ChatGPT

Prompt Engineering – Umfassende Recherche

1. Arten von Prompts

Zero-Shot Prompting: Bei Zero-Shot Prompts erhält das Sprachmodell **keinerlei Beispiele** für die gewünschte Aufgabe. Man formuliert nur die Instruktion oder Frage, und das Modell muss allein aufgrund seines angehäuften Wissens antworten [1] [2]. *Anwendungsfälle:* Zero-Shot eignet sich für **einfache oder allgemein bekannte Aufgaben**, z.B. Übersetzungen, Zusammenfassungen oder Sentiment-Analysen ohne besondere Kontexte [3]. *Vorteile:* Diese Methode ist **einfach und flexibel** – die Prompts sind kurz und erfordern keine Vorbereitung von Beispielen [4]. Da keine Beispieldaten nötig sind, ist Zero-Shot nützlich, wenn kaum Ressourcen oder wenig Zeit vorhanden sind [5]. *Nachteile:* Die **Leistung kann variieren**. Ohne spezifische Hinweise kann das Modell bei komplexen oder sehr **fachspezifischen Aufgaben** deutlich ungenau sein [6]. Zudem hängt alles vom bereits gelernten Wissen des Modells ab – wenn dem Modell zu einem Thema während des Trainings zu wenig begegnet ist, wird es in Zero-Shot mit dieser Aufgabe eher Schwierigkeiten haben [6]. In manchen Szenarien kann eine geschicktere Formulierung den Mangel an Beispielen ausgleichen – Studien zeigen, dass mit **gut strukturierten Anweisungen** Zero-Shot manchmal ähnlich gut oder sogar besser abschneiden kann als Few-Shot [7] (wenn das Modell die Aufgabe prinzipiell beherrscht). Insgesamt gilt Zero-Shot als Basistechnik, die man oft zuerst ausprobiert, bevor man aufwändigere Prompting-Strategien anwendet [8].

One-Shot und Few-Shot Prompting: *Few-Shot Prompts* geben dem Modell **einige Beispiele** (Eingabe-Output-Paare) vor, bevor die eigentliche neue Anfrage gestellt wird [9]. One-Shot ist ein Spezialfall mit genau **einem** Beispiel. Durch diese Demonstrationen lernt das Modell im Kontext ("In-Context Learning"), was für eine Art von Antwort erwartet wird [10] [9]. *Anwendungsfälle:* Few-Shot wird verwendet, wenn die Aufgabe **komplexer** ist oder ein bestimmtes Format erfordert – z.B. **Klassifikationen mit spezifischen Labels**, komplexe **Transformationsaufgaben** (wie ein bestimmtes Antwortformat) oder Aufgaben, bei denen das Modell ohne Beispiele zu unpräzise wäre [9]. Mit Beispielen kann man auch **Stilvorgaben** machen (z.B. eine sachliche vs. humorvolle Antwort) oder das Modell auf seltene Aufgaben trimmen. *Vorteile:* Few-Shot Prompts liefern dem Modell eine **Mini-Lerneinheit** im Prompt: Das Modell sieht direkt, welche **Muster** es befolgen soll [11] [9]. Dies kann die Leistung deutlich **verbessern**, gerade bei Aufgaben, für die das Modell in Zero-Shot unsicher wäre. Man kann so die Ausgabe gezielter steuern – z.B. durch Vorlage eines **Beispiels im gewünschten Format** erhöht sich die Wahrscheinlichkeit, dass die Antwort diesem Format entspricht [9]. *Nachteile:* Few-Shot Beispiele verbrauchen **Token-Platz** im Prompt (gerade bei längeren Kontexten ein limitierender Faktor) und müssen erst ausgewählt oder erstellt werden. **Unpassende oder irrelevante Beispiele** können die Ausgabe sogar verschlechtern, indem sie das Modell auf die falsche Fährte führen. Zudem besteht die Gefahr des **Overfitting an die Beispiele**: das Modell passt seine Antworten zu stark an die gegebenen Beispiele an und generalisiert dann schlechter außerhalb dieses Schemas [12]. Man muss also Beispiele mit Bedacht wählen (möglichst repräsentativ für die Bandbreite der Anfragen). In der Praxis bringt Few-Shot oft einen spürbaren Performance-Boost gegenüber Zero-Shot, *insbesondere* bei Aufgaben mit **variablen Eingaben oder kniffligen Anforderungen** [13] [14]. Allerdings zeigen neuere Untersuchungen, dass sehr große Modelle mit durchdachter reiner Anleitung (Zero-Shot) in manchen Fällen schon sehr gut abschneiden können [7] – hier lohnt es sich, beide Ansätze zu vergleichen und je nach Anwendungsfall abzuwägen.

Chain-of-Thought Prompting: Diese Technik – oft abgekürzt als **CoT** – bedeutet, das Modell zu **Zwischenschritten im Denken** anzuleiten. Anstatt direkt die Endantwort zu verlangen, fordern Chain-of-Thought-Prompts das Modell auf, den Lösungsweg aufzuschreiben oder in Etappen vorzugehen [15]. Dies kann explizit geschehen (z.B. "Denke Schritt für Schritt und erkläre deine Herleitung, bevor du die endgültige Lösung gibst") oder implizit durch Few-Shot-Beispiele, die Lösungswege enthalten [16] [17]. *Anwendungsfälle:* CoT ist besonders nützlich für **komplexe Aufgaben, die logisches Denken, mehrstufige Berechnungen oder Schlussfolgern erfordern** [18]. Beispiele sind Mathematikaufgaben, Logikrätsel, Programmieraufgaben oder allgemein Fragen, bei denen mehrere Fakten verknüpft werden müssen. Durch die Kette von Gedanken kann das Modell seine **"Gedanken" ordnen**, was oft zu **korrekteren Ergebnissen** führt, da Fehler in einem Zwischenschritt eher auffallen und korrigiert werden können [19]. *Vorteile:* Chain-of-Thought fördert **transparente und nachvollziehbare Antworten**. Das Modell legt seine Gedankengänge offen, was dem Nutzer erlaubt, die Plausibilität nachzuvollziehen und eventuelle Irrtümer im reasoning zu erkennen [19]. CoT kann die Leistung bei schwierigen Problemen drastisch **steigern**, weil das Modell sich auf Teilprobleme fokussiert und systematisch zum Resultat kommt [20] [19]. Tatsächlich wurde beobachtet, dass große Modelle mit CoT-Prompts Aufgaben lösen können, an denen sie in "direkten" Prompts scheitern würden (z.B. komplexe Rechenaufgaben) [21] [15]. *Nachteile:* Die Ketten können das Modell zu **weitschweifigen Ausgaben** verleiten. Bei einfachen Fragen ist CoT überdimensioniert – das Modell könnte unnötig ausschweifend antworten. Außerdem können **Fehler im Gedankengang** passieren: wenn das Modell im ersten Schritt falsch abbiegt, führt auch eine ausführliche Erklärung zu einem falschen Endergebnis. CoT braucht meist **mehr Tokens** (längere Antworten) und ist daher kostspieliger in Umgebung mit begrenztem Kontext. Kleinere Modelle profitieren weniger von CoT; bei unzureichender Kapazität verwirrt eine explizite Denkweise ggf. mehr, als dass sie nützt [22]. Insgesamt ist Chain-of-Thought ein mächtiges Werkzeug für tiefergehende **Reasoning-Aufgaben** und wird häufig mit Few-Shot kombiniert (d.h. man gibt ein Beispiel mit Lösungsweg vor, dann eine neue Frage) [17]. Es hilft dem Modell "wie ein Mensch" Zwischenschritte zu formulieren, was oft zu **logischeren und konsistenteren Lösungen** führt [19].

Rollen-Prompting (Persona Prompts): Hierbei weist man dem Modell eine **spezifische Rolle oder Identität** zu, um Stil und Fokus der Antwort zu steuern [23]. Typische Formulierungen sind etwa *"Du bist ein erfahrener Lehrer und sollst folgendes erklären: …"* oder *"Handle als Marketing-Experte und erstelle …"*. Das Modell wird also angewiesen, aus der Perspektive dieser Rolle zu antworten [24] [25]. *Anwendungsfälle:* Rollen-Prompts sind nützlich, um **den Ton oder Duktus der Antwort** anzupassen (z.B. laienverständlich vs. fachlich, förmlich vs. locker) [23]. Bei kreativen Aufgaben kann eine Persona helfen, **Stil- und Kontexttreue** zu gewährleisten – etwa wenn das Modell im Stil eines bestimmten Autors schreiben oder einen Charakter verkörpern soll. Auch für **Beratungs- oder Dialogszenarien** nutzt man Rollen (z.B. der Bot soll sich wie ein Psychologe oder ein Kundendienst-Mitarbeiter verhalten). Zudem gibt es Hinweise, dass Rollenprompts bei bestimmten **Reasoning-Aufgaben** die Leistung steigern können, wenn die gewählte Persona fachlich passend ist [26] [27]. *Vorteile:* Ein gut gewählter Persona-Prompt kann die **Klarheit und Relevanz** der Antwort erhöhen, indem das Modell seinen Output an der vorgegebenen Rolle ausrichtet [23]. Zum Beispiel liefert *"Du bist ein History-Professor…"* in der Regel fundiertere historische Erklärungen, da das Modell Teile seines Wissens hervorholt, die zu einem Professor passen. Für **Tonfall und Stil** sind Rollen-Prompts sehr wirkungsvoll – man kann die gleiche Info einmal humorvoll als "Comedian" oder sachlich als "Analyst" ausgeben lassen. Damit sind Antworten **zielgruppengerechter** anpassbar (z.B. kindgerecht erklären vs. für Experten) [23]. *Nachteile:* Die Wirksamkeit von Rollenprompts ist nicht in jedem Fall garantiert. **Studien ergeben ein gemischtes Bild**: In einigen Fällen verbessern Personas die Zero-Shot-Genauigkeit überhaupt nicht oder verschlechtern sie sogar [28]. Insbesondere bei **faktenbasierten Aufgaben** kann das Einfügen einer Rolle (z.B. "Du bist ein Botaniker…") manchmal keinerlei Genauigkeitsgewinn bringen, in manchen Tests führte es gar zu *mehr* Fehlern [28]. Ein weiteres Risiko sind **Stereotype und Verzerrungen**: Die Modellantwort könnte Klischees der vorgegebenen Rolle reproduzieren [29]. Eine Aufforderung "Sprich wie ein Millennial" könnte z.B. in unnatürliche Umgangssprache abdriften. Schließlich hängt der Erfolg

stark davon ab, **wie die Rolle im Trainingsmaterial des Modells repräsentiert** ist [29]. Ist die Persona exotisch oder selten, weiß das Modell evtl. wenig damit anzufangen. In der Praxis sind Rollen-Prompts dennoch beliebt, um Antworten *kontextuell einzurahmen* und den **gewünschten Ton** zu treffen [30]. Best Practices empfehlen, Personas sparsam und **präzise** einzusetzen (also eher berufliche oder sachliche Rollen statt intime/private Rollen, um dem Modell klare Leitplanken zu geben) [31]. Zusammenfassend erhöhen Persona-Prompts die **Anpassungsfähigkeit** des Outputs – sie sind aber kein Allheilmittel für bessere Qualität und sollten mit Bedacht gewählt werden, je nach Aufgabe und Zielgruppe.

Weitere Prompt-Typen: Neben den oben genannten Hauptkategorien gibt es diverse spezielle Prompting-Techniken. Zum Beispiel kann man **Multi-Turn-Prompting** nutzen – also über mehrere Runden hinwegarbeiten (siehe *Prompt Chaining* unten) – um einen Inhalt iterativ zu erarbeiten. Auch **"System Prompts"** (siehe Abschnitt 4) sind eine besondere Form: hierbei handelt es sich um Anweisungen, die kontextuell festlegen, wie das Modell *grundsätzlich* reagieren soll (z.B. durch Vorgaben im System-Message-Feld einer Chat-API). Solche Systemrollen dienen dazu, **globale Richtlinien** oder Charakterzüge festzulegen, während die eigentliche Nutzeranfrage im User-Prompt folgt [32] [33]. Darüber hinaus existieren technik-spezifische Varianten wie **Visual Prompts** (bei Bildgeneratoren), **Programmierprompts** (die Code als Input/Output haben) usw. Im Kontext von Text-LLMs sind jedoch Zero-Shot, Few-Shot (inkl. One-Shot), Chain-of-Thought und Rollen-Prompts die geläufigsten Kategorien, die oft auch **kombiniert** eingesetzt werden.

2. Merkmale eines guten Prompts

Einen guten Prompt zu schreiben, ist *entscheidend*, um nützliche und präzise Antworten von einem KI-Modell zu erhalten. Einige **Merkmale** zeichnen gelungene Prompts aus: **Klarheit**, **Kontext**, **Zielorientierung** und **Strukturiertheit**. Im Folgenden die wichtigsten Punkte, wie man einen guten Prompt verfasst:

- **Klar und spezifisch formulieren:** Ein guter Prompt lässt **keinen Raum für Mehrdeutigkeiten**. Vage oder allgemeine Anweisungen führen oft zu generischen oder ungenauen Antworten [34] [35]. Statt z.B. *"Schreibe etwas über Klimawandel"* sollte man präzisieren: *"Schreibe eine 500-Wörter-Erklärung, wie sich der Klimawandel auf marine Ökosysteme auswirkt, insbesondere auf Korallenriffe. Nenne 3 konkrete Auswirkungen und mögliche Lösungen, und richte dich an ein Publikum von Oberstufenschülern."* – dieses konkrete Briefing gibt Länge, Fokus, Anzahl der Beispiele, Zielgruppe etc. vor [35]. Je detaillierter die Anforderungen (Thema, Umfang, Tonfall, gewünschte Details), desto eher trifft die KI ins Schwarze. Wichtig ist aber, **Relevantes hervorzuheben und Überflüssiges wegzulassen**, um den Prompt nicht unnötig zu verwässern.

- **Ausreichend Kontext bereitstellen:** Das Modell kennt zwar viel, kann aber nicht erraten, *welchen* Kontext der Nutzer meint, wenn dieser nicht im Prompt steht. Daher sollte man relevante **Hintergrundinformationen im Prompt** mitliefern [36] [37]. Beispielsweise bei *"Ist dieser Code korrekt?"* weiß das Modell nicht, welchen Code man meint – besser wäre, den Codeblock einzufügen und genau zu sagen, worauf geachtet werden soll [38] [37]. Ebenso bei Wissensfragen: Falls man eine bestimmte Quelle zusammengefasst haben möchte, sollte man sie (oder ihre Kerndaten) mitgeben. Der Prompt sollte die Situation umreißen, damit das Modell seine Antwort darauf abstimmen kann. **Besonders bei spezifischen Problemen** (z.B. firmenspezifische Fragen, besondere Abkürzungen) ist Kontext der Schlüssel, um eine relevante Antwort zu erhalten [39] [37]. Faustregel: Alles, was ein *menschlicher* Experte wissen müsste, um die Frage zu beantworten, sollte entweder offensichtlich sein oder im Prompt erwähnt werden.

- **Klares Ziel und Format definieren:** Man sollte dem Modell mitteilen, **was genau am Ende erwartet wird** – z.B. *"Erstelle eine Zusammenfassung in 3 bullet points"* oder *"Gib das Ergebnis als JSON-Struktur aus"*. Diese **Output-Spezifikation** hilft ungemein, um formgerechte Resultate zu bekommen [40] [41]. Auch Tonalität oder Stil können zum Ziel gehören: *"im Tonfall eines formellen Berichts"* oder *"locker und humorvoll geschrieben"*. Durch solch klare Endziele vermeidet man, dass das Modell raten muss, was der Nutzer möchte [42] [43]. Gute Prompts enthalten oft eine **Gliederung**: z.B. erst Instruktion („Was tun?"), dann eventuell Constraints („Wie/Wie lang?/In welcher Form?") und ggf. Kontext oder Zusatzinfos. Diese Struktur kann sogar explizit gemacht werden (siehe Formatierung unten). **Beispiel:** *"Fasse den folgenden Text in einem Absatz zusammen. Wichtig: Erwähne mindestens zwei Datenpunkte aus dem Text und schreibe in einem sachlichen, neutralen Stil."* – Hier sind Aufgabenstellung, Länge und Stil klar vorgeschrieben. Solche expliziten Zielformulierungen sorgen für konsistente, vergleichbare Ausgaben.

- **Einschränkungen und Kriterien mitgeben:** Falls es bestimmte **Do's and Don'ts** gibt, sollte man diese nennen. Z.B. *"Vermeide Fachjargon"*, *"Nutze keine Aufzählungszeichen"*, *"Antwort maximal 100 Wörter"*, *"Falls Unsicherheit besteht, gib keine erfundenen Infos"* usw. Modelle halten sich in der Regel an solche Vorgaben, wenn sie klar formuliert sind. Ebenso kann man Quality Criteria einbauen, wie *"Antwort soll faktisch korrekt und knapp sein"*. Durch solche **Einschränkungen** grenzt man den Raum der möglichen Antworten ein, was besonders bei heiklen Themen (zur Vermeidung von Halluzinationen) oder formatkritischen Outputs (wie Code, JSON) wichtig ist. Eine Einschränkung könnte auch sein, eine bestimmte Perspektive einzunehmen (*"aus Sicht eines Anfängers erklären..."*). Insgesamt gilt: besser zu viel Guidance als zu wenig – solange die Einschränkungen nicht widersprüchlich sind.

- **Geeigneter Stil und Rollenansprache:** Wie im Abschnitt zu Rollen-Prompts beschrieben, lohnt es sich zu überlegen, ob die KI eine gewisse **Rolle annehmen** oder für ein **bestimmtes Publikum** schreiben soll. Diese Info kann man in den Prompt aufnehmen, um den Stil zu steuern. Beispiel: *"Du bist ein geduldiger Lehrer. Erkläre das Konzept so, dass ein 10-jähriges Kind es versteht."* Dadurch wird Klarheit im **Adressatenbezug** geschaffen: das Modell wählt Worte und Satzbau entsprechend einfacher. Solche Persona- oder Audience-Vorgaben helfen, den Tonfall konsistent zu halten [23]. Wichtig ist aber, dass die Rolle zum **Ziel passt** – eine unpassende Persona bringt Verwirrung (z.B. *"Du bist Shakespeare, schreibe einen technischen Bericht"* wäre kontraproduktiv). Im Zweifel kann man Rollen auch weglassen, wenn sie keinen erkennbaren Mehrwert für die Aufgabe bieten.

- **Beispiele (Exemplars) nutzen, wenn sinnvoll:** Wie oben bei Few-Shot beschrieben, können **Beispielfragen mit Lösungen** dem Modell einen Rahmen geben [10]. In vielen Fällen verbessern Beispiele die Ergebnisse deutlich [44], vor allem wenn es um **Stilfragen oder komplexe Formate** geht. Beispiel: *"Formatiere die folgenden Daten als Tabelle. Beispiel:\nEingabe: ... -> Ausgabe: (Tabelle) \nNun formatiere:\n..."*. Durch ein Beispiel in der gewünschten Tabellen-Notation versteht das Modell besser, was zu tun ist. Aber Vorsicht: **Beispiele sollten wirklich relevant** sein. Keine Beispiele sind besser als verwirrende Beispiele. Auch sollte man nicht so viele Beispieldaten geben, dass der eigentliche Prompt untergeht. Ein bis drei gute Demonstrationen reichen meist, um dem Modell die Richtung zu weisen [9]. Wenn man keine konkreten Beispiele hat, aber trotzdem den Prompt nicht komplett "kalt" lassen will, kann man stattdessen **Platzhalter** oder Pseudobeispiele nutzen, um zumindest das Format zu illustrieren.

Typische Fehler vermeiden: Genauso wichtig wie die positiven Merkmale sind Fehler, die man **nicht** begehen sollte [45]. Hier eine kurze Liste häufiger Prompting-Pfallen und wie man sie umgeht:

- *Zu vage:* Undeutliche Prompts wie *"Schreib was über X"* liefern nur oberflächliche, generische Antworten [34]. **Fix:** Spezifizieren, *welchen* Aspekt von X, in welcher Tiefe, für welches Ziel etc. behandelt werden soll [35].

- *Zu viele Aufgaben auf einmal:* Wenn man das Modell in **einem einzigen Prompt mit Bitten überhäuft**, wird die Antwort chaotisch oder lückenhaft [46] [47]. **Fix:** Lieber *einen Fokus pro Prompt*. Bei komplexen Anforderungen diese **in Teilprompts aufteilen** [48]. Beispiel: Statt *"Erkläre Quantencomputing, vergleiche es mit klassischem Computing, liste alle Algorithmen, erkläre Quantum Supremacy und gib Code-Beispiele"* in einem Rutsch – besser mehrere geordnete Anfragen daraus machen [49] [50].

- *Kontext ignoriert:* Wenn man ohne nötigen Kontext fragt, muss das Modell raten und liegt oft daneben [39]. **Fix:** Immer fragen: *"Was sollte die KI wissen, um das korrekt zu beantworten?"* und diese Infos im Prompt mitgeben [37]. Beispiele: Quelltext beim Code-Check, spezifische Daten bei Datenfragen, Vorgeschichte bei Dialogfortsetzungen etc.

- *Keine Formatvorgaben:* Lässt man das **Output-Format völlig offen**, bekommt man evtl. Fließtext, wo eine Liste gefragt war, oder umgekehrt. **Fix:** Das gewünschte Format deutlich nennen (Liste, Tabelle, Aufzählung, Aufsatz etc.) und ggf. Beispielausgabe skizzieren.

- *Widersprüchliche oder offene Anforderungen:* Prompts wie *"Antworte kurz, aber ausführlich …"* verwirren das Modell. Auch Mehrdeutigkeiten (z.B. *"Gib einen Bericht – oder falls du willst, einige Bulletpoints"*) sind kontraproduktiv. **Fix:** Konsistente, eindeutige Instruktionen geben. Wenn mehrere Alternativen möglich sind, entscheide dich im Prompt für *eine*.

- *Modell über seine Grenzen hinausschicken:* Z.B. nach **aktuellen Ereignissen fragen, die nach dem Trainingscutoff passiert sind**, oder um legal/medizinischen Rat bitten (führt oft zu Ablehnungen aufgrund der KI-Richtlinien). **Fix:** Prompt ggf. umformulieren, dass das Modell *hypothetisch* oder allgemein antwortet, oder anerkennen, dass es nur Wissen bis Datum X hat. Ebenfalls sollte man keine Aufgaben stellen, die **bekannt gegen die Nutzungsrichtlinien** verstoßen – das erzwingt nur Defensivantworten.

- *Keine Iteration:* Viele Anwender geben einen Prompt ein und sind enttäuscht vom ersten Output, **ohne Anpassungen zu versuchen**. Dabei kann eine leichte Modifikation (ein zusätzlicher Satz, ein klärendes Wort) den Unterschied machen [51]. **Fix:** Prompt-Entwicklung ist ein **iterativer Prozess** – man sollte Ergebnisse prüfen und den Prompt verfeinern (siehe Abschnitt 3 über iterative Verfeinerung). Auch kann man das Modell *selbst* um Verbesserung bitten, z.B. "Gib mir 3 verschiedene Versionen dieser Antwort" oder "Gehe ins Detail über Punkt 2".

Sollte ein Prompt in bestimmtem Format verfasst sein?

Diese Frage zielt darauf ab, ob **Formatierung (z.B. Markdown, Listen, XML)** des Prompts einen Einfluss auf die Qualität der KI-Antwort hat. Tatsächlich zeigen Erfahrungen und Studien, dass eine **klare Struktur im Prompt sehr hilfreich** sein kann [52]. Es gibt jedoch *kein universelles Idealformat*, das immer die besten Ergebnisse liefert – es hängt vom Modell und der Aufgabe ab [52]. Dennoch empfiehlt es sich, Prompts wie **kleine Dokumente** aufzubauen statt als unstrukturierten Fließtext [53]. Markdown

ist dabei ein praktisches Hilfsmittel, um Abschnitte, Überschriften oder Listen übersichtlich zu gestalten [54] [52]. Ein gegliedertes Prompt-Beispiel könnte so aussehen:

```
# Rolle/Perspektive:
Du bist ein erfahrener Datenanalyst.

# Aufgabe:
Analysiere den folgenden Text und extrahiere die drei wichtigsten
Erkenntnisse.

## Vorgaben:
- Antworte in **Stichpunkten**.
- Nutze einen sachlichen, nüchternen Ton.
- Jeder Stichpunkt max. 15 Wörter.

## Text:
"[Hier steht der zu analysierende Text]"
```

Durch solche Markdown-Strukturen mit **Überschriften und Aufzählungen** kann das Modell klar die verschiedenen Teile des Prompts erkennen (Rolle, Aufgabe, Regeln, Input) [55] [56]. Viele LLMs scheinen auf diese visuelle Gliederung positiv zu reagieren und liefern entsprechend organisierte Antworten [40] [41]. Interessant ist, dass verschiedene Formate unterschiedlich wirken: In einer Studie schnitt für GPT-4 ein sauber strukturierter Markdown-Prompt bei einer Reasoning-Aufgabe besser ab (81,2% Genauigkeit) als derselbe Inhalt im JSON-Format (73,9%). Bei GPT-3.5 war es umgekehrt – dort war JSON geringfügig besser als Markdown [52]. Das zeigt, dass man **experimentieren** darf: Markdown ist ein guter Startpunkt, da es auch für uns Menschen gut lesbar ist [57]. Manche Modelle reagieren evtl. auf andere Strukturen besser (bei streng formatierten Outputs kann z.B. ein XML-artiger Prompt hilfreich sein). Generell gilt: **Übersichtlichkeit schlägt Chaos**. Ein reiner "Wall of Text"-Prompt, in dem Aufgabe, Kontext und Formatwünsche ungeordnet in einem Absatz stehen, ist schwerer interpretierbar [58]. Besser ist es, diese Elemente **sichtbar zu trennen**, sei es durch Absätze, Spiegelstriche oder eben Überschriften [40] [41]. Wichtig: Diese Formatierung dient vor allem der Klarheit – das Modell selbst "sieht" den Prompt als reinen Text. Aber die Muster (z.B. `## Überschrift`) sind auch dem Modell aus Trainingsdaten bekannt (es hat z.B. viele Markdown-Dokumente gesehen) und es versteht daher meistens, dass dies eine Gliederung andeutet. Unterm Strich: Man *muss* keinen speziellen Zeichensatz nutzen, um eine gute Antwort zu bekommen, aber es lohnt sich, den Prompt **wie einen strukturierten Auftrag** zu schreiben. Klare Sektionen (z.B. *Kontext*, *Anweisung*, *Formatvorgabe*) machen es dem Modell leichter, den Input zu interpretieren und die **Erwartungen des Nutzers richtig umzusetzen** [41].

3. Best Practices & Frameworks

Bei der Konstruktion von Prompts haben sich verschiedene **Frameworks, Methoden und Heuristiken** etabliert, die als Leitfaden dienen. Diese Ansätze fassen die oben beschriebenen Prinzipien in Merkschemata zusammen, um nichts Wichtiges zu vergessen. Im Folgenden einige bekannte Frameworks sowie Methoden und anschließend praktische Beispiele erfolgreicher Prompts aus verschiedenen Anwendungsgebieten.

Frameworks für Prompt-Struktur:

- **CLEAR-Framework:** Dieses Akronym (engl. für *Concise, Logical, Explicit, Action-oriented, Relevant*) taucht in einigen Guides auf [59]. Es betont, dass ein Prompt **prägnant** (keine unnötigen Ausschweifungen), **logisch gegliedert**, **explizit** in der Angabe des gewünschten Outputs, **aktionsorientiert** (klarer Auftrag) und **relevant** (kontextbezogen) sein soll. Ähnliche Prinzipien findet man in akademischen Leitfäden, die vorschlagen, einen Prompt auf diese Kriterien hin zu überprüfen [59].

- **RISEN-Framework:** Vorgestellt von Prompt-Ingenieur Kyle Balmer, steht RISEN für **Role, Instruction, Steps, End Goal, Narrowing** [60]. Diese fünf Elemente bauen den Prompt systematisch auf: Zuerst definiert man die **Rolle** (Role) der KI oder Perspektive. Als nächstes kommt eine klare **Anweisung** (Instruction), was getan werden soll [42]. Danach werden **Schritte** oder Unteraufgaben (Steps) angegeben – quasi eine kleine Checkliste, wie die Aufgabe zu bearbeiten ist [61]. Darauf folgt das **Endziel** (End Goal), d.h. eine genaue Beschreibung der erwarteten Ausgabe oder des Zwecks [43]. Zuletzt erfolgt eine **Eingrenzung** (Narrowing): Hier legt man zusätzliche Constraints fest, z.B. Länge, Stil, Format, bestimmte Inhalte einzuschließen oder auszuschließen [62]. RISEN führt also viele der oben genannten Best Practices in einer Erinnerung zusammen. Beispielhaft würde ein RISEN-Prompt etwa so aufgebaut: *"Du bist ROLE (z.B. professioneller Texter). INSTRUCTION: Verfasse einen Blog-Artikel über Yoga. STEPS: 1. Beginne mit einem fesselnden Hook, 2. drei Hauptabschnitte mit Beispielen, 3. Abschluss mit Call-to-Action. END GOAL: Artikel für fortgeschrittene Yoga-Praktizierende, der Mehrwert bietet. NARROWING: ca. 800 Wörter, lockerer Ton, Zielgruppe: Intermediate Yoga-Fans."* – So ähnlich wurde es im Beispiel von Balmer umgesetzt und führte zu deutlich spezifischeren und ansprechenderen Resultaten als das einfache *"Schreib einen Blog über Yoga"* [63] [43]. RISEN ist eine **Gedächtnisstütze**, um wichtige Prompt-Elemente nicht zu vergessen, und kann gerade Einsteigern helfen, systematisch an das Prompting heranzugehen.

- **C.R.E.A.T.E.-Framework:** Entwickelt von Dave Birss (AI-Autor) und propagiert z.B. in LinkedIn-Kursen [64] [65]. C.R.E.A.T.E. steht für **Character, Request, Examples, Additions, Type of output, Extras**. Die **Character**-Phase entspricht dem Rollenprompt: man beschreibt die KI (z.B. "Du bist eine erfahrene Schriftstellerin ohne Jargon") [66]. **Request** ist die eigentliche Aufgabe: "Ich möchte, dass du …" – klar und spezifisch [67]. **Examples**: Man liefert Beispiele oder Referenzen, falls vorhanden, um Erwartungshaltung zu zeigen [68]. **Additions**: Hier verfeinert man den Task durch eine bestimmte Perspektive oder Stilrichtung (z.B. "Betone dabei die Vorteile für Umwelt und nutze inspirierende Sprache") [68]. **Type of Output** legt fest, was am Ende rauskommen soll (Format oder Umfang: "100-Wörter-Zusammenfassung", "tabellarische Liste", etc.) [69]. **Extras** schließlich sind sonstige Infos oder Materialien, z.B. ein bereitgestellter Text, den es zu verarbeiten gilt [70]. In Summe ähnelt CREATE anderen Frameworks, hebt aber die Komponente **Beispiele & Zusatzinfos** explizit hervor. In der Praxis ergibt sich ein ausführlicher Prompt, der wie ein **Aufgabenbriefing** wirkt – die Erfahrung zeigt, dass solche vollständigen Prompts sehr zielgerichtete Ausgaben liefern [71].

- **Strukturierter Ansatz (nach Lance Cummings):** Ein einfaches 4-Schritt-Schema, das im Prinzip "START" entsprechen könnte: **1. Starte mit Rolle und grobem Ziel**, **2. Gib Kontext/ Hintergrund**, **3. Definiere die Aufgabe explizit (inkl. Erwartungen)**, **4. Stelle Referenzmaterial bereit (falls nötig)** [72]. Dieses Vorgehen ist quasi eine Zusammenfassung der meisten oben genannten Frameworks ohne fancy Akronym. Es wurde z.B. von Cummings in *"The Anatomy of a Prompt"* beschrieben und kann als Universalrezept dienen, um eine KI-Aufgabe zu formulieren [72]. Es deckt sich auch mit den Empfehlungen von OpenAI & Co: **Rolle -> Kontext -> Instruktion -> (optional: Beispiel/Content)** [72] [73].

- **Weitere Methoden:** Es gibt noch diverse andere heuristische Ansätze. Z.B. der **"rhetorische Ansatz"** von Sébastien Bauer, der vorschlägt, den Prompt in rhetorische Elemente zu gliedern: erst Kernthese oder Hauptpunkt, dann die rhetorische Situation (Audience, Kontext, Rolle des Autors, emotionaler Ton, logische Punkte, Struktur, Stilvorgaben) [74] [75]. Das ist besonders für persuasive oder argumentierende Texte hilfreich, um die KI gezielt nach einem bestimmten *Appeal* schreiben zu lassen. Auch gibt es Anleihen aus dem Projektmanagement: Einige sehen im Prompting einen agilen Prozess und formulieren etwa **"User Story"**-artige Prompts, wie *"Als [Rolle] möchte ich [Aktion], um [Ziel] zu erreichen."*. Solche Formate können helfen, dem Modell den **Use-Case** klarzumachen.

- **Iterative Verfeinerung:** Ein methodischer Ansatz, der eher Prozess als Struktur ist: Man beginnt mit einem einfachen Prompt (**Start simple**), analysiert das Ergebnis und **verfeinert in Schritten** weiter. Dieses *Prompt-Refining* kann man auch teilweise die KI erledigen lassen, indem man Folgeprompts wie "Verbessere obige Antwort hinsichtlich XYZ" stellt. Einige generative Agenten verfolgen genau diese Strategie (s.u. bei Auto-Prompting). Für den Anwender bedeutet iterative Verfeinerung: **nicht beim ersten Entwurf stehenbleiben**, sondern den Prompt systematisch anpassen, bis die Ausgabe passt [76]. Dabei kann man Frameworks wie oben als Checkliste nehmen: Habe ich eine Rolle angegeben? Ist der Kontext klar? etc. – und diese Elemente in der nächsten Iteration ergänzen. Dieser Ansatz spiegelt auch die Realität wider: Oft braucht es mehrere Versuche, bis ein Prompt wirklich optimale Ergebnisse liefert [76].

Praxisbeispiele erfolgreicher Prompts: Zum Abschluss dieses Abschnitts nun **einige Beispiele** für gut gestaltete Prompts in unterschiedlichen Anwendungsgebieten. Sie verdeutlichen Best Practices im konkreten Kontext:

- **Recherche:** *"Du bist ein wissenschaftlicher Rechercheassistent. Lies die Zusammenfassung einer Studie (unten stehend) und verfasse in 3-4 Stichpunkten die wichtigsten Ergebnisse für einen Fachartikel. Achte auf einen sachlich-neutralen Ton und nenne nach Möglichkeit Zahlen oder Statistiken aus der Studie. \n\nStudienzusammenfassung: …"* – **Erläuterung:** Hier wird eine Rolle (wissenschaftlicher Assistent) definiert, der Kontext (eine gegebene Studienzusammenfassung) geliefert und das Ziel (Stichpunkte der Ergebnisse für Fachpublikum) sehr klar umrissen, inkl. Formatvorgabe. So ein Prompt würde z.B. im Recherche-/Journalismus-Bereich genutzt, um schnell Kernaussagen zu extrahieren.

- **Kreativität (kreatives Schreiben):** *"Übernimm die Rolle eines Fantasy-Romanautors. Schreibe eine kurze Geschichte (etwa 200 Wörter) über einen jungen Drachen, der fliegen lernt. Stil: bildhaft, märchenhaft und mit humorvollen Elementen. Wichtig: Gib der Geschichte eine überraschende Wendung am Ende."* – **Erläuterung:** Der Prompt setzt Persona (Fantasy-Autor), Genre und klare Vorgaben zum Inhalt und Stil. Damit wird die KI sehr wahrscheinlich eine fantasievolle, stilistisch passende Story generieren. Die spezifischen Stichworte ("bildhaft, märchenhaft, humorvoll") dienen der kreativen **Leitplanke**, um den gewünschten Ton zu treffen.

- **Problemlösung (Logik/Analyse):** *"Du bist ein Detektiv, der ein Rätsel löst. Hier ist das Problem:* Ein Mann liegt tot in einem Feld, neben ihm ein ungeöffneter Rucksack. Was ist passiert? *Denke Schritt für Schritt laut nach und präsentiere deine Schlussfolgerung am Ende."* – **Erläuterung:** Dieses Prompt-Beispiel fordert explizit zu *Chain-of-Thought* auf ("Schritt für Schritt nachdenken"). Die Detektiv-Rolle soll das deduktive Denken fördern. Durch die Aufforderung zum lauten Nachdenken wird die KI ihre Überlegungen ausformulieren und vermutlich am Ende die Lösung des Rätsels präsentieren. Solche Prompts eignen sich für Rätsel, mathematische Beweise oder komplexe Problemanalysen, wo der Weg zum Ergebnis wichtig ist.

- **Programmierung:** *"Du bist ein erfahrener Python-Entwickler und Code Reviewer. Analysiere den folgenden Python-Code und erkläre, warum er einen Fehler produziert. Gebe anschließend eine korrigierte Version des Codes. \n\n*`python\n# Code\ndef factorial(n):\n    if n == 1:\n        return 1\n    else:\n        return n * factorial(n-1)\nprint(factorial(0))\n`*\n"* – **Erläuterung:** Hier haben wir einen klaren technischen Prompt. Die Rolle als Entwickler/Reviewer stellt sicher, dass die Antwort fachlich und prägnant ausfällt. Der Kontext (der Code) ist im Prompt enthalten, eingerahmt als Codeblock. Das Ziel (Fehlerursache erklären und Code fixen) ist eindeutig formuliert. Die KI wird höchstwahrscheinlich erkennen, dass der Basisfall 0 nicht abgedeckt ist, das Problem erläutern und einen Codevorschlag machen. Dieses Muster kann man für viele Programmier-Aufgaben nutzen: Immer Code + klare Fragestellung + gewünschtes Ergebnis (Analyse, Optimierung, Dokumentation etc.).

- **Data Science/Analyse:** *"Agiere als Datenanalyst. Ich beschreibe dir einen Datensatz und eine Frage dazu; du sollst einen Analyseplan erstellen. Datensatz: Ein Monat an Webshop-Verkaufsdaten mit Spalten (Datum, Produkt, Kategorie, Preis, Verkaufte_Menge, Nutzer_ID, Land). Fragestellung: Warum sind die Verkäufe in Kategorie X im letzten Monat zurückgegangen? Aufgabe: Nenne 5 mögliche Hypothesen und beschreibe, wie du diese mit Datenanalyse prüfen würdest. Gehe dabei auf mögliche Kennzahlen oder Visualisierungen ein."* – **Erläuterung:** Dieser Prompt setzt die KI in die Analystenrolle und gibt die nötigen Informationen (Struktur des Datensatzes, konkrete Frage). Das Ziel ist definiert (Hypothesen + Vorgehensweise), inkl. Anzahl (5) und inhaltlicher Hinweis (Kennzahlen, Visualisierungen). Ein derartig präziser Prompt sollte eine gut strukturierte Antwort liefern, die Hypothesen aufzählt (z.B. Saisoneffekt, Preisänderung, Marketing, Konkurrenz etc.) und jeweils andeutet, wie man das testen würde (z.B. Zeitreihenplot, Segmentanalyse…). Für Business-Analysen oder Data-Science-Interviews sind solche Prompts hilfreich.

- **Business (z.B. Beratung/Marketing):** *"Du bist ein Unternehmensberater, spezialisiert auf Start-ups. Auftrag: Erstelle eine SWOT-Analyse für ein junges Startup im Bereich erneuerbare Energie, das Solarzellen verkauft. Struktur: Liste je 4 Punkte für* Strengths*,* Weaknesses*,* Opportunities *und* Threats*. Stil: Präzise Bulletpoints, maximal ein Satz pro Punkt."* – **Erläuterung:** Der Prompt gibt einen klaren Business-Kontext (Berater für Startup, Branche erneuerbare Energien) und fordert ein gängiges Beratungsformat (SWOT) mit spezifischen Mengenangaben. Dadurch weiß das Modell genau, was es liefern soll. Das Ergebnis wäre vermutlich eine ordentliche SWOT-Liste mit branchentypischen Stärken/Schwächen (z.B. innovative Technologie vs. geringe Markenbekanntheit) und Chancen/Risiken (Marktwachstum vs. Wettbewerbsdruck etc.). In Business-Settings, wo oft bestimmte Frameworks erwartet werden, sollte man diese im Prompt benennen (hier SWOT) und falls nötig erläutern, damit die KI sauber in diesem Rahmen antwortet.

Anhand dieser Beispiele sieht man: **Erfolgreiche Prompts sind wie gute Briefings** – sie enthalten alle wesentlichen Infos (und nur diese), sie sind eindeutig in ihren Anforderungen und oft in sinnvolle Sektionen gegliedert. Zudem sind sie an den jeweiligen **Use-Case angepasst** (technisch, kreativ, analytisch, geschäftlich), was durch Rollenbeschreibung und Tonvorgabe erreicht wird.

4. Fortgeschrittene Techniken

Über die Grundtechniken hinaus gibt es fortgeschrittene Prompting-Strategien, die vor allem bei sehr komplexen Anwendungen oder im Entwicklungsprozess von KI-Systemen genutzt werden. Dazu zählen *Meta-Prompting*, *Prompt Chaining*, der geschickte Einsatz von Kontext (Persona, Zielgruppe etc.) und das

Feintuning von Parametern wie Temperatur oder Top-p sowie systemischen Prompts. Im Folgenden werden diese Aspekte erläutert:

Meta-Prompting und Prompt Chaining: Unter *Meta-Prompting* versteht man das Verwenden eines Prompts, um **neue Prompts zu generieren** oder den Promptprozess selbst zum Gegenstand zu machen. Praktisch könnte man z.B. die KI bitten: *"Schlage 3 Varianten für einen Prompt vor, der XY erreicht"* – hier hilft die KI beim *Prompt schreiben*. Das Modell wird also genutzt, um bessere Instruktionen zu finden. *Prompt Chaining* bezeichnet eine Vorgehensweise, bei der man **mehrere Prompt-Antwort-Schritte hintereinander schaltet**, um eine komplexe Aufgabe etappenweise zu lösen. Die **Ausgabe eines Schritts wird dabei zum Input des nächsten**. Zum Beispiel könnte man in einem ersten Prompt Ideen sammeln lassen, in einem zweiten Prompt dann basierend auf diesen Ideen einen Plan ausarbeiten lassen, in einem dritten Prompt Details verfeinern usw. Ein Anwendungsfall sind **LLM-Agenten**, die autonom arbeiten: Tools wie AutoGPT oder BabyAGI setzen Prompt Chaining ein, indem sie sich selbst Unteraufgaben stellen und diese der Reihe nach abarbeiten [77]. Dabei generiert der Agent zunächst einen Plan (eine Liste von ToDos als Prompt), führt die erste Aktion aus, wertet das Ergebnis aus und schickt bei Bedarf einen Folgesprompt usw. – alles automatisch. Hier sieht man Meta-Prompting und Chaining kombiniert: Die KI erstellt eigene Prompts, um ein übergeordnetes Ziel zu erreichen [77]. Auch im einfacheren Rahmen kann man Chaining manuell nutzen: **Komplexe Probleme in Teilprobleme zerlegen** und für jeden Teil einen eigenen KI-Aufruf machen. Beispiel: Erst *"Finde alle relevanten Fakten zu Thema X"*, dann *"Mache daraus eine Gliederung"*, dann *"Schreibe zu jedem Gliederungspunkt einen Abschnitt"*. Dieser Ansatz imitiert das modulare Arbeiten eines Menschen. *Vorteil:* Ergebnisse werden oft konsistenter und detaillierter, da die KI sich jeweils auf ein eng umrissenes Stück konzentriert. *Nachteil:* Es erfordert mehr Aufwand und evtl. Koordination (ggf. mit Scripts oder Tools, wenn man es automatisieren will). In der Forschung gibt es Erweiterungen dieses Prinzips, etwa **Tree-of-Thoughts (ToT)**: Hier wird kein linearer Kettenablauf, sondern ein **Baum von Gedanken** verfolgt, d.h. die KI exploriert verschiedene Lösungswege oder Hypothesen parallel und evaluiert diese [78]. Solche Methoden (teils mit heuristischen Suchen wie Breadth-First oder A *auf "Gedankenbäumen") sollen die Erfolgsquote bei schwierigen Aufgaben weiter erhöhen* [79]*. Insgesamt sind Meta-Prompting und Prompt-Chaining mächtige Techniken, um die KI selbstreflexiver oder zielstrebiger** zu machen – sie ermöglichen gewissermaßen mehrschrittige Dialoge mit sich selbst, was insbesondere bei Planungs-, Kreativ- oder Problemlöseaufgaben zu besseren Ergebnissen führen kann.

Kontextbezogene Informationen (Persona, Rolle, Zielgruppe): Dieser Punkt knüpft an Rollen-Prompts an, geht aber noch darüber hinaus. Kontextualisierung kann viele Facetten haben: Man kann der KI eine **detaillierte Persona** geben (inkl. Hintergrundinfos, Temperament, Wissensstand), man kann **Zielgruppenmerkmale** nennen, oder auch situative Kontexte schildern (z.B. "Du sprichst gerade mit einem verärgerten Kunden…"). Solche **kontextuellen Prompts** beeinflussen stark die Wortwahl und den Inhalt der generierten Antworten [23]. Beispielsweise führt die Angabe *"Zielgruppe: Grundschulkinder"* dazu, dass Fachbegriffe vermieden und einfache Sätze genutzt werden; *"Persona: KI-Experte mit 10 Jahren Erfahrung"* lässt die KI wahrscheinlich präzisere technische Auskünfte geben. **Rollenbeschreibungen** können auch Arbeitsweisen implizieren: *"Du bist Projektmanager – beginne mit einer kurzen Zusammenfassung, dann liefere einen Meilensteinplan."* Der Trick ist, der KI möglichst **lebensechte Hinweise** zu geben, wer spricht und zu wem. Ein weiteres Kontextmittel sind **Beispiele im Kontext**: Nehmen wir an, man möchte eine bestimmte **Antwortstruktur**. Man kann dem Prompt einen Rahmenkontext geben wie: "Im Folgenden siehst du einen Ausschnitt eines Experteninterviews. Am Ende soll eine Zusammenfassung stehen." – und dann quasi den Dialog starten. Die KI "denkt" dann, sie ist Teil dieser Kontextwelt und wird die Antwort entsprechend stilisieren. Allerdings muss man aufpassen, den Prompt nicht unnötig aufzublähen oder zu fabulieren – Kontextinfos sollen **zielrelevant** sein. Im fortgeschrittenen Gebrauch werden Kontext und Persona oft mit anderen Techniken kombiniert: z.B. in einem System-Message (siehe unten) kann man dauerhaft festlegen *"Du bist ein hilfsbereiter, aber strenger Korrektor, der…"*, und alle Antworten werden das widerspiegeln [32] [33]. Oder

man nutzt “Few-Shot Persona”: d.h. man zeigt durch Beispiele, *wie* ein bestimmter Charakter redet, und dann soll die KI im selben Stil weitermachen. Insgesamt spielen kontextuelle Angaben eine große Rolle bei der **Feinsteuerung** von LLMs – sie sind oftmals der Unterschied zwischen einer generischen 08/15-Antwort und einer **maßgeschneiderten Reaktion**, die genau den richtigen Ton trifft. Allerdings ist wie erwähnt zu beachten: Persona-Prompting ist kein Garant für bessere Faktenqualität [28] – es dient vorrangig der **Stil- und Perspektiv-Anpassung** und kann in bestimmten reasoning-Aufgaben helfen, aber sollte mit Bedacht eingesetzt werden, um keine falschen Effekte (Bias, Performance-Verlust [28]) zu provozieren.

Steuerungsparameter (Temperatur, Top-p, System-Prompts etc.): Neben dem Prompt-Text selbst gibt es technische Parameter, die das Antwortverhalten beeinflussen. Die wichtigsten sind **Temperatur** und **Top-p** (bei OpenAI-Modellen auch **Top-k** in ähnlicher Funktion). **Temperatur** steuert die “Randomness” der Ausgabe: Eine höhere Temperatur (z.B. 1.0 oder 1.2) führt dazu, dass das Modell **kreativere und vielfältigere** Wörter auswählt, während eine niedrige Temperatur (nahe 0) sehr **deterministische und konservative** Antworten erzeugt [80]. Praktisch heißt das: Bei *Temperatur = 0* bekommt man bei gleichem Prompt fast immer die gleiche Antwort, die dem wahrscheinlichsten Pfad folgt – gut für **präzise, faktische Outputs**, aber evtl. etwas trocken. Bei *Temperatur = 1* könnten viele verschiedene Formulierungen oder Ideen auftauchen – gut für **Brainstorming, Geschichten, kreative Texte**, aber manchmal weniger stringent [80] [81]. Extrem hohe Werte (>1.5) machen die Ausgabe sehr zufällig – die KI kann dann unsinnige Sätze produzieren [81]. **Top-p (Nucleus Sampling)** dagegen legt fest, dass das Modell pro Schritt nicht strikt das wahrscheinlichste Wort nimmt, sondern aus den **Top X% der wahrscheinlichsten Fortsetzungen** wählt [82]. Beispiel: Bei Top-p = 0.9 berücksichtigt das Modell nur die wahrscheinlichsten Wörter, deren kombinierte Wahrscheinlichkeit 90% ergibt, und wählt daraus per Zufall aus. Das **begrenzt die Auswahl** auf sinnvolle Möglichkeiten, während es aber immer noch Variation ermöglicht. Top-p und Temperatur können zusammenwirken; oft stellt man einen Parameter fest und justiert den anderen. Für praktische Zwecke: *Niedrige Temperatur, hoher Top-p* -> **fokussierte, genaue Antworten** (das Modell “traut” sich wenig Abweichung). *Hohe Temperatur, hoher Top-p* -> **sehr kreative, aber potentiell abschweifende Antworten** [83]. *Mittlere Temperatur, mittleres Top-p* -> Balance zwischen Genauigkeit und Kreativität. Je nach Aufgabe lohnt es sich, diese Parameter anzupassen: z.B. **Codierung oder Mathe** lieber mit Temperatur 0 (deterministisch, immer gleiche Lösung), **Storytelling** ruhig mit Temperatur 0.8-1.0 (für mehr Originalität).

Ein weiterer wichtiger “Parameter” ist der **System-Prompt** (System-Message) in Chat-Modellen. Er ist keine Zahl, sondern ein versteckter initialer Prompt, der die **Grundrolle und Verhalten** des Modells festlegt, bevor die Nutzereingabe kommt. Entwickler können dort Vorgaben machen wie *“Du bist ein hilfreicher Assistent, der in einem höflichen, professionellen Ton antwortet.”* [32]. Der System-Prompt wird vom Modell bei **jeder Anfrage mit berücksichtigt** und kann starke Auswirkungen haben – er kann z.B. das Modell dazu bringen, bestimmte Informationen nie preiszugeben oder einen bestimmten Stil strikt einzuhalten [32] [33]. Für Endnutzer ist der System-Prompt meist nicht direkt zugänglich (vordefiniert vom Service), aber in einigen Interfaces (z.B. OpenAI API, manche Chat-Tools) kann man ihn setzen. Durch geschicktes System-Prompting lassen sich **globale Rahmenbedingungen** definieren. Beispiel aus einer Dokumentation: Ein System-Prompt *“You are a helpful and polite assistant. Answer in one sentence using a very formal language.”* führt dazu, dass egal was der User fragt, das Modell stets mit einer sehr formellen Ein-Satz-Antwort beginnt [32] [33]. System-Prompts sind immens nützlich, um **Konsistenz** zu gewährleisten, ohne es jedes Mal im User-Prompt wiederholen zu müssen. Fortgeschrittene Anwender nutzen sie, um Agenten Persönlichkeit und Ziel einzuhauchen, bevor die eigentliche Konversation startet.

Daneben existieren weitere Parameter wie **Maximum Length** (Antwortlänge begrenzen, um z.B. Endlosschleifen zu verhindern), **Stop-Sequenzen** (definierte Zeichenketten, bei deren Auftreten die Ausgabe endet), sowie **Frequency Penalty** und **Presence Penalty** [84] [85]. Frequency/Presence

Penalties beeinflussen, ob das Modell sich **wiederholt**: Eine positive Penalty bestraft häufige Tokens, wodurch die Ausgabe *variabler* wird (weniger Wiederholungen) [86]. Diese sind nützlich, wenn man z.B. in langen Antworten vermeiden will, dass der gleiche Satz ständig neu formuliert wird. Für die meisten Anwendungen braucht man diese Feineinstellungen aber selten bewusst anzupassen – sie kommen ins Spiel, wenn man sehr spezifische Optimierungen will (z.B. Gedicht ohne Wortwiederholungen -> Frequency Penalty hoch).

Zusammengefasst: **Temperatur und Top-p regeln die Kreativität vs. Zuverlässigkeit** der KI-Antwort. Mit *Temperatur = 0* und *Top-p = 1* erhält man deterministische, i.d.R. faktenorientierte Outputs [83]. Mit *Temperatur = 0.7* und *Top-p = 0.9* bekommt man abwechslungsreichere, aber immer noch qualitativ hochwertige Texte [87]. **System-Prompts** und Rollen helfen, global den gewünschten Stil vorzugeben [32]. Und weitere Parameter erlauben Feinjustierung bei Bedarf. Fortgeschrittene Prompt-Ingenieure experimentieren mit diesen Hebeln, um je nach Aufgabe das Optimum herauszuholen – sei es maximaler Ideenreichtum oder streng regelkonforme Ausgabe. Die Kunst besteht darin, das **Zusammenspiel von Prompt-Text und Parametern** zu beherrschen, um die gewünschte Ergebnisqualität zu erzielen.

5. Zukunft & Forschung

Aktuelle Trends und offene Fragen im Prompt Engineering: Prompt Engineering entwickelt sich rasant weiter, getrieben von neuen Fähigkeiten der Modelle und neuen Anforderungen. Ein bedeutender Trend ist, dass **neuere Modelle immer besser auch mit vagen oder knappen Prompts zurechtkommen** [88] [89]. Während man heute noch detaillierte Prompts schreiben muss, könnten künftige Modelle so konzipiert sein, dass sie die Intention des Nutzers aus minimalen Hinweisen verstehen und fehlende Details selbstständig erfragen oder ergänzen. Einige Experten glauben daher, dass langfristig prompt engineering als breite Fähigkeit weniger kritisch wird, weil die KI die *"Übersetzung"* von Nutzerwünschen intern übernimmt [88]. Dennoch bleibt das Feld aktuell sehr wichtig und **viel diskutiert**. Offene Forschungsfragen sind zum Beispiel: *Wie können wir systematisch optimale Prompts finden?* – Man spricht hier von **Prompt-Optimierung** oder gar **Automated Prompt Design**. Erste Arbeiten nutzen etwa evolutionäre Algorithmen oder Reinforcement Learning, um maschinell verbesserte Prompts zu generieren, die noch bessere Outputs liefern. Eine andere Frage: *Wie erklärt sich eigentlich der Erfolg von In-Context Learning?* – Also warum ein LLM aus 2–3 Beispielen im Prompt plötzlich Verallgemeinerungen ziehen kann, ohne seine Gewichte zu ändern. Dieses Phänomen wird theoretisch untersucht (Stichwort **"In-Context Learning Theory"**), um herauszufinden, ob das Modell quasi eine mini-Finetuning-Simulation im Forward-Pass macht. Auch die **Robustheit von Prompts** ist ein Thema: Kleine Änderungen in der Formulierung können überraschend große Auswirkungen haben – hier versucht man zu verstehen, wie Modelle Prompts *parsen* und was semantisch wirklich ankommt. Damit zusammen hängt die Abwehr von **Prompt Injection** und fehlerhaften Eingaben: Wenn LLMs in wichtige Anwendungen eingebunden werden, muss man sicherstellen, dass bösartig gestaltete Prompts (oder Teilprompts) sie nicht aushebeln. Das ist ein aktives Forschungsfeld (Sicherheit im Prompting). Ebenso interessiert die Community, wie man Prompts **standardisieren** oder **wiederverwendbar** machen kann – es gibt Überlegungen zu Prompt-Markup-Languages oder Repositorien von getesteten Prompt-Modulen, die man zusammensetzen kann (sogenanntes *Prompt Programming*). Insgesamt bewegt sich Prompt Engineering von einer rein empirischen "Kunst" hin zu einer Kombination aus Best Practices, **wissenschaftlichem Verständnis** und tooling-gestützter Automatisierung.

Auto-Prompting und generative Agenten: Ein besonders spannender Ausblick sind Systeme, die das Prompting **teil- oder vollautomatisch übernehmen**. Unter *Auto-Prompting* versteht man, dass die KI selbst mögliche Anschlussfragen oder Verfeinerungen vorschlägt [90]. Aktuelle Chatbots zeigen bereits Ansätze davon: Googles Bard z.B. schlägt Nutzern Folgeprompts vor, um die Konversation zu vertiefen

[90]. Auch ChatGPT bietet manchmal Button-Vorschläge an (“Frag dies als Nächstes …”). Diese Funktionen führen den Benutzer, ohne dass er selbst die genauen Worte finden muss – quasi *KI-gestütztes Prompting*. In Zukunft könnten Assistenten so weit gehen, dass der Nutzer nur noch grob sein Ziel formuliert (z.B. “Ich brauche eine Marketing-Strategie für Produkt X”), und der Agent dann **durch gezielte Rückfragen** und interne Kettenschritte die Details erarbeitet – am Ende präsentiert er die fertige Strategie. Hier verschwimmt die Grenze zwischen Benutzerprompt und KI-Aktion: Die KI generiert sich ihre *Sub-Prompts* selbst, um das Ziel zu erreichen.

Generative Agenten wie **AutoGPT, BabyAGI** und ähnliche verkörpern bereits die Vision eines *autonomen KI-Agenten*, der auf ein Ziel hin selbständig plant und handelt [77]. Diese Agenten nehmen einen hochgradig abstrakten Auftrag (“Finde die profitabelsten Aktien und kaufe eine Auswahl”) und erzeugen dann eigenständig eine Reihe von Aktionen: Sie formulieren sich Fragen, rufen ggf. Tools (z.B. Websuche oder Rechner) via Unter-Prompts auf, werten Ergebnisse aus und passen ihren Plan an – bis das Ziel erreicht ist. Dabei wird die **Rolle des Prompt Engineers teilweise von der KI selbst übernommen**: Das System hat Module, die neue Prompts schreiben (z.B. eine Google-Suchanfrage generieren), das Resultat lesen und den nächsten Prompt basierend darauf erstellen [77]. Dieser recursive Prompting-Mechanismus erlaubt es, Aufgaben zu lösen, die ein einziger statischer Prompt kaum bewältigen könnte. Das heißt aber nicht, dass menschliches Prompting obsolet wird – vielmehr verlagert es sich auf eine **höhere Ebene**: Man entwirft jetzt *Agent-Prompts*, in denen man den Agenten umreißt (Ziele, Tools, Grenzen) und der Agent erledigt die Feinplanung. In Zukunft könnte dies sehr weit gehen: KI-Agenten, die fast wie kleine autonom forschende “Mitarbeiter” Aufgaben entgegennehmen und sich selbst beibringen, was zu tun ist. Prompt Engineering würde dann bedeuten, die **Agentenumgebung** richtig zu gestalten (eine Art Meta-Prompt-Engineering).

Weiterentwicklung der Modelle: Ein weiterer Trend ist, dass **Modelle durch RLHF und Feintuning immer besser** darin werden, Nutzanweisungen zu folgen – auch wenn diese unklar sind [91]. GPT-4 und neuere (GPT-4.5, GPT-5?) legen noch mehr Gewicht auf Verständnis des Nutzerziels, selbst wenn der Prompt nicht perfekt ist [92]. Gleichzeitig werden Kontextfenster größer (es gibt Modelle mit 100k+ Token Kontext); dadurch kann man einfach **sehr viel Rohmaterial** (Daten, Dokumente) ins Prompt packen, was den Bedarf an cleverer Kondensation reduziert (man muss nicht mehr so knapp sein, man kann alles reinwerfen). Dennoch: Große Kontexte bringen eigene Herausforderungen (man muss relevanten Kontext filtern, sonst *verwässert* der Prompt). Auch hier sind neue Ansätze gefragt, etwa **Retrieval-Augmented Generation (RAG)**, wo externe Wissensdatenbanken durchsucht werden und nur passende Auszüge ins Prompt kommen. Das Zusammenspiel von **Prompting und Werkzeugnutzung** (Tool-Use) wird ebenfalls wichtiger: KIs, die wissen, wann sie lieber eine Datenbank fragen oder einen Rechner nutzen sollten, müssen im Prompt (bzw. Systemmessage) dahingehend instruiert werden – hierzu gibt es das Forschungsfeld der **ReAct-Prompts** (Reasoning+Acting), in denen Modelle lernen, zwischendrin Aktionen (API-Aufrufe etc.) einzuschieben.

Auto-Prompting-Forschung: Es gibt auch Ansätze, die prompt engineering weitgehend automatisieren wollen. Ein Beispiel ist das **DSPy-Framework** (von Microsoft-Forschern entwickelt), welches versucht, Prompting und Programmfluss zu vereinen [93]. Die Idee dahinter: Entwickler beschreiben auf höherer Ebene, was sie vom Modell wollen, und DSPy übernimmt das Ausformulieren der optimalen Prompts und Feintuning der Modellgewichte teilweise automatisch [93]. Solche Tools könnten die “Black Box” Prompting in strukturierte Bahnen lenken, ähnlich wie höhere Programmiersprachen Maschinencode ersetzen – der Entwickler sagt *was* er will, das System erledigt das *wie (Prompt)* im Hintergrund.

Ausblick: Die Frage “ob wir in Zukunft noch Prompt Engineers brauchen” wird kontrovers diskutiert [94] [88]. Einerseits sieht man Bemühungen, die Notwendigkeit komplexer Prompts zu reduzieren (durch intelligentere Modelle, UI-Assistenten, Auto-Prompting). Andererseits wächst der Einsatzbereich von LLMs rasant, sodass immer neue Domänenwissen und Feinsteuerung gefragt sind – hier sind

spezialisierte Prompts nach wie vor gefragt. Wahrscheinlich verschiebt sich die Rolle eher: vom einfachen *Schreiben einer Anfrage* hin zum *Orchestrieren komplexer KI-Workflows*. Prompt Engineering könnte integraler Bestandteil von **"AI Agent Design"** werden. In jedem Fall ist es sinnvoll, die aktuellen Best Practices zu kennen, da sie unmittelbar dabei helfen, bessere Resultate aus heutigen Systemen zu holen – und viele Prinzipien (Klarheit, Struktur, Kontext) werden auch mit cleveren KI-Systemen von morgen gültig bleiben, denn sie sind im Kern *gute Kommunikation*. Auch wenn zukünftig KI-Modelle einen Teil dieser Kommunikation selbst regeln, wird menschliche Kreativität und Präzision gefragt bleiben, um die **richtigen Ziele und Constraints zu setzen**. Die Forschung bleibt aktiv: sei es in technischen Verbesserungen (bessere In-Context-Learning-Algorithmen, Mechanismen gegen Halluzinationen), in Tools (Agenten, Prompt-Datenbanken) oder in der Ausbildung (immer mehr Kurse und Literatur entstehen zum Thema Prompt Engineering). Es bleibt spannend zu sehen, wie sich dieses Feld mit den KI-Fähigkeiten weiterentwickelt – ob es in zehn Jahren selbstverständlich zum Allgemeinwissen gehört, wie man KI-Anfragen formuliert, oder ob die KI uns bis dahin einen Großteil dieser Arbeit abgenommen hat.

Checkliste für praxisgerechte Prompts: Zum Abschluss eine kompakte Zusammenfassung der wichtigsten Tipps, die beim Erstellen von Prompts beherzigt werden sollten:

- **Klarer Auftrag:** Formuliere eindeutig, was die KI tun soll (keine vagen "Erzähl mal was…"-Aufträge).
- **Kontext einbinden:** Gib alle relevanten Informationen oder Daten, die die KI zur Bearbeitung braucht, im Prompt mit.
- **Ziel und Format angeben:** Beschreibe, wie das Ergebnis aussehen soll (z.B. Bulletpoints, Tabelle, bestimmte Länge, Tonfall).
- **Beispiele nutzen (wenn sinnvoll):** Zeige der KI durch 1-3 Beispiele, was du erwartest – insbesondere bei spezifischen Formaten oder Stilen.
- **Komplexität aufteilen:** Bei mehreren Teilaufgaben oder sehr komplexen Fragen lieber Schritt für Schritt vorgehen oder mehrere Prompts verwenden, statt alles in einen Prompt zu packen.
- **Auf Widersprüche prüfen:** Stelle sicher, dass deine Anforderungen im Prompt nicht unabsichtlich kollidieren (z.B. nicht gleichzeitig "ausführlich" und "kurz" verlangen).
- **Iteration einplanen:** Sei bereit, den Prompt mehrmals zu justieren. Starte einfach und füge Details hinzu, oder lass die KI erste Ergebnisse liefern und präzisiere dann weiter.
- **Struktur und Formatierung:** Verwende bei längeren Prompts Abschnitte, Überschriften oder Aufzählungen, um dem Modell (und dir selbst) Übersicht zu geben. Ein gut gegliederter Prompt führt zu fokussierteren Antworten [40] [41].
- **Parameter beachten:** Falls möglich, wähle die passenden Einstellungen (Temperatur niedrig für Fakten, höher für Kreatives; ggf. System-Prompt definieren für konsistenten Stil).

Wenn du diese Punkte beachtest, bist du auf dem besten Weg, effektive Prompts zu erstellen. So erhältst du konsistente, relevante und qualitativ hochwertige Antworten von deinem KI-Modell – und nutzt sein Potenzial optimal aus. [40] [41]

1 2 3 9 11 12 14 Zero-Shot vs. Few-Shot Prompting: Key Differences - Shelf
https://shelf.io/blog/zero-shot-and-few-shot-prompting/

4 5 6 7 8 What is zero-shot prompting? | IBM
https://www.ibm.com/think/topics/zero-shot-prompting

10 Few-Shot Prompting | Prompt Engineering Guide
https://www.promptingguide.ai/techniques/fewshot

13 Zero-Shot, One-Shot, and Few-Shot Prompting
https://learnprompting.org/docs/basics/few_shot?srsltid=AfmBOoraeBZtO3xVmdPEBZN09XFtkZrmgRBrCFQAOAGC7glnLHmI6Qm9

15 16 17 18 19 20 21 22 How Chain of Thought (CoT) Prompting Helps LLMs Reason More Like Humans | Splunk
https://www.splunk.com/en_us/blog/learn/chain-of-thought-cot-prompting.html

23 24 25 29 30 31 Role Prompting: Guide LLMs with Persona-Based Tasks
https://learnprompting.org/docs/advanced/zero_shot/role_prompting?srsltid=AfmBOorqrEoF1bYG_dFXwRdUv8JjtP_OYEWDkBqbESteNGetic7n2osJ

26 27 28 Role-Prompting: Does Adding Personas to Your Prompts Really Make a Difference?
https://www.prompthub.us/blog/role-prompting-does-adding-personas-to-your-prompts-really-make-a-difference

32 33 Understanding User, Assistant, and System Roles in ChatGPT | Baeldung on Computer Science
https://www.baeldung.com/cs/chatgpt-api-roles

34 35 36 37 38 39 44 45 46 47 48 49 50 51 7 Prompt Engineering Mistakes Beginners Must Avoid (and How to Fix Them) | PromptJesus
https://www.promptjesus.com/blog/7-prompt-engineering-mistakes-beginners-must-avoid

40 41 52 53 54 55 56 57 58 Markdown for Prompt Engineering Best Practices - Tenacity
https://tenacity.io/snippets/supercharge-ai-prompts-with-markdown-for-better-results/

42 43 60 61 62 63 Prompt Engineering, Explained. A great tool for teaching the ‘art’ of... | by Sunil Manghani | Electronic Life | Medium
https://medium.com/electronic-life/prompt-engineering-explained-3b83ba347722

59 Generative Artificial Intelligence: Crafting a Prompt - Research Guides
https://risd.libguides.com/genAI/prompts

64 65 66 67 68 69 70 71 72 74 75 76 Prompt Engineering: The Art of Getting What You Need From Generative AI | Ivan Allen College of Liberal Arts
https://iac.gatech.edu/featured-news/2024/02/AI-prompt-engineering-ChatGPT

73 Prompt Engineering for Generative AI | Machine Learning
https://developers.google.com/machine-learning/resources/prompt-eng

77 88 89 90 91 92 93 94 The Future of Prompt Engineering: Evolution or Extinction? | by Code and Theory | Code and Theory | Medium
https://medium.com/code-and-theory/the-future-of-prompt-engineering-evolution-or-extinction-2a74f183fae1

78 What is Tree Of Thoughts Prompting? - IBM
https://www.ibm.com/think/topics/tree-of-thoughts

79 Tree of Thought Prompting - Walking the Path of Unique Approach ...
https://promptengineering.org/tree-of-thought-prompting-walking-the-path-of-unique-approach-to-problem-solving/

80 81 82 84 85 86 Understanding Temperature, Top P, and Maximum Length in LLMs
https://learnprompting.org/docs/intermediate/configuration_hyperparameters?srsltid=AfmBOopo8p1gWSCx-ixNzqO6HSRvE5Bb5Cg3gyZjEGK6mY7RcN4pmyc6

83 Temperature, top_p and top_k for chatbot responses - Prompting
https://community.openai.com/t/temperature-top-p-and-top-k-for-chatbot-responses/295542

87 Understanding prompt engineering parameters - Portkey
https://portkey.ai/blog/understanding-prompt-engineering-parameters