Praxisleitfaden für den Einsatz generativer KI beim Erlernen einer Programmiersprache

Version: Okt-2025

Raluca–Maria Vedislav, Dennis Grewe, W. Daniel Scherz

***Zusammenfassung*—Künstliche Intelligenz (KI) verändert derzeit, wie wir lernen und lehren – besonders, wenn es ums Programmieren geht. Tools wie ChatGPT, GitHub Copilot oder Gemini können dir beim Coden helfen, indem sie Vorschläge machen, Fehler finden oder Konzepte erklären. Das kann den Einstieg in eine neue Programmiersprache deutlich erleichtern und dir helfen, schneller Fortschritte zu machen.**

Doch KI kann noch mehr: Sie kann sich an dein Lerntempo anpassen, passende Übungen vorschlagen oder dich Schritt-für-Schritt durch schwierige Aufgaben führen. So entsteht eine Lernumgebung, die dich individuell unterstützt – fast wie ein persönlicher Tutor.

Gleichzeitig bringt der Einsatz von KI auch Herausforderungen mit sich. Wenn man sich zu sehr auf automatische Vorschläge verlässt, kann leicht das Verständnis für die Grundlagen und das eigene Problemlösevermögen zurück bleiben. Anstelle eine neue Programmiersprache zu meistern, versteht man die Konzepte nicht wirklich, oder kann die Sprache nicht gezielt für das Finden einer Problemlösung einsetzen. Es ist also wichtig, KI bewusst und reflektiert einzusetzen – als Werkzeug, nicht als Ersatz fürs eigene Denken.

Dieser Praxisleitfaden zeigt dir, wie du KI sinnvoll beim Erlernen einer Programmiersprache nutzen kannst. Er stellt bewährte Methoden, Tipps und Beispiele vor, die dir helfen, von den Stärken der KI zu profitieren, ohne dabei das selbstständige Denken zu vernachlässigen. Ziel ist es, dich dabei zu unterstützen, KI so einzusetzen, dass du langfristig besser, kreativer und nachhaltiger programmierst.

***Index Terms*—Generative KI, Programmieren lernen, Prompt Engineering, LLM, Chatbots, GitHub Copilot, Hochschullehre**

I. Einführung

Künstliche Intelligenz (KI) ist heute allgegenwärtig – auch im Studium. Ob beim Schreiben, Recherchieren oder Programmieren: KI-Tools können uns das Lernen und Arbeiten enorm erleichtern. Gleichzeitig stellen sie uns vor neue Fragen: Wie nutze ich KI sinnvoll? Wie viel Unterstützung ist „okay“? Und wie kann ich sicherstellen, dass ich dabei trotzdem selbst etwas lerne?

Dieser Praxisleitfaden soll dir helfen, genau das herauszufinden. Wenn du gerade mit dem Programmieren beginnst, findest du hier viele praktische Tipps, wie du KI-Tools gezielt einsetzen kannst – um deinen Lernprozess zu verbessern. In den folgenden Abschnitten erfährst du, welche Chancen KI im Programmieralltag bietet, wo mögliche Stolperfallen liegen und wie du sie vermeidest. Außerdem bekommst du eine Übersicht über die bekanntesten KI-Tools, die bei der Programmierung zum Einsatz kommen und wie du sie optimal nutzen kannst.

A. Ziel dieses Leitfadens

KI verändert, wie wir lernen. Tools wie ChatGPT, GitHub Copilot oder Lernplattformen auf Basis künstlicher Intelligenz können Inhalte auf dich zuschneiden und dich beim Lösen von Aufgaben unterstützen. Doch der richtige Einsatz von KI wirft auch Fragen auf – zum Beispiel zum Datenschutz oder zur Nachvollziehbarkeit deiner Ergebnisse.

Dieser Praxisleitfaden soll dir zeigen, wie du KI beim Programmieren so einsetzt, dass sie dir wirklich hilft, anstatt dich zu bremsen. Der Leitfaden gibt dir Orientierung, wann KI sinnvoll ist, wann du besser selbst nachdenken solltest – und wie du mit den Ergebnissen kritisch umgehst.

B. Warum ist der Leitfaden wichtig?

KI Tools können beim Lernen unglaublich nützlich sein – aber nur, wenn du weißt, wie du sie richtig einsetzt. Ohne klare Regeln kann es passieren, dass man sich zu sehr auf die KI verlässt oder Inhalte übernimmt, ohne sie zu verstehen.

Dieser Leitfaden hilft dir, KI verantwortungsbewusst und reflektiert zu nutzen, damit du langfristig Fähigkeiten entwickelst, wie z.B. problemorientiertes Denken, Konzepte von Programmiersprache verstehst und somit nachhaltig programmieren lernst.

1) Risiken bei der KI-Nutzung:: Der Einsatz von KI Tools birgt einige Risiken, je nachdem wie du diese Tools nutzt. Die wohl offensichtlichsten lauten:

- **Copy & Paste ohne Verständnis:** Es ist verlockend, KI-Code einfach zu übernehmen – aber ohne echtes Verständnis lernst du wenig und es ist nicht nachhaltig. Das ist so ähnlich wenn man Google nutzt um etwas in einem Online-Lexikon nachzuschlagen: hilft uns kurzfristig, aber wir merken uns meist nur bedingt die Inhalte.
- **Abhängigkeit von KI:** Wenn du dich zu stark auf Vorschläge der KI verlässt, trainierst du dein eigenes Problemlösungsdenken nicht mehr. Die KI erledigt die größten Teil der "Denkarbeit", was eine *Illusion von Fortschritt* darstellt, obwohl du eigentlich nicht wirklich viel geleistet hast.
- **Datenschutz und Ethik:** KI-Systeme arbeiten mit großen Datenmengen -– achte darauf, keine sensiblen Informationen (z.B.: persönliche Daten, Daten von Dritten wie Studienkollegen, Unternehmensinformationen, Hochschulinterna, etc.) zu teilen.

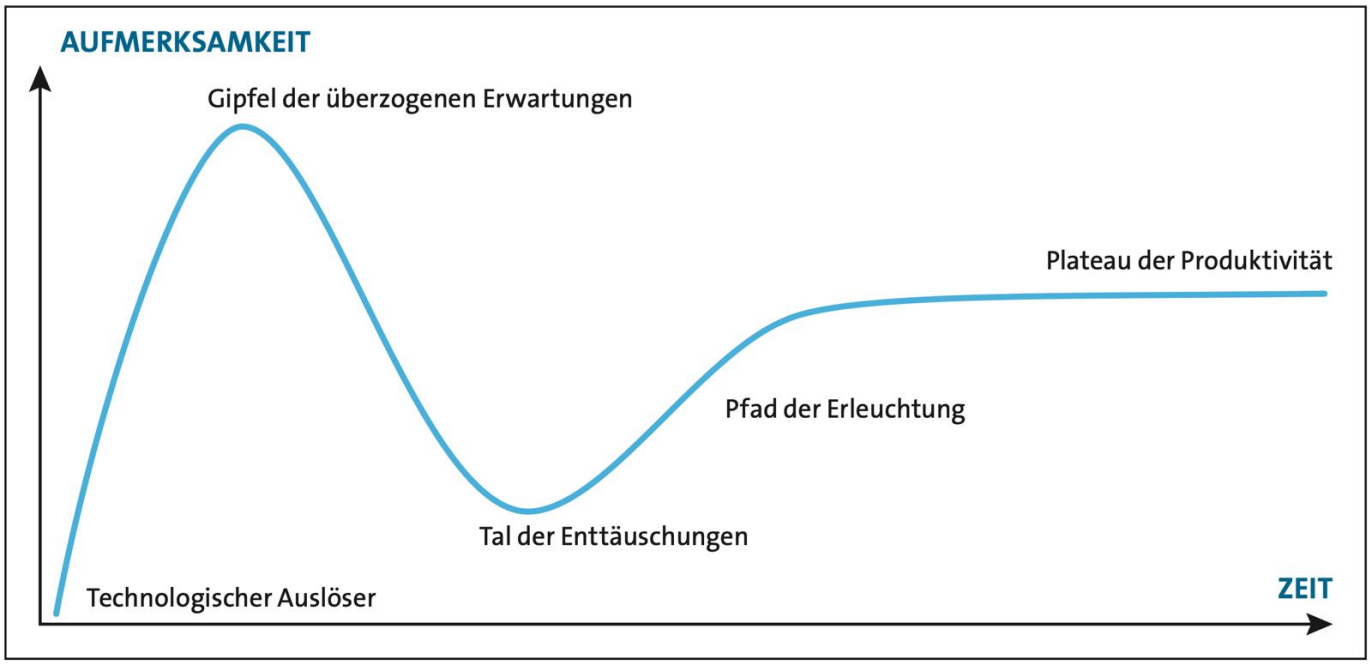

Abbildung 1. Verlauf der Nutzererwartung beim Einsatz von KI – angelehnt an den Hype–Zyklus *Quelle: [1]*

2) Chancen durch KI:: Der Einsatz von KI Tools verspricht folgende Chancen:

- **Codegenerierung:** KI Tools helfen bei der Generierung von Programmcode oder sogar größere Bestandteile eines Software-Projekts. Dies kann eine zeitliche und inhaltliche Unterstützung bedeuten.
- **Debugging:** KI Tools können gezielt genutzt werden um Fehler im Programmcode zu finden und Vorschläge zur Behebung machen.
- **Personalisierung:** KI kann Lerninhalte an dein Tempo und deinen Wissensstand anpassen und dir somit bei der Erreichung deiner Lernziele helfen.

C. Der typische Lernverlauf mit KI beim Programmieren

Wenn du beginnst, KI-Tools beim Programmieren zu nutzen, durchläufst du oft mehrere Phasen. Man spricht auch oft von dem sogenannten *Hypecycle*. Abbildung 1 stellt die Phasen bildhaft dar:

- Am Anfang ist die Begeisterung groß – plötzlich erscheint alles einfacher.
- Dann kommt die Ernüchterung, wenn du merkst, dass KI nicht alles kann
- Schließlich findest du eine gesunde Balance: Du nutzt KI als Unterstützung, aber nicht als Krücke

Sich diesem Zyklus bewusst zu sein hilft bereits den Einsatz eines Werkzeugs, wie ein KI Tool, besser einzuschätzen.

Merke: KI wird Programmiererinnen und Programmierer nicht zeitnah ersetzen – aber diejenigen, die KI klug nutzen, werden auf Dauer einen klaren Vorteil gegenüber jenen haben, welche die Tools falsch oder nicht anwenden.

II. Einführung in die KI

KI ist längst kein Zukunftsthema mehr – sie prägt heute, wie wir lernen, arbeiten und programmieren. Im Folgenden erhältst du einen verständlichen Überblick über die wichtigsten Grundlagen, Begriffe und Modelle, damit du weißt, was hinter Begriffen wie „Machine Learning“, „LLM“ oder „ChatGPT“ steckt und wie du sie sinnvoll beim Programmieren einsetzt.

A. Arten von KI-Modellen – und was sie für dich bedeuten

KI ist nicht gleich KI. Es gibt viele verschiedene Ansätze – manche sind auf spezielle Aufgaben trainiert, andere versuchen, allgemeiner zu „denken“. In Tabelle I siehst du eine vereinfachte Übersicht, die dir hilft, die wichtigsten Konzepte zu unterscheiden.

1) Nach Fähigkeiten: Von spezialisierter bis hypothetischer Intelligenz.: Die *Narrow AI* (auch „schwache KI") ist die heute dominierende Form. Sie ist darauf spezialisiert, klar abgegrenzte Aufgaben besonders gut zu lösen – etwa Text generieren, Bilder erkennen oder Sprache übersetzen. Beispiele sind ChatGPT, Siri oder GitHub Copilot. Ihre Stärke liegt in der hohen Präzision und Zuverlässigkeit innerhalb ihres Spezialgebiets. Ihre Schwäche: Sie besitzt kein echtes Verständnis über den Kontext hinaus und kann Wissen nicht flexibel übertragen.

Die *General AI* (auch „starke KI") strebt danach, menschenähnlich zu denken – also flexibel zu lernen, Wissen zu abstrahieren und komplexe Probleme in unterschiedlichen Domänen zu lösen. Solche Systeme existieren bislang nur in der Theorie. Ihre Entwicklung ist eines der großen Forschungsziele der KI, bringt aber zugleich ethische, sicherheitstechnische und gesellschaftliche Herausforderungen mit sich.

Die *Superintelligente KI* ist schließlich ein hypothetisches Konzept, das eine Intelligenz beschreibt, die menschliche

Tabelle I
KLASSIFIKATIONSANSÄTZE VON KÜNSTLICHER INTELLIGENZ.

Klassifikation	**Typ**	**Beschreibung**
Nach Fähigkeiten	Narrow AI (Schwache KI)	Kann nur bestimmte Aufgaben lösen (z. B. Chatbots, Codeassistenten)
	General AI (Starke KI)	Ziel: menschenähnliche Denkfähigkeit (noch Zukunftsmusik)
	Superintelligente KI	Hypothetisch: würde Menschen übertreffen
Nach Funktionalität	Reaktive Maschinen	Reagieren nur auf aktuelle Eingaben
	KI mit Gedächtnis	Nutzen vergangene Daten für bessere Ergebnisse
	Theory of Mind	Könnte Gefühle oder Absichten erkennen (aktuell theoretisch)
	Selbstbewusste KI	Gibt es (noch) nicht
Nach Technologien	Machine Learning (ML)	Systeme, die aus Daten lernen
	Deep Learning	Komplexe neuronale Netze mit vielen Schichten
	Natural Language Processing (NLP)	Versteht und verarbeitet menschliche Sprache
	Robotik	Intelligente Maschinensteuerung und Interaktion
	Computer Vision	Erkennt Objekte und Muster in Bildern oder Videos
	Expertensysteme	Treffen Entscheidungen auf Basis vordefinierter Regeln

Fähigkeiten in nahezu allen Bereichen übertrifft. Dieses Szenario wird oft in der Zukunftsforschung diskutiert, bleibt aber spekulativ. Ihre potenziellen Stärken – unbegrenzte Lernfähigkeit, Problemlösungskompetenz, Effizienz – sind zugleich ihre größten Risiken, wenn Kontrolle und Transparenz fehlen.

2) Nach Funktionalität: Wie KI denkt und lernt.: Diese Einteilung betrachtet, wie ein KI-System Informationen verarbeitet. *Reaktive Maschinen* wie frühe Schachcomputer (z.B.: IBM Deep Blue) handeln ausschließlich auf Basis aktueller Eingaben und können keine früheren Erfahrungen berücksichtigen. Sie sind zuverlässig und berechenbar, aber wenig anpassungsfähig.

KI mit Gedächtnis stellt eine Weiterentwicklung dar: Systeme lernen aus vergangenen Daten, um zukünftige Entscheidungen zu verbessern. Diese Kategorie umfasst die meisten modernen Machine-Learning-Modelle. Ihr Vorteil liegt in der Fähigkeit, Muster über die Zeit zu erkennen; ihre Schwäche ist jedoch die starke Abhängigkeit von der Qualität und Repräsentativität der Trainingsdaten.

Theory of Mind-Modelle stellen einen theoretischen Zwischenschritt dar. Sie sollen nicht nur Daten verarbeiten, sondern auch Emotionen, Absichten und soziale Dynamiken verstehen. Aktuell existieren hierzu nur Forschungsprototypen. Eine voll funktionsfähige *selbstbewusste KI*, die über ein eigenes Bewusstsein oder Selbstverständnis verfügt, gibt es hingegen noch nicht. Beide Konzepte verdeutlichen, wie weitreichend die Vision von KI über heutige Anwendungen hinausgeht.

3) Nach Technologien: Die Bausteine moderner KI-Systeme.: Die technologische Klassifikation beschreibt, auf welchen Prinzipien KI-Systeme beruhen. *Machine Learning (ML)* bildet dabei das Fundament vieler Anwendungen. Es nutzt statistische Verfahren, um Muster in Daten zu erkennen. ML ist besonders effektiv, wenn große Datenmengen vorliegen, kann aber anfällig für Verzerrungen (Bias) sein.

Diese Kategorisierung hilft dir, KI-Systeme besser zu verstehen – von einfachen Sprachassistenten bis hin zu komplexen lernenden Modellen. Die meisten heute eingesetzten Systeme, auch ChatGPT oder GitHub Copilot, gehören zur sogenannten *Narrow AI*.

B. Wichtige Teilgebiete der KI

KI besteht aus vielen Teilgebieten, die sich gegenseitig ergänzen. Einige konzentrieren sich auf das Lernen aus Daten, andere auf Sprache, Bilder oder Entscheidungsfindung. Die folgende Tabelle gibt dir einen Überblick, wofür sie stehen und wo du ihnen im Alltag begegnest.

Tabelle II
ÜBERBLICK ÜBER ZENTRALE TEILGEBIETE DER KÜNSTLICHEN INTELLIGENZ

Teilgebiet	**Beschreibung**	**Typische Anwendungen**
Maschinelles Lernen (ML)	Systeme lernen selbstständig aus Daten	Bilderkennung, Spamfilter, Vorhersagen
Deep Learning (ML)	Nutzt neuronale Netze, um komplexe Muster zu erkennen	Sprach- und Bilderkennung, Chatbots
Natural Language Processing (NLP)	KI versteht und erzeugt Sprache	Chatbots, Übersetzungen, Textanalyse
Expertensysteme	Treffen Entscheidungen auf Basis von Regeln	Medizin, Finanzen, Supportsysteme

Note: Viele moderne KI-Anwendungen, die du täglich nutzt – von Übersetzungsdiensten bis zu Empfehlungssystemen – basieren auf einer Kombination dieser Teilgebiete.

C. Large Language Models (LLMs)

Chatbots wie ChatGPT, Anthropic Claude oder Google Gemini basieren auf sogenannten *Large Language Models* (LLMs), also großen Sprachmodellen. Diese Modelle sind darauf trainiert, natürliche Sprache zu verstehen, Muster zu erkennen und neue Texte zu erzeugen. Grundlage sind Milliarden von Textbeispielen aus dem Internet, Büchern und anderen Quellen und typischerweise der Transformer–Ansatz **vaswani2017attention**. Aus technischer Sicht handelt es sich

um statistische, probabilistische Modelle: Sie schätzen aus dem bisherigen Kontext die Wahrscheinlichkeitsverteilung $p(w_t \mid w_{<t})$ für das nächste Token/Wort und generieren Ausgaben autoregressiv, also Token für Token. Die Ausgabe entsteht durch wiederholte Auswahl (deterministisch oder stochastisch) des nächsten Tokens. In anderen Worten: Solche Modelle schätzen das nächste Wort in einem Satz basierend auf dem bisherigen generierten Text und den Informationen, welche Sie von dem Nutzer als Eingabe erhalten haben.

D. Übersicht über aktuelle LLMs

Nachfolgend findest du eine Übersicht der wohl am meisten genutzten LLMs zur heutigen Zeit. Es kommen regelmäßig neue dazu – also keine Garantie auf Vollständigkeit.

1) GPT (Generative Pre-Trained Transformer): Das Generative Pre-trained Transformer, bekannt als GPT, ist ein von OpenAI entwickeltes Modell, eignet sich hervorragend für Anfänger im Programmieren. Es bietet präzise Erklärungen, kann Programmiercode generieren, analysieren und verbessern [2]. Das Modell wurde seit der Veröffentlichung im November 2022 kontinuierlich weiterentwickelt, wobei jedes neue Modell neue Funktionen und verbesserte Leistung bietet. In der kostenlosen Version sind die kompakteren Modelle wie GPT-4o mini und GPT-4.1 mini enthalten, wobei die Nutzung in dieser Version oft Einschränkungen hinsichtlich der Anzahl der Anfragen oder der unterstützten Funktionen haben kann.

Darüber hinaus gibt es noch zwei weitere kostenpflichtige Pakete [3]:

- **Plus-Paket:** Bietet erweiterten Zugang zu leistungsfähigeren Modellen wie GPT-4 und GPT-4.1 mit zusätzlichen Funktionen und ohne die Einschränkungen der kostenlosen Version.
- **Pro-Paket:** Ermöglicht den Zugriff auf die neuesten und leistungsfähigsten Modelle, darunter GPT-4o und GPT-4.1, mit maximaler Funktionalität und Priorität bei der Nutzung.

Die Tabelle III zeigt dir eine Übersicht der aktuellen Modelle mit dem Datum der Veröffentlichung, einer kurzen Beschreibung und in welchem Paket sie enthalten sind.

Tabelle III
ÜBERSICHT DER GPT-MODELLE

Modellname	**Datum**	**Stichwörter**	**Verfügbar im Preisplan**
GPT-4	März 2023	Multimodal, Text & Bild	Plus, Pro, Team, Enterprise
GPT-4o	Mai 2024	Sprache, Bild, Audio	Plus, Pro, Team, Enterprise
GPT-4o mini	Juli 2024	Schnell, günstig	Kostenlos, Plus, Pro, Team, Enterprise
GPT-4.1	April 2025	Verbesserte Programmier-hilfe	Plus, Pro, Team, Enterprise
GPT-4.1 mini	Mai 2025	Kompakt und effizient	Kostenlos, Plus, Pro, Team, Enterprise
GPT-5	August 2025	Zusammenschluss von "Denken-denünd Nicht Denken-denModellen	Kostenlos, Plus, Pro, Team, Enterprise

2) Anthropic's Claude: Claude ist ein fortschrittliches KI-Modell von Anthropic, das mit dem Ziel entwickelt wurde, verantwortungsbewusste und ethische KI-Systeme zu schaffen. Claude zeichnet sich durch seine Fähigkeit aus, in Bildungseinrichtungen und anderen Bereichen als unterstützendes Werkzeug zu fungieren, wobei besonderer Wert auf Datenschutz und akademische Integrität gelegt wird. So werden beispielsweise keine Studierendendaten verwendet, um die generativen Modelle zu trainieren [4].

Eine umfassende Analyse von über einer halben Million anonymisierter Gespräche von Studierenden mit Claude hat gezeigt, dass die KI hauptsächlich in den Bereichen Erstellung und Verbesserung von Lernmaterialien (39,3% der Gespräche) sowie technische Erklärungen und Problemlösungen (33,5%) eingesetzt wird. Dazu gehören das Erstellen von Übungsfragen, das Bearbeiten von Essays, das Zusammenfassen akademischer Inhalte sowie das Debuggen von Programmieraufgaben und das Lösen mathematischer Probleme [5], [6].

Im Februar 2025 wurde Claude 3.7 Sonnet vorgestellt. Die Besonderheit: Benutzer können wählen, ob sie eine schnelle Antwort oder eine Version bevorzugen, bei der das Modell länger nachdenkt und Schritt-für-Schritt Lösungen ausgibt, bevor es eine Antwort liefert [6], [7]. Mit jeder neuen Version von Claude steigen die Fähigkeiten des Modells komplexe Programmieraufgaben und Programmcode zu erzeugen.

3) GitHub Copilot: GitHub Copilot wurde von Microsoft entwickelt und sollte als eine Erweiterung des IntelliSense-Konzepts dienen. InelliSense ist eine integrierte KI, deren Vorschläge sich auf Namen von Methoden oder Eigenschaften beschränkt. GitHub Copilot kann Vorschläge für Schleifen, Bedingungen oder ganze Methoden übernehmen [1]. Zum

Beispiel kann bei der Erstellung einer neuen Funktion der Code basierend auf dem Rest der Datei erstellt werden. Copilot hat auch eine Chat-Integration, mit der direkt in der IDE kommuniziert werden kann.

Es gibt ein paar Tipps und Tricks, wie GitHub Copilot bessere Vorschläge liefern kann. Die Funktionen sollten möglichst aussagekräftige Namen haben. Eventuell ist auch ein Kommentar vor der Funktion, der kurz beschriebt, was die Funktion machen soll, sinnvoll. Umgekehrt kann GitHub Copilot auch bei der Erstellung von Kommentaren helfen. Falls es weitere relevante Dateien gibt, die für das Projekt relevant sind, sollten diese auch geöffnet sein, damit sie von dem Copilot berücksichtigt werden können [1].

GitHub Copilot hat drei verschiedene Preispakete, die jeweils mit verschiedenen Modellen arbeiten, wie zum Beispiel Anthropic Claude 3.5 Sonnet, OpenAi GPT-4.1 oder die neueren Modelle von Antropic's Claude oder OpenAI GPT Modelle. Das kostenlose Paket enthält bis zu 50 Chat-Interaktionen im Monat und Zugriff zu den Modellen von Anthropic Claude, Google Gemini, OpenAI GPTs. GitHub Copilot kann in mehrere IDEs integriert werden, wie Visual Studio Code, Visual Studio, JetBrainsIDE, Eclipse, Xcode oder Azure Data Studio [8].

4) Weitere Modelle und Werkzeuge im Programmierkontext: Neben den großen multimodalen Sprachmodellen wie GPT oder Claude gibt es eine wachsende Zahl spezialisierter KI-Modelle und Entwicklungsumgebungen, die gezielt auf das Programmieren ausgerichtet sind.

Cursor ist eine moderne, KI-gestützte Entwicklungsumgebung, die auf OpenAI- und Anthropic-Modellen basiert, jedoch stark auf den *Entwicklungsworkflow* optimiert wurde. Cursor bietet kontextbezogene Codevorschläge, Refactoring-Hilfen und ermöglicht es, direkt im Editor mit einem Chatbot über Codeänderungen zu diskutieren. Besonders für Lernende bietet dies einen intuitiven Zugang, da KI-Vorschläge im direkten Arbeitskontext sichtbar sind [9].

Gemini Code, entwickelt von Google DeepMind, kombiniert Sprachverständnis mit multimodalen Eingaben und ist über die Plattform *Google Colab* oder in IDEs wie *VS Code* verfügbar. Es bietet Lernenden interaktive Unterstützung durch kontextuelle Erklärungen, Testgenerierung und Codekommentierung [10].

Code Llama (Meta AI) ist ein auf Programmieraufgaben spezialisiertes Derivat von LLaMA, das frei verfügbar ist und in vielen Open-Source-IDE-Integrationen genutzt wird. Es ist besonders interessant für Lehrzwecke, da es lokal betrieben werden kann und damit auch *datenschutzfreundliche Szenarien* im Bildungsumfeld erlaubt [11].

StarCoder und **SantaCoder** (Hugging Face und ServiceNow) sind offene Alternativen mit Fokus auf reproduzierbare Forschung und Ausbildung. Sie unterstützen zahlreiche Programmiersprachen und können in universitären Laboren selbst gehostet werden, um Studierenden den Aufbau eigener Codemodelle zu demonstrieren [12].

Kommerzielle Assistenten wie **Amazon CodeWhisperer** oder **Tabnine** setzen dagegen auf die Integration in bestehende Unternehmens-IDE-Ökosysteme. Sie bieten stabile, produktionsnahe Unterstützung, sind aber weniger transparent in Bezug auf Trainingsdaten und Modellarchitektur [13], [14].

Zusammengefasst unterscheiden sich die Systeme vor allem in *Zugänglichkeit, Spezialisierungsgrad und Transparenz*:

- **Proprietäre Systeme** (z.B.: Copilot, CodeWhisperer, Gemini) sind leistungsstark, aber kostenpflichtig und weniger offen.
- **Offene Modelle** (z.B.: Code Llama, StarCoder) sind ideal für Lehre, Forschung und Datenschutz-Szenarien.
- **Integrierte Umgebungen** wie Cursor oder Replit AI schaffen neue Lernparadigmen, bei denen KI direkt zum Bestandteil des Programmierprozesses wird.

III. Einsatzmöglichkeiten von KI beim Programmieren

Generative KI-Tools wie ChatGPT, Claude oder GitHub Copilot bieten vielfältige Unterstützungsmöglichkeiten im Programmierlernprozess. Dabei kommt es darauf an, für was und wie du die Werkzeuge einsetzt. Die nachfolgenden Abschnitte geben dir einen Überblick über typische Anwendungsbereiche, konkrete Hinweise für die Nutzung und zeigen auf, worauf du beim Formulieren von Prompts, also den Befehlen und Instruktionen zwischen dir und der KI, achten solltest.

A. Typische Anwendungsbereiche

Wie in Kapitel I bereits beschrieben können KI-Systeme, im Kontext der Programmierung, in folgenden Situationen zum Einsatz kommen:

- **Codegenerierung:** Erstellung von Programmcode wie Funktionen, Codeblöcken oder sogar größere, komplexere Codepassagen und -dateien auf Basis einer Aufgabenbeschreibung.
- **Debugging:** Analyse und Korrektur von Fehlermeldungen oder fehlerhaftem Verhalten.
- **Konzepterklärung:** Erläuterung von Programmierkonzepten und Zusammenhängen wie beispielsweise Arrays, Schleifen oder Zeigern.
- **Syntaxhilfe:** Nachschlagen von Befehlen, Parametern oder Funktionssignaturen.
- **Strukturvorschläge:** Unterstützung beim Aufbau einer Programmstruktur oder Planung des Lösungswegs.
- **Fehlersuche und -behebung:** Identifikation und Lösung typischer Laufzeit- oder Kompilierfehler.

B. Hinweise zur effektiven Nutzung

Im Umgang mit KI-Tools sollten ein paar Dinge beachtet werden, welche das Ergebnis positiv oder negativ beinflussen können. Bei der Interaktion mit dem KI-Tool, das sogenannte Prompting, sind folgende Aspekte für ein positives Ergebnis zu beachten.

- **Kontext geben:** Beschreibe den Kontext in der du mit der KI interagierst, z.B.: Welche Programmiersprache nutzt du? Was genau ist dein Ziel mit der Interaktion:

Unterstützung bei der Problemlösung, Fehleranalyse, geforderte Erklärung, etc.? Du solltest beschreiben, was du von der KI erwartest und in welcher Rolle du die KI bei deiner Interaktion siehst (z.B.: Erklärhilfe, Tutor, etc.). Je genauer du den Kontext beschreibst, desto besser kann dir die KI helfen.
- **Konkret statt allgemein:** Je klarer deine gesamte Anfrage ist, desto hilfreicher wird die Antwort sein. Wenn du nur grob beschreibst was du möchtest kann das Ergebnis weniger hilfreich sein, ähnlich wie bei einer Google Suche.
- **Fehlermeldung einfügen:** Beim Debugging möglichst den exakten Fehlertext (evtl. die Fehlermeldung kopieren) inklusive dem betroffenen Code mitliefern. Je mehr Informationen die KI hat, desto besser kann sie dich bei der Fehleranalyse und Fehlerkorrektur beraten.
- **Ziel formulieren:** Beschreibe im Detail wie du dir die Rückmeldung der KI vorstellst, z.B.: ob eine Erklärung, Verbesserung oder ein Beispiel gewünscht wird.

C. Warnhinweis: Kein Ersatz für eigenes Denken

Die vorgestellten Möglichkeiten machen deutlich, dass KI-Tools eine hilfreiche Ergänzung sein können, sie ersetzen aber nicht den aktiven Lernprozess. Ein bloßes Übernehmen von Lösungen ohne Reflexion führt langfristig nicht zum Kompetenzerwerb. Die eigene Auseinandersetzung mit Problem, Lösungsweg und Funktionalität des Codes bleibt zentral.

IV. Hinweise für die KI-Nutzung beim Programmieren

Bevor mit KI-Tools gearbeitet wird, sollten sich Nutzer einige grundlegende Hinweise bewusst machen und die Fähigkeiten der KI realistisch einschätzen, um die Gefahr fehlerhafter oder irreführender Ergebnisse zu minimieren.

A. KI kann halluzinieren

KI-gestützte Entwicklungswerkzeuge können beeindruckende Unterstützung leisten, sie liefern oft schnelle, gut formulierte Lösungen für alltägliche Programmieraufgaben. Doch sie haben auch ihre Schwächen: Gerade bei weniger bekannten Programmiersprachen, neuen Bibliotheken oder sehr spezifischen Fragestellungen treten häufig Fehler auf. Besonders problematisch ist, dass KI-Tools auch falsche Antworten selbstsicher und sprachlich überzeugend präsentieren. Da sie bei Unsicherheiten oft „halluzinieren“[1], also plausibel klingende, aber erfundene Informationen liefern, ist kritisches Hinterfragen unerlässlich [1].

B. KI-Ergebnisse sind nicht reproduzierbar

Selbst bei identischer Eingabe eines Prompts [2] und gleicher Modellversion kann die KI bei jedem Ausführen unterschiedliche Ergebnisse erzeugen. Diese Outputs können in ihrer Qualität variieren, mal sind sie hilfreicher, mal weniger präzise [1].

[1] In der KI bezeichnet „halluzinieren“ die Erzeugung von plausibel klingenden, aber faktisch falschen oder erfundenen Inhalten [15].

[2] Ein Prompt ist eine gezielte Eingabe, z.B. eine Anweisung, mit der ein KI-System zu einer bestimmten Reaktion oder Ausgabe angeregt wird [16].

C. Umgangssprache ist auch ok

In dieser Richtlinie werden wir uns damit beschäftigen, wie der Umgang mit KI am besten gelingt und wie am besten mit der KI kommuniziert werden kann. Prinzipiell ist es wichtig, dass die KI klar formulierte Anweisungen bekommt, damit sie weiß, was der Nutzer konkret möchte. Allerdings ist es auch in Ordnung, wenn die Prompts umgangssprachlich beschrieben werden und klar hervorgeht, was die KI tun soll [1].

D. Verantwortung

Der Nutzer ist alleine verantwortlich was mit dem Output der KI passiert. Informationen sollten immer kritisch hinterfragt und analysiert werden bevor sie übernommen oder weiterverwendet werden [1].

V. Prompt Engineering

Prompt Engineering ist ein Feld, das sich mit der Erstellung, Optimierung und Verfeinerung von Prompts beschäftigt [17]. Prompts bilden die Basis der Interaktion zwischen Mensch und KI [18] und lassen sich einfach als konkrete Anweisungen oder Fragen definieren, die einem Sprachmodell gegeben werden, um dessen Verhalten zu steuern und gewünschte Ausgaben zu erzeugen [19] (z.B.: in Form eines Textes in einem Chatfenster). Die Qualität der Outputs, die von KI generiert werden, hängt stark von der Formulierung des Prompts ab, das der Nutzer eingibt [17].

A. Die Bestandteile eines Prompts

Ein effektiver Prompt besteht typischerweise aus vier Komponenten [19], die in Folgendem vorgestellt werden:

- **Anweisung *Instruction*:** Die Anweisung beschreibt die konkrete Aufgabe, die das Modell erfüllen soll. Sie steuert das Verhalten des Modells auf das gewünschte Ergebnis.
- **Kontext *Context*:** Der Kontext liefert dem Modell zusätzliche Hintergrundinformationen oder Rahmenbedingungen, die für die Bearbeitung der Aufgabe relevant sind. Kontext hilft dem Modell, die Aufgabe besser einzuordnen und passendere Antworten zu generieren. Dazu zählen zum Beispiel Definitionen, Beispiele, Informationen zu der Zielgruppe oder thematische Einschränkungen.
- **Eingabedaten *Input data*.** Die Eingabedaten sind der konkrete Inhalt, den das Modell verarbeiten soll. Sie stehen im Zentrum des Prompts und definieren, worauf sich die Antwort beziehen soll. Inputdaten können beispielsweise. eine Frage, ein Textabschnitt, ein Code-Snippet oder eine Problemstellung sein.
- **Output-Indikator *Output indicator*:** Der Output-Indikator beschreibt, in welcher Form das Ergebnis geliefert werden soll. Er legt fest, ob eine z.B. kurze Antwort, eine ausführliche Erläuterung, eine Liste, eine Tabelle oder ein bestimmter Tonfall gewünscht ist.

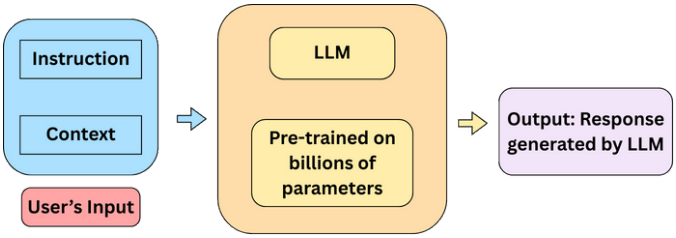

Abbildung 2. Die Bestandteile eines Prompts - Quelle: [20]

B. Verfahren in Prompt Engineering

Im Bereich des Prompt Engineerings sind zahlreiche Methoden bekannt, um das Verhalten vortrainierter Modelle für konkrete Aufgaben gezielt zu steuern [20]. Eine Studie von Sahoo P. et al. [20] aus dem Jahr 2024 präsentiert eine Übersicht von mehr als 29 Verfahren, die nach verschiedenen Kategorien aufgeteilt sind. In Folgendem wird eine Auswahl dieser Verfahren vorgestellt und jeweils an einem Beispiel kurz erklärt.

1) Zero-Shot Prompting: Bei dieser Technik erhält das Modell lediglich eine Aufgabenbeschreibung – ohne zusätzliche Beispiele. Es löst die Aufgabe allein durch sein im Vortraining erlerntes Wissen über Sprache, Bedeutung und logisches Denken [18], [21]. Im Unterschied zu klassischen ML-Methoden, die auf Datensätzen mit vorgegebenen richtigen Antworten (sogenannten Labels) angewiesen sind, nutzt Zero-Shot Prompting die Verallgemeinerungsfähigkeit großer Sprachmodelle. So können Aufgaben wie Textklassifikation, logisches Schließen oder Textgenerierung ohne domänenspezifisches Training gelöst werden [22].

Die Qualität der Ausgabe hängt jedoch stark von der Klarheit und Struktur des Prompts ab. Unpräzise Anweisungen führen leicht zu falschen oder unpassenden Ergebnissen [23], daher ist ein gut gestalteter Prompt entscheidend [24].

Abbildung 3 zeigt ein Beispiel für ein Zero-Shot Prompting mit der Antwort der KI. So ist zu sehen, dass der KI kein Beispiel gegeben wurde, sondern nur die Anweisung.

Abbildung 3. Beispiel für ein Zero-Shot Prompting

2) Few-Shot Prompting: Beim Few-Shot Prompting werden dem KI-Modell wenige ausgewählte Eingabe-Ausgabe-Beispiele als Teil der Eingabe mitgegeben. Diese dienen dazu, die Struktur und Anforderungen einer Aufgabe zu verdeutlichen und so das Antwortverhalten des Modells gezielt zu steuern [18], [20]. Im Gegensatz zum Zero-Shot Prompting, bei dem keine Beispiele verwendet werden, hat sich gezeigt, dass selbst wenige hochwertige Demonstrationen die Leistung des Modells bei komplexen Aufgaben deutlich verbessern können [22].

Ein Nachteil des Few-Shot Prompting liegt jedoch im erhöhten Platzbedarf: Die Beispieldaten müssen innerhalb des Prompts eingebettet werden, was je nach Länge der Beispiele und des eigentlichen Texteingabe zu Einschränkungen führen kann, insbesondere bei Modellen mit begrenztem Kontextfenster [20].

Abbildung 4 zeigt ein Beispiel für ein Zero-Shot Prompting mit der Antwort der KI. So ist zu sehen, dass der KI zwei Beispiele gegeben wurden, anhand deren das Modell Struktur und Stil erkannt hat.

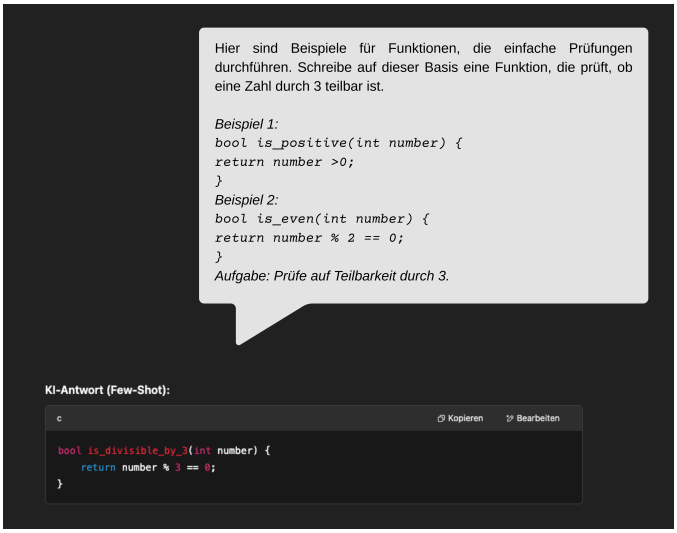

Abbildung 4. Beispiel für ein Few-Shot Prompting

3) Chain-of-Thought (CoT): Dieses Prompting wurde für Aufgaben eingeführt, die logisches oder mehrstufiges Denken erfordern, etwa bei Mathematik- oder Alltagslogik-Problemen, um große Sprachmodelle zu strukturierterem und schrittweisem Denken anzuregen. Die Vorteile bei diesem Verfahren sind, dass ein großes Problem in mehrere kleinere Probleme aufgeteilt wird. Dadurch kann mehr Rechenleistung für Aufgaben verwendet werden, die komplexeres Denken erfordern. Zweitens bietet eine „Denkkette“ einen Einblick in das Verhalten des Modells und kann zeigen, wie dieses zu einer bestimmten Antwort gekommen ist. So können Fehler im Denkprozess erkannt und korrigiert werden [20], [25].

Abbildung 5 zeigt ein Beispiel für ein Chain-of-Thought Prompting mit der Antwort der KI. Die Schritte im Kommentar zeigen den Denkprozess der KI.

4) Scratchpad Prompting: Scratchpad Prompting ist ein Ansatz, der dem Modell erlaubt, bei komplexen, mehrschrittigen Aufgaben Notizen zu machen, bevor es zur

Abbildung 5. Beispiel für ein Chain-of-Thought Prompting

endgültigen Lösung kommt. Anstatt das Modell sofort nach der finalen Antwort zu fragen, wird es dazu angeregt, seine Zwischenschritte explizit niederzuschreiben [20], ähnlich wie Menschen beim Rechnen einen Notizblock („Scratchpad“) verwenden. Diese Technik erhöht die Nachvollziehbarkeit der Ergebnisse und reduziert Fehler, insbesondere bei algorithmischen oder mathematischen Fragestellungen [26].

Abbildung 6 zeigt ein Beispiel für ein Scratchpad Prompting mit der Antwort der KI. Die Ausgabe von Zwischenschritten und erklärende Kommentare simulieren ein „Scratchpad“.

Abbildung 6. Beispiel für ein Scratchpad Prompting

C. Herausforderungen in Prompt Engineering

Trotz zahlreicher Verfahren im Prompt Engineering gibt es weiterhin eine Reihe an Herausforderungen im Umgang mit großen SPrachmodellen. Prompts spielen eine wichtige Rolle bei der Steuerung von KI, doch sie sind anfällig für verschiedene Fallstricke: Mehrdeutige Formulierungen, Überanpassung an Trainingsdaten (Overfitting) oder unrealistische Erwartungen. Es ist wichtig, dass Nutzer sich dieser Herausforderungen bewusst sind und sie aktiv in die Gestaltung von Prompts einbeziehen. In Folgendem werden zentrale Herausforderungen des Prompt Engineerings mit einem dazugehörigen Beispiel vorgestellt [19], [20].

1) Mehrdeutigkeit (Ambiguity): Ein häufiges Problem im Prompt Engineering ist die Ambiguität, also die Mehrdeutigkeit oder Unklarheit eines Prompts. Wenn Anweisungen zu allgemein oder unspezifisch formuliert sind, führt dies oft zu oberflächlichen oder unpräzisen Antworten seitens der KI [19], [20].

Abbildung 7 zeigt ein Beispiel für einen unpräzisen Prompt.Dieser Prompt lässt offen, welche Art von Liste gemeint ist (z.B. Array oder verkettete Liste), welche Daten verarbeitet werden sollen (z.B. Ganzzahlen, Zeichenketten), und was „verarbeiten“ konkret bedeutet (z.B. sortieren, durchsuchen, berechnen).

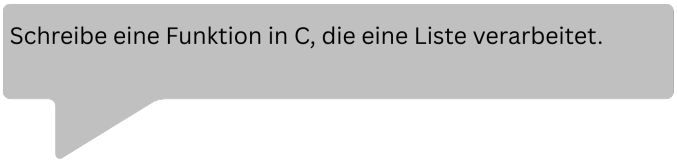

Abbildung 7. Beispiel für einen schlechten Prompt zu Mehrdeutigkeit

Abbildung 8 zeigt einen überarbeiteten, besseren Prompt. Die überarbeitete Version gibt klare Anweisungen zum Datentyp, zur konkreten Aufgabe und zur erwarteten Ausgabe. Das Sprachmodell kann dadurch eine passende und funktionale Lösung generieren.

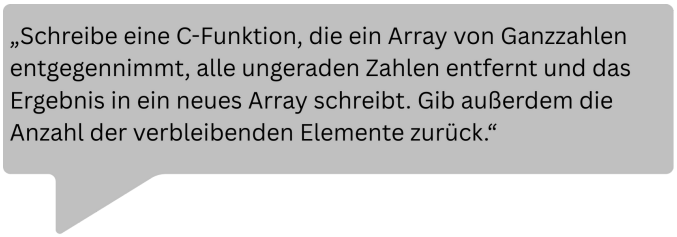

Abbildung 8. Beispiel für einen überarbeiteten Prompt zu Mehrdeutigkeit

2) Überanpassung Overfitting: Ein weiteres Problem im Prompt Engineering ist das sogenannte Overfitting – also die Überanpassung eines Prompts an ein ganz bestimmtes Beispiel. Ein Prompt kann so spezifisch sein, dass die KI zwar eine passende Antwort generiert, diese aber nur auf genau dieses Beispiel zutrifft und nicht auf ähnliche Fragestellungen übertragbar ist [19].

Abbildung 9 zeigt ein Beispiel für einen überangepassten Prompt. Dieser Prompt liefert zwar eine korrekte Funktion, aber die Aufgabe ist so stark auf eine ganz bestimmte

Zahlenfolge zugeschnitten, dass die Lösung für andere Fälle unbrauchbar ist. Die KI „lernt“ dadurch nichts über eine zugrundeliegende Logik oder ein allgemeineres Prinzip.

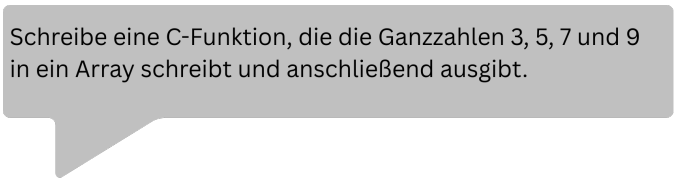

Abbildung 9. Beispiel für einen überangepassten Prompt

Abbildung 10 zeigt ein Beispiel für einen allgemein folrmulierten Prmpt. Dieser Prompt fordert dieselbe Funktionalität, erlaubt aber eine breitere Anwendung und fördert die Generalisierungsfähigkeit des Modells.

Ein zu stark auf ein Beispiel fokussierter Prompt kann zwar kurzfristig zur gewünschten Ausgabe führen, verhindert aber, dass das Modell flexibel oder kreativ auf ähnliche Aufgaben reagieren kann [19].

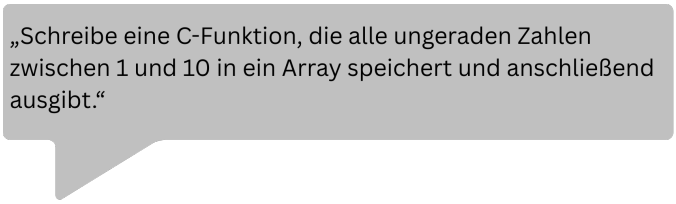

Abbildung 10. Beispiel für einen allgemein formulierten Prompt

3) Unrealistische Erwartungen an die Fähigkeiten der KI: Ein häufiger Fehler im Prompt Engineering ist die Annahme, dass KI-Modelle wie ChatGPT in der Lage seien, perfekte oder deterministisch richtige Lösungen für beliebige Aufgaben zu liefern. Sprachmodelle sind keine Allwissenssysteme – sie erzeugen Antworten basierend auf Mustern aus Trainingsdaten, nicht auf echtem Weltwissen oder deterministischer Logik. In diesem Zusammenhang kann es auch zu sogenannten Halluzinationen kommen – also zu scheinbar plausiblen, aber faktisch falschen oder erfundenen Ausgaben [19].

Abbildung 11 zeigt ein Beispiel für einen Prompt mit zu hohen Erwartungen an die KI. Dieser Prompt geht davon aus, dass die KI in der Lage ist, für beliebige Datenmengen und -verteilungen stets den optimalen Sortieralgorithmus auszuwählen.

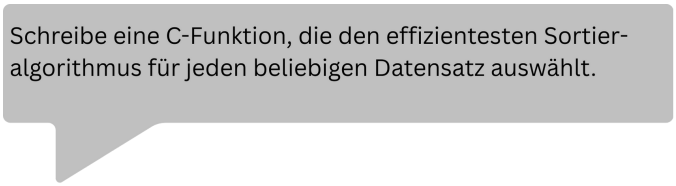

Abbildung 11. Beispiel für einen schlechten Prompt mit unrealistischen Erwartungen an die KI

Abbildung 12 zeigt die überarbeitete Anforderung an die KI. Durch diesen Prompt wird eine konkrete, realistisch umsetzbare Aufgabe gestellt, die das Modell sicher bewältigen kann.

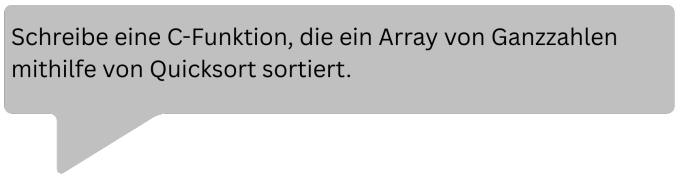

Abbildung 12. Beispiel für einen überarbeiteten Prompt mit realistischen Erwartungen an die KI

Statt überhöhte Erwartungen an das Modell zu stellen, sollten Prompts so formuliert werden, dass sie sich an den realistischen Fähigkeiten der KI orientieren. Zudem ist es wichtig, Antworten stets kritisch zu reflektieren, denn durch Halluzinationen kann die KI auch fehlerhafte oder erfundene Inhalte mit großer Überzeugung präsentieren. Fachliche Prüfung und zusätzliche Recherche bleiben daher unerlässlich, besonders bei technischen Anwendungen.

VI. Allgemeine Best Practices

In diesem Kapitel werden allgemeine praktische Tipps vorgestellt, die das Verständnis der KI für Anweisungen verbessern sollen.

1) **Deutsch vs. Englisch** Obwohl KI-Modelle wie ChatGPT auch deutschsprachige Prompts verstehen, empfiehlt es sich, Anfragen möglichst auf Englisch zu stellen. Der Grund: Ein Großteil des Trainingsmaterials ist englischsprachig, das erhöht die Wahrscheinlichkeit für präzisere und qualitativ bessere Antworten, insbesondere bei komplexen oder technischen Fragestellungen. Zudem entspricht Englisch dem Sprachstandard vieler Softwareprojekte und ist Voraussetzung für die Nutzung vieler anderer KI-Modelle, die nur eingeschränkt mit anderen Sprachen arbeiten [1].
2) **Zusatzinformationen** Vor der eigentlichen Frage sollte der fachliche Rahmen genannt werden – zum Beispiel verwendete Programmiersprache, Bibliotheken, Tools oder Entwicklungsumgebung. Dadurch kann die Antwort gezielter und passender ausfallen [1], [19].
3) **Nebenbedingungen definieren** Wichtige Zusatzinformationen sollten auch benannt werden, z.B. ob kompakter Code gewünscht ist, möglichst wenige Bibliotheken verwendet werden sollen, Sicherheitsaspekte zu beachten sind oder besondere Anforderungen an Effizienz oder Rekursion bestehen [1].
4) **Zweiter Versuch** Falls die KI beim ersten Versuch ein ganz anderes Ergebnis liefert als erwartet, oder auf der „falschen Spur„ liegt, kann ein zweiter Versuch helfen. Dafür einfach einen neuen Prompt probieren.
5) **Neue Session bei Themenwechsel** Da die Modelle sich die Informationen aus einem Chatverlauf merken, sollte bei einem neuen Thema eine neue Session begonnen werden, damit die alten Information nicht auf das neue Thema übertragen werden [1].

6) **Klarheit und Präzision** Je klarer und präziser ein Prompt formuliert ist, desto gezielter und relevanter ist die Antwort der KI. Allgemeine Anfragen führen oft zu unspezifischen Ergebnissen. Konkrete Angaben und Kontext reduzieren Interpretationsspielraum und verbessern die Qualität der Ausgabe [27].

VII. Fazit

Die Nutzung von KI-Tools, wie LLMs oder Coding Agenten, kann beim Programmierenlernen eine wertvolle Unterstützung sein, insbesondere beim Verstehen von Konzepten, beim Lösen konkreter Programmierprobleme und beim Reflektieren über eigene Lösungswege. Damit diese Potenziale sinnvoll ausgeschöpft werden, ist ein bewusster und reflektierter Umgang mit den KI-Ausgaben unerlässlich.

Dieser Praxisleitfaden zeigt auf, dass das Verständnis grundlegender Konzepte wie der Funktionsweise von LLMs, der Aufbau effektiver Prompts sowie das Wissen um die Stärken und Grenzen von KI entscheidend dafür sind, qualitativ hochwertige Ergebnisse zu erzielen. Gleichzeitig wird deutlich, dass KI-Systeme keine fehlerfreien Expertensysteme sind – sie halluzinieren, liefern nicht immer reproduzierbare Ergebnisse und ersetzen keine eigenständige Denk- oder Lernleistung.

Ziel sollte es daher sein, KI als Werkzeug zu begreifen: richtig eingesetzt, kann sie Denkprozesse anstoßen, Lösungswege vorschlagen und das Lernen effizienter gestalten. Falsch oder unkritisch genutzt, besteht dagegen die Gefahr von Missverständnissen, fehlerhaftem Code oder einer Überbewertung maschineller Antworten.

Durch die Beachtung der Ratschläge dieses Praxisleitfadens, die Definition klarer Prompts sowie die kritische Reflexion der Ergebnisse kann der Umgang mit KI beim Programmierenlernen nicht nur effektiver, sondern auch nachhaltiger gestaltet werden.

Literatur

[1] M. Kofler, B. Öggl und S. Springer, *Coding mit KI: das Praxisbuch für die Softwareentwicklung*, ger, 1. Auflage. Bonn: Rheinwerk Computing, 2025, ISBN: 978-3-367-10346-1.

[2] G. Yenduri u. a., *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*, en, Mai 2023. besucht am 18. Mai 2025. Adresse: https://arxiv.org/abs/2305.10435v2.

[3] *Models - OpenAI API*, de-DE. besucht am 18. Mai 2025. Adresse: https://platform.openai.com.

[4] *Claude for Education | Partnering with Universities on Responsible AI \ Anthropic*, en. besucht am 18. Mai 2025. Adresse: https://www.anthropic.com/education.

[5] *Anthropic Education Report: How University Students Use Claude*, en. besucht am 18. Mai 2025. Adresse: https://www.anthropic.com/news/anthropic-education-report-how-university-students-use-claude.

[6] *Claude 3.7 Sonnet System Card*, en. besucht am 18. Mai 2025. Adresse: https://www.semanticscholar.org/paper/Claude-3.7-Sonnet-System-Card/9ff93dfa8f445c932415d335c88852ef47f1201e.

[7] *Claude 3.7 Sonnet and Claude Code*, en. besucht am 18. Mai 2025. Adresse: https://www.anthropic.com/news/claude-3-7-sonnet.

[8] *Take flight with GitHub Copilot*. besucht am 19. Mai 2025. Adresse: https://github.com/features/copilot/plans.

[9] C. AI, *Cursor – The AI-Powered Code Editor*, https://www.cursor.com/, Accessed: 2025-10-06, 2025.

[10] G. DeepMind, *Introducing Gemini for Developers*, https://deepmind.google/technologies/gemini/, Accessed: 2025-10-06, 2025.

[11] M. AI, *Code Llama: Open Foundation Models for Code*, https://ai.meta.com/research/publications/code-llama/, Accessed: 2025-10-06, 2023.

[12] H. Face und S. Research, *StarCoder: A Large Language Model for Code*, https://huggingface.co/bigcode/starcoder, Accessed: 2025-10-06, 2023.

[13] A. W. Services, *Amazon CodeWhisperer – AI Coding Companion*, https://aws.amazon.com/codewhisperer/, Accessed: 2025-10-06, 2024.

[14] T. Ltd., *Tabnine – The AI Assistant for Developers*, https://www.tabnine.com/, Accessed: 2025-10-06, 2024.

[15] D. J. Siebert, *Halluzinationen von generativer KI und großen Sprachmodellen (LLMs) - Blog des Fraunhofer IESE*, de, Sep. 2024. besucht am 17. Mai 2025. Adresse: https://www.iese.fraunhofer.de/blog/halluzinationen-generative-ki-llm/.

[16] S. Burgermeister, *Was ist Prompting | IU Akademie Blog*, de, Nov. 2023. besucht am 19. Mai 2025. Adresse: https://www.iu-akademie.de/blog/was-ist-prompting/.

[17] S. Arvidsson und J. Axell, „Prompt engineering guidelines for LLMs in Requirements Engineering," eng, Aug. 2023, Accepted: 2023-08-07T08:59:27Z. besucht am 16. Mai 2025. Adresse: https://gupea.ub.gu.se/handle/2077/77967.

[18] C. I. Chang, „A Survey of Techniques, Key Components, Strategies, Challenges, and Student Perspectives on Prompt Engineering for Large Language Models (LLMs) in Education," en, März 2025, Publisher: Preprints. DOI: 10.20944/preprints202503.1808.v1. besucht am 16. Mai 2025.

[19] L. Giray, „Prompt Engineering with ChatGPT: A Guide for Academic Writers," en, *Annals of Biomedical Engineering*, Jg. 51, Nr. 12, S. 2629–2633, Dez. 2023, ISSN: 1573-9686. DOI: 10.1007/s10439-023-03272-4. besucht am 16. Mai 2025.

[20] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal und A. Chadha, *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*, en, Feb. 2024. besucht am 17. Mai 2025. Adresse: https://arxiv.org/abs/2402.07927v2.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei und I. Sutskever, „Language models are unsupervised multitask learners," *OpenAI blog*, Jg. 1, Nr. 8, S. 9, 2019.

[22] T. Brown u. a., „Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F.

Balcan und H. Lin, Hrsg., Bd. 33, Curran Associates, Inc., 2020, S. 1877–1901. Adresse: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[23] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo und Y. Iwasawa, *Large Language Models are Zero-Shot Reasoners*, en, Mai 2022. besucht am 17. Mai 2025. Adresse: https://arxiv.org/abs/2205.11916v4.

[24] T. Debnath, M. N. A. Siddiky, M. E. Rahman, P. Das und A. K. Guha, „A Comprehensive Survey of Prompt Engineering Techniques in Large Language Models," *TechRxiv*, 2025.

[25] J. Wei u. a., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, en, Jan. 2022. besucht am 17. Mai 2025. Adresse: https://arxiv.org/abs/2201.11903v6.

[26] E. Zelikman, Y. Wu, J. Mu und N. Goodman, „STaR: Bootstrapping Reasoning With Reasoning," en, Okt. 2022. besucht am 17. Mai 2025. Adresse: https://openreview.net/forum?id=_3ELRdg2sgI.

[27] B. Chen, Z. Zhang, N. Langrené und S. Zhu, „Unleashing the potential of prompt engineering for large language models," English, *Patterns*, Jg. 0, Nr. 0, Mai 2025, Publisher: Elsevier, ISSN: 2666-3899. DOI: 10.1016/j.patter.2025.101260. besucht am 19. Mai 2025.