HAW HAMBURG

BACHELOR THESIS
Philipp Riefer

Verbesserung kontextbasierter KI-Tutoren im Hochschulbereich: Einsatz von Large Language Models zur Erkennung von Benutzerintentionen und Wissenslücken

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Philipp Riefer

Verbesserung kontextbasierter KI-Tutoren im Hochschulbereich: Einsatz von Large Language Models zur Erkennung von Benutzerintentionen und Wissenslücken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Paweł Buczek
Zweitgutachter: Dr. Marcel Völschow

Eingereicht am: 29. September 2025

Philipp Riefer

Thema der Arbeit

Verbesserung kontextbasierter KI-Tutoren im Hochschulbereich: Einsatz von Large Language Models zur Erkennung von Benutzerintentionen und Wissenslücken

Stichworte

Künstliche Intelligenz, Tutorensystem, Large Language Model, Bayesian Knowledge Tracing, Skill-Erkennung, adaptive Lernsysteme, Hochschulbildung

Kurzzusammenfassung

Diese Bachelorarbeit beschreibt die Entwicklung eines Prototyps für einen KI-basierten Tutor, der auf einem lokal betriebenen Large Language Model (LLM) aufsetzt und um eine automatische Skill-Erkennung sowie Bayesian Knowledge Tracing (BKT) erweitert wurde. Ziel war es, den Wissensstand von Studierenden während der Interaktion abzuschätzen und adaptiv darauf zu reagieren.

Philipp Riefer

Title of Thesis

Improving Contextual AI Tutors in Higher Education: Using Large Language Models for the Detection of User Intentions and Knowledge Gaps

Keywords

Artificial Intelligence, Tutoring System, Large Language Model, Bayesian Knowledge Tracing, Skill Detection, Adaptive Learning Systems, Higher Education

Abstract

This bachelor thesis presents the development of a prototype for an AI-based tutoring system built on a locally hosted Large Language Model (LLM) and extended with automatic skill detection and Bayesian Knowledge Tracing (BKT). The aim was to estimate students' knowledge states during interaction and adaptively respond to them.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Kaum ein anderer Meilenstein in der modernen Menschheitsgeschichte hatte einen größeren Einfluss auf Bildung, Arbeitswelt und allgemeine Gesellschaft zugleich als die Veröffentlichung von ChatGPT. Unter den vielseitigen Nutzungsmöglichkeiten des Large Language Models der Firma OpenAI findet sich auch die Verwendung durch Schüler*innen und Student*innen als Lernhilfe oder zum Lösen von Aufgaben.

Diese neue Form der Interaktion mit KI hat das Lernverhalten nachhaltig verändert: Während zuvor eigenständige Recherche und aktives Problemlösen zentrale Bestandteile des Lernprozesses waren, übernehmen heute KI-Systeme häufig große Teile dieser kognitiven Arbeit. Dies führt einerseits zu einer gesteigerten Verfügbarkeit von Wissen, andererseits bringt es die Gefahr einer zunehmenden kognitiven Abhängigkeit mit sich [42]. Ein erhöhter Verlass auf KI-generierte Antworten kann dazu führen, dass lernende Personen ihre eigenen Kompetenzen unterschätzen und weniger kritisch reflektieren [14].

Im Vergleich zu menschlichen Tutor*innen ist es für eine künstliche Intelligenz schwieriger, Rückschlüsse auf das tatsächliche Verständnisniveau der lernenden Person zu ziehen. Trotz der fehlenden Erkenntnisse über mögliche Missverständnisse oder unklare Intentionen der Benutzer*innen können formal korrekte Antworten generiert werden, allerdings verfehlen diese zumeist das Wissensniveau der Benutzer*innen, was Missverständnisse zunehmend vertiefen kann. Durch die zunehmende Nutzung von KI-Systemen zur automatisierten Lösung von Aufgaben ohne ein fundiertes Verständnis der zugrunde liegenden Theorie wächst die Kluft zwischen praktischer Anwendung und theoretischem Verständnis. Dies kann dazu führen, dass Student*innen zwar Lösungen reproduzieren, diese jedoch nicht eigenständig herleiten oder kritisch hinterfragen können [35].

Gleichzeitig ist der positive Einfluss von LLMs auf Lernleistung, Lernwahrnehmung und höhere Denkprozesse belegt. Eine kontrollierte Integration von LLMs in schulische und hochschulische Bildung wird daher, unter bestimmten Voraussetzungen, als förderlich betrachtet [38].

Die Bereitstellung solcher Systeme über nahezu ausschließlich kommerzielle Anbieter wie OpenAI, Meta oder Google wirft jedoch im europäischen Hochschulkontext Fragen des Datenschutzes und der digitalen Souveränität auf [23]. Eine datenschutzkonforme Alternative könnten lokal betriebene, hochschuleigene LLMs darstellen, unter der Voraussetzung, sie erreichen eine vergleichbare Qualität und Funktionalität wie ihre kommerziellen Pendants.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, die Interaktion zwischen Student*innen und KI-basierten Tutorensystemen näher zu untersuchen. Im Fokus steht dabei die Frage, wie ein KI-Tutor die Intention hinter studentischen Anfragen erkennen und mögliche Wissenslücken identifizieren kann, um dadurch individualisierte, lernförderliche Antworten zu erzeugen.

Die Analyse von wissenschaftlichen Arbeiten bezüglich der Wissenslücken- und Intentionserkennung steht hierbei im Vorderpunkt. Aufbauend darauf werden die in den Arbeiten beschriebenen Systeme rekonstruiert und prototypisch umgesetzt.

Ein weiteres Ziel der Arbeit ist es, die Auswirkungen dieser Funktionen auf die Qualität der Tutor-Antworten und die wahrgenommene Nützlichkeit durch die Nutzer*innen zu evaluieren. Dabei soll auch der Aspekt der digitalen Souveränität berücksichtigt werden: Die entwickelten Konzepte sollen explizit auf lokal ausgeführte, datenschutzkonforme KI-Modelle übertragbar sein, wie sie im Hochschulkontext zunehmend an Relevanz gewinnen.

1.3 Aufbau der Arbeit

Die vorliegende Bachelorarbeit gliedert sich in sieben Hauptkapitel. Nach der Einleitung in Kapitel 1, in welcher Motivation, Problemstellung und Zielsetzung der Arbeit erläutert werden, folgt in Kapitel 2 eine Darstellung des Hintergrunds. Dort werden zunächst

das Emmy-Projekt und dessen Rahmenbedingungen beschrieben, anschließend die Konzepte der Intentionserkennung und der Wissenslückenerkennung eingeführt und auf den Kontext von KI-Tutorensystemen übertragen.

Kapitel 3 stellt die theoretischen Grundlagen bereit. Es führt den Leser in die Begriffe der Künstlichen Intelligenz und Large Language Models ein, erläutert deren Funktionsweise, Grenzen und Potenziale sowie die Konzepte Retrieval-Augmented Generation (RAG) und Bayesian Knowledge Tracing (BKT) als zentrale Bausteine dieser Arbeit.

Auf dieser Basis werden in Kapitel 4 die Anforderungen an den zu entwickelnden Prototyp formuliert. Hier werden sowohl funktionale als auch nicht-funktionale Anforderungen definiert, der Verzicht auf eine Q-Matrix begründet und das Anforderungsmodell zusammengefasst.

Kapitel 5 beschreibt die Umsetzung des Prototyps. Es werden die Gesamtarchitektur, der lokale LLM-Server, die Gradio-Benutzeroberfläche, die Erweiterungen der RAG-Funktionalität, die automatische Skill-Erkennung sowie das BKT-Modul und der adaptive Antwortstil des Tutors detailliert dargestellt.

In Kapitel 6 folgt die Auswertung. Hier werden der entwickelte Benchmark erläutert, die erzielten Ergebnisse vorgestellt und im Hinblick auf die in Kapitel 1 formulierten Ziele interpretiert.

Den Abschluss bildet Kapitel 7 mit einer Zusammenfassung der Ergebnisse, der Diskussion von Limitationen und einem Ausblick auf mögliche Weiterentwicklungen des Systems.

Im Anhang schließlich sind verwendete Hilfsmittel, zusätzliche Abbildungen und der Quellcode des Projekts dokumentiert.

2 Hintergrund

2.1 Das Emmy Projekt

Im Oktober 2024 wurde ich von Prof. Völschow in das *Emmy* Projekt eingebunden, welches nach der Mathematikerin Emmy Noether benannt ist. Das Vorhaben wird von der Claussen-Simon-Stiftung finanziell unterstützt und ist Teil der Hamburg Open Online University (HOOU).

Ziel des Projekts ist die Entwicklung eines Chatbots, welcher auf der Plattform der HOOU Studierende bei der Bearbeitung von Onlinekursen unterstützt. Auf der Webseite der HAW Hamburg wird das Vorhaben von Prof. Völschow als „ein Jupyter-basiertes E-Learning-Portal für das Deep Learning“ beschrieben [36].

Das Projekt soll einen virtuellen Tutor bereitstellen, welcher den Lernenden ähnlich wie eine menschliche Lehrkraft Hilfestellung bieten kann. Ein zentrales Ziel besteht darin, Kontextbewusstsein zu ermöglichen, sodass der Chatbot erkennt, welchen Kurs eine Person aktuell bearbeitet, um darauf abgestimmte Unterstützung leisten zu können.

In der ersten Projektphase wurde die Entwicklung eines Chatbots mit Retrieval-Augmented Generation (RAG) angestrebt. Durch den Einsatz von RAG wird eine höhere Antwortqualität erwartet sowie eine erleichterte Bedienung, da Nutzende auf Details des Kurses und der zu behandelnden Aufgaben in ihren Prompts verzichten können.

Zu Beginn wurde ein RAG-basierter Chatbot mit einem lokal betriebenen Large Language Model (LLM) auf einem Rechner der HAW Hamburg implementiert. Um eine benutzerfreundliche Oberfläche zu schaffen und nicht ausschließlich terminalbasiert mit dem LLM zu interagieren, kam das Python-Framework *Gradio* (s. Abb. 2.1) zum Einsatz, mit welchem die ersten Prototypen einer Benutzeroberfläche erstellt wurden.

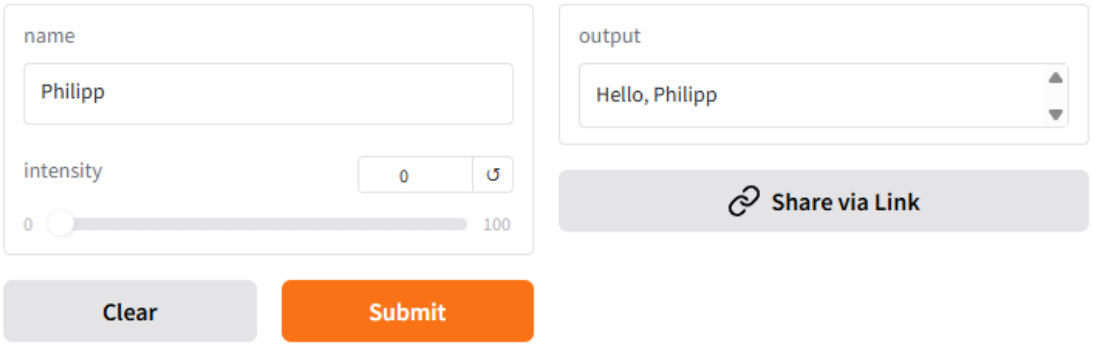


Abbildung 2.1: Beispiel einer mit dem Gradio Framework erstellten Grafischen Benutzeroberfläche [33]

Im Juli 2025 wurde ein erstes Feldexperiment durchgeführt. Im Rahmen einer Physik 1 Vorlesung, geleitet von Prof. Nourmofidi, wurde den 32 anwesenden Studierenden (Erstsemester Bachelor Elektro- und Informationstechnik) ein Übungstest vorgelegt. Die darin enthaltenen Aufgaben, alle drei im Bereich des Kurses Physik 1, sollten ohne Hilfsmittel bis auf Taschenrechner und den Emmy KI Tutor bearbeitet werden. Dieser hatte bereits die Aufgaben in einer Datenbank gespeichert, wodurch die Studierenden über ein Drop-Down-Menü die Aufgabe auswählen konnten, zu welcher Hilfestellung benötigt wurde, ähnlich zu Abbildung 2.1. Um eventuell auftretende technische Probleme zu behandeln waren die Mitarbeitenden des Emmy Projekts anwesend.

Im Anschluss an das Feldexperiment wurde unter den teilnehmenden Studierenden eine Befragung durchgeführt, um deren Nutzungserfahrungen sowie die Zufriedenheit mit dem Chatbot zu evaluieren. Von den 32 Teilnehmenden des Feldexperiments haben 17 Personen diese Befragung ausgefüllt.

Die Ergebnisse zeigen insgesamt ein gemischtes Bild. Der Chatbot wurde mit durchschnittlich 2,71 von 5 Sternen als nicht besonders hilfreich bewertet. Auch seine Unterstützung beim Verständnis grundlegender Konzepte von Aufgaben erhielt lediglich 3,5 von 5 Sternen. Das Vertrauen in die gemeinsam mit dem Chatbot erarbeiteten Lösungen lag bei 2,94 von 5 Sternen. Die Nachvollziehbarkeit der Erklärungen wurde mit 3,41 von 5 Sternen eingestuft, und hinsichtlich der Vollständigkeit der Antworten gaben nur 9 von 17 Personen an, dass alle relevanten Formeln und Gesetze genannt wurden. Entsprechend lag die Bewertung dieser Dimension bei 3,29 von 5 Sternen, während der Informationsgehalt der Antworten mit 3,24 von 5 Sternen beurteilt wurde.

Lediglich 7 der 17 Umfrageteilnehmenden hatten den Eindruck, dass der Chatbot die gestellten Prompts korrekt verstand. Zudem mussten 14 von 17 Teilnehmenden ihre Prompts mindestens einmal umformulieren, um zufriedenstellende Antworten zu erhalten. Lediglich drei Personen berichteten, dass keine Umformulierung notwendig war. Dieses Ergebnis führte zur Überlegung, eine Intentionserkennung mit automatisierter Promptumschreibung zu implementieren, um die Erwartungen der Nutzenden an die Antworten präziser erfassen zu können.

In den Freitextantworten wurde vor allem der Kontextverlust nach mehreren Prompts im Gespräch kritisiert. Dieses Problem wurde nachträglich, außerhalb des Rahmens der Bachelorarbeit, behoben. Eine weitere Rückmeldung betraf die Ausrichtung des Chatbots: es wurde angemerkt, dass der Chatbot „alles zu beantworten versucht und nicht korrigiert“, also komplette Lösungen ausgibt, anstatt sich auf den jeweils aktuellen Bearbeitungsschritt zu beschränken. Diese Rückmeldung gab den Impuls, eine Wissenslückenerkennung und damit zusammenhängend eine Wissensstandsverfolgung zu entwickeln, die es erlaubt, die Unterstützung gezielter an den latenten Wissensstand der Nutzenden anzupassen.

Die Ergebnisse der Umfrage sind im Anhang A.1 zu sehen.

2.2 Die Intentionserkennung

Die Ergebnisse des Feldexperiments führten zunächst zur Überlegung, eine gesonderte Intentionserkennung zu implementieren, um die Eingaben der Studierenden automatisch zu analysieren und gegebenenfalls zu reformulieren. Ziel dieser Maßnahme wäre es gewesen, Missverständnisse zwischen Chatbot und Nutzenden zu reduzieren und die erwartete Antwortqualität zu erhöhen.

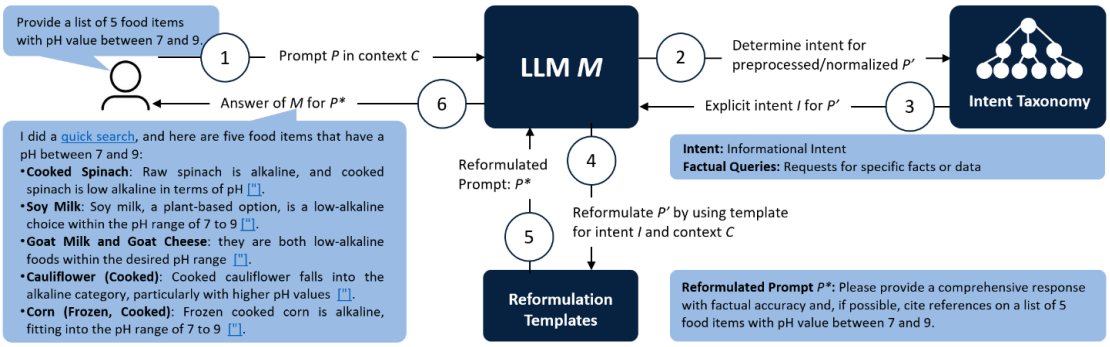


Abbildung 2.2: Übersicht über das vorgeschlagene Framework zur Prompt-Reformulierung von Bodonhelyi u. a. [7]

Aktuelle Forschungsergebnisse legen jedoch nahe, dass ein zusätzlicher Intent-Klassifikationsschritt bei modernen LLM-basierten Tutorsystemen nur eingeschränkt Nutzen bringt und teilweise sogar nachteilig sein kann. Cutler u. a. [12] untersuchten ein Chat-basiertes intelligentes Tutorsystem (ITS) mit mehr als 10.000 Studentennachrichten und entwickelten einen Intent-Klassifikator, der zwischen verschiedenen Handlungsabsichten unterscheiden sollte. Obwohl GPT-4o mit einem F1-Wert[1] von 0,77 das beste Ergebnis erzielte, lagen alle Ansätze nur im mittleren Genauigkeitsbereich und erforderten teilweise mehr als eine Sekunde zusätzliche Latenz pro Nachricht. Die Autorinnen und Autoren betonen, dass jedes dieser Modelle zahlreiche Fehlklassifikationen produziert, die das System dennoch abfangen muss. Gleichzeitig erschweren fehlende, kalibrierte Konfidenzwerte eine zuverlässige Nutzung im Live-Betrieb. Damit wird deutlich, dass eine separate Intent-Klassifikation nicht nur ressourcen- und zeitintensiv ist, sondern auch nur moderate Verbesserungen bringt und zusätzliche Dialogflüsse zur Bestätigung erfordert, um Fehlklassifikationen abzufangen.

[1] Das F1-Maß ist ein in der Informatik weit verbreitetes Gütemaß für Klassifikationssysteme. Es kombiniert *Precision* (Genauigkeit) und *Recall* (Vollständigkeit) zu ihrem harmonischen Mittel und erlaubt so eine ausgewogene Bewertung von Klassifikationsmodellen [4].

Bodonhelyi u. a. [7] führten eine groß angelegte Nutzungsstudie mit GPT-3.5 Turbo und GPT-4 Turbo durch, in der die Erkennung von feingranularen Benutzerintentionen und deren anschließende Prompt-Reformulierung evaluiert wurden. Sie entwickelten zunächst eine detaillierte Intent-Taxonomie mit 24 Kategorien (u.a. Informationssuche, Problemlösen, Lernunterstützung, Kreativarbeit) und ließen die LLMs Prompts den Kategorien zuordnen und reformulieren.

Das zentrale Ergebnis dieser Studie war, dass die Teilnehmenden bei beiden Modellen zufriedener mit Antworten auf ihre Originalprompts waren als mit Antworten auf reformulierte Prompts, selbst dann, wenn die Reformulierung korrekt war. Die Autorinnen und Autoren führen dies darauf zurück, dass bei der automatischen Umschreibung Nuancen, Offenheit und implizite Hinweise verloren gehen, die für eine hilfreiche Antwort wichtig sein können. Außerdem zeigte sich, dass sich Nutzende in realen Interaktionen häufig schnell anpassen und ihre Prompts bei Bedarf selbst umformulieren.

Vor diesem Hintergrund wird in dieser Arbeit bewusst auf die Implementierung einer eigenen Intentionserkennung mit automatischer Prompt-Reformulierung verzichtet. Stattdessen wird auf die inhärenten Fähigkeiten zur Intentionserkennung des eingesetzten Large Language Models gesetzt. Diese Strategie minimiert technische Komplexität und Antwortlatenz und erhält zugleich die Offenheit und Nuancen der ursprünglichen Nutzereingaben.

2.3 Die Wissenslückenerkennung

Ziel der Wissenslückenerkennung ist es, den LLM-basierten Tutor stärker an das Verhalten eines menschlichen Tutors anzunähern. Während der derzeitige KI-Tutor dazu neigt, auf eine Anfrage hin vollständige Lösungen auszugeben und Studierende damit zu überfordern, soll eine Wissenslückenerkennung ermöglichen, den aktuellen Wissensstand der Studierenden einzuschätzen und gezielt auf diejenigen Themen einzugehen, die noch Schwierigkeiten bereiten. Die für eine Aufgabe notwendigen Konzepte sollen so schrittweise vermittelt werden.

2.3.1 Menschliche Tutor*innen

Menschliche Tutor*innen verfügen über ein ausgeprägtes Gespür für den Wissensstand ihrer Schüler*innen und können Lernprozesse flexibel und adaptiv gestalten. Scarlatos und Lan zeigen in ihrer Arbeit [30], dass insbesondere die Analyse offener studentischer Beiträge, im Gegensatz zur Auswertung von Antworten auf Multiple-Choice-Fragen, eine wertvolle Form der formativen Diagnostik darstellt. Offene Äußerungen der Studierenden können Unsicherheiten und Verständnislücken offenlegen. Erfahrene Tutor*innen nutzen solche Signale, um Lernstände zu modellieren und Fehlvorstellungen zu identifizieren. Anhand dieser Informationen können sie ihr Feedback auf die Studenten abstimmen.

Wang u. a. verdeutlichen in ihrer Studie [39] am Beispiel von Mathematik-Nachhilfe, dass erfahrene Lehrkräfte ein implizites Entscheidungsmodell anwenden, um auf Fehler der Lernenden zu reagieren. Dieses Modell lässt sich in drei Schritte zerlegen: zunächst wird der Fehler der Schülerin oder des Schülers diagnostiziert (z. B. geraten, falsch interpretiert, unpräzise geantwortet); darauf folgt die Auswahl einer passenden pädagogischen Strategie (z.B. eine gezielte Rückfrage, Vereinfachung des Problems, ein worked example oder eine Ermutigung); und schließlich wird die Intention der Intervention bestimmt (z.B. motivieren, Missverständnis klären, Diagnose vertiefen). Diese Schritte beruhen auf kognitiver Aufgabenanalyse (Cognitive Task Analysis) und machen sichtbar, dass Expert*innen nicht einfach spontan reagieren, sondern systematisch entscheiden, wie sie Lernende unterstützen.

Durch dieses Vorgehen können menschliche Tutor*innen nicht nur punktuelle Fehler erkennen, sondern auch Muster in den Antworten der Lernenden wahrnehmen, die auf

tiefere Wissenslücken hinweisen. Sie passen ihr Feedback dann gezielt an, durch Methoden wie Rückfragen stellen, Denkanstöße geben, alternative Darstellungen anbieten oder die Aufgabenstellung an das aktuelle Verständnis anpassen. Damit fördern sie nicht nur die Korrektur einzelner Fehler, sondern unterstützen den Aufbau tragfähiger Konzepte. Genau diese Fähigkeit, aus studentischen Äußerungen latente Wissensstände abzuleiten und darauf abgestimmt zu handeln, ist zentral für wirksame menschliche Nachhilfe und bildet die Grundlage für Ansätze wie das von Scarlatos und Lan vorgeschlagene „Knowledge Tracing in Dialogues“.

2.3.2 Übertragbarkeit auf den KI-Tutor

Bei einem KI-Tutor stehen für eine Wissenslückenerkennung ausschließlich die Texteingaben der Studierenden zur Verfügung. Faktoren wie die Zeit zwischen den Prompts sind nicht zuverlässig interpretierbar, da Pausen durch vielfältige äußere Einflüsse entstehen können. Der Ansatz muss sich daher auf die Analyse der studentischen Eingaben selbst stützen.

Scarlatos und Lan zeigen in ihrer Arbeit [30], dass offene studentische Dialogbeiträge in Tutor-Studenten-Interaktionen wertvolle Hinweise auf den Wissensstand und konkrete Fehlkonzepte enthalten. Sie schlagen vor, diese Äußerungen mit Methoden des *Knowledge Tracing* zu analysieren, um Wissenszustände über den gesamten Dialog hinweg zu modellieren. In ihren Experimenten konnten neuartige LLM-basierte Knowledge-Tracing-Methoden studentische Antwortkorrektheit in Dialogen deutlich besser vorhersagen als klassische Verfahren und lieferten qualitativ sinnvolle Schätzungen von Wissenszuständen. Damit wird ein erster praktikabler Ansatz aufgezeigt, wie LLMs aus den Texteingaben der Studierenden latente Wissensstände ableiten und gezieltere Hilfestellungen geben könnten.

Darüber hinaus verdeutlichen Kucharavy u. a. in ihrer Studie [17], dass die klassische Vorstellung eines „LLM als Tutor“ problematisch ist. LLMs neigen zu sogenannten Halluzinationen, also plausibel klingenden, aber falschen Aussagen, und ihre guten Ergebnisse in Standardtests beruhen häufig auf der reinen Wiedergabe gespeicherter Inhalte statt echter Generalisierung. Zudem kann die Autorität eines als Tutor präsentierten LLMs zu einem gefährlichen unkritischen Vertrauen in besagtes LLM führen, bei der Studierende Fehler des Modells übernehmen. Das von Kucharavy u. a. vorgeschlagene *LLM-Protégé-*

Konzept dreht den Ansatz um: Das Modell wird absichtlich mit Wissenslücken versehen und agiert als „Schüler“, den die Studierenden korrigieren müssen.

Dieses Vorgehen nutzt den sogenannten Protégé-Effekt („Learning by Teaching“), fördert aktives Materialengagement und verringert übermäßige Abhängigkeit von LLMs. In ihrer Pilotstudie verbesserten Studierende, die erfolgreich Wissenslücken im LLM diagnostizierten, ihre Noten im Mittel um 0,72 Punkte auf einer Sechs-Punkte-Skala und reduzierten die Durchfallquote im zweiten Zwischentest von 28% auf 8%.

Schließlich hebt Seemann in seiner Literaturstudie [31] hervor, dass Large Language Models zwar zunehmend leistungsfähiger werden, aber ihr „Verstehen“ fachlich umstritten bleibt und uns belastbare Metriken fehlen, um ihr tatsächliches semantisches Erfassen zu messen. Vor diesem Hintergrund empfiehlt sich ein vorsichtiger Umgang mit der Modellierung von Lernprozessen durch LLMs und ein Fokus auf die transparenten, überprüfbaren Stärken dieser Systeme.

Zusammenfassend bietet die Analyse von Studenteneingaben mittels Knowledge-Tracing-Methoden eine vielversprechende Grundlage für eine Wissenslückenerkennung im KI-Tutor. Ansätze wie das von Kucharavy u. a. vorgeschlagene LLM-Protégé-Modell zeigen, dass gezielt eingebaute Wissenslücken den Lernprozess fördern können. Eine reine Übernahme menschlicher Tutorenfähigkeiten (etwa nonverbale Diagnostik) ist für LLMs derzeit nicht realistisch, doch durch systematische Auswertung der Texteingaben können zentrale Elemente einer Wissensdiagnose nachgebildet werden. In dieser Arbeit wird daher der Schwerpunkt auf textbasierte, transparente Verfahren gelegt, um Studierende schrittweise und adaptiv zu unterstützen.

3 Theoretische Grundlagen

3.1 Begriff und Abgrenzung von Künstlicher Intelligenz

Künstliche Intelligenz (KI) kann als der Versuch verstanden werden, Maschinen mit geistigen Fähigkeiten auszustatten, die typischerweise dem Menschen vorbehalten sind. Während klassische Automatisierungssysteme primär auf die Substitution physischer Arbeit abzielen, erweitert KI diese um kognitive Funktionen wie Wahrnehmung, Entscheidungsfindung, Problemlösen und Lernen [22]. Russell und Norvig definieren KI-Systeme als technische Systeme, die ihre Umwelt wahrnehmen, das Wahrgenommene verarbeiten, selbstständig Entscheidungen treffen und aus den Konsequenzen ihrer Handlungen lernen können [29].

Ein grundlegendes konzeptuelles Unterscheidungsmerkmal innerhalb der KI-Forschung ist die Differenzierung zwischen *schwacher* und *starker* KI. Unter **schwacher KI** (engl.: *narrow AI*) versteht man Systeme, die auf klar abgegrenzte Anwendungsbereiche spezialisiert sind und intelligentes Verhalten lediglich simulieren. Sie basieren in der Regel auf fest definierten Algorithmen oder datengetriebenen Modellen, verfügen jedoch über kein echtes Verständnis oder Bewusstsein. Systeme wie ChatGPT zählen zu dieser Kategorie. Im Gegensatz dazu steht die hypothetische **starke KI** (engl.: *general AI*), die über ein dem Menschen vergleichbares, allgemeines Problemlösevermögen sowie über Selbstbewusstsein verfügen soll. Obwohl starke KI ein häufig diskutiertes Ziel der Forschung darstellt, existieren bislang keine praktischen Umsetzungen [28].

Zur weiteren Einordnung und technischen Abgrenzung von KI-Systemen lassen sich die Begriffe **Künstliche Intelligenz**, **Maschinelles Lernen** (ML) und **Deep Learning** (DL) hierarchisch gliedern. Abbildung 3.1 verdeutlicht diese Einordnung grafisch: Künstliche Intelligenz bildet den umfassenden Oberbegriff für alle Systeme, die in der Lage sind, intelligentes Verhalten zu zeigen. Maschinelles Lernen ist ein Teilbereich der KI, in

dem Algorithmen durch Trainingsdaten Muster erkennen und Vorhersagen treffen. Innerhalb des maschinellen Lernens bildet Deep Learning einen spezialisierten Bereich, der auf tiefen künstlichen neuronalen Netzen basiert. Diese ermöglichen es Systemen, sehr große Datenmengen zu verarbeiten und komplexe Strukturen zu erlernen.

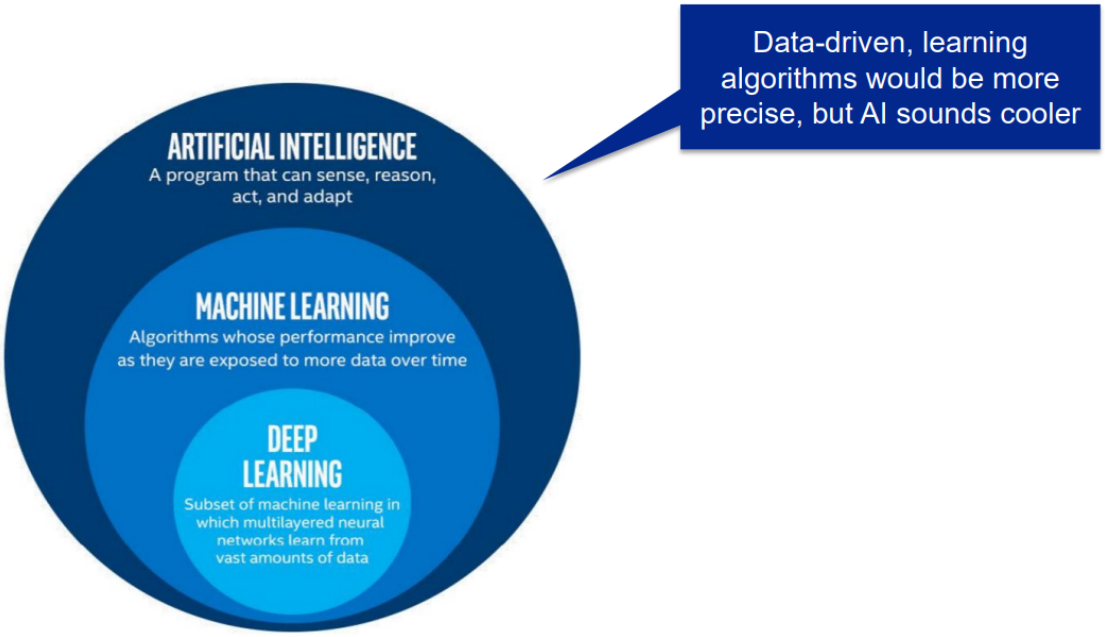


Abbildung 3.1: Abstufungen Künstlicher Intelligenz [37]

Diese hierarchische Abstufung ist insbesondere im Hinblick auf die vorliegende Arbeit von Bedeutung, da sie die technologische Grundlage für sogenannte *Large Language Models* (LLMs) bildet. LLMs sind Deep-Learning-basierte Modelle, die im Rahmen schwacher KI zur Generierung und Verarbeitung natürlicher Sprache eingesetzt werden. Ihre Anwendung als Tutorensysteme steht im Mittelpunkt dieser Arbeit.

3.2 Large Language Models (LLMs)

3.2.1 Transformer-Architektur

Die Transformer-Architektur wurde 2017 von Vaswani u. a. in ihrer Studie „Attention Is All You Need“ vorgestellt und hat sich seither als Standard für große Sprachmodelle (Large Language Models, LLMs) etabliert [34]. Ursprünglich von Google entwickelt für akkurate Übersetzungen, ist die Variante welche für heutige KI-Modelle zum Einsatz kommt trainiert auf Eingabenanalyse und subsequenter Ausgabe von Nachfolgetokens in Form einer Wahrscheinlichkeitsverteilung.

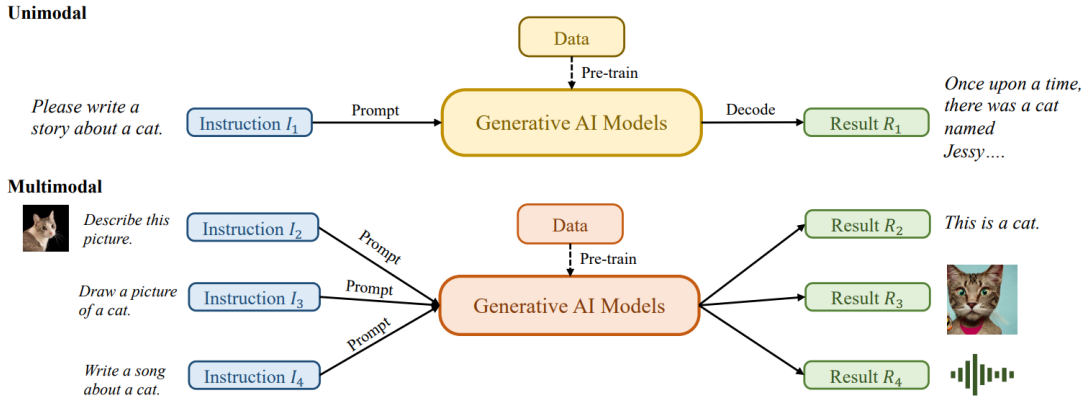


Abbildung 3.2: Übersicht über KI-Generierten Content (AI-Generated Content, AIGC). Generell lassen sich generative KI-Modelle in zwei Typen einteilen: unimodale Modelle und multimodale Modelle. Unimodale Modelle erhalten Anweisungen aus derselben Modalität wie der erzeugte Inhalt, während multimodale Modelle modalitätsübergreifende Anweisungen akzeptieren und Ergebnisse in unterschiedlichen Modalitäten erzeugen. [10]

Einbettung

Zunächst wird der Input in sogenannte Tokens aufgeteilt (s. Abb. 3.3). Diese Tokens werden Vektorisiert um deren Bedeutungen mathematisch darzustellen. Die länge der Vektoren hängt von der Embedding-Dimension des KI Modells ab: Bei GPT-4 beispielsweise sind es je nach Version ca. 2.000 bis 4.000 Dimensionen. Hierbei steht jede Zahl für einen reellen Wert, welcher die latente Semantik entlang einer bestimmten Dimension beschreibt. Jede dieser Dimensionen kodiert eine eindeutige Eigenschaft (z.B. „ist ein

Verb“). Stellt man sich diese Vektoren in einem hochdimensionalen Graphen vor, würden die Vektoren von Wörtern mit ähnlicher Semantik nahe beieinander liegen.

Abbildung 3.3: Beispielhafte Unterteilung eines Liedtextes in Tokens [21]

Aufmerksamkeit

Diese Sequenz von Vektoren wird dem sogenannten *Attention Block* (Aufmerksamkeitsblock) übergeben, welche den Vektoren erlaubt, miteinander zu Kommunizieren und ihre Werte anzupassen, abhängig von den Werten der anderen Vektoren. Die Tokens werden somit im Kontext des gesamten Prompts ausgewertet. Somit kann beispielsweise das Wort „Bau“ in den Kontexten „Die Kollegen auf dem Bau“, „Im Bau die Haftstrafe absitzen“ oder „Im Bau vom Murmeltier“ unterschiedliche Vektoren aufweisen. Hierbei ist es die Aufgabe des Attention Blocks, zu erkennen, welche Wörter im Kontext wichtig sind, um die Vektoren welcher anderen Wörter anzupassen und wie genau die Vektoren angepasst werden sollen [6].

Mehrschichtige Perzeptrons

Die vom Attention-Block angepassten Vektoren werden an das *Multi-Layer Perceptron* (MLP, Mehrschichtiges Perzeptron) bzw. Feed-Forward-Netzwerk weitergegeben. Dieses besteht aus mehreren hintereinandergeschalteten linearen Transformationen und nichtlinearen Aktivierungsfunktionen. Ziel ist es, die vom Attention-Mechanismus erzeugten

Kontextrepräsentationen in eine für die nächste Schicht geeignete Form zu überführen. Während der Self-Attention-Mechanismus also Beziehungen zwischen Tokens herstellt, dient das MLP dazu, komplexe nichtlineare Transformationen auf jeden einzelnen Tokenvektor anzuwenden und damit die Ausdruckskraft des Modells zu erhöhen [34, 6].

Durch das wiederholte Stapeln von Attention-Blöcken und MLP-Schichten in sogenannten Transformer-Blöcken wird die Repräsentation der Eingabesequenz schrittweise verfeinert. Moderne LLMs enthalten Dutzende bis Hunderte solcher Blöcke. Zwischen den Schichten kommen außerdem Residual Connections und Layer Normalization zum Einsatz, um den Gradientenfluss zu stabilisieren und ein tieferes Training zu ermöglichen.

Ausbettung

Nachdem die Token-Vektoren mehrere Transformer-Blöcke durchlaufen haben, werden sie im sogenannten *Output Layer* (Ausbettung) wieder in eine für den Menschen lesbare Form übersetzt. Konkret bedeutet dies, dass jeder Vektor über eine Ausgabematrix in eine Wahrscheinlichkeitsverteilung über das Vokabular des Modells projiziert wird. Das Modell wählt dann (deterministisch oder stochastisch) das wahrscheinlichste nächste Token aus. Auf diese Weise kann das Modell Sequenzen generieren, indem es iterativ Token für Token vorhersagt und das gerade erzeugte Token wieder als Eingabe für den nächsten Schritt nutzt [9].

Dieser letzte Schritt stellt die Verbindung zwischen der hochdimensionalen semantischen Repräsentation und der tatsächlichen Textausgabe her. Moderne LLMs nutzen zusätzliche Techniken wie Sampling-Strategien (z. B. Top-k oder Temperature Sampling), um die Ausgabesequenzen zu steuern und abwechslungsreicher oder kreativer zu gestalten.

3.2.2 Training und Feinabstimmung von LLMs

Large Language Models werden in einem zweistufigen Verfahren entwickelt. Im *Pretraining* werden sie zunächst auf riesigen Textkorpora trainiert, die aus öffentlich zugänglichen Webseiten, Büchern, wissenschaftlichen Artikeln und Programmcode bestehen. Ziel ist die Vorhersage des jeweils nächsten Tokens in einer Sequenz (next-token prediction). Auf diese Weise lernt das Modell statistische Muster, semantische Zusammenhänge und

syntaktische Strukturen der Sprache [9]. Das Pretraining erfolgt üblicherweise auf Tausenden von Grafikprozessoren (GPUs) oder speziellen Beschleunigern über Wochen bis Monate und erfordert erhebliche Rechenressourcen.

Nach dem Pretraining folgt die *Feinabstimmung* (Fine-Tuning) auf spezifische Anwendungsdomäne oder Interaktionsstile. Hierbei werden die Parameter des Modells mit kuratierten, meist qualitativ hochwertigeren Daten nachtrainiert. Eine verbreitete Technik ist das *Instruction Tuning*, bei dem Modelle lernen, auf natürlich formulierte Anweisungen zu reagieren. Ergänzend wird zunehmend *Reinforcement Learning from Human Feedback* (RLHF) eingesetzt, bei dem menschliche Rückmeldungen genutzt werden, um die Antworten des Modells an gewünschte Kriterien wie Hilfsbereitschaft oder Sicherheit anzupassen [25]. Durch diese Verfahren werden LLMs von reinen Sprachmodellen zu interaktiven Dialogsystemen.

3.2.3 Skalierung und Modellgrößen

Ein zentraler Erfolgsfaktor moderner LLMs ist ihre Skalierung. Forschungsergebnisse zeigen, dass die Leistungsfähigkeit neuronaler Sprachmodelle systematisch mit der Zahl der Parameter, der Größe des Trainingskorpus und der Rechenkapazität wächst [16]. Mit zunehmender Modellgröße verbessert sich nicht nur die Genauigkeit bei bekannten Aufgaben, sondern es treten auch emergente Fähigkeiten auf, die bei kleineren Modellen nicht beobachtet werden.

Gleichzeitig führt die Skalierung zu erheblichen Ressourcenanforderungen. Große Modelle wie GPT-4 oder Gemini bestehen aus Hunderten Milliarden Parametern und benötigen für Training und Inferenz spezialisierte Hardware. Auch die *Kontextlänge*, also die Anzahl der Tokens, die das Modell gleichzeitig verarbeiten kann, wurde in den letzten Jahren massiv erweitert und ermöglicht komplexere Aufgaben wie Dokumentanalyse oder Code-Verständnis in einem Durchgang.

3.2.4 Multimodalität

Während klassische LLMs auf Textdaten beschränkt sind, setzt sich zunehmend der Trend zur *Multimodalität* durch (s. Abb. 3.2). Multimodale Modelle können nicht nur Text, sondern auch Bilder, Audio oder Video verarbeiten und erzeugen [10]. Dies geschieht, indem verschiedene Modalitäten in einen gemeinsamen semantischen Vektorraum

eingebettet werden. So können beispielsweise Bildunterschriften erzeugt oder Textanfragen direkt mit Bildinhalten beantwortet werden.

Die Erweiterung hin zu multimodalen Modellen eröffnet neue Anwendungsfelder, von der Bildbeschreibung und Videoanalyse bis hin zu komplexen Assistenzsystemen. Für Bildungstechnologien bedeutet dies, dass Lerninhalte nicht mehr nur in Textform, sondern auch visuell oder auditiv personalisiert aufbereitet werden können.

3.2.5 Anwendungsfelder von LLMs

LLMs werden heute in einer Vielzahl von Bereichen eingesetzt. Klassische Anwendungsfelder sind die maschinelle Übersetzung, Textzusammenfassungen, automatisierte Kundenkommunikation oder die Unterstützung von Wissensarbeit durch Recherche und Ideengenerierung. Neuere Modelle werden zudem für Code-Vervollständigung, Datenanalyse und Kreativprozesse eingesetzt [8].

Im Bildungskontext bieten LLMs besondere Chancen. Sie können, wie im Emmy Projekt vorgesehen, als Tutorensysteme fungieren, welche Lernende individuell begleiten, Verständnisfragen beantworten oder Aufgaben schrittweise erklären. Gerade im Hinblick auf personalisiertes Lernen eröffnet dies neue Möglichkeiten, die mit traditionellen Lehrmethoden nur schwer skalierbar sind [38].

3.2.6 Grenzen und Herausforderungen

Trotz ihrer Leistungsfähigkeit weisen LLMs inhärente Grenzen auf. Ein zentrales Problem sind sogenannte *Halluzinationen*: inhaltlich falsche, aber plausibel klingende Antworten. Zudem spiegeln LLMs die Biases und Wertvorstellungen der Trainingsdaten wider, was zu unfairen oder diskriminierenden Ausgaben führen kann [8]. Auch Fragen des Datenschutzes und der Urheberrechte beim Training auf öffentlichen Daten sind ungelöst.

Darüber hinaus ist der Energie- und Ressourcenverbrauch von LLMs beträchtlich. Das Training großer Modelle verursacht erhebliche CO_2-Emissionen und stellt Unternehmen und Forschungseinrichtungen vor ökologische und ökonomische Herausforderungen. Diese Aspekte sind bei der Planung und Implementierung von KI-Systemen, insbesondere im öffentlichen Sektor und im Bildungsbereich, zu berücksichtigen.

3.2.7 Personalisierung und Adaptivität

Ein vielversprechender Ansatz zur Überwindung einiger dieser Grenzen ist die Personalisierung. Hierbei wird das Verhalten eines LLMs an die individuellen Bedürfnisse und Vorkenntnisse der Nutzerinnen und Nutzer angepasst. Techniken wie *Retrieval-Augmented Generation* (RAG) ermöglichen es, aktuelle und kontextbezogene Informationen aus externen Wissensquellen in die Modellantworten einzubinden, ohne das Basismodell neu trainieren zu müssen [19].

In Kombination mit Verfahren wie dem *Bayesian Knowledge Tracing* (BKT) lassen sich Lernfortschritte erfassen und adaptive Lernpfade gestalten. Dies eröffnet die Möglichkeit, Tutorensysteme zu entwickeln, die nicht nur inhaltlich korrekt, sondern auch didaktisch wirksam agieren. Für die vorliegende Arbeit bildet dies die konzeptionelle Grundlage zur Untersuchung von KI-gestützten Tutorensystemen.

3.3 Retrieval-Augmented Generation (RAG)

3.3.1 Motivation und Grundidee

Große Sprachmodelle verfügen über ein enormes in den Modellparametern gespeichertes Faktenwissen, das sie während des Pretrainings aus riesigen Textkorpora gelernt haben. Dieses Wissen ist jedoch statisch und nicht überprüfbar: Es lässt sich nicht gezielt aktualisieren, es kann veralten und es spiegelt potenziell Verzerrungen der Trainingsdaten wider. Zudem neigen Sprachmodelle dazu, bei Wissenslücken plausible, aber falsche Antworten zu generieren (s. Kap. 3.2.6).

Retrieval-Augmented Generation (RAG) wurde entwickelt, um genau diese Probleme zu adressieren. Das Modell kombiniert generative Textproduktion mit einem externen, abrufbaren Wissensspeicher und kann so aktuelle und belegbare Informationen in seine Antworten einfließen lassen [19].

3.3.2 Systemarchitektur und Ablauf

Die Grundarchitektur von RAG trennt zwei Komponenten: den Retriever und den Generator. Bei einer Anfrage wird zunächst die Eingabe in einen semantischen Vektorraum eingebettet und mit einem vorab erstellten Index verglichen.

Der Retriever wählt die relevantesten Passagen aus einer externen Wissensbasis aus. Diese Passagen werden zusammen mit der ursprünglichen Eingabe an das Sprachmodell übergeben. Der Generator konditioniert seine Antwort auf diese zusätzlichen Informationen und kann dadurch auch Fragen beantworten, deren Inhalte nicht in seinen Gewichten gespeichert sind. Dieser Ablauf verläuft für die Nutzenden transparent und wirkt sich vor allem in einer höheren Faktentreue und Aktualität der Ausgaben aus.

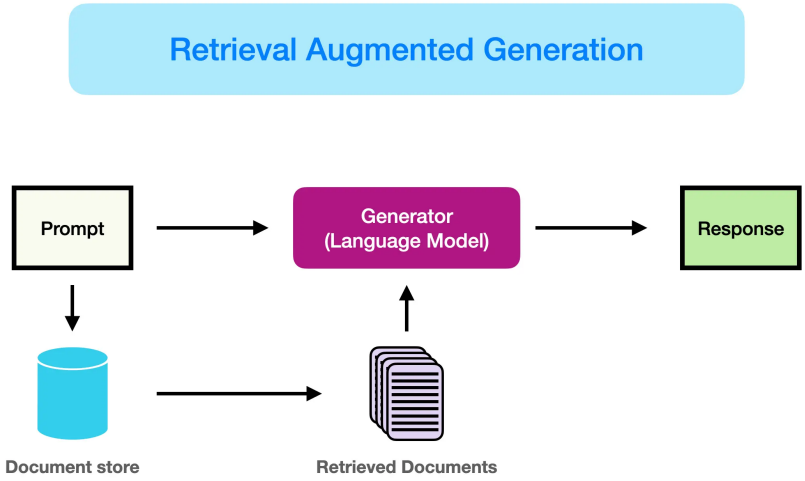


Abbildung 3.4: Schematische Darstellung des Retrieval-Augmented-Generation-Ansatzes (RAG). Ein Nutzereingabeprompt wird zunächst mit einer externen Dokumentbasis abgeglichen. Die dabei abgerufenen relevanten Textpassagen werden zusammen mit dem Prompt an ein Sprachmodell übergeben, das darauf aufbauend eine kontextualisierte und faktenbasierte Antwort generiert. [5]

3.3.3 Indexierung und Repräsentation

Damit der Retriever performant und präzise arbeitet, müssen die Wissensquellen vorab indexiert werden. Üblicherweise werden Dokumente in kleinere, semantisch sinnvolle Einheiten zerlegt und jede Einheit mit Metadaten versehen, etwa Quelle, Erstellungsdatum oder Themengebiet. Diese Einheiten werden anschließend in Vektoren eingebettet, die semantische Ähnlichkeiten in einem hochdimensionalen Raum abbilden. Moderne Systeme nutzen dichte Vektorrepräsentationen (Embeddings) und Distanzmaße wie die Kosinusähnlichkeit.

In der Praxis haben sich hybride Ansätze bewährt, bei denen zunächst klassische Schlüsselwortmethoden wie BM25 eine Kandidatenliste erzeugen, die anschließend durch ein denses Modell neu gerankt wird. So lassen sich Präzision und Recall ausbalancieren und die Robustheit des Systems erhöhen [32, 43].

3.3.4 Einbettung in Prompts und Generierung

Gerade bei längeren Kontexten ist ein sorgfältiges Kontextmanagement erforderlich, da die maximale Tokenlänge des Modells begrenzt ist. Heuristiken und Priorisierungsmethoden entscheiden, welche Passagen aufgenommen werden, wenn mehr relevante Informationen vorliegen als in ein Kontextfenster passen. Dieses Problem ist uns im Emmy Feldexperiment deutlich bewusst geworden.

Die vom Retriever gefundenen Passagen müssen sinnvoll in den Prompt des Sprachmodells integriert werden. Häufig werden sie in einem separaten Block vor der eigentlichen Nutzereingabe eingefügt, ergänzt um Quellenangaben oder Markierungen. Der Prompt weist das Modell an, sich ausschließlich auf die bereitgestellten Passagen zu stützen und Unwissen explizit zu kennzeichnen. Auf diese Weise kann die generative Komponente „geerdet“ werden, ohne ihre Ausdrucksfähigkeit zu verlieren [2, 1].

3.3.5 Evaluation und Herausforderungen

Die Qualität von RAG-Systemen wird nicht nur an der Güte des Abrufs, sondern auch an der Faktentreue und Nützlichkeit der generierten Antworten gemessen. Typische Metriken sind die Trefferquote des Retrievers, die Genauigkeit bei Frage-Antwort-Aufgaben oder automatisierte Konsistenzprüfungen zwischen Zitat und Antwort.

Falsch ausgewählte Passagen können zu „grounded hallucinations“ führen, also fehlerhaften, aber belegten Antworten. Hinzu kommen Datenschutzfragen, wenn sensible Materialien indexiert werden, sowie erhöhte Latenz durch den zusätzlichen Retrieval-Schritt. In der Praxis sind daher kuratierte Wissensbasen, Logging und kontinuierliche Qualitätskontrolle entscheidend für ein stabiles System [13, 20].

3.4 Bayesian Knowledge Tracing (BKT)

3.4.1 Grundprinzip und Motivation

Bayesian Knowledge Tracing (BKT) ist ein etabliertes Verfahren, um den Wissensstand von Lernenden über die Zeit zu modellieren [11]. Es bildet für jede*n Lernende*n u und jeden einzelnen Skill k einen latenten Zustand $p(L_t)$ ab, der beschreibt, mit welcher Wahrscheinlichkeit dieser Skill bereits beherrscht wird. Nach jeder Interaktion mit einem Lernsystem wird diese Wahrscheinlichkeit aktualisiert.

Das Ziel ist hierbei, Lernverläufe zu modellieren und adaptive Tutorensysteme zu entwickeln, die den Schwierigkeitsgrad oder die Art der Rückmeldung an den tatsächlichen Lernfortschritt anpassen können. BKT ist dabei bewusst einfach gehalten und gut interpretierbar.

3.4.2 Parameter und Update-Mechanismus

Das BKT-Modell verwendet vier intuitive Parameter pro Skill: die anfängliche Beherrschungswahrscheinlichkeit $p(L_0)^k$ („p-init“), die Lernwahrscheinlichkeit zwischen zwei Aufgaben $p(T)$ („p-transit“), die Rate zufälligen Erratens $p(G)$ („p-guess“) und die Rate an Fehlern trotz Beherrschung $p(S)$ („p-slip“) [3].

Nach jeder Antwort wird in einem Bayes-Schritt zunächst die Posteriorwahrscheinlichkeit für den aktuellen Zustand berechnet, unter Berücksichtigung der Wahrscheinlichkeiten für „guess“ und „slip“. Anschließend erfolgt ein Übergangsschritt, welcher die Lernwahrscheinlichkeit „transit“ einbezieht. So entsteht eine aktualisierte Beherrschungswahrscheinlichkeit $p(L_{t+1})^k_u$ für die nächste Aufgabe. Dieses zweistufige Verfahren stellt sicher, dass sowohl diagnostische Evidenz aus der letzten Antwort als auch kontinuierlicher Lernfortschritt abgebildet werden. Die Formeln hierfür werden nachfolgend im Detail erläutert.

Die anfängliche Wahrscheinlichkeit, dass ein*e Student*in u einen Skill k gemeistert hat, wird auf den Wert des „p-init“ Parameter für diesen Skill gesetzt:

$$p(L_1)^k_u = p(L_0)^k \tag{3.1}$$

Abhängig davon, ob der/die Student*in u den Skill k gelernt und korrekt oder nicht korrekt angewendet hat, wird die konditionelle Wahrscheinlichkeit $p(L_t)_u^k$ mit Gleichung 3.2 bei korrekter Anwendung oder Gleichung 3.3 bei nicht korrekter Anwendung berechnet.

$$p(L_t \mid \text{obs} = \text{correct})_u^k = \frac{p(L_t)_u^k \cdot \big(1 - p(S)^k\big)}{p(L_t)_u^k \cdot \big(1 - p(S)^k\big) + \big(1 - p(L_t)_u^k\big) \cdot p(G)^k} \tag{3.2}$$

$$p(L_t \mid \text{obs} = \text{wrong})_u^k = \frac{p(L_t)_u^k \cdot p(S)^k}{p(L_t)_u^k \cdot p(S)^k + \big(1 - p(L_t)_u^k\big) \cdot \big(1 - p(G)^k\big)} \tag{3.3}$$

Die konditionelle Wahrscheinlichkeit $p(L_t)_u^k$ wird benutzt, um die Wahrscheinlichkeit, dass der/die Student*in u den Skill k gemeistert hat, neu zu berechnen:

$$p(L_{t+1})_u^k = p(L_t \mid \text{obs})_u^k + \big(1 - p(L_t \mid \text{obs})_u^k\big) \cdot p(T)^k \tag{3.4}$$

Um die Wahrscheinlichkeit zu erhalten, ob der/die Student*in u den Skill k in Zukunft korrekt anwendet, wird Gleichung 3.5 eingesetzt:

$$p(C_{t+1})_u^k = p(L_{t+1})_u^k \cdot \big(1 - p(S)^k\big) + \big(1 - p(L_{t+1})_u^k\big) \cdot p(G)^k \tag{3.5}$$

3.4.3 Zuordnung von Aufgaben und Skills

Eine zentrale Voraussetzung für BKT ist die Zuordnung von Aufgaben zu Skills. Diese erfolgt typischerweise über eine sogenannte Q-Matrix, in der jede Zeile eine Aufgabe und jede Spalte einen Skill repräsentiert. Ein Eintrag markiert, ob ein bestimmter Skill für eine Aufgabe erforderlich ist. In vielen Curricula decken einzelne Aufgaben mehrere Skills ab; in diesem Fall wird das Update für jeden beteiligten Skill durchgeführt. Die Qualität dieser Zuordnung hat erheblichen Einfluss auf die Güte der Beherrschungsschätzung. Sie kann von Expertinnen und Experten vorgegeben oder datengetrieben verfeinert werden, etwa durch Analysen von Antwortmustern.

3.4.4 Anwendung in Tutorensystemen

In klassischen BKT-Szenarien stammen die Beobachtungen aus strukturierten Formaten wie Multiple-Choice-Fragen. In tutorbasierten Dialogsystemen wie in dieser Arbeit gestaltet sich dies schwieriger, da die Eingaben der Lernenden frei formuliert sind. Hier wird zunächst mit Hilfe eines Sprachmodells ermittelt, welche Skills in einem Beitrag berührt und ob sie korrekt oder fehlerhaft angewendet wurden. Erst diese evidenzbasierten Labels können dann in das BKT-Modell eingespeist werden. Auf diese Weise entsteht eine zeitliche Sequenz von Beherrschungswerten, was dem Tutor ermöglicht, den Antwortstil dynamisch anzupassen. Beispielsweise kann er bei niedriger Beherrschung eines Skills kleinschrittiger vorgehen und bei hoher Beherrschung stärker zusammenfassen oder sich auf andere Skills konzentrieren.

3.4.5 Erweiterungen und Alternativen

In den letzten Jahren sind zahlreiche Erweiterungen von BKT entstanden. *Deep Knowledge Tracing* (DKT) ersetzt die pro Skill parametrisierte Bayes-Struktur durch rekurrente neuronale Netze, die Antwortsequenzen direkt modellieren und Interaktionen zwischen Skills berücksichtigen [27]. Auch LLM-basierte Ansätze gewinnen an Bedeutung, da sie semantische Informationen aus freien Texteingaben und Begründungen nutzen können [24]. Diese Verfahren sind leistungsfähig, aber weniger transparent und schwerer zu kalibrieren. Für didaktische Szenarien, in denen Nachvollziehbarkeit und einfache Implementierung wichtig sind, bleibt BKT deshalb attraktiv. Es erlaubt eine unmittelbare Interpretation der Beherrschungswerte und lässt sich mit wenig Rechenaufwand in Echtzeitsysteme integrieren.

Ein weiteres häufig zitiertes Verfahren ist die *Performance Factor Analysis* (PFA) [26]. PFA verzichtet auf ein latentes Zustandsmodell und schätzt stattdessen direkt die Wahrscheinlichkeit einer korrekten Antwort in Abhängigkeit von der Anzahl richtiger und falscher Übungsversuche eines Skills. Für jeden Skill werden separate Gewichtungen für „richtige“ und „falsche“ Übungsgelegenheiten gelernt, sodass sich die Effekte von Übung und Fehlern parametrisch erfassen lassen. Im Gegensatz zu BKT benötigt PFA keine Annahmen über „slip“ und „guess“ und erlaubt eine kontinuierliche Repräsentation des Lernfortschritts. Damit stellt PFA eine rechnerisch einfache und oft besser kalibrierbare Alternative zu klassischen BKT-Modellen dar, bleibt aber wie diese transparent und interpretierbar.

Neuere Arbeiten vergleichen PFA, BKT und DKT systematisch und zeigen, dass PFA trotz seiner Einfachheit in vielen Datensätzen ähnliche oder bessere Vorhersagegüten erzielt, während DKT vor allem bei sehr großen, heterogenen Datenmengen Vorteile bringt. Für Szenarien wie in dieser Arbeit, wo mit begrenzten Datenmengen, hoher Interpretierbarkeit und Echtzeitanforderungen gearbeitet wird, bietet BKT jedoch weiterhin die besten Voraussetzungen.

3.4.6 Relevanz für diese Arbeit

Für das hier entwickelte Tutorensystem ist BKT besonders geeignet. Es lässt sich in Python mit geringem Aufwand umsetzen, liefert pro Skill einen klaren Beherrschungswert und kann mit einer LLM-basierten Skill-Erkennung kombiniert werden. Diese Kombination ermöglicht es, offene Texteingaben der Lernenden auszuwerten, die Beherrschungswerte laufend zu aktualisieren und den Tutorantwortstil entsprechend zu steuern.

4 Anforderungen

4.1 Ausgangssituation und Zielsetzung

Das BKT-System wurde im Rahmen der Bachelorarbeit als theoretische Maßnahme zur Verbesserung des Emmy-Prototyps für die HOOU konzipiert. Es ist wichtig zu betonen, dass das Emmy-System selbst bislang noch nicht einsatzbereit ist, sondern nur prototypisch existiert. Die in dieser Arbeit entwickelte Komponente ist kein fertiges System für den produktiven Einsatz, sondern ein Nachweis, wie adaptives Tutoring durch Skill-Modellierung eingebracht werden könnte. Für eine HOOU-weite Übernahme wären weitere Anpassungen, Skalierungen und Validierungen nötig.

Die Anforderungen sollten daher nicht nur technische Machbarkeit gewährleisten, sondern auch klar machen, welche Zielsetzungen im prototypischen Rahmen verfolgt werden und wo Grenzen der Realisierbarkeit liegen. Das System sollte also robust und modular sein, aber keine unerreichbaren Standards für einen Prototyp setzen.

4.2 Funktionale Anforderungen

Ein zentrales funktionales Ziel war, dass das System aus einer freien Nutzereingabe automatisch relevante Skills erkennt und die Beherrschungswahrscheinlichkeiten (Mastery) entsprechend anpasst. Klassischerweise würde man hierzu eine Q-Matrix verwenden, bei welcher jede Aufgabe mit Skills verknüpft ist. In dieser Arbeit wurde jedoch bewusst darauf verzichtet. Stattdessen besitzt jede Aufgabe eine eigene, hart kodierte Liste von Skills. Dieses Vorgehen erlaubt explizit, dass Lehrende für jeden Kurs oder jede Aufgabe individuelle Skillsets definieren, die genau auf den Lehrplan bzw. Aufgabenkontext abgestimmt sind.

Weiterhin muss das BKT-Update bei jeder erkannten Skill-Beobachtung online erfolgen, mit konfigurierbaren Parametern für Lernrate ($p(T)$), Erraten ($p(G)$) und Fehler trotz

Beherrschung ($p(S)$). Die Tutor-Antwort soll anschließend adaptiv generiert werden, abhängig vom aktuellen Mastery-Stand. Bei niedriger Beherrschung etwa hilfsorientierter, bei hoher eher zusammenfassend oder weiterführend.

Außerdem soll das Modul modular angelegt sein, sodass es als Add-on zu bestehenden Chatbot-Prototypen integriert werden kann, ohne dass das Basis-LLM oder die Chat-Infrastruktur stark verändert werden muss.

4.3 Nicht-funktionale Anforderungen

Da das System als Prototyp angelegt ist, war einfache Implementierbarkeit ein wichtiger Leitgedanke. Die Referenzimplementierung erfolgte in Python ohne exotische Abhängigkeiten, damit Änderungen und Erweiterungen leicht möglich sind.

Transparenz ist eine weitere nicht-funktionale Anforderung: Alle Klassifikationen, Mastery-Werte und Systementscheidungen müssen nachvollziehbar sein. Dazu gehören Speicherung des BKT-Systems in einer lesbaren JSON-Datei und ausführliche Logs.

Auch die Performanz wurde als Kriterium berücksichtigt: Der zusätzliche Overhead durch Skill-Klassifikation und BKT-Update sollte pro Nutzerbeitrag möglichst gering bleiben, damit die Nutzerinteraktion nicht spürbar verzögert wird.

4.4 Reflexion der Designentscheidung ohne Q-Matrix

In vielen realen Lehrkontexten werden Inhalte von Lehrenden individuell formuliert. Durch fest kodierte Skilllisten pro Aufgabe können gezielt jene Konzepte modelliert werden, die spezifisch für den Kurs relevant sind, ohne aus einem generischen Pool ausgesucht werden zu müssen. Zwar erfordert diese Herangehensweise manuelle Arbeit, doch maximiert sie Passgenauigkeit und Transparenz gegenüber Lehrenden.

Ein solcher expliziter Modellierungsansatz findet Parallelen in Arbeiten, die den Einsatz von Wissen von Domänenexpert*innen in adaptiven Systemen betonen. So zeigen Ansätze zur Integration expliziten Expertenwissens, dass manuelle Konstruktionsschritte in hybriden Systemen nicht nur akzeptiert, sondern oft notwendig sind, um kontextsensibles Verhalten zu ermöglichen [15].

Da dieses System als Prototyp im Rahmen der Bachelorarbeit konzipiert wurde und das Emmy-System selbst noch nicht produktiv eingesetzt ist, ist eine pragmatische Entscheidung zu einer manuell definierten Skillstruktur angemessen. Sie ermöglicht schnellen Einstieg, einfache Integration und klare Kontrolle über die Modellierung. Später lässt sich dieses System modular erweitern: Mit zunehmender Nutzungsdatenbasis könnte man automatisierte Verfahren (z. B. Q-Matrix-Schätzungen oder Clustering von Skills) ergänzen, ohne die Kernkomponenten zu überarbeiten.

In zukünftigen Versionen wäre es demnach denkbar, eine hybride Lösung zu nutzen: manuelle Skilllisten als Basissatz, die bei genügend Nutzungsdaten durch automatische Q-Matrix-Vorschläge ergänzt werden. Damit wäre ein pragmatischer Weg zur Skalierbarkeit umsetzbar.

4.5 Zusammenfassung der Anforderungen

Das Anforderungsmodell zielt darauf ab, ein adaptives Tutor-Modul von überschaubarer Komplexität zu realisieren, das dennoch in Open-Online-Kontexten wie der HOOU modular einsetzbar ist. Die Funktionalität zur Skill-Erkennung und dynamischen Mastery-Anpassung bildet den Kern der Arbeit. Der Verzicht auf eine Q-Matrix erlaubt hohe Passgenauigkeit für Kurse, fordert jedoch manuelle Pflege. Nicht-funktionale Anforderungen wie Transparenz und low-overhead Design sichern die Realisierbarkeit im Prototypenkontext. Für eine produktive Nutzung müssten diese Anforderungen später um Aspekte wie Skalierbarkeit, Robustheit, Benutzerverwaltung und Datenschutz ergänzt werden.

5 Umsetzung

In diesem Kapitel wird die technische Realisierung des im Rahmen dieser Arbeit entwickelten Prototyps beschrieben. Im Mittelpunkt steht die Erweiterung des bestehenden Emmy-Chatbots um eine automatische Skill-Erkennung und ein Bayesian-Knowledge-Tracing-(BKT)-System, die es ermöglichen sollen, den Wissensstand von Studierenden während der Interaktion abzuschätzen und adaptiv zu reagieren.

Der Fokus liegt hier auf der Gradio-basierten Chat-Applikation und den zugehörigen Komponenten wie dem lokalen LLM-Server, der Benutzeroberfläche, der Retrieval-Augmented-Generation (RAG)-Integration, der Skill-Erkennung und dem BKT-Modul. Aspekte der Evaluation und des Benchmarkings werden im nachfolgenden Kapitel 6 behandelt.

5.1 Gesamtarchitektur des Prototyps

Der in dieser Arbeit entwickelte Prototyp baut funktional auf den bereits im Emmy-Projekt vorhandenen Komponenten auf, erweitert diese jedoch um zusätzliche Module. Im Zentrum steht ein lokal betriebenes Large Language Model (LLM), das über eine browserbasierte Benutzeroberfläche angesprochen wird und durch eine automatische Skill-Erkennung sowie ein Bayesian-Knowledge-Tracing-(BKT)-System ergänzt wird. Auf diese Weise entsteht ein Tutor, der nicht nur Antworten generiert, sondern gleichzeitig den Wissensstand der Studierenden modelliert und sein Antwortverhalten daran ausrichtet.

Die Grundlage bildet ein lokaler LLM-Server, der über ein Shell-Skript (`llm_server.sh`) gestartet wird und anschließend eine HTTP-Schnittstelle bereitstellt. Diese Schnittstelle wird von der in `gradio_chat_app.py` implementierten Chat-Applikation genutzt, um Prompts an das Modell zu senden und generierte Antworten zurückzuerhalten.

Als Frontend kommt das Python-Framework Gradio zum Einsatz, welches eine einfache, plattformunabhängige Bereitstellung einer grafischen Weboberfläche erlaubt. Innerhalb dieser Oberfläche können die Nutzenden zunächst über Dropdown-Menüs die Aufgabe und Teilaufgabe auswählen und darunter in einem Chatfenster Fragen stellen und Antworten empfangen.

Um die Kontextprobleme der ersten Emmy-Version zu lösen, wird die Retrieval-Augmented-Generation (RAG) dynamisch gesteuert: Anstatt die Aufgabenbeschreibung nur zu Beginn des Chats einmalig an das Modell zu übergeben, werden relevante Informationen aus `questions_full.json` bei jeder neuen Nutzereingabe erneut eingebettet. Dadurch bleibt der Aufgabenkontext auch in längeren Sitzungen erhalten.

Parallel dazu wurde eine automatische Skill-Erkennung implementiert. Ein eigenes Klassifikator-Prompting gibt dem Modell neben der aktuellen Eingabe auch die Aufgabenbeschreibung, die Teilaufgabe und den bisherigen Chatverlauf vor. Das Modell gibt dann in JSON-Form zurück, welche Skills aus `skill_tree.json` korrekt oder fehlerhaft angewendet wurden. Auf Basis dieser Klassifikation wird in einer JSON-Datenbank das BKT-Modul ausgeführt. Es aktualisiert für jeden Skill die Beherrschungswahrscheinlichkeit und stellt diese Information wiederum dem Tutor zur Verfügung. Diese Informationen ermöglichen es dem Tutor, die Antworten granular anzupassen auf das Wissensniveau der Benutzer*innen.

Diese modulare Architektur erlaubt es, das System ohne tiefgreifende Änderungen am Basis-LLM zu erweitern. Neue Aufgaben und Skillsets können einfach über JSON-Dateien eingepflegt werden, und BKT-Parameter lassen sich konfigurieren, ohne den Code zu verändern. Damit erfüllt der Prototyp die in Kapitel 4 formulierten funktionalen und nichtfunktionalen Anforderungen und bildet die Grundlage für die in Kapitel 6 beschriebene Evaluation.

5.2 LLM Server

Das Sprachmodell des Tutors wird lokal auf dem Hochschulrechner betrieben, um Datenschutz und digitale Souveränität zu gewährleisten. Für den Start des Modells dient ein Shell-Skript (`llm_server.sh`), das den benötigten Serverprozess automatisiert hochfährt. Innerhalb dieses Skripts wird der ausführbare `llama-server` (bzw. ein kompatibler LLM-Server) mit den gewünschten Parametern wie Modellpfad, Portnummer und maximaler Kontextlänge gestartet. Modellpfad und Port sind dabei als Variablen hinterlegt und können leicht an unterschiedliche Modelle oder Umgebungen angepasst werden.

Durch den Aufruf

```
bash ./llm_server.sh start
```

wird der Serverprozess im Hintergrund gestartet und ist anschließend über eine HTTP-Schnittstelle erreichbar. Er lässt sich jederzeit über den Befehl

```
bash ./llm_server.sh stop
```

stoppen. Dies wird dadurch ermöglicht, dass beim Starten die Prozess-ID (PID) des Servers gespeichert wird, sodass diese PID beim Stoppen gezielt beendet werden kann.

Die Chat-Applikation `gradio_chat_app.py` kommuniziert zur Laufzeit direkt mit diesem LLM-Server. Hierzu werden mit dem Python-Paket `requests` HTTP-POST-Anfragen an die Server-URL gesendet, wobei der jeweilige Prompt in JSON-Form übergeben wird. Der Server streamt die generierte Antwort tokenweise zurück; die App liest diesen Stream und gibt ihn in Echtzeit in der Gradio-Oberfläche aus. Durch diese Entkopplung von Frontend und Modell kann das zugrunde liegende LLM leicht ausgetauscht oder neu gestartet werden, ohne die Gradio-Applikation selbst zu ändern.

5.3 Benutzeroberfläche

Die Benutzeroberfläche des Prototyps wurde mit dem Python-Framework `Gradio` umgesetzt. Gradio eignet sich besonders für schnelle Prototypenentwicklung, da es eine einfache API für Web-Interfaces bereitstellt und ohne zusätzlichen Servercode sofort eine browserbasierte Oberfläche erzeugt. Dadurch können auch Nutzer*innen ohne lokale Entwicklungsumgebung bequem über einen Webbrowser mit dem Tutor interagieren. Ein weiterer Vorteil ist die enge Integration mit Python: Eingaben, Ausgaben und Streaming-Events lassen sich direkt aus Python-Funktionen ansteuern, sodass keine separate Frontend-Entwicklung in HTML oder JavaScript erforderlich ist.

In `gradio_chat_app.py` wird diese Oberfläche so aufgebaut, dass sie die spezifischen Anforderungen des Tutorsystems abbildet. Oberhalb des Chatbereichs befindet sich ein Dropdown-Menü zur Auswahl der Aufgabe, darunter ein zweites Dropdown für die Teilaufgabe. Unterhalb dieser Auswahl ist der eigentliche Chatbereich platziert, in dem die bisherigen Nachrichten zwischen Tutor und Nutzer*in angezeigt werden. Rechts daneben oder darunter befindet sich ein Texteingabefeld, in das die Nutzer*innen ihre Fragen oder Antworten eingeben und per Knopfdruck absenden können. Das System zeigt dann die generierte Antwort des Tutors in Echtzeit an.

Abbildung 5.1 zeigt einen Screenshot der implementierten Oberfläche. Diese einfache, klar strukturierte Anordnung unterstützt den geplanten Ablauf: Auswahl einer Aufgabe, Eingabe der eigenen Frage oder Lösungsschritte, und direkte Rückmeldung durch den Tutor. Durch die Dropdown-Auswahl werden zudem die notwendigen Metadaten für die Skill-Erkennung und das BKT-System bereitgestellt, ohne dass der oder die Studierende diese Informationen manuell eingeben muss.

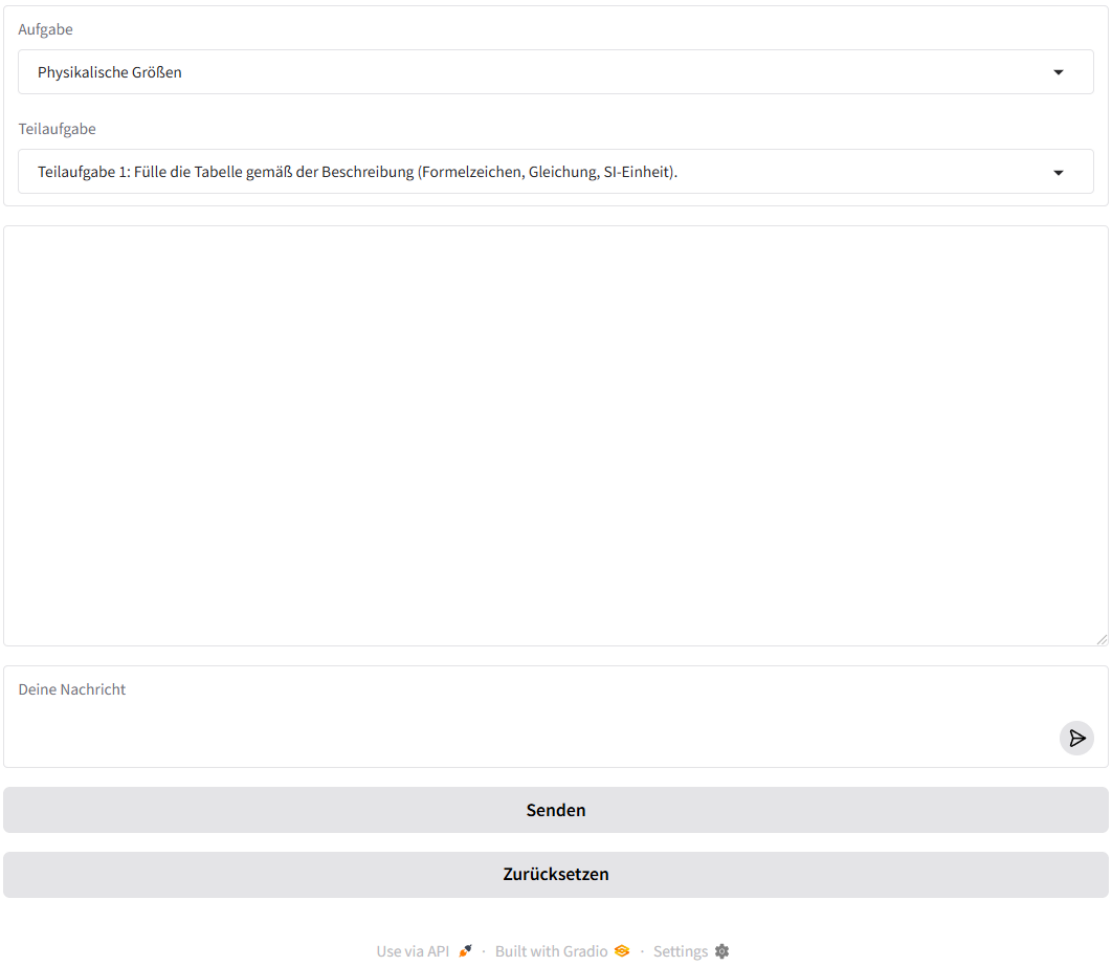


Abbildung 5.1: Benutzeroberfläche des Prototyps: Auswahl von Aufgabe und Teilaufgabe sowie Eingabe eigener Fragen in der Gradio-Weboberfläche.

5.4 Verbesserungen der RAG Funktionalität

In der im Feldexperiment verwendeten Frühversion wurde die Aufgabenbeschreibung lediglich zu Beginn des Chats einmalig in den Prompt eingefügt. In längeren Dialogen „wanderte“ dieser Kontext aus dem verfügbaren Kontextfenster des Modells heraus, so dass der Tutor die Aufgabe nicht mehr zuverlässig im „Gedächtnis“ hatte und der Chat neu gestartet werden musste um den Kontext der Aufgabe zurückzuholen.

Der Prototyp behebt dieses Problem durch eine dynamische, turnweise Kontext-Einbettung. Anstatt die Aufgabe statisch in einen großen Systemprompt zu packen oder sie am Anfang vor dem ersten User-Prompt mitzugeben, wird bei jeder neuen Nachricht die jeweils gewählte Aufgabe und Teilaufgabe als zusätzliche System-Nachricht vor dem aktuellen User-Prompt in den Nachrichtenstrom eingefügt. Ergänzend wird der aktuelle Skill-Zustand als JSON-Block übergeben, so dass der Tutor auch die (durch BKT fortgeschriebene) Beherrschungsinformation kennt.

Konkret erzeugt `gradio_chat_app.py` aus der bisherige `chat_history` eine Renderliste und hängt dann, sofern vorhanden, drei systematische Kontexte an: (1) `[Aufgaben-Beschreibung]` mit dem Aufgabenfließtext, (2) `[Aktuelle Teilaufgabe]` mit der Unteraufgabe, (3) `[Skill-Mastery]` mit dem aktuellen Skill-JSON. Erst danach folgt der neue User-Prompt.

Für den lokalen `llama-server` wird der Nachrichtenverlauf anschließend in das vom Server erwartete Chat-Format serialisiert (jeweils <im_start> Rolle \n Inhalt <im_end> , mit offenem `assistant`-Turn). Dadurch kann die App die Vorteile eines dialog-orientierten Prompts nutzen, obwohl der Server eine `/completion`-Schnittstelle erwartet:

```
def build_prompt(messages):
    prompt = ""
    for m in messages:
        prompt += f"<|im_start|>{m['role']}\n{m['content']}<|im_end|>\n"
    prompt += "<|im_start|>assistant\n"
    return prompt
```

Der fertige Prompt wird an `http://127.0.0.1:8082/completion` gesendet; die Antwort wird in der Oberfläche gestreamt. Da Aufgabe und Teilaufgabe turnweise als System-Nachrichten „frisch“ eingebettet werden, bleibt ihr Kontext auch in sehr langen

Unterhaltungen präsent, ohne dass ein Neustart nötig ist. Gleichzeitig bleibt der Tutor-Systemprompt absichtlich generisch; die fachlichen Inhalte kommen ausschließlich aus der dynamischen Einbettung.

Ein zweiter, praxisrelevanter Aspekt ist die gezielte Kürzung von Verlaufskontexten für Nebenaufrufe: Der Skill-Klassifikator erhält eine gekürzte History (z.B. maximal acht Nachrichten und eine Zeichenobergrenze), um Tokenbudget zu sparen, ohne den Tutor-Kontext zu gefährden. Der Tutor-Call selbst nutzt die vollständige (bzw. serverseitig begrenzte) History plus die drei oben genannten Systemblöcke. In Summe entsteht so ein schlankes, robustes RAG-Verhalten: kuratierter Kurskontext wird bei jedem Turn eingebunden, die Dialogkohärenz bleibt gewahrt, und der zusätzliche Overhead bleibt gering.

5.5 Skill-Erkennung

Die Erkennung fachlicher Teilfähigkeiten (Skills) erfolgt turnweise über einen separaten Klassifikationsaufruf an das LLM. Grundlage ist eine pro Aufgabe fest kodierte Skillliste in `skill_tree.json`. Für jede Aufgabe sind dort die relevanten Skills mit *Name* und *Beschreibung* hinterlegt; die *Mastery*-Werte werden zur Laufzeit fortgeschrieben (s. BKT). Der Klassifikator erhält bei jedem Nutzereingabeturn (a) den eigenen Klassifikator-Systemprompt (`classification_system_prompt.txt`), (b) die aktuell gewählte Aufgabe und Teilaufgabe, (c) eine gekürzte Chat-History als Kontext und (d) die aktuelle Nutzeräußerung. Ausgegeben wird *ausschließlich* ein JSON-Objekt, das erkannte Skills mit 1 (korrekt angewandt) bzw. 0 (fehlerhaft / nicht korrekt) markiert. Sind in der Äußerung keine fachlichen Inhalte enthalten (z. B. reine Meta-Fragen), gibt der Klassifikator ein leeres JSON-Objekt ({}) zurück.

Der Ablauf ist bewusst strikt gehalten, um Parsingfehler und Halluzinationen zu vermeiden. Ungültiges oder nicht-parsbares JSON wird als Klassifikationsfehler geloggt und für den Turn verworfen; der Tutor antwortet dann ohne BKT-Update, aber mit normalem didaktischen Verhalten. Ein vereinfachter Ausschnitt:

```
def classify_skills(allowed_skills, task_desc, subtask_text, history,
    user_msg):
    sys_prompt = load_text("system_prompts/classification_system_prompt.txt")
    # kompakten Kontext bauen
    ctx = {
        "task": task_desc, "subtask": subtask_text,
        "history": truncate_history(history, max_msgs=8, max_chars=2500),
        "allowed_skills": allowed_skills,
        "last_user": user_msg
    }
    messages = [
        {"role": "system", "content": sys_prompt},
        {"role": "user",   "content": json.dumps(ctx, ensure_ascii=False)}
    ]
    payload = {"prompt": serialize_chat(messages), "stream": False}
    r = requests.post(LLM_URL, json=payload, timeout=60)
    try:
        result = json.loads(r.json().get("content","{}"))
        assert isinstance(result, dict)
        return result  # z.B. {"S01 Grundbegriffe verstehen": 1, "S02
    Beobachtbarkeit und Messbarkeit": 0}
    except Exception as e:
```

```
21          log.warning("Skill Classification Error: %s", e)
22          return {}
```

Die Trennung von Tutor-Dialog und Klassifikator-Call stellt sicher, dass didaktische Regeln des Tutors unabhängig bleiben, während die Klassifikation rein diagnostisch arbeitet. Zudem schont die gekürzte History Token-Budget und Latenz, ohne den Kernkontext zu verlieren.

5.6 Bayesian Knowledge Tracing (BKT)

Die von der Skill-Erkennung gelieferten Beobachtungen (pro Skill 1 = korrekt, 0 = falsch) werden unmittelbar in ein BKT-Modell eingespeist. Nach jedem Turn berechnet der Prototyp (1) den Bayes-Posterior gegeben der Beobachtung und (2) den Übergangsschritt mit $p(T)$. Die Mastery-Werte werden als JSON persistiert, sodass zwischen Sitzungen konsistente Zustände vorliegen. Reset-Skripte ermöglichen das Zurücksetzen der Mastery-Werte je Skill.

Der folgende Ausschnitt zeigt die zentralen Update-Formeln in Codeform (Notation analog zu Kapitel 3):

```
def bkt_update(p_L_t: float, obs_is_correct: bool, pS: float, pG: float, pT:
    float) -> float:
    pLt = max(0.0, min(1.0, float(p_L_t)))
    if obs_is_correct:
        numer = pLt * (1.0 - pS)
        denom = pLt * (1.0 - pS) + (1.0 - pLt) * pG
        p_L_t_given_obs = numer / denom if denom > 0 else pLt
    else:
        numer = pLt * pS
        denom = pLt * pS + (1.0 - pLt) * (1.0 - pG)
        p_L_t_given_obs = numer / denom if denom > 0 else pLt
    p_L_next = p_L_t_given_obs + (1.0 - p_L_t_given_obs) * pT
    return max(0.0, min(1.0, p_L_next))
```

5.7 Adaptiver Antwortstil des Tutors

Die Mastery-Werte aus der Skill-Erkennung und dem BKT-System werden im aktuellen Prototypen nach jedem Nutzereingabeturn berechnet und zusammen mit der Aufgaben- und Teilaufgabenbeschreibung an das LLM übergeben. Damit verfügt das Modell bei jeder Antwort über einen aktualisierten Überblick, welche Skills als beherrscht und welche als noch unsicher gelten. Der Tutor-Systemprompt selbst bleibt jedoch generisch; eine explizite, fest kodierte Entscheidungslogik zum Antwortstil ist in der vorliegenden Implementierung noch nicht integriert. Das Modell kann seinen Stil implizit aus den bereitgestellten Kontextinformationen ableiten, die Art und Tiefe der Hilfestellungen sind aber nicht deterministisch vorgegeben.

Für eine weitergehende Ausbaustufe wäre es denkbar, eine „Tutoring-Policy“ zu implementieren, die aus den Mastery-Werten automatisch einen Fokus-Skill und eine Detailstufe ableitet und diese Leitplanken in den Prompt einbettet. Ein solches Verfahren könnte etwa wie folgt aussehen: Zunächst wird aus den beobachteten Skills derjenige mit der niedrigsten Beherrschung ausgewählt, anschließend anhand vordefinierter Schwellenwerte (z. B. $p \leq 0{,}4$: Micro-Hints, $p < 0{,}7$: Guided Steps, $p \geq 0{,}7$: Zusammenfassungen) eine Detailstufe bestimmt und daraus ein kurzer Richtlinienblock („Tutoring-Policy“) generiert, der vor jeder Antwort in den Systemprompt geschrieben wird. Ein vereinfachter Codeentwurf dazu wäre:

```
def select_focus_skill(observations: dict[str,int], mastery: dict[str,float])
     -> str | None:
    # bevorzuge Skills, die im aktuellen Turn beobachtet wurden
    candidates = [s for s,obs in observations.items() if obs in (0,1)]
    if not candidates:
        # Fallback: global niedrigste Mastery
        return min(mastery, key=mastery.get, default=None)
    return min(candidates, key=lambda s: mastery.get(s, 0.5))

def guidance_level(m: float) -> str:
    if m <= 0.40:   return "micro_hint"      # sehr kleinschrittig, Rü
    ckfragen, nächste Mini-Aktion
    if m <  0.70:   return "guided_steps"    # 2–3 gezielte Schritte mit
    kurzer Begründung
    return "summary"                          # knapp zusammenfassen, nächste
     Lernschritte skizzieren

focus = select_focus_skill(observations, mastery)
```

```
level = guidance_level(mastery.get(focus, 0.5))

policy = (
    "[Tutoring-Policy]\n"
    f"Fokus-Skill: {focus or '-'}\n"
    f"Mastery: {mastery.get(focus, 0.5):.2f}\n"
    "Regeln:\n"
    "- Keine vollständigen Lösungen; nur den nächsten sinnvollen Schritt erläutern.\n"
    "- Prüffragen statt Behauptungen, um Verständnis zu testen.\n"
    "- Verweise auf Aufgabe/Teilaufgabe beibehalten.\n"
)
if level == "micro_hint":
    policy += "- Gib eine sehr kleine, konkrete Hilfestellung und stelle eine Rückfrage.\n"
elif level == "guided_steps":
    policy += "- Skizziere 2–3 Schritte (kurz) und fordere zur Ausführung auf.\n"
else:
    policy += "- Fasse den Stand kurz zusammen und schlage nächste Vertiefungen vor.\n"

render_messages.append({"role": "system", "content": policy})
# danach: user-Nachricht anhängen und an den LLM-Server senden
```

Eine solche Logik ist im vorliegenden Prototypen noch nicht implementiert, würde aber erlauben, den Antwortstil stärker und reproduzierbarer an den aktuellen Lernstand anzupassen. Sie stellt daher einen sinnvollen Ansatz für zukünftige Arbeiten dar.

6 Auswertung

In diesem Kapitel wird der entwickelte Prototyp hinsichtlich seiner Funktionsweise und seiner angestrebten Ziele überprüft. Im Mittelpunkt steht die Frage, ob die implementierte Skill-Erkennung und das Bayesian-Knowledge-Tracing-(BKT)-Modul wie vorgesehen arbeiten. Konkret soll untersucht werden, ob das Klassifikationsmodell in der Lage ist, die von Studierenden gezeigten Teilfähigkeiten (Skills) den im `skill_tree.json` hinterlegten Skills korrekt zuzuordnen und zwischen richtiger und fehlerhafter Anwendung zu unterscheiden.

Da im Rahmen dieser Arbeit kein Zugriff auf eine größere Zahl realer Studierender bestand, wurde auf einen automatisierten Benchmarking-Ansatz zurückgegriffen. Hierzu wurde ein Large-Language-Model (LLM) so gesteuert, dass es die Rolle von Studierenden übernimmt und Prompts generiert, die sonst echte Nutzer*innen in das System eingeben würden. Der Skill-Klassifikator wurde anschließend auf diese künstlich erzeugten Prompts angewendet und die resultierenden Mastery-Werte im BKT-System fortgeschrieben. Auf diese Weise sollte überprüft werden, ob das Gesamtsystem aus Klassifikator und BKT prinzipiell wie vorgesehen funktioniert.

Die nachfolgenden Abschnitte erläutern zunächst den Aufbau des Benchmarks, stellen die wichtigsten Ergebnisse dar und diskutieren diese kritisch im Hinblick auf die in Kapitel 1 und Kapitel 4 formulierten Ziele.

6.1 Quantitativer Benchmark

Die Entwicklung eines automatisierten Benchmarks für ein adaptives Tutorsystem ist nicht trivial. Um manuelles Erstellen und Annotieren hunderter Nutzereingaben zu vermeiden, wurde in dieser Arbeit ein generativer Ansatz gewählt: Ein LLM wurde angewiesen, Prompts zu formulieren, in welchen vorgegebene Skills vorkommen sollen und die, je nach Vorgabe, eine korrekte oder fehlerhafte Anwendung dieser Skills demonstrieren. Der

Skill-Klassifikator sollte anschließend erkennen, welche Skills in diesen Prompts enthalten sind und ob sie korrekt oder fehlerhaft genutzt wurden; das BKT-Modul sollte daraufhin die Mastery-Werte entsprechend aktualisieren.

Dieses Verfahren hat jedoch inhärente Grenzen. Es setzt voraus, dass das generierende LLM die gewünschten Skills korrekt auswählt, es Prompts erstellt, die diese Skills tatsächlich klar zeigen, und dass das Klassifikator-LLM anschließend die intendierten Skills zuverlässig erkennt. Damit entstehen zwei vorgeschaltete Unsicherheitsstufen, bevor überhaupt der eigentliche Klassifikator getestet wird. In der Praxis zeigte sich, dass das generierende LLM die gewünschten Skills nicht immer konsistent umsetzt, sodass der Benchmark nur bedingt Aussagen über die echte Klassifikationsgüte erlaubt.

Vor diesem Hintergrund ist der hier durchgeführte Benchmark vor allem als Funktionsnachweis des Prototyps zu verstehen: Er zeigt, dass das System auf Prompts reagiert, Skill-Klassifikationen vornimmt und Mastery-Werte gemäß den BKT-Formeln fortschreibt. Er belegt jedoch ausdrücklich nicht, dass diese Klassifikationen bei realen Studierenden inhaltlich korrekt wären.

6.2 Ergebnisse des Benchmarks

Trotz der Unsicherheiten dieses Benchmark-Verfahrens lässt sich aus den Ergebnissen ein gewisser Lichtblick ableiten: Der Klassifikator konnte auch ohne spezifisches Training viele Skills konsistent zuordnen und reagierte nicht empfindlich auf untypische Prompts. Dies deutet darauf hin, dass sich das System mit echten Trainingsdaten und besser gesteuerten Eingabe-Prompts künftig weiter verbessern und robuster machen ließe.

Abbildungen 6.1 bis 6.3 zeigen exemplarisch den Verlauf der vom System ermittelten Mastery-Werte. Bei Skill S18[1] der Aufgabe „Physikalische Größen“ folgt die Kurve der vom Benchmark intendierten Entwicklung, wenn auch mit einzelnen Abweichungen. Bei Skill S13[2] derselben Aufgabe stimmt die Mastery-Kurve noch genauer mit dem Sollverlauf überein. Hingegen ist eine konsistente Klassifikation bei Skill S03[3] der Aufgabe „Raketenstart A“ nicht gelungen (vgl. Abbildung 6.3).

[1] Kann Ergebnisse verständlich dokumentieren, zum Beispiel mit geeigneten Bezeichnungen und Einheiten in Worten.
[2] Beschreibt, wie sich zusammengesetzte Größen aus grundlegenden Größen verbal herleiten lassen.
[3] Beschreibt den zeitlichen Verlauf von Beschleunigung, Geschwindigkeit und Ort bei konstanter Beschleunigung.

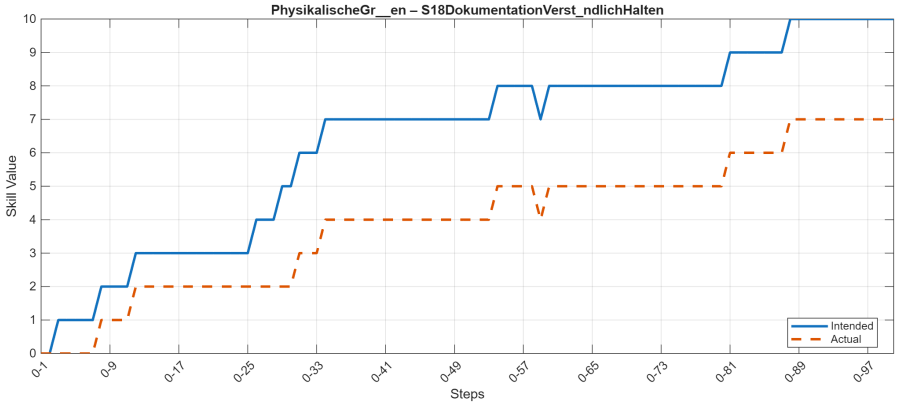


Abbildung 6.1: Bei Skill S18 der Aufgabe Physikalische Größen folgt die vom Klassifikator ermittelte Mastery-Kurve dem intendierten Verlauf, wenn auch mit einzelnen Fehlern.

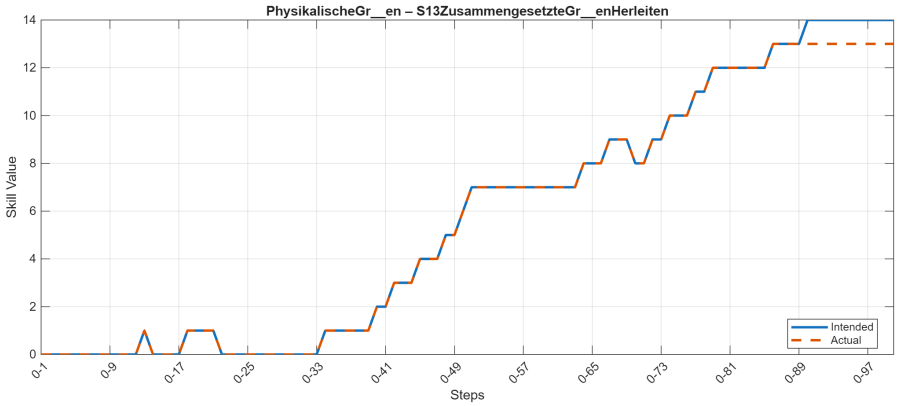


Abbildung 6.2: Bei Skill S13 derselben Aufgabe wird die Mastery-Kurve noch genauer getroffen.

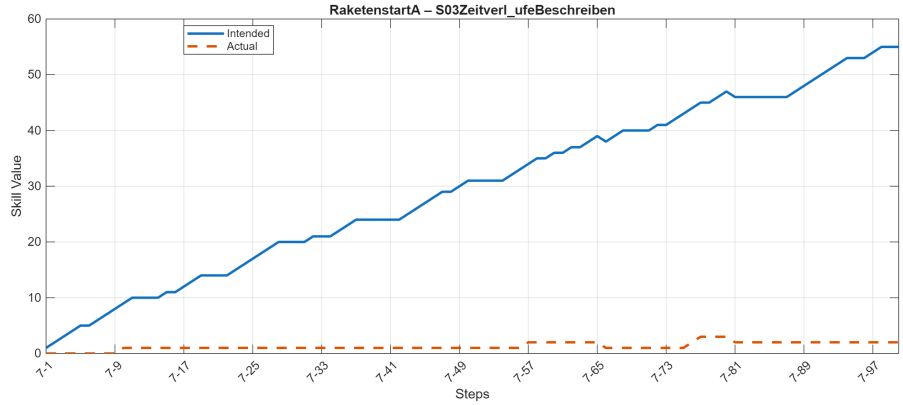


Abbildung 6.3: Bei Skill S03 der Aufgabe Raketenstart A ist eine deutliche Abweichung zur erwarteten Kurve zu erkennen.

Die nachfolgenden JSON-Ausschnitte verdeutlichen, wie Prompts, Skills und Klassifikationsausgabe im Benchmark zusammenhängen.

Das erste Beispiel (Aufgabe „Translation und Rotation“) zeigt einen vom generierenden LLM erstellten Prompt, in welchem zwei Skills (S04[4] und S06[5]) demonstriert werden sollten. Der Klassifikator hat diese Skills korrekt erkannt und die Mastery-Werte entsprechend angepasst:

```
{
  "role": "user",
  "content": "Ich verstehe, dass die Winkelbeschleunigung die analoge Größe
    zur Beschleunigung ist, aber warum wird das Drehmoment nicht als analoge
    Größe zur Kraft betrachtet?",
  "prompt_id": "2_21"
}

{
  "exercise": "Translation und Rotation",
  "prompt_id": "2_21",
  "skills_proposed": {
    "S04 Beschleunigung und Winkelbeschleunigung zuordnen": 1,
    "S06 Kraft und Drehmoment gegenüberstellen": 1
  },
  "skills_changed": {
    "S04 Beschleunigung und Winkelbeschleunigung zuordnen": 1,
```

[4]Ordnet Beschleunigungen und Winkelbeschleunigungen zu und erklärt, welche Rolle sie in Modellen spielen.

[5]Kann Kräfte und Drehmomente inhaltlich gegenüberstellen und die Wirkung erklären.

```
16      "S06 Kraft und Drehmoment gegenüberstellen": 1
17    }
18 }
```

Das zweite Beispiel (Aufgabe „Vakuumröhre“) vermittelt die Grenzen des Benchmarks. Hier sollten die Skills S02[6] und S04[7] demonstriert werden. Das Prompt-LLM hat es jedoch nicht geschafft, einen Prompt zu erzeugen, der diese Skills klar zeigt; entsprechend konnte der Klassifikator keine Mastery-Werte anpassen:

```
1 {
2   "role": "user",
3   "content": "Hat der Mittelwert der Messzeit eine Einheit?",
4   "prompt_id": "4_97"
5 }
6
7 {
8   "exercise": "Vakuumröhre",
9   "prompt_id": "4_97",
10  "skills_proposed": {
11    "S02 Zusammenhang Höhe und Zeit beschreiben": 1,
12    "S04 Arithmetisches Mittel erläutern": 1
13  },
14  "skills_changed": {}
15 }
```

Dieses Beispiel unterstreicht, dass ein Benchmark, der vollständig auf generierten Prompts basiert, die inhaltliche Qualität der Klassifikation nicht zuverlässig abbildet: Wenn die generierten Prompts die intendierten Skills nicht klar enthalten, ist eine valide Bewertung des Klassifikators unmöglich.

[6] Beschreibt, wie Fallhöhe und Fallzeit zusammenhängen und wie daraus die Fallbeschleunigung abgeleitet wird.

[7] Berechnet in Worten das arithmetische Mittel und erklärt seine Aussage.

6.3 Interpretation im Hinblick auf die Ziele

Die im Rahmen dieser Arbeit entwickelte Komponente verfolgt das Ziel, einen prototypischen KI-Tutor mit automatischer Skill-Erkennung und Bayesian Knowledge Tracing (BKT) zu realisieren. Der quantitative Benchmark diente ursprünglich dazu, zu prüfen, ob das System in der Lage ist, aus offenen studentischen Prompts relevante Skills zu erkennen und die Beherrschungswerte für diese Skills entsprechend anzupassen.

Die Auswertung hat gezeigt, dass dieser Ansatz nur eingeschränkt geeignet ist, um eine echte Leistungsbewertung vorzunehmen. Der Benchmark beruht auf einer Kette von LLM-Aufrufen: Ein Modell wählt Skills aus, ein zweites Modell erzeugt dazu passende Prompts, und erst das dritte Modell, welches der eigentliche Klassifikator ist, ordnet Skills zu und aktualisiert die Mastery-Werte. Somit hängt das Ergebnis stark von der Qualität der ersten beiden LLM-Schritte ab. Wie die Beispiele zeigen, gelingt es dem Prompt-generierenden LLM nicht immer, tatsächlich einen Prompt zu formulieren, der die gewünschten Skills demonstriert (z.B. im Fall „Vakuumröhre“). In solchen Fällen kann auch ein korrekt arbeitender Klassifikator keine sinnvollen Updates erzeugen.

Dennoch lassen sich zwei zentrale Beobachtungen ableiten: Erstens zeigt der Klassifikator-LLM bereits ohne spezielles Training eine beachtliche Robustheit beim Erkennen von Skills in realistisch wirkenden Prompts (z.B. im Beispiel „Translation und Rotation“). Zweitens funktioniert die BKT-Implementierung technisch zuverlässig: Erfasste Beobachtungen werden turnweise verarbeitet und führen zu aktualisierten Mastery-Werten, die in den Tutor-Prompt eingebettet werden können. Ob diese Werte inhaltlich stimmen, konnte mit dem gewählten Setup zwar nicht validiert werden, doch das Grundprinzip (laufendes Tracking von Skill-Beherrschung in einem dialogbasierten Tutor) wurde erfolgreich demonstriert.

Im Hinblick auf die in Kapitel 1 und Kapitel 4 formulierten Ziele lässt sich damit festhalten: Es ist gelungen, einen funktionierenden Prototypen für Skill-Erkennung und BKT in ein LLM-basiertes Tutorensystem zu integrieren. Die Implementierung zeigt, dass adaptive Elemente wie Mastery-Tracking technisch umsetzbar sind und mit den bestehenden Emmy-Komponenten harmonieren. Zugleich macht der Benchmark deutlich, dass für eine echte Wirksamkeitsmessung qualitativ hochwertige, manuell annotierte Testdaten erforderlich sind, anstatt Prompts und Labels automatisch generieren zu lassen.

Diese Arbeit liefert somit vor allem einen Proof-of-Concept: Das System kann auf Basis erfasster Beobachtungen Mastery-Werte ändern und diesen Kontext an das Tutor-LLM

weitergeben. Ob und in welchem Umfang dies zu didaktisch besseren Antworten führt, bleibt zukünftigen Arbeiten vorbehalten. Auf dieser Grundlage kann in künftigen Projekten mit echten Studierendendaten, klar definierten Skillbeschreibungen und ggf. trainierten Klassifikatoren untersucht werden, wie gut sich Wissensstände tatsächlich modellieren und adaptive Tutorstrategien ableiten lassen.

7 Fazit

7.1 Zusammenfassung der Ergebnisse

Ausgangspunkt dieser Arbeit war die Zielsetzung, den bestehenden Emmy-Chatbot um Mechanismen zur automatischen Skill-Erkennung und zum Bayesian Knowledge Tracing (BKT) zu erweitern, um den Wissensstand von Studierenden besser abzuschätzen und adaptiv auf ihn reagieren zu können. Im Rahmen dieser Arbeit wurde ein funktionsfähiger Prototyp entwickelt, der auf einem lokal betriebenen LLM-Server basiert und über eine Gradio-Weboberfläche zugänglich ist.

Kernmaßnahmen waren:

- die dynamische Einbettung von Aufgaben und Teilaufgaben in jede Nutzeranfrage (Verbesserung der RAG-Funktionalität),
- ein separater Skill-Klassifikator, der pro Eingabe Skills aus `skill_tree.json` erkennt und als korrekt bzw. fehlerhaft markiert,
- sowie die direkte Fortschreibung der Mastery-Werte durch ein BKT-Modul in Echtzeit.

Damit konnte gezeigt werden, dass sich Mastery-Werte abhängig von den erkannten Skills automatisch aktualisieren lassen und dem Tutor zur Steuerung seines Antwortverhaltens bereitstehen. Auch wenn die Qualität der Klassifikationen und Updates noch nicht validiert ist, bildet der Prototyp eine belastbare Grundlage für künftige Arbeiten an einem adaptiven KI-Tutor.

7.2 Limitationen

Die Evaluation hat die Grenzen des aktuellen Ansatzes deutlich gemacht. Der durchgeführte Benchmark beruht auf einer Kette mehrerer LLMs: Zunächst wählt ein Modell zufällig Skills aus und simuliert eine studentische Eingabe, anschließend soll der Klassifikator diese Skills erkennen und ein BKT-Update auslösen. Dieses Vorgehen kann die eigentliche Klassifikationsleistung nur sehr eingeschränkt bewerten und ist anfällig für Fehler in jeder vorgelagerten Stufe. Entsprechend lassen sich aus dem Benchmark keine belastbaren Aussagen darüber ableiten, ob die Mastery-Werte inhaltlich korrekt angepasst werden.

Hinzu kommt, dass weder der Skill-Tree noch die Systemprompts auf fachlich geprüften Daten beruhen. Die Skilllisten wurden prototypisch und ohne Expertenvalidierung erstellt; für einen produktiven Einsatz müssten sie von Fachpersonen entwickelt und gepflegt werden. Auch die Formulierungen der Systemprompts wurden ohne tatsächliche Kenntnisse erstellt und sind nicht systematisch getestet. Schließlich fehlt bislang ein gezieltes Training oder Fine-Tuning des Klassifikators auf echten Studierendendaten, was die Robustheit erheblich steigern könnte.

7.3 Ausblick

Trotz der aufgezeigten Einschränkungen verdeutlicht der entwickelte Prototyp das Potenzial adaptiver KI-Tutoren im Hochschulkontext. Aufbauend auf den hier implementierten Komponenten eröffnen sich verschiedene Entwicklungsmöglichkeiten. Zunächst könnte die derzeit rein manuell erstellte Skillmodellierung durch hybride Verfahren ergänzt werden, die auf Basis realer Nutzungsdaten automatisch Vorschläge für Q-Matrizen oder Skill-Cluster generieren. Auf diese Weise ließe sich die Skalierbarkeit eines solchen Systems erhöhen, ohne die notwendige Transparenz gegenüber Lehrenden zu verlieren.

Darüber hinaus erscheint es sinnvoll, die Evaluation nicht länger auf generierten Prompts aufzubauen, sondern mit echten Studierendendaten durchzuführen. Ein Benchmark auf realen Eingaben würde nicht nur die Validität der Klassifikationen erhöhen, sondern auch eine gezielte Feinabstimmung des Klassifikators erlauben, um seine Präzision bei der Skill-Erkennung deutlich zu verbessern.

Ein weiterer vielversprechender Ansatz wäre die Individualisierung der BKT-Parameter. Wie Yudelson u. a. [41] zeigen, führt eine studentenspezifische Parametrisierung zu messbaren Verbesserungen bei der Vorhersage von Lernergebnissen[1]. Eine Anpassung von Lernrate, Guess- und Slip-Werten an einzelne Nutzer*innen könnte somit die Prognosequalität und Adaptivität des Systems erheblich steigern. Zugleich stoßen klassische BKT-Modelle bei groß skalierten Anwendungen an ihre Grenzen, da sie keine Beziehungen zwischen unterschiedlichen Skills abbilden. Hier legen Arbeiten wie Käser u. a. nahe, dass hierarchische oder graphbasierte Modelle[2] eine sinnvolle Weiterentwicklung darstellen könnten, um komplexe Skillstrukturen abzubilden.

Schließlich ließe sich das System auch in seiner Modalität erweitern. Künftige Tutorensysteme könnten Text, Code, Diagramme oder Audioausgaben kombinieren, sodass der KI-Tutor nicht nur in Worten erklärt, sondern automatisch Skizzen oder Graphen generiert, um Konzepte zu visualisieren. Ebenso ließe sich die in dieser Arbeit skizzierte „Tutoring-Policy“, bei welcher Fokus-Skills und Hilfestufen automatisch bestimmt werden, implementieren und empirisch evaluieren. Auf diese Weise könnte der Antwortstil des Tutors reproduzierbarer und lernförderlicher gesteuert werden.

Insgesamt stellt der entwickelte Prototyp damit keinen fertigen Tutor dar, sondern einen „Proof of Concept“. Er zeigt, dass sich Skill-Erkennung und BKT technisch mit einem lokal betriebenen LLM verknüpfen lassen und damit eine Grundlage für datenschutzkonforme, adaptive KI-Tutoren geschaffen werden kann. Mit gezielter Datenbasis, validierter Skillmodellierung und systematischer Evaluation ließe sich dieser Ansatz in Zukunft zu einem robusten System ausbauen, welches Studierende individualisiert, transparent und wirksam unterstützen kann.

[1] Dies stellt Yudelson selbst allerdings in einer 3 Jahre später erschienenen Veröffentlichung [40] wieder in Frage.

[2] z.B. Deep Knowledge Tracing [27]

Literaturverzeichnis

[1] : *Evaluating LLM Answers with the Groundedness Score | deepset Blog*. – URL https://www.deepset.ai/blog/rag-llm-evaluation-groundedness. – Zugriffsdatum: 2025-09-27

[2] : *Ground responses using RAG | Generative AI on Vertex AI*. – URL https://cloud.google.com/vertex-ai/generative-ai/docs/grounding/ground-responses-using-rag. – Zugriffsdatum: 2025-09-27

[3] *Bayesian knowledge tracing*. Juni 2025. – URL https://en.wikipedia.org/w/index.php?title=Bayesian_knowledge_tracing&oldid=1296367418. – Zugriffsdatum: 2025-09-27. – Page Version ID: 1296367418

[4] *F1 Score in Machine Learning*. Juli 2025. – URL https://www.geeksforgeeks.org/machine-learning/f1-score-in-machine-learning/. – Zugriffsdatum: 2025-09-26. – Section: Machine Learning

[5] : *Retrieval Augmented Generation (RAG) für LLMs – Nextra*. Juli 2025. – URL https://www.promptingguide.ai/de/research/rag. – Zugriffsdatum: 2025-09-27

[6] 3Blue1Brown: *Transformers, the tech behind LLMs | Deep Learning Chapter 5*. April 2024. – URL https://www.youtube.com/watch?v=wjZofJX0v4M. – Zugriffsdatum: 2025-09-27

[7] Bodonhelyi, Anna ; Bozkir, Efe ; Yang, Shuo ; Kasneci, Enkelejda ; Kasneci, Gjergji: *User Intent Recognition and Satisfaction with Large Language Models: A User Study with ChatGPT*. November 2024. – URL http://arxiv.org/abs/2402.02136. – Zugriffsdatum: 2025-09-25. – arXiv:2402.02136 [cs]

[8] Bommasani, Rishi ; Hudson, Drew A. ; Adeli, Ehsan ; Altman, Russ ; Arora, Simran ; Arx, Sydney v. ; Bernstein, Michael S. ; Bohg, Jeannette ; Bosselut, Antoine ; Brunskill, Emma ; Brynjolfsson, Erik ; Buch, Shyamal ;

Card, Dallas ; Castellon, Rodrigo ; Chatterji, Niladri ; Chen, Annie ; Creel, Kathleen ; Davis, Jared Q. ; Demszky, Dora ; Donahue, Chris ; Doumbouya, Moussa ; Durmus, Esin ; Ermon, Stefano ; Etchemendy, John ; Ethayarajh, Kawin ; Fei-Fei, Li ; Finn, Chelsea ; Gale, Trevor ; Gillespie, Lauren ; Goel, Karan ; Goodman, Noah ; Grossman, Shelby ; Guha, Neel ; Hashimoto, Tatsunori ; Henderson, Peter ; Hewitt, John ; Ho, Daniel E. ; Hong, Jenny ; Hsu, Kyle ; Huang, Jing ; Icard, Thomas ; Jain, Saahil ; Jurafsky, Dan ; Kalluri, Pratyusha ; Karamcheti, Siddharth ; Keeling, Geoff ; Khani, Fereshte ; Khattab, Omar ; Koh, Pang W. ; Krass, Mark ; Krishna, Ranjay ; Kuditipudi, Rohith ; Kumar, Ananya ; Ladhak, Faisal ; Lee, Mina ; Lee, Tony ; Leskovec, Jure ; Levent, Isabelle ; Li, Xiang L. ; Li, Xuechen ; Ma, Tengyu ; Malik, Ali ; Manning, Christopher D. ; Mirchandani, Suvir ; Mitchell, Eric ; Munyikwa, Zanele ; Nair, Suraj ; Narayan, Avanika ; Narayanan, Deepak ; Newman, Ben ; Nie, Allen ; Niebles, Juan C. ; Nilforoshan, Hamed ; Nyarko, Julian ; Ogut, Giray ; Orr, Laurel ; Papadimitriou, Isabel ; Park, Joon S. ; Piech, Chris ; Portelance, Eva ; Potts, Christopher ; Raghunathan, Aditi ; Reich, Rob ; Ren, Hongyu ; Rong, Frieda ; Roohani, Yusuf ; Ruiz, Camilo ; Ryan, Jack ; Ré, Christopher ; Sadigh, Dorsa ; Sagawa, Shiori ; Santhanam, Keshav ; Shih, Andy ; Srinivasan, Krishnan ; Tamkin, Alex ; Taori, Rohan ; Thomas, Armin W. ; Tramèr, Florian ; Wang, Rose E. ; Wang, William ; Wu, Bohan ; Wu, Jiajun ; Wu, Yuhuai ; Xie, Sang M. ; Yasunaga, Michihiro ; You, Jiaxuan ; Zaharia, Matei ; Zhang, Michael ; Zhang, Tianyi ; Zhang, Xikun ; Zhang, Yuhui ; Zheng, Lucia ; Zhou, Kaitlyn ; Liang, Percy: *On the Opportunities and Risks of Foundation Models*. Juli 2022. – URL http://arxiv.org/abs/2108.07258. – Zugriffsdatum: 2025-09-27. – arXiv:2108.07258 [cs]

[9] Brown, Tom ; Mann, Benjamin ; Ryder, Nick ; Subbiah, Melanie ; Kaplan, Jared D. ; Dhariwal, Prafulla ; Neelakantan, Arvind ; Shyam, Pranav ; Sastry, Girish ; Askell, Amanda ; Agarwal, Sandhini ; Herbert-Voss, Ariel ; Krueger, Gretchen ; Henighan, Tom ; Child, Rewon ; Ramesh, Aditya ; Ziegler, Daniel ; Wu, Jeffrey ; Winter, Clemens ; Hesse, Chris ; Chen, Mark ; Sigler, Eric ; Litwin, Mateusz ; Gray, Scott ; Chess, Benjamin ; Clark, Jack ; Berner, Christopher ; McCandlish, Sam ; Radford, Alec ; Sutskever, Ilya ; Amodei, Dario: Language Models are Few-Shot Learners. In: *Advances in Neural Information Processing Systems* Bd. 33, Curran Associates, Inc., 2020, S. 1877–1901. – URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bf

b8ac142f64a-Abstract.html. – Zugriffsdatum: 2025-09-27

[10] Cao, Yihan ; Li, Siyu ; Liu, Yixin ; Yan, Zhiling ; Dai, Yutong ; Yu, Philip S. ; Sun, Lichao: *A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT*. März 2023. – URL http://arxiv.org/abs/2303.04226. – Zugriffsdatum: 2025-09-27. – arXiv:2303.04226 [cs]

[11] Corbett, Albert T. ; Anderson, John R.: Knowledge tracing: Modeling the acquisition of procedural knowledge. In: *User Modeling and User-Adapted Interaction* 4 (1994), Dezember, Nr. 4, S. 253–278. – URL https://doi.org/10.1007/BF01099821. – Zugriffsdatum: 2025-09-27. – ISSN 1573-1391

[12] Cutler, Ella ; Levonian, Zachary ; Christie, S. T.: *Detecting Student Intent for Chat-Based Intelligent Tutoring Systems*. Februar 2025. – URL http://arxiv.org/abs/2502.15096. – Zugriffsdatum: 2025-09-11. – arXiv:2502.15096 [cs]

[13] Gerner, Assaf ; Madvil, Netta ; Barak, Nadav ; Zaikman, Alex ; Liberman, Jonatan ; Hamra, Liron ; Brazilay, Rotem ; Tsadok, Shay ; Friedman, Yaron ; Harow, Neal ; Bresler, Noam ; Chorev, Shir ; Tannor, Philip: *Grounded in Context: Retrieval-Based Method for Hallucination Detection*. April 2025. – URL http://arxiv.org/abs/2504.15771. – Zugriffsdatum: 2025-09-27. – arXiv:2504.15771 [cs] version: 1

[14] Goutier, Marc ; Diebel, Christopher ; Adam, Martin ; Benlian, Alexander: Humans Over-rely On Help From Artificial Intelligence In Problem-Solving. In: *ECIS 2025 Proceedings*. Amman, Jordan, Juni 2025. – URL https://aisel.aisnet.org/ecis2025/hci/hci/5. – Zugriffsdatum: 2025-07-10

[15] Górski, Franciszek ; Wysocki, Oskar ; Valentino, Marco ; Freitas, Andre: *Integrating Expert Knowledge into Logical Programs via LLMs*. Mai 2025. – URL http://arxiv.org/abs/2502.12275. – Zugriffsdatum: 2025-09-27. – arXiv:2502.12275 [cs]

[16] Kaplan, Jared ; McCandlish, Sam ; Henighan, Tom ; Brown, Tom B. ; Chess, Benjamin ; Child, Rewon ; Gray, Scott ; Radford, Alec ; Wu, Jeffrey ; Amodei, Dario: *Scaling Laws for Neural Language Models*. Januar 2020. – URL http://arxiv.org/abs/2001.08361. – Zugriffsdatum: 2025-09-27. – arXiv:2001.08361 [cs]

[17] Kucharavy, Andrei ; Vallez, Cyril ; Percia David, Dimitri: LLMs Protégés: Tutoring LLMs with Knowledge Gaps Improves Student Learning Outcome. In: Kochmar, Ekaterina (Hrsg.) ; Alhafni, Bashar (Hrsg.) ; Bexte, Marie (Hrsg.) ; Burstein, Jill (Hrsg.) ; Horbach, Andrea (Hrsg.) ; Laarmann-Quante, Ronja (Hrsg.) ; Tack, Anaïs (Hrsg.) ; Yaneva, Victoria (Hrsg.) ; Yuan, Zheng (Hrsg.): *Proceedings of the 20th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2025)*. Vienna, Austria : Association for Computational Linguistics, Juli 2025, S. 248–257. – URL https://aclanthology.org/2025.bea-1.19/. – Zugriffsdatum: 2025-09-26. – ISBN 979-8-89176-270-1

[18] Käser, Tanja ; Klingler, Severin ; Schwing, Alexander G. ; Gross, Markus: Beyond Knowledge Tracing: Modeling Skill Topologies with Bayesian Networks. In: Trausan-Matu, Stefan (Hrsg.) ; Boyer, Kristy E. (Hrsg.) ; Crosby, Martha (Hrsg.) ; Panourgia, Kitty (Hrsg.): *Intelligent Tutoring Systems*. Cham : Springer International Publishing, 2014, S. 188–198. – ISBN 978-3-319-07221-0

[19] Lewis, Patrick ; Perez, Ethan ; Piktus, Aleksandra ; Petroni, Fabio ; Karpukhin, Vladimir ; Goyal, Naman ; Küttler, Heinrich ; Lewis, Mike ; Yih, Wen-tau ; Rocktäschel, Tim ; Riedel, Sebastian ; Kiela, Douwe: *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. April 2021. – URL http://arxiv.org/abs/2005.11401. – Zugriffsdatum: 2025-09-27. – arXiv:2005.11401 [cs]

[20] Magesh, Varun ; Surani, Faiz ; Dahl, Matthew ; Suzgun, Mirac ; Manning, Christopher D. ; Ho, Daniel E.: Hallucination-Free? Assessing the Reliability of Leading AI Legal Research Tools. In: *Journal of Empirical Legal Studies* 22 (2025), Juni, Nr. 2, S. 216–242. – URL https://onlinelibrary.wiley.com/doi/10.1111/jels.12413. – Zugriffsdatum: 2025-09-27. – ISSN 1740-1453, 1740-1461

[21] Maret, Adrien: *Inside LLMs: understanding tokens*. Juni 2024. – URL https://gen-ai.fr/en/blog/atelier/large-language-models/inside-llm-understanding-tokens/. – Zugriffsdatum: 2025-09-27

[22] Menzel, Christoph ; Winkler, Christian: Zur Diskussion der Effekte Künstlicher Intelligenz in der wirtschaftswissenschaftlichen Literatur. (2018), August. – URL https://www.bundeswirtschaftsministerium.de/Redaktion/DE/Downloads/Diskussionspapiere/20190205-diskussionspapier-effekte

-kuenstlicher-intelligenz-in-der-wirtschaftswissenschaftlichen-literatur.pdf?__blob=publicationFile&v=6

[23] Mienye, Ibomoiye D. ; Swart, Theo G.: ChatGPT in Education: A Review of Ethical Challenges and Approaches to Enhancing Transparency and Privacy. In: *Procedia Computer Science* 254 (2025), Januar, S. 181–190. – URL https://www.sciencedirect.com/science/article/pii/S1877050925004272. – Zugriffsdatum: 2025-07-12. – ISSN 1877-0509

[24] Neshaei, Seyed P. ; Davis, Richard L. ; Hazimeh, Adam ; Lazarevski, Bojan ; Dillenbourg, Pierre ; Käser, Tanja: *Towards Modeling Learner Performance with Large Language Models*. Februar 2024. – URL http://arxiv.org/abs/2403.14661. – Zugriffsdatum: 2025-09-06. – arXiv:2403.14661 [cs]

[25] Ouyang, Long ; Wu, Jeff ; Jiang, Xu ; Almeida, Diogo ; Wainwright, Carroll L. ; Mishkin, Pamela ; Zhang, Chong ; Agarwal, Sandhini ; Slama, Katarina ; Ray, Alex ; Schulman, John ; Hilton, Jacob ; Kelton, Fraser ; Miller, Luke ; Simens, Maddie ; Askell, Amanda ; Welinder, Peter ; Christiano, Paul ; Leike, Jan ; Lowe, Ryan: *Training language models to follow instructions with human feedback*. März 2022. – URL http://arxiv.org/abs/2203.02155. – Zugriffsdatum: 2025-09-27. – arXiv:2203.02155 [cs]

[26] Pavlik, Philip I. ; Cen, Hao ; Koedinger, Kenneth R.: Performance Factors Analysis – A New Alternative to Knowledge Tracing. (2009). – URL https://pact.cs.cmu.edu/pubs/AIED%202009%20final%20Pavlik%20Cen%20Keodinger%20corrected.pdf. – Zugriffsdatum: 2025-09-27

[27] Piech, Chris ; Spencer, Jonathan ; Huang, Jonathan ; Ganguli, Surya ; Sahami, Mehran ; Guibas, Leonidas ; Sohl-Dickstein, Jascha: *Deep Knowledge Tracing*. Juni 2015. – URL http://arxiv.org/abs/1506.05908. – Zugriffsdatum: 2025-09-06. – arXiv:1506.05908 [cs]

[28] Raman, Raghu ; Kowalski, Robin ; Achuthan, Krishnashree ; Iyer, Akshay ; Nedungadi, Prema: Navigating artificial general intelligence development: societal, technological, ethical, and brain-inspired pathways. In: *Scientific Reports* 15 (2025), März, Nr. 1, S. 8443. – URL https://www.nature.com/articles/s41598-025-92190-7. – Zugriffsdatum: 2025-07-14. – Publisher: Nature Publishing Group. – ISSN 2045-2322

[29] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence, Global Edition : A Modern Approach*. Pearson Deutschland, Mai 2021. – URL https://elibrary.pearson.de/book/99.150005/9781292401171. – ISBN ISBN 9781292401133

[30] Scarlatos, Alexander ; Lan, Andrew: *Exploring Knowledge Tracing in Tutor-Student Dialogues*. September 2024. – URL http://arxiv.org/abs/2409.16490. – Zugriffsdatum: 2025-09-26. – arXiv:2409.16490 [cs] version: 1

[31] Seemann, Michael: Künstliche Intelligenz, Large Language Models, ChatGPT und die Arbeitswelt der Zukunft. URL https://www.boeckler.de/de/faust-detail.htm?sync_id=HBS-008697, September 2023. – Forschungsbericht

[32] Sultania, Dewang ; Lu, Zhaoyu ; Naik, Twisha ; Dernoncourt, Franck ; Yoon, David S. ; Sharma, Sanat ; Bui, Trung ; Gupta, Ashok ; Vatsa, Tushar ; Suresha, Suhas ; Verma, Ishita ; Belavadi, Vibha ; Chen, Cheng ; Friedrich, Michael: *Domain-specific Question Answering with Hybrid Search*. Dezember 2024. – URL http://arxiv.org/abs/2412.03736. – Zugriffsdatum: 2025-09-27. – arXiv:2412.03736 [cs] version: 1

[33] Team, Gradio: *Quickstart*. – URL https://www.gradio.app/guides/quickstart. – Zugriffsdatum: 2025-09-28

[34] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: *Attention Is All You Need*. Juni 2017. – URL http://arxiv.org/abs/1706.03762. – Zugriffsdatum: 2025-09-27. – arXiv:1706.03762 [cs]

[35] Vieriu, Aniella M. ; Petrea, Gabriel: The Impact of Artificial Intelligence (AI) on Students' Academic Development. In: *Education Sciences* 15 (2025), März, Nr. 3, S. 343. – URL https://www.mdpi.com/2227-7102/15/3/343. – Zugriffsdatum: 2025-07-11. – Number: 3 Publisher: Multidisciplinary Digital Publishing Institute. – ISSN 2227-7102

[36] Völschow, Marcel: *Dr. Marcel Völschow — HAW Hamburg, Beschäftigte Detailseite*. – URL https://www.haw-hamburg.de/hochschule/beschaeftigte/detail/person/person/show/marcel-voelschow/253/. – Zugriffsdatum: 2025-09-26

[37] Völschow, Marcel: *Neural Networks in Data Science: Session 1: Introduction*. April 2024. – URL https://hawhamburgedu.sharepoint.com/:b:/s/SoS

e24_NNDS/EeUAeQqAVnVPptsYSahvF1MBk2MBSP4aXErr-FQnzWEqHQ?e=GykaEQ. – Zugriffsdatum: 2025-09-27

[38] Wang, Jin ; Fan, Wenxiang: The effect of ChatGPT on students' learning performance, learning perception, and higher-order thinking: insights from a meta-analysis. In: *Humanities and Social Sciences Communications* 12 (2025), Mai, Nr. 1, S. 621. – URL https://www.nature.com/articles/s41599-025-04787-y. – Zugriffsdatum: 2025-07-12. – Publisher: Palgrave. – ISSN 2662-9992

[39] Wang, Rose E. ; Zhang, Qingyang ; Robinson, Carly ; Loeb, Susanna ; Demszky, Dorottya: *Bridging the Novice-Expert Gap via Models of Decision-Making: A Case Study on Remediating Math Mistakes*. April 2024. – URL http://arxiv.org/abs/2310.10648. – Zugriffsdatum: 2025-09-26. – arXiv:2310.10648 [cs] version: 3

[40] Yudelson, Michael V.: Individualizing Bayesian Knowledge Tracing. Are Skill Parameters More Important than Student Parameters? / International Educational Data Mining Society. URL https://eric.ed.gov/?id=ED592687. – Zugriffsdatum: 2025-09-06, 2016. – Forschungsbericht. ERIC Number: ED592687

[41] Yudelson, Michael V. ; Koedinger, Kenneth R. ; Gordon, Geoffrey J.: Individualized Bayesian Knowledge Tracing Models. In: Lane, H. C. (Hrsg.) ; Yacef, Kalina (Hrsg.) ; Mostow, Jack (Hrsg.) ; Pavlik, Philip (Hrsg.): *Artificial Intelligence in Education*. Berlin, Heidelberg : Springer, 2013, S. 171–180. – ISBN 978-3-642-39112-5

[42] Zhai, Chunpeng ; Wibowo, Santoso ; Li, Lily D.: The effects of over-reliance on AI dialogue systems on students' cognitive abilities: a systematic review. In: *Smart Learning Environments* 11 (2024), Juni, Nr. 1, S. 28. – URL https://slejournal.springeropen.com/articles/10.1186/s40561-024-00316-7. – Zugriffsdatum: 2025-07-11. – ISSN 2196-7091

[43] Zhang, Peitian ; Liu, Zheng ; Xiao, Shitao ; Dou, Zhicheng ; Yao, Jing: *Hybrid Inverted Index Is a Robust Accelerator for Dense Retrieval*. Oktober 2023. – URL http://arxiv.org/abs/2210.05521. – Zugriffsdatum: 2025-09-27. – arXiv:2210.05521 [cs]

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug zur Erstellung dieses Dokuments
Visual Studio Code	Entwicklungsumgebung für Python- und Shellskripte
Python 3.11 mit Bibliotheken	Umsetzung der Chat-Applikation, des BKT-Moduls und der Auswertungen
llama-server / llama.cpp	Lokaler LLM-Server zur Bereitstellung des Sprachmodells
Git	Versionsverwaltung des Codes
Zotero	Verwaltung von Literaturquellen und Zitaten
LLM-gestützte Schreib- und Codehilfe (z.B. ChatGPT)	Unterstützung bei der Ideenfindung und beim Debuggen von Code, ohne eigenständige Autorenschaft

A.2 Bilder

A.2.1 Umfrageergebnisse des Emmy Feldexperiments

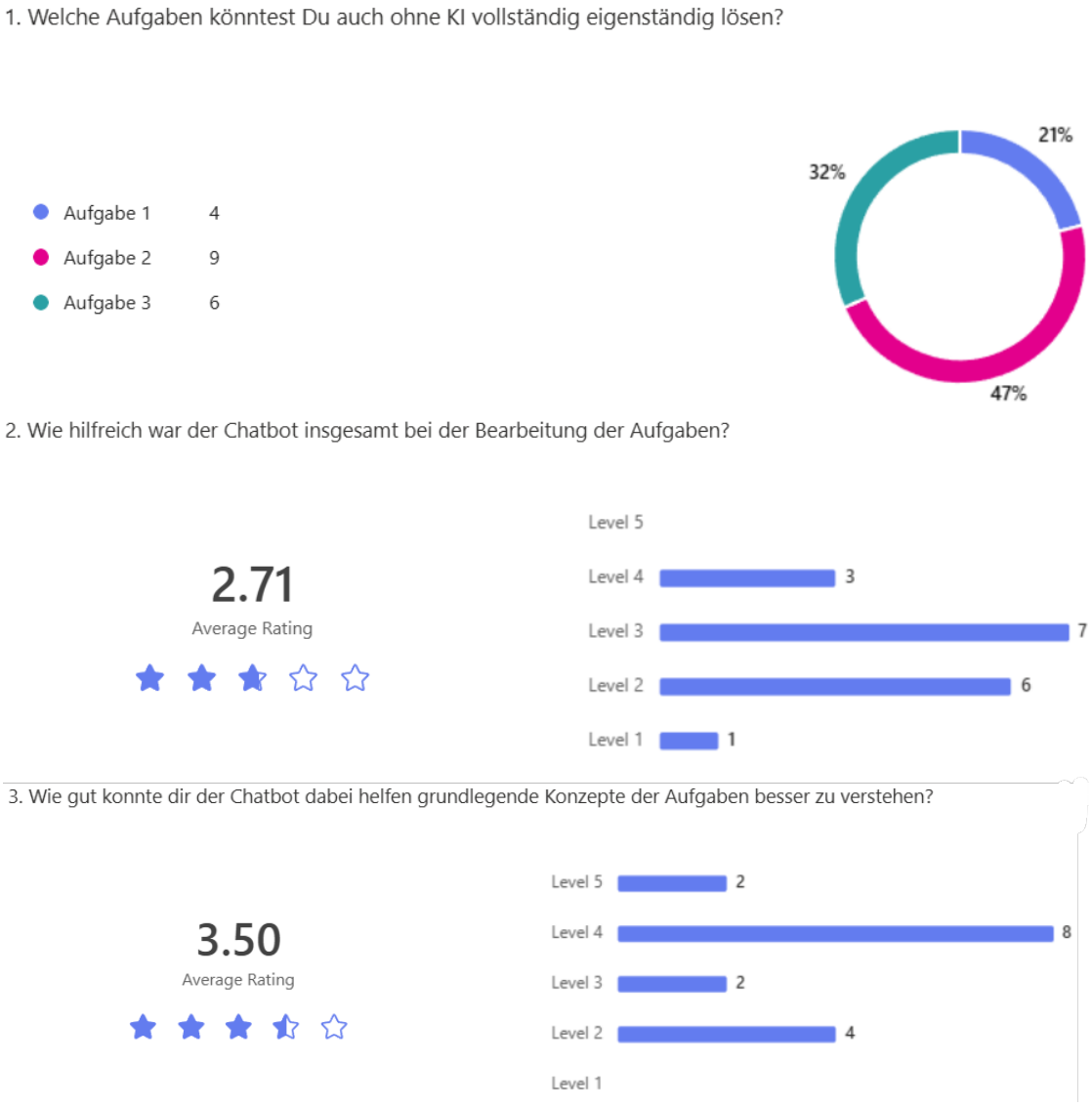


Abbildung A.1: Umfrageergebnisse des Emmy Feldexperiments

Abbildung A.1: Umfrageergebnisse des Emmy Feldexperiments (Fortsetzung)

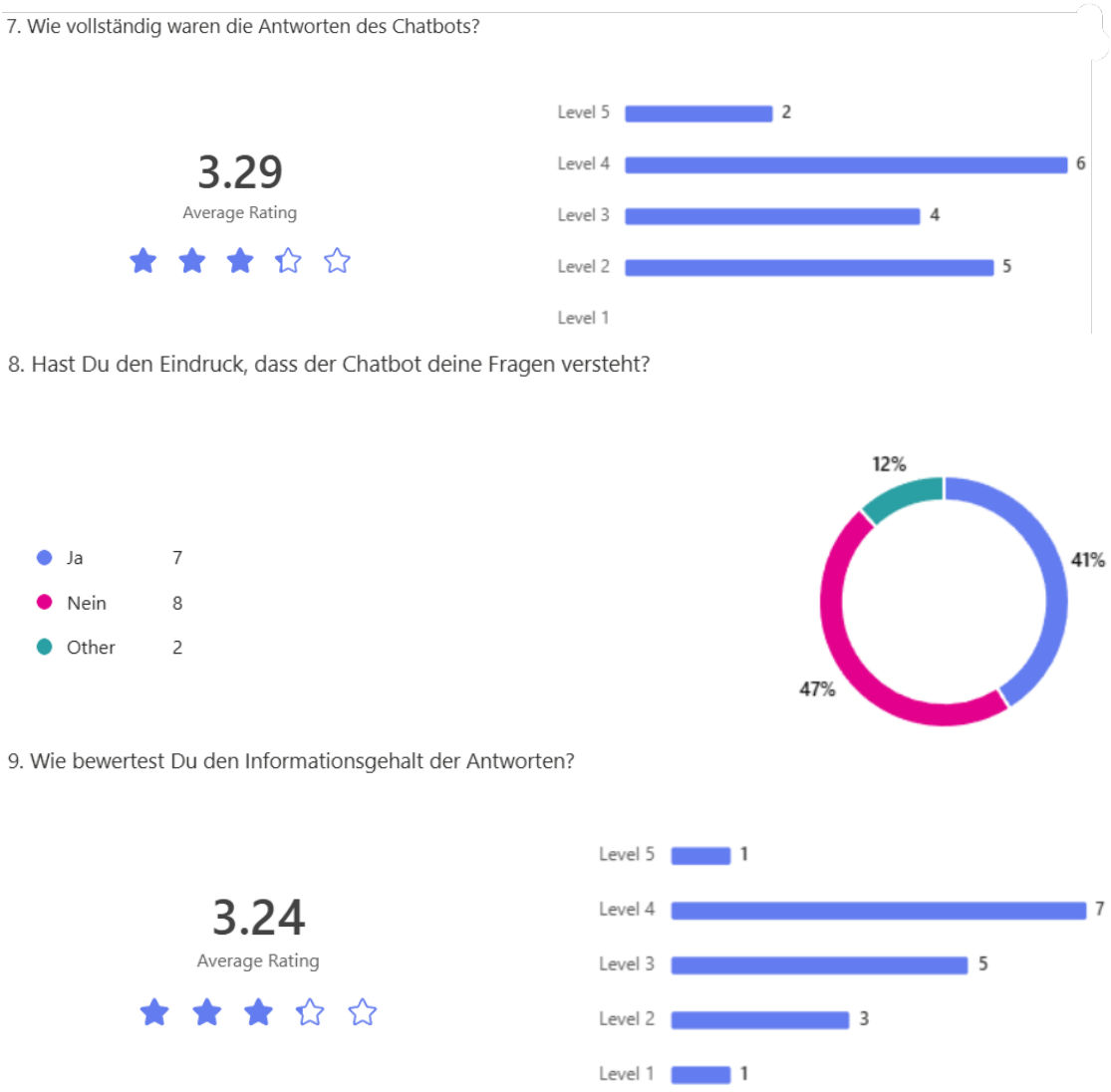
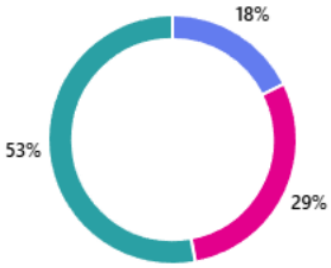


Abbildung A.1: Umfrageergebnisse des Emmy Feldexperiments (Fortsetzung)

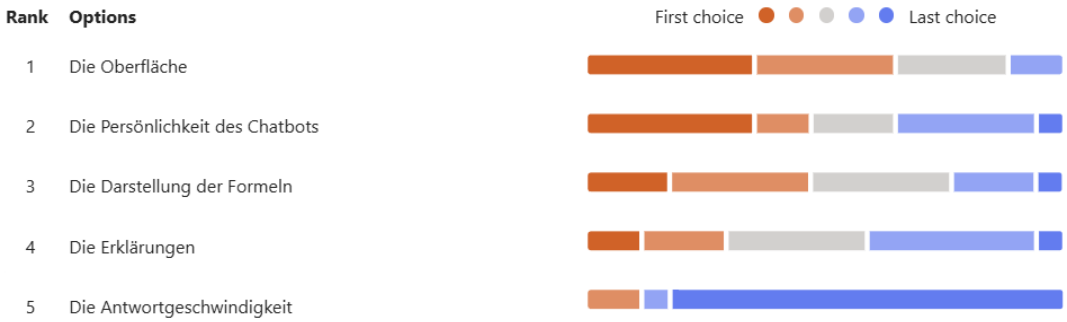
10. Musstest Du deine Anfragen umformulieren, um gute Antworten zu erhalten?

Nie	3
Einmal	5
Mehr als einmal	9
Jedes mal	0
Other	0



11. Wie zufrieden warst du mit den folgenden Aspekten des Chatbots? Sortiere. Beginne mit dem besten Aspekt.

17 Responses



12. Was möchtest Du uns noch mitteilen?

6 Responses

ID ↑	Name	Responses
1	anonymous	Die Werte waren Quatsch
2	anonymous	Kennt seine Grenzen nicht. Versucht alles zu beantworten und korrigiert nicht.
3	anonymous	Er kann sehr schöne ASCII Code Zeichnungen machen, allerdings leider nur eine Art von Gesicht
4	anonymous	Wenn man Fragen zu speziellen Aufgaben gestellt hat, wurde nicht zwangsläufig eine Erklärung gegeben, die zu der spezifischen Aufgabe gepasst hat. Die Zahlen, mit denen gerechnet wird, waren häufig nicht die der Aufgabe.
5	anonymous	Gutes Modell, ab und zu jedoch falsche Werte benutzt. Je länger und genauer formuliert meine prompts waren, desto länger und ausführlicher wurden die antworten der KI
6	anonymous	Idee ist super, durch verlorenen Kontext werden ab und zu bestimmte Größen "vergessen", vielleicht einen bestimmten Prompt-Typus zur Eingabe der Aufgabenstellung

Abbildung A.1: Umfrageergebnisse des Emmy Feldexperiments (Fortsetzung)

A.3 Programmcode und Benchmark

Der vollständige Programmcode und die Benchmarkergebnisse befinden sich auf CD, einzusehen bei Prof. Buczek (Erstprüfer).

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original