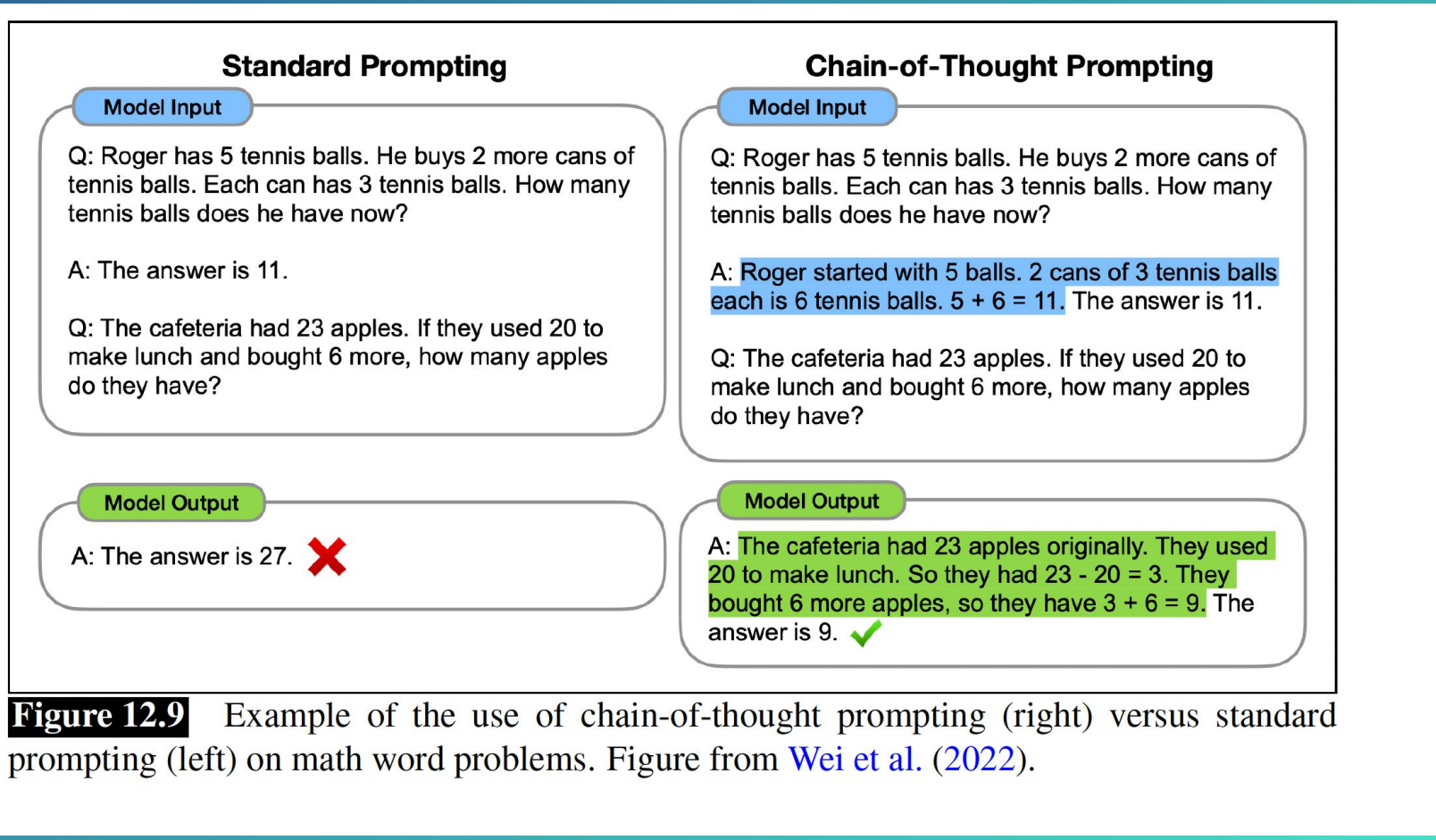


Figure 12.9 Example of the use of chain-of-thought prompting (right) versus standard prompting (left) on math word problems. Figure from Wei et al. (2022).

Jurafsky/Martin – Prompting

Beispiel aus Suzgun et al. Zu zeitlichen Abfolgen

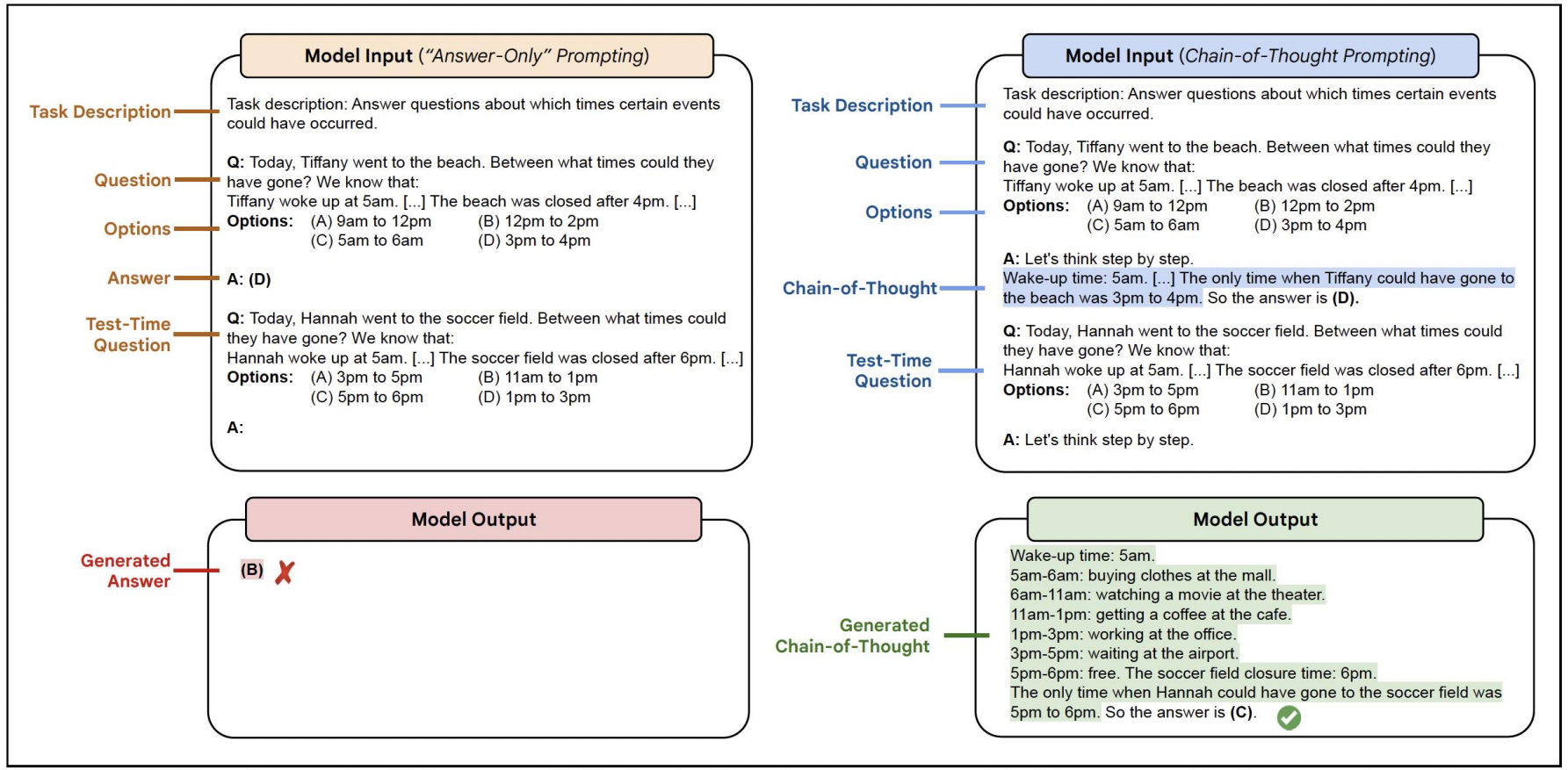


Figure 12.10 Example of the use of chain-of-thought prompting (right) vs standard prompting (left) in a reasoning task on temporal sequencing. Figure from Suzgun et al. (2023).

Suzgun, M., N. Scales, N. Sch¨arli, S. Gehrmann, Y. Tay, H.W. Chung, A. Chowdhery, Q. Le, E. Chi, D. Zhou, and J. Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. ACL Findings.

Jurafsky/Martin – Prompting – Automatische Optimierung

Automatische Prompt Optimierung

geg. Prompt und Task: Prompt Optimierung sucht nach Prompts mit verbesserter Performanz (ähnlich iterative improvement search: Russel/Norvig) im Raum möglicher Prompts. Optimiert nach Performanz der Aufgabe.

1) Start Prompt

2) Scoring Metrik

3) Expansion: Variation des Prompts

Einordnung der Prompts/Suchverfahren → Fixierte Werte für Iteration oder Stop

Jurafsky/Martin – Prompting – Automatische Optimierung

Scoring: Performanz von potentiellen Prompts bewerten um eine Richtung der Expansion zu bestimmen. Kosten sind kritisch da alle Prompts durchgespielt werden müssen.

Ggf: Zugang zu gelabelten Trainingsdaten möglich. Score basierend auf Task Execution Akkuratheit.

Task spezifische Scores:

- Klassifikation: 0/1 loss
- Generative: BERT Score, Bleu, Rouge → Kleine Samples wegen der Kosten.

Jurafsky/Martin – Prompting – Automatische Prompt Optimierung

Prompt Expansion über das LLM:

Generate a variation of the following instruction while keeping the semantic meaning.
Input: {Instruction}
Output: {Complete}

Variante: Current Prompt abschneiden (Präfix) und dann per LLM vervollständigen lassen.

Jurafsky/Martin – Prompting – Automatische Prompt Optimierung

Explizit bessere Prompts erzeugen.

Jetziger Kandidat: Aus Sample von Trainingsbeispielen – soll den Expansionsprozess anleiten.

Idee: Inkorrekt/Verbesserungswürdig: „den Prompt kritisieren“

Automatic Prompt Optimization with “Gradient Descent” and Beam Search

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, Michael Zeng
Microsoft Azure AI
{reidpryzant,iterdan,jerrl,yintatlee,chezhu,nzeng}@microsoft.com

Jurafsky/Martin – Prompting – Automatische Prompt Optimierung

Gradientenmethode

1) Prompt auf ein Sample von Trainigsbeispielen anwenden
2) Identifizieren von Fails
3) LLM soll Kritik/Verbesserung in Hinblick auf die negativen Beispiele generieren
4) Kritik und Prompt an LLM → generiere einen verbesserten Prompt

Langchain – Moderation, Guardrails – Automatische Promptoptimierung

Critiquing Prompt

```
I'm trying to write a zero-shot classifier prompt.
My current prompt is:  {prompt}
But this prompt gets the following examples wrong:
{error_string}
Give {num_feedbacks} reasons why the prompt could have
gotten these examples wrong.
```

Prompt Improvement Prompt

```
I'm trying to write a zero-shot classifier.  My current prompt is:
{prompt}
But it gets the following examples wrong:  {error_str}

Based on these examples the problem with this prompt is that {gradient}.
Based on the above information, I wrote {steps_per_gradient} different
improved prompts.  Each prompt is wrapped with <START> and <END>.

The {steps_per_gradient} new prompts are:
```

Automatic Prompt Optimization with "Gradient Descent" and Beam Search Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, Michael Zeng: Microsoft Azure AI

Langchain – Retrieval und Vektoren – Evaluation von LLMs/Prompts

Acc auf NLP Tasks

Toxikalität, Fairness

Multiple Choice sets: MMLU Testset mit 15908 Fragen auf 57 Gebieten

Qualitativ: Papers zu GPT4 und GPT1o wie besprochen.