Von Prompt Engineering zu Context Engineering

Ein praxisorientiertes Kapitel für den Einsatz im Unterricht Stand: März 2026. Autor Thomas Jörg

1. Paradigmenwechsel: „Prompt Engineering“ reicht nicht mehr

Im Juni 2025 prägte Andrej Karpathy (ehemaliger KI-Chef bei Tesla und OpenAI-Gründer) auf der Plattform X einen neuen Begriff: ***Context Engineering***. Seine Aussage: „Ich bevorzuge ‘Context Engineering’ gegenüber ‘Prompt Engineering’, weil es die eigentliche Kompetenz besser beschreibt: die Kunst, dem LLM genau den richtigen Kontext bereitzustellen, damit es die Aufgabe lösen kann.“

Seine Analogie ist dabei besonders einprägsam und eignet sich hervorragend als Erklärung für Schüler:

> **Karpathys CPU-Analogie**
>
> **Das LLM ist die CPU** (der Prozessor, der die eigentliche Arbeit macht).
>
> **Das Context Window ist der RAM** (der Arbeitsspeicher, in dem alles liegt, worauf das Modell zugreifen kann).
>
> **Du bist das Betriebssystem** – du entscheidest, was in den Arbeitsspeicher geladen wird.
>
> Zu wenig Kontext: Das Modell raten muss und halluziniert. Zu viel Kontext: Die Kosten steigen, die Qualität sinkt. Die richtige Balance zu finden, ist die eigentliche Kunst.

Prompt Engineering (alt)	Context Engineering (neu)
„Stell eine gute Frage“	„Baue die gesamte Umgebung auf“
Ein einzelner Satz oder Absatz	System-Prompt + Rolle + Beispiele + Daten + Tools
Fokus auf geschickte Formulierung	Fokus auf Informationsarchitektur
Trial & Error	Systematische Iteration mit Evaluation
Statisch: einmal geschrieben, fertig	Dynamisch: passt sich an Aufgabe und Nutzer an

Für den Unterricht bedeutet das:

> **Wir bringen Schülern nicht bei, „gute Fragen“ zu stellen.**
> **Wir bringen ihnen bei, systematisch den richtigen Kontext aufzubauen.**

Das ist eine Kompetenz, die weit über die Nutzung von ChatGPT hinausgeht – es ist strukturiertes Denken, Informationsarchitektur und kritische Reflexion in einem.

2 Die Bausteine eines guten Kontexts

Ein leistungsfähiger Kontext für ein LLM besteht aus mehreren Schichten, die zusammenspielen. Jede Schicht erfüllt eine bestimmte Funktion. Im Folgenden werden die wichtigsten Bausteine vorgestellt – jeweils mit einem konkreten Beispiel, das direkt im Unterricht einsetzbar ist.

2.1 Der System-Prompt: Die „Grundhaltung“ des Modells

Der System-Prompt ist die wichtigste Neuerung gegenüber dem einfachen Prompting. Er wird *vor* dem eigentlichen Gespräch gesetzt und definiert, **wer** das Modell ist, **wie** es sich verhalten soll und **welche Regeln** gelten. Er ist vergleichbar mit der Rollenbeschreibung eines Schauspielers, bevor die Szene beginnt.

> **Hands-on: System-Prompt für einen Physik-Tutor**
>
> ```
> Du bist ein geduldiger Physik-Nachhilfelehrer für
> Schüler der 10. Klasse Gymnasium.
>
> Regeln:
> - Antworte immer auf Deutsch.
> - Verwende maximal 150 Wörter pro Antwort.
> - Benutze Alltagsbeispiele statt Fachjargon.
> - Wenn der Schüler einen Denkfehler macht, korrigiere
>   ihn nicht sofort, sondern stelle eine Rückfrage, die
>   den Schüler zum Nachdenken bringt (sokratische Methode).
> - Gib NIEMALS eine vollständige Lösung, wenn der
>   Schüler eine Aufgabe lösen soll. Leite ihn stattdessen
>   Schritt für Schritt an.
> - Wenn du dir bei einer Antwort unsicher bist, sage es.
> ```

Warum ist das so viel besser als ein einzelner Prompt? Weil der System-Prompt für das *gesamte Gespräch* gilt. Jede nachfolgende Frage des Schülers wird im Lichte dieses Kontexts beantwortet. Das Modell „vergisst“ die Rolle nicht mitten im Gespräch.

Tipp für die Praxis: In ChatGPT kann der System-Prompt über „Custom Instructions“ oder die API gesetzt werden. In Claude über das „System“-Feld. In LM Studio und AnythingLLM ebenfalls im Konfigurationsbereich. In der Web-Oberfläche von ChatGPT kann man alternativ den System-Prompt als *erste Nachricht* eintippen, bevor der Schüler seine Fragen stellt.

2.2 Strukturierte Prompts mit XML-Tags

Eine der wirksamsten Techniken aus der professionellen Prompt-Entwicklung ist die Verwendung von **XML-artigen Tags** zur Strukturierung des Kontexts. Tags wie <aufgabe>, <kontext>, <format> oder <beispiel> helfen dem Modell, verschiedene Informationsarten sauber voneinander zu trennen – ähnlich wie HTML-Tags einer Website Struktur geben.

> **Hands-on: Strukturierter Prompt für Klassenarbeits-Erstellung**

```
<rolle>
Du bist ein erfahrener Informatik-Lehrer an einem
deutschen Gymnasium.
</rolle>

<aufgabe>
Erstelle 5 Aufgaben für eine Klassenarbeit zum Thema
Binäre Suche in Python.
</aufgabe>

<rahmenbedingungen>
- Klassenstufe: 11, Leistungskurs
- Bearbeitungszeit: 45 Minuten
- Schwierigkeitsverteilung: 2 leicht, 2 mittel, 1 schwer
- Punkteverteilung angeben
- Erwartungshorizont mit Musterlösung beifügen
</rahmenbedingungen>

<format>
Gib die Aufgaben in nummerierten Abschnitten aus.
Trenne Aufgabenblatt und Erwartungshorizont deutlich.
</format>
```

Warum funktioniert das so gut? Moderne LLMs wie Claude, GPT-4 und Gemini sind auf riesigen Mengen strukturierter Daten (HTML, XML, Markdown) trainiert worden. Die Tags signalisieren dem Modell: „Hier endet die Rollenanweisung, hier beginnt die eigentliche Aufgabe.“ Das reduziert Missverständnisse drastisch.

Übung für Schüler: Schüler nehmen einen einfachen, unstrukturierten Prompt („Mach mir ein Quiz über Sortieralgorithmen“) und strukturieren ihn mit Tags um. Anschließend vergleichen sie die Qualität der Antworten. Die Erkenntnis ist oft verblüffend.

2.3 Few-Shot-Beispiele: Dem Modell zeigen, was man meint

Statt dem Modell lang und breit zu *erklären*, was man möchte, ist es oft effektiver, es zu *zeigen*. Das Prinzip heißt **Few-Shot Prompting**: Man gibt dem Modell 2–3 Beispiele des gewünschten Eingabe-Ausgabe-Paares, und es leitet daraus das Muster ab.

Hands-on: Few-Shot für Code-Review-Feedback

```
Bewerte den folgenden Schüler-Code nach diesem Muster:

<beispiel>
Code: x = x + 1
Bewertung: Korrekt, aber kürzer als x += 1.
Tipp: Verwende den +=-Operator für kompakteren Code.
Note: Gut (2)
</beispiel>

<beispiel>
Code: for i in range(0, len(liste), 1):
Bewertung: Funktional korrekt, aber unnötig verbose.
Tipp: range(len(liste)) reicht, da Start=0 und
  Schritt=1 die Standardwerte sind.
Note: Befriedigend (3)
</beispiel>

Bewerte nun diesen Code:
[hier den Schülercode einfügen]
```

Wichtig: Aktuelle Forschung (Stand 2025/26) zeigt, dass Modelle die Details in Beispielen *sehr genau* imitieren. Wenn deine Beispiele Fehler enthalten oder einen bestimmten Stil zeigen, wird das Modell diesen übernehmen. Beispiele sind daher sowohl mächtig als auch riskant – ein gutes Thema für eine Unterrichtsdiskussion über Bias in KI-Systemen.

2.4 Chain-of-Thought: Schritt für Schritt denken lassen

Chain-of-Thought (CoT) ist eine Technik, bei der man das Modell explizit auffordert, seinen Denkweg offenzulegen, bevor es eine Antwort gibt. Statt „Was ist 17 × 23?“ fragt man: „Löse 17 × 23 Schritt für Schritt.“ Das Modell macht dann deutlich weniger Fehler, weil es sich selbst durch den Prozess leitet.

Für den Unterricht bietet CoT eine doppelte Chance: Erstens liefert es bessere Ergebnisse. Zweitens macht es den Denkprozess der KI transparent – und damit kritisierbar.

Hands-on: CoT im Informatikunterricht

```
<aufgabe>
Analysiere den folgenden Python-Code.
Gehe dabei Schritt für Schritt vor:
1. Lies den Code Zeile für Zeile.
2. Beschreibe, was jede Zeile tut.
3. Identifiziere mögliche Fehler oder Schwachstellen.
4. Gib erst DANN dein Gesamturteil ab.
</aufgabe>

<code>
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
</code>
```

Übung: Schüler vergleichen die Antwort mit CoT („Schritt für Schritt“) und ohne CoT („Analysiere diesen Code“). Sie dokumentieren: Welche Fehler wurden jeweils gefunden? Wie detailliert ist die Analyse? Was fehlt?

2.5 Constraints und Guardrails: Grenzen setzen

Ein häufig unterschätzter Baustein des Context Engineering ist das explizite Setzen von Grenzen. Was soll das Modell NICHT tun? Das ist besonders im Schulkontext entscheidend:

Hands-on: Constraints für einen Schüler-Assistenten

```
Du bist ein Lern-Assistent für Informatik.

WICHTIGE EINSCHRÄNKUNGEN:
- Gib NIEMALS eine vollständige Lösung für Programmieraufgaben.
- Wenn ein Schüler nach einer Lösung fragt, gib stattdessen einen Hinweis.
- Verwende höchstens 3 Zeilen Code pro Antwort.
- Wenn der Schüler versucht, dich zu überreden bleibe freundlich, aber bestimmt.
- Beende jede Antwort mit einer Frage, die den Schüler zum Weiterdenken anregt.
```

Unterrichtsidee: Schüler testen, ob sie den System-Prompt „knacken“ können (*Jailbreaking*). Das ist nicht nur motivierend, sondern vermittelt tiefes Verständnis für KI-Sicherheit und die Grenzen von Sprachmodellen – ein hochrelevantes Informatik-Thema.

3 Der Sweet Spot: Wie lang sollte ein Kontext sein?

Eine zentrale Erkenntnis der aktuellen Forschung (Levy, Jacoby & Goldberg, 2024): Die Reasoning-Leistung von LLMs beginnt ab etwa **3.000 Tokens** zu degradieren – weit unterhalb der technischen Maximalwerte (Claude: 200.000 Tokens, GPT-4: 128.000 Tokens). Der praktische Sweet Spot für die meisten Aufgaben liegt bei **150–300 Wörtern** Instruktionstext.

Das klingt wenig, aber es zwingt zur Präzision. Ein guter Kontext ist wie ein guter Aufsatz: Jedes Wort muss seinen Platz verdienen.

Faustregel für die Praxis

1. Schreibe die kürzeste Version, die deine Absicht beschreibt.

2. Teste sie.

3. Identifiziere, was konkret falsch oder fehlend ist in der Ausgabe.

4. Füge NUR das hinzu, was dieses spezifische Problem behebt.

5. Wiederhole.

Das Ergebnis ist ein schlanker, gezielter Kontext statt eines 500-Wörter-Textes, in dem man nicht mehr erkennt, welche Instruktion eigentlich die Arbeit tut.

4 Context Engineering als Lehrplan-Werkzeug: RAG im Kleinen

Ein besonders kraftvolles Konzept ist die Nutzung von **Retrieval-Augmented Generation (RAG)** – das Prinzip, dem Modell zusätzliches Wissen als Kontext mitzugeben, über das es sonst nicht verfügt. Im Schulalltag bedeutet das: **Den Lehrplan, Aufgabensammlungen oder Bewertungskriterien direkt in den Kontext einfüttern.**

Hands-on: Lehrplan als Kontext

```
<rolle>
Du bist ein Aufgabenentwickler für den
Informatik-Unterricht in Baden-Württemberg.
</rolle>

<lehrplan>
Bildungsplan BW Informatik Klasse 10:
- Algorithmen und Datenstrukturen
  - Sortierverfahren (Selection Sort, Insertion Sort)
  - Laufzeitanalyse (O-Notation qualitativ)
  - Suchen in sortierten Daten
- Objektorientierte Modellierung
  - Klassen und Objekte
  - Attribute und Methoden
</lehrplan>

<aufgabe>
Erstelle eine Übung, die Sortierverfahren mit
OOP verbindet: Die Schüler sollen eine Klasse
„Sortierer“ modellieren, die verschiedene
Sortieralgorithmen als Methoden enthält.
</aufgabe>
```

Dieses Prinzip skaliert: Man kann den kompletten Kernlehrplan oder ein Schulcurriculum als Textdatei vorbereiten und bei jeder Aufgabenerstellung als Kontext mitgeben. So stellt man sicher, dass generierte Aufgaben, Quizze oder Klausuren *lehrplankonform* sind – ohne dass man jedes Mal alles von Hand prüfen muss.

In AnythingLLM oder LM Studio lässt sich so ein Lehrplan-Dokument sogar als permanenter Kontext für einen Workspace hinterlegen. Tools wie SchulKI bieten ähnliche Funktionen.

5 Typische Fehler – und wie Context Engineering hilft

5.1 Der „Ein-Satz-Prompt“

Schwach (typischer Schülerprompt)	Stark (mit Context Engineering)
„Erklär mir Binärbäume“	„Erkläre Binärbäume für einen 10.-Klässler, der Arrays kennt aber noch keine rekursiven Strukturen. Verwende eine Analogie.“
„Schreib mir ein Programm“	„Schreibe ein Python-Programm (max. 20 Zeilen), das BubbleSort auf einer Integer-Liste implementiert. Kommentiere jede Zeile.“
„Was ist KI?“	„Erkläre den Unterschied zwischen regelbasierter KI und maschinellem Lernen anhand eines Alltagsbeispiels. Maximal 100 Wörter.“

5.2 „Die KI hat recht“ – Unkritische Übernahme

Einer der größten Fehler ist die blinde Übernahme von LLM-Antworten. Context Engineering hilft auch hier: Indem man das Modell explizit anweist, Unsicherheiten zu benennen.

Hands-on: Unsicherheits-Instruktion

```
Wenn du dir bei einer Aussage nicht sicher bist,
markiere sie mit [UNSICHER] und erkläre kurz,
warum du unsicher bist.
Erfinde keine Quellen. Wenn du keine Quelle kennst,
sage „mir ist keine spezifische Quelle bekannt“.
```

Diese einfache Ergänzung im System-Prompt reduziert Halluzinationen signifikant – weil das Modell die „Erlaubnis“ erhält, „ich weiß es nicht“ zu sagen, statt eine plausible Antwort zu erfinden.

5.3 Der „Copy-Paste-Schüler“: Ergebnis ohne Verständnis

Wenn Schüler einfach die Ausgabe des LLMs kopieren, ohne sie zu verstehen, hat die KI die Aufgabe erledigt – aber der Schüler hat nichts gelernt. Context Engineering bietet eine elegante Lösung: den Assistenten so konfigurieren, dass er gar nicht erst komplette Lösungen liefert (siehe den Constraint-Prompt in Abschnitt 2.5).

6 Unterrichtssequenz „Context Engineering“ (2–3 Doppelstunden)

Stunde 1: Vom naiven Prompt zum strukturierten Kontext

Einstieg: Schüler stellen einem LLM die gleiche Frage in drei Varianten: (a) einzeiliger Prompt, (b) Prompt mit Rolle und Constraints, (c) vollständig strukturierter Prompt mit Tags. Sie dokumentieren und vergleichen die Qualität der Antworten. Erarbeitungsphase: Die Bausteine (System-Prompt, Tags, Beispiele, Constraints) werden gemeinsam systematisiert.

Stunde 2: Chain-of-Thought und Halluzinationserkennung

Einstieg: Schüler lassen das LLM eine fachlich anspruchsvolle Frage beantworten – einmal mit und einmal ohne „Schritt für Schritt“-Anweisung. Sie bewerten: Wo stimmt die Argumentation? Wo halluziniert das Modell? Transfer: Schüler testen die Unsicherheits-Instruktion ([UNSICHER]-Markierung) und diskutieren: Warum halluziniert ein LLM? Was bedeutet „Next Token Prediction“ für die Zuverlässigkeit?

Stunde 3: Eigene System-Prompts bauen und testen

Projektarbeit: Jede Gruppe baut einen spezialisierten Assistenten für ein konkretes Szenario (z.B. Mathe-Tutor, Bewerbungstrainer, Python-Debugger, Quiz-Master). Der System-Prompt muss Rolle, Constraints, Format und mindestens ein Beispiel enthalten. Abschluss: Gruppen testen gegenseitig die Assistenten der anderen – gelingt ein „Jailbreak“? Was hält stand?

7.1 Prompt Chaining: Aufgaben zerlegen

Statt eine komplexe Aufgabe in einem einzigen Prompt zu lösen, zerlegt man sie in mehrere Schritte. Die Ausgabe von Schritt 1 wird zum Input für Schritt 2. Beispiel: (1) Zusammenfassung eines Fachtextes erstellen, (2) daraus Quiz-Fragen generieren, (3) Erwartungshorizont ableiten. Jeder Schritt kann einzeln optimiert und geprüft werden.

7.2 Temperature und Sampling: Der „Kreativitätsregler“

Der Parameter **Temperature** steuert, wie „kreativ“ das Modell antwortet. Temperature 0 = deterministisch und konsistent (ideal für Faktenfragen). Temperature 0.7–1.0 = variabel und kreativer (ideal für Brainstorming oder kreatives Schreiben). In LM Studio und über die API ist dieser Parameter direkt einstellbar – ein hervorragendes Unterrichtsthema, um die probabilistische Natur von LLMs zu demonstrieren.

7.3 Multimodale Prompts: Bilder als Kontext

Moderne Modelle (GPT-4o, Claude 3.5, Gemini) können Bilder als Input verarbeiten. Im Informatikunterricht eröffnet das faszinierende Möglichkeiten: ein Foto eines handschriftlichen Flussdiagramms analysieren lassen, UML-Diagramme interpretieren oder Screenshots von Fehlermeldungen debuggen. Der Kontext ist dann nicht mehr nur Text, sondern multimodal – daher der erweiterte Begriff „Context Engineering“.

8 Zusammenfassung: Die fünf Prinzipien des Context Engineering

Nr.	Prinzip	Kernaussage
1	**Rolle definieren**	Gib dem Modell eine klare Identität im System-Prompt.
2	**Struktur geben**	Nutze Tags, Abschnitte und klare Trennungen.
3	**Zeigen statt erklären**	Verwende Few-Shot-Beispiele für das gewünschte Format.
4	**Grenzen setzen**	Definiere explizit, was das Modell NICHT tun soll.
5	**Iterativ verbessern**	Teste, identifiziere Schwächen, ergänze gezielt.

9 Weiterführende Ressourcen

Anthropic Prompt Engineering Guide: https://docs.claude.com/en/docs/build-with-claude/prompt-engineering/overview

Prompting Guide (Community): https://www.promptingguide.ai/guides/context-engineering-guide

Context Engineering Repository (GitHub): https://github.com/davidkimai/Context-Engineering

Karpathys Post zu Context Engineering (X/Twitter): https://x.com/karpathy/status/1937902205765607626

GenAI in CS Education Consortium: https://www.teachcswithai.org/

Google Teachable Machine (Hands-on ML im Browser): https://teachablemachine.withgoogle.com/

Machine Learning for Kids (Scratch + ML): https://machinelearningforkids.co.uk/