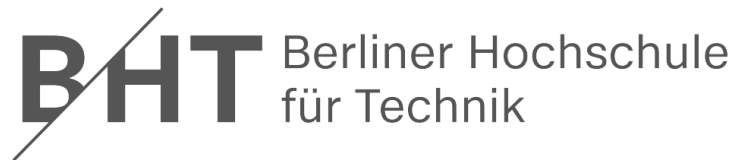


Evaluierung und Optimierung von Large Language Models für die Entwicklung von Webanwendungen

Ein Ansatz zur Verbesserung des Entwicklungsprozesses bei Softwareprojekten



Masterthesis

für den angestrebten akademischen Grad
Master of Science im Studiengang Medieninformatik

Eingereicht von: Wilfried Pahl
Matrikelnummer: 901932
Studiengang: Online Medieninformatik
Berliner Hochschule für Technik

Betreuer Prof. Dr. S. Edlich
Berliner Hochschule für Technik
Gutachter Prof. Dr. Alexander Löser
Berliner Hochschule für Technik

Temmen-Ringenwalde, der 8. Januar 2025

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel „Evaluierung und Optimierung von Large Language Models für die Entwicklung von Webanwendungen (*Ein Ansatz zur Verbesserung des Entwicklungsprozesses bei Softwareprojekten*)“ selbstständig und ohne unerlaubte Hilfe verfasst habe. Alle benutzten Quellen und Hilfsmittel sind vollständig angegeben und wurden entsprechend den wissenschaftlichen Standards zitiert.

Ich versichere, dass alle Passagen, die nicht von mir stammen, als Zitate gekennzeichnet wurden und dass alle Informationen, die ich aus fremden Quellen übernommen habe, eindeutig als solche kenntlich gemacht wurden. Insbesondere wurden alle Texte und Textpassagen anderer Autoren sowie die Ergebnisse von Sprachmodellen wie OpenAI's GPT-3 entsprechend den wissenschaftlichen Standards zitiert und referenziert.

Ich versichere weiterhin, dass ich keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe und dass ich keine Teile dieser Arbeit in anderer Form für Prüfungszwecke vorgelegt habe.

Mir ist bewusst, dass eine falsche eidesstattliche Erklärung strafrechtliche Konsequenzen haben kann.

Temmen-Ringenwalde, 8. Januar 2025

Ort, Datum

, _____
Unterschrift

ABSTRACT

Abstract in Englisch.

ZUSAMMENFASSUNG

Zusammenfassung in Deutsch.

Inhaltsverzeichnis

Abstract	i
Abbildungsverzeichnis	vi
Tabellenverzeichnis	vii
Listings	viii
Abkürzungsverzeichnis	ix
1 Einleitung	1
1.1 Hintergrund und Kontext	1
1.2 Problemstellung	2
1.3 Stand der Forschung	3
1.4 Zielsetzung und Forschungsfragen	3
1.5 Aufbau der Arbeit	4
1.6 Abgrenzung	4
2 Grundlagen	7
2.1 Künstliche Intelligenz	8
2.1.1 Maschinelles Lernen	8
2.1.2 Neuronale Netze	9
2.1.3 Deep Learning	12
2.2 Natural Language Processing	12
2.3 Large Language Model	13
2.3.1 Grundlagen	13
2.3.2 Grenzen und Probleme bei LLMs	14
2.3.3 Verständnis für die LLMs	15
2.4 Koordinationsstrategien für LLMs	15
2.4.1 Orchestrierung von LLMs	16
2.4.2 Multi-Agenten-Systeme	17
2.5 Prompt Engineering	18

2.5.1	Prompt-Techniken	18
2.5.2	Grenzen beim Prompt-Engineering für LLMs	19
2.6	Grundlagen der Webentwicklung	20
2.6.1	Programmiersprachen	20
2.6.2	Entwicklung	20
2.7	Benchmark für LLM	21
3	Konzeption / Design	23
3.1	Definition der Evaluierungsziele	23
3.2	Auswahl der LLMs und deren Konfiguration	24
3.3	Design der Experimente	27
3.4	Konzeption des Prompt-Engineerings	29
3.5	Evaluationsumgebung	30
4	Implementierung	31
4.1	Lokale Modelle	32
4.1.1	Modellbereitstellung mit Ollama	32
4.1.2	Modellbereitstellung als Datei	32
4.2	Benchmark Codeevaluation	33
4.2.1	Problemabfragen an die Modelle	33
4.2.2	Auswertung der Modellantworten	33
4.3	Codeevaluation mit Frameworks	34
4.3.1	PHP Codeevaluation	34
4.3.2	JavaScript Codeevaluation	35
5	Evaluation	37
5.1	Modellbewertung mit HumanEval Benchmark	38
5.1.1	Nachteile der Evaluierung	40
5.2	Optimierung der Ergebnisse	40
6	Lessons Learned	41
6.1	Erweiterte Codeevaluation	41
6.1.1	PHPUnit	42
6.1.2	PHPMetrics	42
6.1.3	SonarQube	42
7	Diskussion und Ausblick	43
7.1	Impulse für zukünftige Forschungen	44
7.2	Praktische Anwendung	45
7.2.1	Anwendung für Entwickler	45
8	Fazit	47

Literatur	49
Glossar	51
Anhang	53
A Installationshinweise	53
A.1 Python	53
A.2 Installation und Konfiguration von Ollama	53
A.3 Open WebUI Installationshinweise	55
A.4 Download der Hugging Face Modelle	56
A.5 Abfragen lokaler Modelle	58
A.6 Evaluation der Antworten von den Modellen	62
B Fragenkataloge	62
B.1 PHP	63
B.2 JavaScript	63

Abbildungsverzeichnis

2.1	LLMs im Kontext der Forschungsbereiche von KI	7
2.2	Biologische Nervenzelle	10
2.3	Künstliche Nervenzelle	10
3.1	Codegeneration	28

Tabellenverzeichnis

3.1	Auswahl der LLMs für die Evaluierung	26
3.2	Einstellungen der Modellparameter	27

Listings

3.1	Beispiel für einen Test aus dem HumanEval-XL Benchmark	28
3.2	Prompt Beispiel für eine Aufgabe aus dem HumanEval-XL Benchmark	29
4.1	Codesnippet zur Ausführung des PHP Interpreters label	33
4.2	Berechnung der pass@k Metrik in Python	34
5.1	Gemini Ergebnis für das PHP-3 Problem Version 1	39
5.2	Gemini Ergebnisse für das PHP-3 Problem Version 2	39
5.3	Gemini Ergebnisse für das PHP-3 Problem Version 3	40
5.4	Gemini Ergebnisse für das PHP-3 Problem Version 4	40
6.1	Beispiel für Bewertungskriterien	41
8.1	Ollama Hostanpassng für Netzwerkbetrieb	53
8.2	Open WebUI installieren	55
8.3	Laden der Modelle von Hugging Face und lokal speichern	56
8.4	Laden der Modelle von Hugging Face und lokal speichern	57
8.5	Abfragen der Modelle von Hugging Face und Ollama	58
8.6	Evaluation der Modellantworten: Suche nach Codeausschnitten	62

1.1 Hintergrund und Kontext

Durch die zunehmende Globalisierung und Digitalisierung wird die Gesellschaft in der Gegenwart und Zukunft geprägt. Der Ausbau von Hochgeschwindigkeitsnetze und die globale Corona-Pandemie haben diese Entwicklung beschleunigt. Immer mehr Unternehmen erkennen die Potenziale der Digitalisierung und passen ihre Geschäftsprozesse an und nutzen die Möglichkeiten der digitalen Systeme. Ganze Wertschöpfungsketten werden auf cloudbasierte Umgebungen umgestellt. Angefangen bei der Kommunikation, über Beschaffung und Produktion bis zum Verkauf der Waren und Dienstleistungen, vergleiche mit [1, Seite 21 ff.] und [2]. In allen Stufen der Prozesse kommen webbasierte Anwendungen zum Einsatz, um die Kommunikation der Anwender mit den Systemen zu ermöglichen oder Schnittstellen für die Datenübertragung zwischen den verschiedenen Systemen zu gewährleisten. Durch die wachsende Anzahl von Web-Anwendungen wächst auch der Druck für die Entwicklungsfirmen, Anwendungen den oft schnell und wechselnden Kundenanforderungen anzupassen.

Durch diesen Prozess getrieben, müssen Entwicklungsfirmen in immer kürzeren Release-Zyklen Softwarekomponenten hinzufügen oder vorhandene erweitern. Gleichzeitig wachsen aber auch die Anforderungen an Stabilität und Sicherheit der cloudbasierten Anwendungen, sowie der Bedarf an kostengünstigeren IT-Abläufen. Ein weiteres Problem ist der wachsende Fachkräftemangel in der Wirtschaft und die damit verbundenen steigenden Gehälter der Entwickler.

Die Verwendung künstlicher Intelligenz bei der Programmierung gewinnt immer mehr an Bedeutung. Eine Technologie die im besonderen Maße an dieser Entwicklung beteiligt ist, sind die Large Language Models. Insbesondere mit der Veröffentlichung vom ChatGPT wurde hier ein regelrechter Hype um die LLMs ausgelöst. Diese Modelle erlauben eine Softwareentwicklung mit natürlicher

Sprache. Dadurch sind viele Nutzer in der Lage Programmcode zu erzeugen und diesen in vorhandene Software zu integrieren. Oft haben die Nutzer aber keine oder wenig Erfahrungen in der Softwareentwicklung und die damit verbundenen Kenntnisse der Programmierung.

1.2 Problemstellung

So groß der Hype um künstliche Intelligenz auch sein mag, zurzeit kann KI nicht alle Anforderungen selbstständig lösen. Dies sollte auch bei der Verwendung von KI generierten Inhalten und Programmcodes beachtet werden.

KI denkt nicht, KI trifft keine Entscheidungen. Eine KI antwortet auf eine Eingabe nicht mit der besten Antwort, sondern mit der Wahrscheinlichsten.

VATTENFALL ONLINE , KI für Unternehmen – Die Grenzen der KI

Der Nutzer muss die generierten Ergebnisse überprüfen, ehe erstellte Programmcodestücke in vorhandene Programme eingefügt und in Produktionsumgebungen implementiert werden. Den im Gegensatz zur natürlichen Sprache, ist bei Problemen der Codegenerierung, die Syntax der jeweiligen Programmiersprache einzuhalten. Andernfalls kann es zu Laufzeitfehlern kommen oder einem unerwarteten Verhalten der Software führen.

Viele Entwickler setzen auf Chatbots, wie ChatGPT oder Gemini zur Generierung von Code, wie eine Umfrage von *stackoverflow* vom Mai 2024 zeigt [3]. Wenn der generierte Code ohne Prüfung und Tests in bestehende Projekte implementiert wird, kann dies dazu führen, dass sich unter anderem technische Schulden anhäufen. Dadurch erhöhen sich langfristig die Wartungsaufwände und das Hinzufügen von Erweiterungen ist ebenfalls mit erhöhtem Aufwand verbunden.

Ein weiteres Problem der generierten Codes sind die vorhandenen Sicherheitslücken. Werden diese fehlerhaften Codes übernommen, ist es oft ein leichtes für Angreifer sensible Kundendatendaten zu stehlen. In der Arbeit [4] wird das Thema Schwachstellen in von ChatGPT generiertem PHP-Code evaluiert und ebenfalls vor der sorglosen Verwendung gewarnt. Inwieweit die erstellten Codes Auswirkungen auf echte Webseiten haben, ist zurzeit noch nicht hinreichend untersucht.

Viele Entwickler und Nichtentwickler sind sich dieser mangelhaften generierten Code nicht bewusst und verwenden diese ohne weitere Prüfung. Hinzu kommt das in den meisten Fällen nur ein Modell befragt wird, nicht aber ein zweites oder die Prüfung durch ein weiteres erfolgt.

1.3 Stand der Forschung

Gerade in den letzten Monaten sind viele Forschungsfelder zum Thema Sprachmodelle hinzugekommen. Diese befassen sich mit der Optimierung und effizienter Nutzung der Modelle bei der Generierung von Codes.

In [5] wird eine bis dato fehlende Literaturrecherche zum Thema „Codegenerierung durch große Sprachmodelle“ bemängelt, was in dieser Arbeit nachgeholt wird und im Juni 2024 wurde die Literatur zusammengetragen, welche sich mit Codegenerierung befasst.

Um die Prompts im Ingenieurwesen zu optimieren, wird in [6] die GPEI (Goal Prompt Evaluation Iteration) Methodik vorgeschlagen, welche aus vier Schritten besteht. Zuerst wird das Ziel definiert, dann ein Entwurf der Anforderung, im Anschluss die Bewertung gefolgt von der Iterationen.

Es gibt Bestrebungen kleinere Modelle, die auf Codegenerierung spezialisiert sind zu verwenden, um große teure Sprachmodelle zu ersetzen, so auch in [7]. Hier werden die Modelle als „Granite Code Models“-Familie zusammengefasst. Eine weitere Arbeit die sich mit kleinen Modellen befasst, ist die Arbeit [8] mit StarCoder 2. Dieses kleine Modell wurde speziell für die Generierung von Codes trainiert.

Der wissenschaftliche Artikel [9] befasst die sich ebenfalls mit der Web-Entwicklung mittel GPT-3. Hierbei wird die Verwendung von Generativ Adversarial Networks (GANs) vorgeschlagen, ein neuer Ansatz, mit der die Nachbearbeitung minimiert und die Codequalität optimiert wird.

Eine weitere Arbeit ist [10]. Diese befasst sich mit einer Umfrage zum Thema „Natural Language-to-Code“ und gibt eine Übersicht über 27 Modelle und geben einen Überblick über Benchmarks und Metriken. Hier wird auch der in dieser Arbeit angewandte Benchmark *HumanEval* vorgestellt.

1.4 Zielsetzung und Forschungsfragen

Das Ziel in der Softwareentwicklung war und ist die Optimierung des Entwicklungsprozesses, um Ressourcen und Kosten einzusparen und dadurch einen Wettbewerbsvorteil zu erlangen. Ein Bereich der Besonders stark von der Digitalisierung profitiert, ist der Bereich der Webentwicklung. Durch die steigende Nachfrage von Cloud-Anwendungen steigt auch der Optimierungsdruck in diesem Bereich besonders stark.

Vor diesem Hintergrund lässt sich die Zielsetzung bereits aus dem Titel „*Evaluierung und Optimierung von Large Language Models für die Entwicklung von Webanwendungen*“ dieser Arbeit herleiten. Sie untersucht die Möglichkeiten mit natürlicher Sprache, Code zu generieren. Wie in [5, vgl. Seite 2] wird auch in dieser Arbeit die Kurzform NL2Code, für Language-to-Code verwendet. Diese Arbeit soll eine Auswahl von

Modellen evaluieren und dessen Brauchbarkeit für die Entwicklung von Webanwendungen aufzeigen. Um die Antworten der Modelle zu optimieren, soll eine Evaluation von Methodiken erfolgen, bei der deren Anwendung eine Verbesserung der Antworten ersichtlich ist.

Als erstes Ziel soll jedoch die Evaluierung des generierten Codes der Modelle stehen und dessen Tauglichkeit für die Programmierung untersucht werden. Sobald Code geschrieben wurde und von anderen Programmierenden überarbeitet oder verstanden werden muss, erfordert dies erheblich mehr Zeit als das ursprüngliche Schreiben des Codes. Daher ist es essenziell, dass der Code gut strukturiert ist und durch Kommentare ergänzt wird, um die Verständlichkeit und Wartbarkeit zu erleichtern. Viele Evaluierungsdatensätze prüfen lediglich die korrekte Funktionalität des Codes.

Die drei Ziele dieser Arbeit lassen sich in den folgenden kurz formulierten Sätzen zusammenfassen,

- Z1** Können LLMs hochwertigen Code für die Entwicklung von Webanwendungen generieren und welche Modelle eignen sich für die Entwicklung von Webanwendungen?
- Z2** Wie gut sind die Ergebnisse, hinsichtlich Syntax, Funktionalität, Codequalität und Coding-Standards?
- Z3** Welches Prompt-Design kann helfen die Codequalität und die Genauigkeit der Ergebnisse zu verbessern?

1.5 Aufbau der Arbeit

Um ein grundlegendes Verständnis der Thematik zu gewährleisten, werden die theoretischen Grundlagen in Kapitel 2 ausführlich erläutert.

Kapitel 4 widmet sich der Implementierung, die für die Codegenerierung und die nachfolgende Evaluierung notwendig ist. Die daraus gewonnenen Ergebnisse werden anschließend in Kapitel 5 analysiert und diskutiert.

Die in dieser Arbeit gesammelten positiven und negativen Erfahrungen sowie die aufgetretenen Herausforderungen werden in Kapitel 6 thematisiert. Zudem werden mögliche Lösungsansätze vorgeschlagen.

Abschließend werden in Kapitel 7 die erzielten Ergebnisse eingehend diskutiert und mögliche Anregungen für zukünftige Arbeiten und den praktischen Einsatz vorgestellt, bevor in Kapitel 8 die Arbeit zusammengefasst und ein abschließendes Fazit gezogen wird.

1.6 Abgrenzung

In dieser Arbeit liegt der Fokus auf der Evaluierung und Optimierung von durch Large Language Models (LLMs) generiertem Code im Kontext der Webanwendungsentwicklung, insbesondere in Bezug auf die

verwendeten Programmiersprachen wie JavaScript, HTML, CSS und PHP. Andere Anwendungsbereiche wie die Entwicklung von Desktop-Anwendungen werden nicht explizit untersucht, obwohl mögliche Parallelen und Erkenntnisse in diesen Bereichen nicht ausgeschlossen werden. Diese Arbeit bleibt bewusst auf den Webentwicklungsbereich beschränkt, um eine gezielte Analyse und Optimierung der Prompts zu ermöglichen.

Rechtliche und ethische Überlegungen im Umgang mit Künstlicher Intelligenz sind zweifellos wichtige Themen. Allerdings werden diese Aspekte in der vorliegenden Arbeit nicht behandelt. Es gibt bereits umfassende Literatur zu diesen Themen, die in der Arbeit zur Kenntnis genommen werden, jedoch erfolgt keine vertiefte Auseinandersetzung damit. Der Schwerpunkt liegt vielmehr auf der technischen Evaluierung und der Verbesserung der Prompt-Strategien, ohne dabei Änderungen an den LLMs selbst vorzunehmen, wie beispielsweise Bias-Anpassungen oder Modifikationen am Modell.

Die Arbeit beschränkt sich auf die Anwendung von LLMs im Bereich der Webanwendungsentwicklung. Andere Anwendungsfälle, wie die Generierung von Texten für kreative Inhalte oder wissenschaftliche Artikel, werden nicht in die Untersuchung einbezogen. Ziel ist es, die spezifischen Anforderungen und Herausforderungen der Webentwicklung in den Fokus zu rücken und gezielt Optimierungen für diesen Bereich zu erarbeiten.

Der Schwerpunkt der Optimierung liegt ausschließlich auf der Anpassung und Verfeinerung der Eingabeprompts. Es wird bewusst darauf verzichtet, Änderungen an den zugrundeliegenden Modellarchitekturen, den Trainingsdaten oder der internen Bias-Reduktion der LLMs vorzunehmen. Diese Arbeit konzentriert sich auf die Möglichkeiten, die sich durch die geschickte Gestaltung der Prompts eröffnen, um die Qualität des generierten Codes zu verbessern.

Der Fokus dieser Arbeit liegt auf der technischen Optimierung und Evaluierung der durch LLMs generierten Codes. Aspekte wie Benutzerfreundlichkeit, Design oder User Experience der resultierenden Webanwendungen werden in dieser Untersuchung nicht betrachtet. Ziel ist es, die technische Qualität und Funktionalität des Codes zu analysieren und zu verbessern.

Die Untersuchung konzentriert sich ausschließlich auf deutschsprachige Prompts und die daraus generierten Codes. Andere Sprachen oder Mehrsprachigkeit werden in dieser Arbeit nicht berücksichtigt. Diese Einschränkung ermöglicht eine präzisere Analyse und Vergleichbarkeit der Ergebnisse innerhalb des gewählten Sprachraums.

In diesem Kapitel werden die grundlegenden Konzepte vorgestellt, die für das Verständnis dieser Arbeit relevant sind. Es wird ein Überblick über die wichtigsten Teilgebiete gegeben, wobei die einzelnen Bereiche nur oberflächlich behandelt werden können, um eine grundlegende Einführung zu bieten.

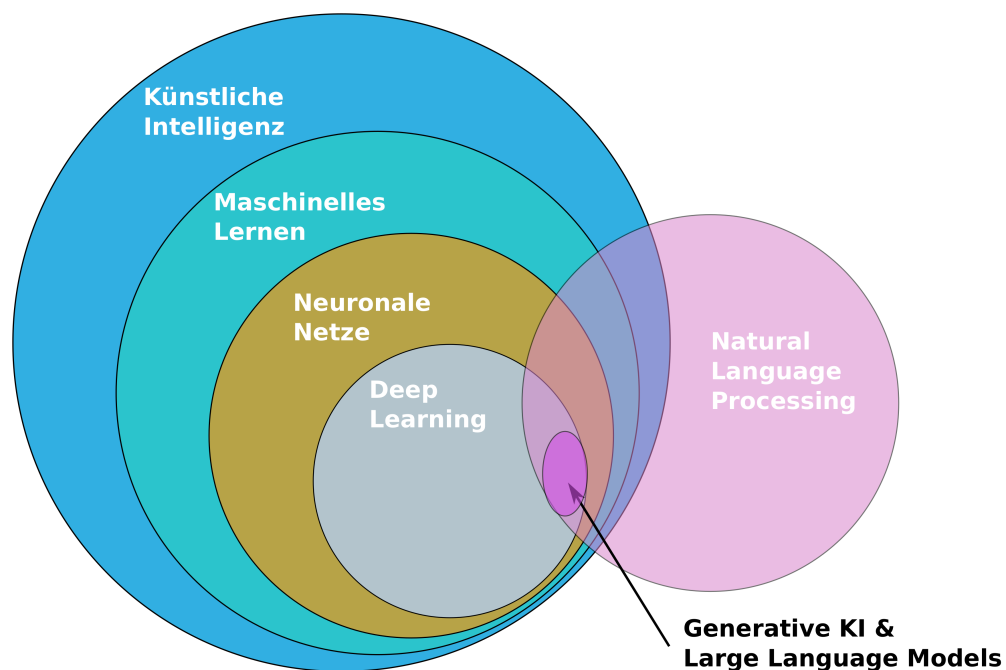


Abbildung 2.1: LLMs im Kontext der Forschungsbereiche von KI

Die Forschungsfelder der großen Sprachmodelle (Large Language Models, LLM) sind eng mit den Teilbereichen Deep Learning und der Verarbeitung natürlicher Sprache (Natural Language Processing, NLP) verbunden. Diese Teilgebiete sind wiederum Unterbereiche des maschinellen Lernens, das selbst ein Teil der umfassenderen Disziplin der künstlichen Intelligenz ist. Die Grafik 2.1 veranschaulicht die sprachliche und

konzeptionelle Einordnung der Begriffe „künstliche Intelligenz“ „maschinelles Lernen“ „neuronale Netze“ „Deep Learning“ und „NLP“.

2.1 Künstliche Intelligenz

Künstliche Intelligenz hat bereits in viele Unternehmensprozesse Einzug gehalten und unterstützt die Nutzer und Systeme. Besonders die generative KI, mit ihren großen Sprachmodellen wird in den nächsten Jahren immer weiter in die Unternehmensbereiche vorstoßen und viele Aufgaben übernehmen. Entscheider und Führungspersonal versprechen sich von der Technologie nicht nur effizientere Prozesse, sondern auch Kosteneinsparungen im Personalbereich.

Eine explizite Definition für *künstliche Intelligenz* ist zurzeit noch nicht einheitlich erfolgt. Geschuldet ist diese Tatsache, dass der Begriff *Intelligenz* nicht eindeutig definiert ist. Somit finden sich viele Versuche eine Definition für künstliche Intelligenz herzuleiten. In dieser Arbeit wird für die künstliche Intelligenz, die Definition aus [11, 6 ff.] verwendet.

Systeme der künstlichen Intelligenz (KI-Systeme) sind vom Menschen entwickelte Softwaresysteme (und gegebenenfalls auch Hardwaresysteme), die in Bezug auf ein komplexes Ziel auf physischer oder digitaler Ebene handeln, indem sie ihre Umgebung durch Datenerfassung wahrnehmen, die gesammelten strukturierten oder unstrukturierten Daten interpretieren, Schlussfolgerungen daraus ziehen oder die aus diesen Daten abgeleiteten Informationen verarbeiten, und über das bestmögliche Handeln zur Erreichung des vorgegebenen Ziels entscheiden. KI-Systeme können entweder symbolische Regeln verwenden oder ein numerisches Modell erlernen, und sind auch in der Lage, die Auswirkungen ihrer früheren Handlungen auf die Umgebung zu analysieren und ihr Verhalten entsprechend anzupassen.

Bitkom e.V.

Aus den Forschungsgebieten der künstlichen Intelligenz sind für die großen Sprachmodelle der Bereich des „Deep Learning“ besonders interessant. Hier findet die Überschneidung mit dem Bereich der NLP statt, welche massiv dazu beitrug, dass die großen Sprachmodelle diesen Erfolg erfahren. In den folgenden Kapiteln wird auf die Teilgebiete eingegangen, in denen „Deep Learning“ angesiedelt ist.

2.1.1 Maschinelles Lernen

Als Teilgebiet der künstlichen Intelligenz befasst sich maschinelles Lernen mit dem Problem wie Maschinen Lernen und Denken können. Wobei hier nicht von selbstständigem Lernen und Denken gesprochen werden kann, sondern lediglich von Imitieren dieser Prozesse. Aber ML ist sehr wohl in der Lage aus großen

Datenmengen komplexe Muster und Funktionen zu erkennen. Für das maschinelle Lernen gibt es mehrere Formen von Lernparadigmen.

Beim *überwachten Lernen* sind für die Eingaben der Trainingsdaten dazugehörige Ausgaben, die Labels definiert. Das Ziel ist es eine Funktion zu trainieren um künftige Eingaben korrekt klassifizieren oder vorhersagen zu können. Dieses Lernparadigma wird häufig eingesetzt, wenn es sich um Regressionens- und Klassifizierungsprobleme handelt.

Die gelabelten Ausgaben sind beim *unüberwachten Lernen* nicht vorhanden. Hierbei wird beispielsweise durch Clustering oder Dimensionsreduktion versucht Muster und Strukturen zu erkennen. Des Weiteren soll die Methode helfen Anomalien in Daten zuerkennen oder Assoziationen zwischen Datenobjekten zu finden.

Das *selbst überwachte Lernen* ermöglicht es Modellen, sich selbst zu überwachen ohne gelabelte Daten. Hierbei lernen die Algorithmen einen Teil der Eingaben von anderen Teilen und generieren automatisch Labels. So werden unüberwachten Problemen in überwachte Probleme überführt. Diese Art des Lernens ist u.a. besonders nützlich bei NLP, da hier die Trainingsdaten in großer Anzahl vorliegen

Beim *verstärkten Lernen* (engl. Reinforcement Learning) werden die Systeme mit Belohnung und Strafe trainiert. Das System wird aufgrund seines Handelns bewertet, dadurch wird es ermutigt gute Praktiken weiterzuverfolgen und schlechte zu verwerfen. Das Lernen wird häufig bei der Videospielentwicklung und in der Robotik eingesetzt.

Das *semi-überwachte Lernen* stellt eine Hybridform aus unüberwachtem und überwachtem Lernen dar. Bei diesem Ansatz werden kleine Mengen gelabelter Datensätze genutzt, um eine große Anzahl ungelabelter Daten effizient zu steuern. Diese Methode ermöglicht es den verwendeten Technologien, neue Inhalte zu generieren und bildet die Grundlage moderner generativer KI-Systeme. Die Bandbreite der Technologien in diesem Bereich reicht von Generative Adversarial Networks (GANs) bis hin zu Diffusionsmodellen, welche wesentliche Fortschritte in der Erzeugung von Texten, Bildern und anderen Inhalten ermöglichen.

2.1.2 Neuronale Netze

Künstliche neuronale Netze (KNN) sind spezifische Algorithmen des maschinellen Lernens, die von der Struktur und der Funktionsweise des menschlichen Gehirns inspiriert sind. Die Abbildung 2.2 von [12] zeigt eine stark vereinfachte biologische Nervenzelle.

Biologische Nervenzellen reagieren auf elektrischen Reize, welche von Sinnesorganen oder anderen Nervenzellen stammen. Die Dendriten nehmen die elektrischen Signale auf und leiten diese an den Zellkern weiter. Dort erfolgt die Zusammenführung der eingehenden Signale und bildet das Aktionspotential. Übersteigt es

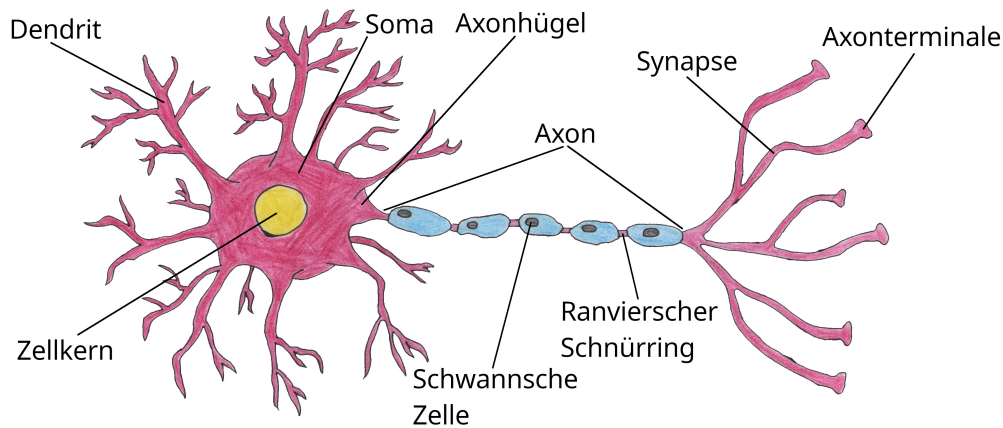


Abbildung 2.2: Biologische Nervenzelle

das Schwellenpotential der Zelle, so wird das Signal über das Axon abgeleitet, die Nervenzelle „*feuert*“. Diese Beschreibung ist die grundlegende Ausführung über die Arbeitsweise der biologischen Nervenzellen und erläutert und beachtet keine tiefer gehenden Prozesse.

Die nachempfundene Nervenzelle stellt die kleinste Einheit in künstlichen neuronalen Netzen und wird als Neuron bezeichnet. Die Abbildung 2.3 zeigt die generelle Funktionsweise eines einfachen Neurons.

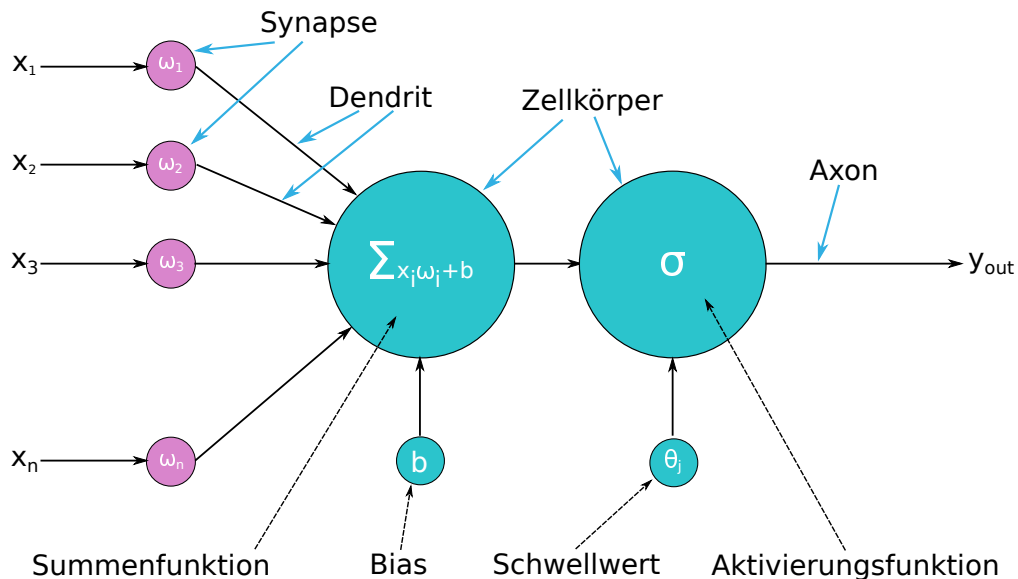


Abbildung 2.3: Künstliche Nervenzelle

Neuronen in neuronalen Netzen erhalten als Eingabewerte typischerweise einen Vektor und liefern ein skalares Ausgangssignal. Abgesehen von der Eingabeschicht, bei der die Eingangssignale direkt aus den Daten

stammen, ist jedes Eingangssignal x_n in den verborgenen und Ausgabeschichten das Ausgangssignal y_{out} eines vorhergehenden Neurons. Die Gewichtungen der Eingangssignale modellieren dabei die synaptischen Verbindungen zwischen biologischen Neuronen, die entweder verstärkend oder hemmend wirken können. Alle gewichteten Eingangssignale werden durch die Summenfunktion aufaddiert, bevor ein Bias hinzugefügt wird. Die Gleichung 2.1 zeigt die Summenfunktion für n Eingangssignale unter Berücksichtigung des Bias-Wertes.

$$y_{sum} = x_1 + x_2 + \dots + x_n + b \quad (2.1)$$

Nach der Summenfunktion wird das Signal an die Aktivierungsfunktion übergeben. Diese Funktion leitet ein Signal erst weiter, wenn ein festgelegter Schwellwert überschritten wird. Die Analogie zur biologischen Nervenzelle ist das Aktionspotential, welches durch die Reize anderer Nervenzellen aufgebaut wird und wie beim künstlichen Neuron führt das Überschreiten eines Schwellenwertes dazu, dass das Neuron „feuert“. Die Formel 2.2 zeigt das Verhalten einer „Binary Step“-Aktivierungsfunktion mit vorgegebenen Schwellenwert S .

$$\sigma(y_{sum}) = \begin{cases} 1 & : y_{sum} > S \\ 0 & : sonst \end{cases} \quad (2.2)$$

Neben dieser einfachen Aktivierungsfunktion wie die *Binary Step* gibt es viele weitere Aktivierungsfunktionen, beispielsweise die *Sigmoidfunktion* oder *ReLU (Rectified Linear Unit)* Funktion. Diese Aktivierungsfunktionen verwenden für die Berechnung immer das Ergebnis der Summenfunktion. Es gibt auch Aktivierungsfunktionen die alle Neuronen einer Schicht zur Berechnung verwenden. Zu diesen Funktionen zählen u.a. die Softmax- und die Maxout-Aktivierungsfunktion.

Das eben beschriebene Neuronen-Modell ist ein einfaches Modell, welches oft in Netzen wie *Feedforward Neural Netzwerke (FNN)*, *Rekurrente neuronale Netze (RNNs)* oder *Long Short-Term Memory Networks (LSTM)* Anwendung findet. Andere Neuronen-Modelle wie beispielsweise das *Leaky-Integrate-And-Fire* Modell, finde seine Anwendung in gepulsten Netzwerken. Mit diesen mathematischen Modellen wird versucht das biologische Nervensystem nachzubilden, mit all seinen Stärken und Schwächen. Die Forschung hat in den letzten Jahren bereits große Fortschritte gemacht, aber mit immer besser werdender Technik und verbessertem Verständnis der biologischen Abläufe ist das Potenzial der neuronalen Netze noch nicht erschöpft.

2.1.3 Deep Learning

Das Teilgebiet *Deep Learning* versucht möglichst präzise Vorhersagen und Entscheidungen aus komplexen Daten zutreffen. Hierfür werden tiefe neuronale Netze verwendet. Das sind Netze mit mehreren versteckten Schichten zwischen der Ein- und Ausgabeschicht. Jede Schicht verarbeitet Daten, extrahiert Merkmale und übergibt diese an die nächste Schicht. Diese Struktur und Arbeitsweise erlauben die Verarbeitung und Analyse komplexer Datenmuster in großen Datenmengen.

Durch den Einsatz von Deep Learning Methoden können relevante Merkmale aus Daten extrahiert werden ohne manuelles Feature-Engineering. Auch erreichen die Methoden eine hohe Genauigkeit und können an eine Vielzahl von Aufgaben angepasst werden.

Die großen Herausforderungen im Bereich Deep Learning liegen in den riesigen Datenmengen, die hohen Rechenleistungen welche das Training benötigt und die schwierige Nachvollziehbarkeit der durch Deep Learning getroffenen Entscheidungen.

Deep Learning hat den vergangenen Jahren viele Anwendungsbereiche erschlossen, darunter auch natürliche Sprachverarbeitung, wie Übersetzung, Textveränderung oder Textgenerierung.

2.2 Natural Language Processing

Natural Language Processing ist ein Teilgebiet der Informatik und nutzt Deep Learning. NLP soll es digitalen Systemen in die Lage versetzen Texte und Sprachen zu erkennen, um diese zu verstehen und verarbeiten zu können. Dabei muss NLP die Bedeutung (Semantik) der Texte erkennen, die Grammatik und Beziehungen zwischen den Teilen der Sprache herstellen, Wortarten wie Verben, Adjektive und Nomen spezifizieren, sowie verschiedene Formen der Sprache beherrschen wie beispielsweise Prosa oder wissenschaftliches Schreiben.

NLP wird aber auch in anderen Bereichen eingesetzt. Mithilfe von NLP können Bilder generiert, Suchmaschinen abgefragt, Chatbots für den Kundenservice betrieben werden und Sprachassistenten wie Amazon Alexa, MS Cortana und Apple Siri nutzen ebenfalls die NLP Techniken.

Zunehmend findet NLP Einsatz im unternehmerischen Bereich. Hier werden vor allem Prozesse automatisiert um die Produktivität der Mitarbeiter zu steigern. Neben Aufgaben wie Kundensupport, Datenanalyse oder Dokumentenverwaltung kommt NLP auch in der Entwicklung von Software zum Einsatz. Hierbei werden fast alle Segmente der Entwicklung abgedeckt, von der Codegenerierung über Test und Qualitätsmanagement bis hin zur Bereitstellung.

Die ersten große Erfolge hatte NLP mit neuronalen Netzen, beispielsweise mit *Feedforward Neural Networks*

und *Convolutional Neural Networks*, wie [13] zeigt. Mit der Einführung von ChatGPT und BERT, wurde auch hier die neuen Transformer Modellen eingesetzt. Die Forschungen im Bereich NLP haben die großen Sprachmodelle erst ermöglicht.

2.3 Large Language Model

Die Teilgebiete Deep Learning und Natural Language Processing haben es den großen Sprachmodellen LLM ermöglicht kommunikationsfähig zu werden. Sie verstehen Anfragen und können Antworten generieren. Die LLMs sind in der Lage Bilder und andere Medien wie Video oder Audio zu generieren.

Diese Modelle wurden mit sehr großen Datenmengen trainiert und sind daher in der Lage natürliche Sprache zu verstehen.

2.3.1 Grundlagen

Die großen Sprachmodelle können menschliche Sprache arbeiten. Sie sind speziell für die Lösung sprachbezogene Probleme geeignet, wie Textgenerierung, Klassifizierung und Übersetzung. Sie nehmen Anfragen sog. *Prompts* entgegen und errechnen daraus die wahrscheinlichste Antwort. Des Weiteren können Prompts als Anweisung (instruction-tuning) oder in Dialogform (chat fine-tuning) gestellt werden. Die meisten der heutigen großen Sprachmodelle verwenden die Transformer Technik.

Die grundlegende Funktionsweise der Large Language Models kann in vier Teilschritte unterteilt werden,

1. Tokenisierung: zerlegen der Texte in einzelne Token.
2. Embedding: Vergleiche mit anderen Vektoren und Einordnung in einer Gesamtstruktur.
3. Vorhersage: Wahrscheinlichkeit des nächsten Tokens berechnen.
4. Decodierung: Auswahl der Ausgabestrategie.

Durch die mögliche hochgradige Parallelisierung kann in diesen Modellen die Trainingszeit erheblich verkürzt werden, was ihre Effizienz steigert. Des Weiteren sind diese Modelle sehr flexibel und für verschiedene Aufgaben eingesetzt werden.

Diese Technik hat die Leistungsfähigkeit der LLMs erheblich gesteigert und versetzt Modelle in der Lage den Kontext von Worten in Sätzen zu verstehen und Abhängigkeiten zu modellieren. Beispiele für Modelle die diese Technik nutzen, sind unter anderem GPT-3, BERT und T5.

2.3.2 Grenzen und Probleme bei LLMs

Auch wenn Künstliche Intelligenz mit ihren großen Sprachmodellen in vielen Bereichen der privaten Nutzung und in den Prozessen von Unternehmen immer präsenter wird, hat diese Technologie auch Grenzen. Im folgenden werden kurz die wichtigsten Grenzen und Probleme erläutert.

Ressourcenverbrauch

Mit dem Aufkommen der großen Sprachmodelle ist auch der Verbrauch an Ressourcen enorm angestiegen. Dabei stehen diese nur in einem begrenzten Maß zur Verfügung. Kleine und mittlere Unternehmen kommen hier schnell an ihre Grenzen und nutzen daher die Modelle der Anbieter wie OpenAI, Google oder Microsoft. Auch hier gilt Ressourcenbegrenzung, sodass die Modelle nicht unendlich groß werden können. Die folgenden Ressourcen, die hier genannt werden, haben direkten Einfluss auf die Modelle und deren Betrieb,

- Speicher
- Rechenleistung
- Netzwerk
- Energie
- Finanzen

Im Lebenszyklus der großen Sprachmodelle werden Ressourcen in unterschiedlichen Mengen benötigt. So ist beispielsweise die Rechenleistung beim Training der Modelle enorm hoch.

Datenmengen

Ein weiteres Problem, vor dem die Entwickler der Modelle stehen, sind die riesigen Datenmengen, die große Sprachmodelle für das Training benötigen. Diese Daten müssen zuvor gesammelt, kuratiert und gelabelt werden. Diese Prozesse sind zeitaufwändig und kostspielig.

Interpretierbarkeit

Da die Ergebnisse stark von den Trainingsdaten abhängen, die nicht immer offen liegen und die Modelle hochkomplex sind, ist ein Ergebnis schwer nachvollziehbar. Dies erschwert auch das Aufspüren von Fehlern im Bias und den Modellen.

Probleme bei der Generierung von Texten

Vor allem bei der Generierung von Texten können Halluzinationen auftreten. Dann erzeugen die Modelle falsche oder unsinnige Informationen, die aber plausibel erscheinen. Ein weiteres Problem ist die Erzeugung diskriminierender Informationen. Diese entstehen beispielsweise durch nicht repräsentative Trainingsdaten, fehlerhafte Label in Trainingsdatensätzen und fehlendes Gesamtbild.

Die genannten Probleme und Grenzen lassen sich auch bei der Codegenerierung beobachten. Die generierten Codes können plausible erscheinen, funktionieren aber nicht oder werfen Fehler. Fehlenden Informationen über geeignete Frameworks und Funktionen veranlassen die Modelle Klassen und Methoden aus anderen Kontexten abzuleiten oder vermischen Codesegmente.

2.3.3 Verständnis für die LLMs

Viele Nutzer (Privatnutzer aber auch Firmen) wissen nicht, was hinter den großen Sprachmodellen steckt oder wie diese funktionieren. Diese Unwissenheit birgt die Gefahr, dass Nutzer nicht korrekte Eingabe in die LLMs übergibt und dann die Ergebnisse der LLMs falsch interpretieren oder die LLMs nicht korrekte Aussagen trifft. Werden aufgrund dieser falschen Ergebnisse Entscheidungen getroffen, können diese enorme finanzielle und personelle Einbußen nach sich ziehen. Zudem kann es weiterhin zu Desinformation, Diskriminierung, juristische Probleme und zum Vertrauensverlust in die Technologie führen.

Um diesen Problemen bei Entwicklern entgegenzuwirken, sind vor, während und nach der Einführung einer LLM zur Codeentwicklung, die Nutzer aufzuklären. Sie müssen sich im Klaren sein, dass LLMs Fehler produzieren können und es erforderlich ist, die Ergebnisse zu validieren. Nur so kann einem Vertrauensverlust entgegengewirkt werden und eine stetige Weiterentwicklung der Modelle erfolgen.

2.4 Koordinationsstrategien für LLMs

Es gibt mehrere Ansätze, dass LLMs komplexe Probleme gemeinsam bearbeiten. Dabei übernehmen die LLMs unterschiedliche Aufgaben oder Teilaufgaben des Problems und tragen somit zur Gesamtlösung bei. Durch diese modulare Struktur ist es möglich, dass für jedes Teilproblem eine spezialisierte LLM einzusetzen. Ein weiterer Vorteil dieser Methoden ist die Verteilung des Problems auf mehrere Systeme, die voneinander unabhängig sind. Durch diese Zusammenarbeit können die LLMs effizienter und genauer antworten

Des Weiteren sind die LLMs in der Lage mit externen Ressourcen zusammenarbeiten und Ergebnisse abfragen. So ist es möglich die Fähigkeiten der LLMs zu erweitern z.B. durch Datenbankabfragen, Web-

Suche oder Steuerung von Hardware. Ein wichtiger Aspekt für die Koordination zwischen LLMs ist die Kommunikation. Diese ist entscheidend für die erfolgreiche Bearbeitung des Problems und koordinieren die LLMs.

In den folgenden Kapiteln werden zwei Ansätze für die Zusammenarbeit von mehreren LLMs erläutert. Zum einen die *Orchestrierung*, zum anderen die *Multi-Agenten-System (MAS)*.

2.4.1 Orchestrierung von LLMs

Bei der Orchestrierung von LLMs wird die Steuerung, der Agenten mittels eines zentralisierten Systems umgesetzt, es erfolgt eine koordinierte Nutzung. Meist wird ein Problem in Teilprobleme zerlegt und die Agenten bearbeiten Teilprobleme meist parallel. Die zentrale Steuerung entscheidet welche Teilaufgabe, welcher Agent am besten geeignet ist für die Lösung der Teilaufgabe.

Die zentrale Rolle in der Orchestrierung von LLMs übernimmt dabei der Orchestrator. Dieser steuert die Aufgabenverteilung, koordiniert und kombiniert die Ergebnisse und leitet sie in die entsprechenden Agenten oder erstellt daraus die Antwort, außerdem kann er zusätzliche Aufgaben wie Fehlerbehandlung, Skalierung, Datenschutz und Sicherheit ausführen.

Im Bereich der Softwareentwicklung mit Spezialisierung auf internetbasierte Anwendungen, bei der bestimmte Standards erwartet, spezielle Frameworks und Bibliotheken eingesetzt werden, könnte eine Orchestrierung bei der Umsetzung der Programmcodeerstellung wie folgt beschrieben, helfen. Bei der Lösung von Anforderungen sind nicht immer alle Agent beteiligt, vielmehr sucht der Orchestrator die jeweiligen optimalen Agenten aus.

Der Orchestrator übernimmt auch hier die oben beschriebenen Aufgaben. Ein Frontend-Agent nutzt eines der großen Sprachmodelle, um Nutzeranforderungen in die Benutzeroberflächen der Anwendungen zu implementieren und könnte das Design verwalten. Gleichzeitig wäre es möglich, dass dieser Agent Tools wie React.js oder Vue.js unterstützen. Für die serverseitigen Anwendungen ist der *Backend-Agent* verantwortlich und verwaltet die Logik der Anwendung. Er könnte mit Frameworks wie Node.js, Express und Django umgehen. Um die Anwendung mit einer Datenbank auszustatten, kann ein *Datenbank-Agent* eingesetzt werden. Er kennt verschiedenen Datenbanken wie MySQL oder PostgreSQL. Dieser verwaltet die Datenbank und deren Abfragen. Der *Test-Agent* testet die Anforderung die von durch den Frontend-, Backend- oder Datenbank-Agent umgesetzt wurden.

Ein letzter wichtiger Agent könnte noch der NLP-Agent sein. Dieser Agent nimmt natürliche Sprachanweisungen und Anforderungen entgegen, übersetzt diese in technische Anforderungen als Prompt für die Sprachmodelle. Die Ergebnisse der Bearbeitung werden zum Schluss von dem Agenten in eine vom

Menschlichen verständliche Sprache überführt und zurückgegeben.

2.4.2 Multi-Agenten-Systeme

Multi-Agenten-Systeme (MAS) bestehen aus mehreren Agenten. Ein Agent ist eine autonome Einheit, die mit der Umwelt interagieren kann. Im Gegensatz zur Orchestrierung sind Multi-Agenten-Systeme in ihrer Steuerung dezentralisiert. Alle Agenten haben unterschiedliche Lösungsansätze für ein Problem. Je nach deren Fähigkeit hat dieser auch seine ganz eigenen Ziele, welche zu den anderen Agenten entweder kollaborativ oder kompetitiv ausgerichtet sind. Die Hauptarbeit zur Lösungsfindung eines Problems übernimmt der Agent, mit dem besten Lösungsansatz für das Problem. Die anderen Agenten können den ausführenden Agenten unterstützen. Um die beste Lösung zu finden, müssen die Agenten untereinander kommunizieren. Teil der Kommunikation kann es sein, einfache Informationen austauschen, um eine gemeinsame Strategie fest zulegen oder um zu Verhandeln, welcher Agent die Lösung eines Problems übernimmt.

Im Bereich der Webentwicklung mit MAS, könnte ein derartiges System wie folgt aussehen. Ein *Frontend-Agent* ist für das Design und die Benutzeroberfläche verantwortlich. Hierbei erzeugt dieser Agent Ausgaben in HTML, JavaScript und CSS um die Oberflächen zu erstellen. Dazu kann der Agent Frameworks, wie React verwenden und auf externe Designer Tool zugreifen. Ein weiterer Agent ist der *Backend-Agent*, welcher für die serverseitige Anwendung zuständig ist. Er erstellt seine Funktionen in PHP, Python oder NodeJS. Der Backend-Agent hat Zugriff auf Frameworks und externe Bibliotheken der gewählten Programmiersprache. Er erstellt und verwaltet zudem die Datenbankoperationen (CRUD-Operations). Hinzu kommt noch ein *Test-Agent*, welcher automatisierte Tests durchführt. Um die Funktionalität der Anwendung zu gewährleisten, arbeitet der Test-Agent mit dem Frontend- und Backend-Agent eng zusammen. Der Test-Agent stellt sicher, dass jegliche Codeänderung getestet wird und führt Unit-, Inetragtions- und End-to-End-Tests durch. Wird ein Fehler festgestellt, kann der Test-Agent ein Ticket erstellen oder direkt mit dem Frontend- oder Backend-Agenten kommunizieren.

Ein weiterer Agent könnte ein *Deploment-Agent* sein. Dieser führt automatische Depolyments in verschiedene Umgebungen (QA, Test oder Produktion) durch. Er ist in den Continuous Integration (CI) und Continuous Deployment (CD) Workflow integriert, welche die Bereitstellung auf verschiedenen Servern (VMware, Bare-Metal) und Cloud-Umgebungen (AWS, Azure, Google) bewerkstelligt. Des weitere könnten beispielsweise Security-Agent, Monitoring-Agent und Optimierungs-Agent Einsatz finden.

Auch hier kann ein NLP-Agent zum Einsatz kommen und die Kommunikation zwischen Mensch und System managen.

2.5 Prompt Engineering

Prompt Engineering optimiert die Antworten große Sprachmodelle, ohne Parameter, wie Bias und Gewichte des Models ändern zu müssen. Dieser Forschungsbereich hat in den letzten Jahren enorm an Bedeutung gewonnen und sich zu einer eigenen Disziplin im Bereich der Künstlichen Intelligenz entwickelt.

Ein Prompt oder Anweisung muss entweder als Anweisung oder als Frage gestellt werden. Dies kann, wie in [14] beschrieben, in Form von einer einfachen Anweisung bis hin zu detaillierten Beschreibungen oder spezifischen Aufgaben erfolgen.

2.5.1 Prompt-Techniken

Die folgenden Techniken dienen dazu die Abfragen zu optimieren und somit eine bessere Antwort von den Sprachmodellen zu erhalten. In dieser Arbeit werden hauptsächlich *Contextual Augmentation* und *Meta-Prompts* verwendet.

Contextual augmentation sind Prompts die neben der eigentlichen Eingabeaufforderung zusätzliche Informationen für die LLMs bereitstellen. Dazu gehören Kontextangaben, anzuwendende Programmiersprache, geforderte Bibliotheken, Angaben der Zielgruppe und Definition über die auszugebende Struktur wie Methoden oder Klassen.

Beispiel für einen kontextuellen Prompt:

Du bist ein Erfahrener PHP Programmierer, hier deine Aufgabe. Erstelle eine PHP-Funktion, die eine Liste von Ganzzahlen sortiert und zugibt.

Ausgabe der LLM:

```
1  function sortiere_und_gebe_aus($zahlen) {  
    // Die Liste wird nach aufsteigender Reihenfolge sortiert  
    sort($zahlen);  
  
    // Die sortierte Liste wird ausgegeben  
6  foreach ($zahlen as $zahl) {  
    echo $zahl . " ";  
    }  
}
```

Diese Art der Prompts können hilfreich sein, wenn Erweiterungen oder Änderungen in bestehende Programme erfolgen. Hier sind oft schon Standards oder Bibliotheken definiert.

Meta-Prompts sind Prompts, die eine LLMs anweisen, einen Prompt zu erstellen.

Meta-Prompt Beispiel:

Du bist eine Prompt-Engineer und hier ist deine Aufgabe. Schreibe einen Prompt, der eine PHP-Funktion generiert, die eine Methode erstellt, die eine Liste von Ganzzahlen sortiert.

Ausgabe der LLM:

Schreibe eine PHP-Funktion namens `sortiereZahlen`, die ein Array von Ganzzahlen als Parameter entgegennimmt. Die Funktion soll das Array in absteigender Reihenfolge sortieren und das sortierte Array zurückgeben. Nutze eine geeignete PHP-Funktion für die Sortierung.

Es gibt weitere Prompt-Typen, die nicht relevant für diese Arbeit sind. Eine Zusammenfassung von Methodiken und deren Beurteilung kann [15] entnommen werden.

Prompttemplates

Um die Eingaben für Programmierer oder Nutzer und dessen Anfragen an LLMs zu vereinfachen, können Prompt-Templates verwendet werden. Diese eignen sich im besonderen Maße für automatisierte Prompts.

```
1  Erstelle eine [Programmiersprache] Funktion namens [Funktionsname], die
    [Beschreibung der Funktion]
    . Die Funktion soll folgende Parameter verwenden:
    [Liste der Parameter]
    . Die Funktion soll folgende Bedingungen erfüllen:
6  [Liste der Bedingungen: z.B. Fehlerbehandlung, bestimmte Algorithmen]
    . Die Ausgabe der Funktion soll in folgendem Format sein:
    [Erwartetes Ausgabeformat]
```

2.5.2 Grenzen beim Prompt-Engineering für LLMs

Trotz der bemerkenswerten linguistischen Leistung, stoßen große Sprachmodelle an ihre Grenzen, unter anderem wie in [14] beschrieben. Oft ist die Komplexität der Sprache für die LLMs ein großes Hindernis, da diese oft mehrdeutig sein kann und zu unerwarteten Aussagen seitens der LLM führen kann. Ebenso haben LLMs Probleme mit Sarkasmus und Ironie, was ebenfalls zu nicht erwünschten Ausgaben führen kann.

Ein weiteres Kriterium sind die Modellbeschränkungen. Die Qualität der Trainingsdaten entscheidet oft über die Qualität der Antwort. Ebenfalls wirkt sich die Architektur auf die Art der zu bewältigenden Aufgaben aus. Nicht alle Modelle sind für alle Prompts geeignet. Modelle können zwar Muster in Daten erkennen, ihnen fehlt aber ein tiefes Verständnis für die Welt und ihnen fehlen Zusammenhänge zwischen Ereignissen.

Ein großes Problem stellen die praktischen Herausforderungen. Da ein guter Prompt ein iterativer Prozess ist, der Zeit und Geduld erfordert, welche oft nicht investiert wird. Hinzu kommen die Kosten kommerzieller Modelle, welche sich negativ auf die Gesamtkosten von Softwareprojekten auswirken kann.

2.6 Grundlagen der Webentwicklung

In diesem Unterkapitel soll kurz auf Anforderungen der Webentwicklung eingegangen werden.

2.6.1 Programmiersprachen

Grundsätzlich kann jede Programmiersprache verwendet werden. Es gibt jedoch Programmiersprachen, die explizit für Webanwendungen entwickelt wurden und einige Funktionen mitbringen, welche die Entwicklung vereinfachen. Die meisten visuellen Anwendungen erstellen HTML (**H**yper**T**ext **M**arkup **L**anguage) Code als Grundgerüst und generieren CSS (**C**ascading **S**tyle **S**heets) Dateien für das Layout, die als Standardformatierungssprache gilt. Anwendungen die als RestAPI (**A**pplication **P**rogramming **I**nterface) fungieren liefern meist Ausgaben in Form von JSON (**J**ava**S**cript **O**bject **N**otation) aus. Neben JSON Format gibt es weitere beispielsweise XML (**X**ML **A**in't **M**arkup **L**anguage) oder YAML (**Y**AML **A**in't **M**arkup **L**anguage).

2.6.2 Entwicklung

Bei der Entwicklung von Webseiten werden längst schon die selben Prozesse und Tools verwendet wie bei anderen Softwareprojekten. Auch hier finden Tolls wie GitLab¹ und Jenkins² Anwendung. Gerade in der Entwicklung von cloudbasierten Anwendungen kommen Containertools wie Docker³ in Verbindung mit Kubernetes⁴ zum Einsatz. Diese Tools lassen sich hervorragend in CI/CD Pipelines integrieren. An deren Anfang steht auch hier der Entwickler, welcher durch KI Unterstützung erhalten kann.

¹Gitlab ist eine webbasierte Anwendung die Issue-Traking, CI/CD Pipelines, Dokumentation und mehr für Entwickler anbietet.

²Jenkins ist ein webbasiertes Tool für die kontinuierliche Integration welches viele Build-Tools, wie Ant und Maven integriert, Testtols wie JUnit und Emma bietet, sowie Verwaltungssystem wie CVS, Subversion und Git unterstützt. Jenkins kann durch viele Plugins erweitert werden.

³Durch die Containerisierung mit Docker können Anwendungen und deren Umgebungen einfach bereitgestellt und bei bedarf skaliert werden. Docker bietet eine Vielzahl von einsatzbereiten Container an, die einzeln oder in Clustern laufen können.

⁴Kubernetes ist Orchestrierungstool für Dockercontainer das von Google entwickelt wurde. Neben den Container-Anwendungen verwaltet Kubernetes auch die Umgebung für Container, wie beispielsweise Netzwerke.

Einsatz von KI

Der Einsatz von Künstlicher Intelligenz kann in allen Entwicklungsphasen eingesetzt werden, angefangen von der Codegenerierung über die Bereitstellung mittels Pipeline bis zur Inhaltserstellung.

Der Einsatz von NL2Code steckt hier noch in den Anfängen, bietet aber sehr gute Ansätze viele Aufgaben zu automatisieren oder als Werkzeug welches die Entwicklung effizienter gestalten kann.

Die Codegenerierung für Designelemente kann ebenso mittels NL2Code erfolgen wie komplexe Backend-funktionalitäten. Ebenso kann die vorherige Konzeption durch eine LLM erfolgen.

2.7 Benchmark für LLM

Bei der Evaluierung großer Sprachmodell hinsichtlich des generierten Codes, gibt es einige Herausforderungen. Herkömmliche Methoden, wie BLUE-Score misst die Textähnlichkeiten nicht aber die funktionale Korrektheit des Codes und vernachlässigt auch den Kontext, in dem der Code erstellt wurde. Ein generierter Code kann in seiner Lösung stark von einer vorgegebenen Beispiellösung abweichen, trotzdem aber seine Funktionalität erfüllen. Menschliche Programmierer würden das mit verschiedenen Unit-Tests überprüfen, aus diesem Grund sollte der Code mit einer weiteren Methode geprüft werden.

Als Benchmark für die Bewertung der Sprachmodelle wird der HumanEval-XL verwendet, welche unter <https://github.com/FloatAI/humaneval-xl/tree/main> heruntergeladen werden können.

In diesem Kapitel werden die Rahmen- und Randbedingungen für das methodische Vorgehen der Evaluation großer Sprachmodelle für die Codegenerierung von webbasiertem Code festgehalten. Dies umfasst die Festlegung der verwendeten LLMs, die geprüfte Programmiersprachen, Framework zur Erstellung und Auswertung der Tests und Systeme für die Bereitstellung und Verarbeitung der Ergebnisse. Die Evaluierung der Modelle erfolgt in deutscher Sprache, was die Prompts und die Tests betrifft. Allein die Methodenbezeichnung ist in englischer Sprache.

3.1 Definition der Evaluierungsziele

Ausgehend von den in Kapitel 1.4 gestellten Ziele dieser Arbeit, werden hier das Konzept und das Design für die Evaluation beschrieben. Wie werden die Ziele erreicht und welche Komponenten sind dazu erforderlich, um die Daten zu erheben und auszuwerten.

Die Evaluation soll zeigen, ob die generierten Codes korrekt funktioniert und es nicht zu Laufzeitfehlern oder Deadlocks kommt. Des Weiteren wird evaluiert, ob der Code den gängigen Programmierstil entsprechen und welche Qualität der generierte Code ausweist. Hier liegt vor allen der Fokus auf Lesbarkeit und Einhaltung von Codingstandards. Als letzter Punkt soll die Dokumentation innerhalb des Codes geprüft werden. Je besser diese Ausfällt je leichter sind spätere Refactorings möglich.

Was nicht im Fokus liegt und in den Test vernachlässigt wird, sind die Performance der Modelle und die Architekturen der Systeme auf denen die Modelle laufen. Es geht in erster Linie um den generierten Code und dessen Brauchbarkeit für die reelle Verwendung in der Praxis.

Für die Messung der Fehler wird die `pass@k` Metrik angewandt. Diese Metrik baut auf einen mitgelieferten Test für jedes Problem auf. Der Test zusammen mit dem generierten Code ergeben einen ausführbaren Code. Jeder Test wird in einer Liste notiert und dazu das Ergebnis ob positiv oder negativ. Daraus kann dann mit der `pass@k` Metrik die repräsentative Zuverlässigkeit des Modells für jedes Problem und für das

Modell errechnet werden.

Für die erweiterte Messung des PHP Codes kommt das Tool `phpmetrics` zum Einsatz. `phpmetrics` liefert eine Vielzahl an Metriken, mit der der Code analysiert werden kann. Bei diesem Tool liegt der Fokus auf die Ergebnisse von `Cyclomatic Complexity`, mit der die Komplexität des Codes gemessen wird. Auf den `Maintainability Index` zur Auswertung der Wartbarkeit. Diese ermittelt wie aufwändig es zukünftig ist den Code zu erweitern und bewertet die Lesbarkeit. Ein weiterer Parameter welcher hilft die Komplexität zu bewerten, ist der `Logical Lines of Code` Parameter. Ein weiteres Indiz zur Wartbar- und Lesbarkeit ist der `Method Length` Parameter. Ein weiterer Parameter zur Bewertung der Lesbarkeit ist der Parameter `Numbers of Parameters`. Diese gibt die Anzahl der Parameter welche in die Methode übergeben werden. Eine hohe Anzahl kann ein Indiz dafür sein, dass der Code schwerer zu lesen ist. Zur Bewertung der Dokumentation wird der Parameter `Comment Density` herangezogen. Mithilfe eines Python-Skripts wird der generierte Code mit `phpmetrics` evaluiert.

(SonarQube, wenn die Zeit es noch erlaubt)

Mit dem SonarQube steht ein Auswertungstool zur Verfügung, welches neben qualitativen Codeanalyse noch eine Sicherheitsanalyse des Codes durchführt und die technischen Schulden ermittelt. Somit ist SonarQube eine Ergänzung zu `phpmetrics`. SonarQube wird eine lokale VM ausgeführt und mittels Integration werden die generierten Codes an den Server übermittelt.

Beide Verfahren `phpmetrics` und SonarQube benötigen für die Analyse Dateien. Somit müssen die Ergebnisse erst als Datei gespeichert werden, bevor die Tools mit der Analyse beginnen.

3.2 Auswahl der LLMs und deren Konfiguration

Für die Evaluation werden experimentell einige freie und kommerzielle Modelle ausgewählt und miteinander verglichen. Hauptsächlich wurden bei den freien Modellen, jene ausgewählt welche den Fokus auf die Codegenerierung legen und mit diesem Argument beworben werden. Als Referenz sollen die kommerziellen Modelle dienen, da durch hier weite Verbreitung und breite Nutzer ihre Daten stetig erweitern und die Entwicklung vorantreiben.

Im Folgenden werden die ausgewählten LLMs kurz vorgestellt und warum diese gewählt wurden. Die Reihenfolge stellt an dieser Stelle keine Wertung der LLM oder über deren generierten Inhalte dar.

Das **Qwen2.5-Coder**-Modell zeichnet sich durch seine spezialisierte Architektur für die Codegenerierung aus. Trainiert, um sowohl syntaktisch korrekten als auch funktional hochwertigen Code zu produzieren, integriert es fortschrittliche Mechanismen zum Kontextverständnis und semantisch sinnvolle Ausgabe. Es findet Anwendung in verschiedenen Bereichen der Softwareentwicklung, insbesondere in der Web- und

Anwendungsprogrammierung.

Deepseek-Coder-V2 ist die zweite Generation der Deepseek-Coder-Reihe und bietet verbesserte Fähigkeiten zur Codegenerierung und -optimierung. Das Modell nutzt fortschrittliche Suchalgorithmen, um präzisere und effizientere Codestücke zu erstellen. Es ist insbesondere für seine hohe Genauigkeit bei der Generierung komplexer Algorithmen und Datenstrukturen bekannt.

Llama3.1-Claude stellt eine spezialisierte Version der Llama3-Serie dar, die auf die Generierung und Optimierung von Code ausgelegt ist. Das Modell kombiniert die Stärken von LLMs mit domänenspezifischen Anpassungen, um robuste und fehlerresistente Codefragmente zu generieren. Es bietet hohe Effizienz und Genauigkeit, besonders bei längeren Codeabschnitten.

Llama3.2 ist eine Weiterentwicklung der Llama-Modellreihe und bietet verbesserte Leistung in der Codegenerierung. Durch Optimierungen in der Modellarchitektur und der Trainingspipeline erreicht Llama3.2 eine gesteigerte Genauigkeit und Effizienz bei der Bearbeitung komplexer Programmieraufgaben, insbesondere in der Webentwicklung und der automatisierten Codierung.

Mistral ist ein modernes Sprachmodell, das speziell für die Codegenerierung und -analyse entwickelt wurde. Es verwendet fortschrittliche Transformer-Architekturen, um kontextbewusste und syntaktisch korrekte Codevorschläge zu machen. Mistral ist bekannt für seine Flexibilität und die Fähigkeit, mehrere Programmiersprachen effizient zu unterstützen.

ChatGPT 3.5, entwickelt von OpenAI, ist ein vielseitiges Sprachmodell, das sowohl für allgemeine textuelle als auch programmatische Aufgaben verwendet wird. Es bietet eine solide Leistung in der Codegenerierung, mit besonderem Fokus auf Benutzerinteraktion und Konversation. Das Modell zeigt gute Ergebnisse in der Erstellung und Anpassung einfacher bis mittelkomplexer Codestrukturen.

ChatGPT 4 ist die neueste und leistungsstärkste Version der ChatGPT-Serie. Mit verbesserten Fähigkeiten zur Sprach- und Kontextverarbeitung übertrifft es seine Vorgänger in der Generierung präziser und komplexer Codes. Das Modell wird häufig für anspruchsvolle Programmieraufgaben eingesetzt und zeichnet sich durch seine hohe Anpassungsfähigkeit an unterschiedliche Codierungsanforderungen aus.

Gemini 1.5 ist ein Modell von Google, das speziell für die Generierung von Code und die Verarbeitung natürlicher Sprache entwickelt wurde. Es basiert auf multimodalen Eingaben und nutzt fortschrittliche Techniken zur Verbesserung der Codequalität und -konsistenz. Gemini 1.5 zeigt eine hohe Effizienz bei der Bearbeitung umfangreicher Programmieraufgaben.

Gemini 2 repräsentiert die neueste Generation der Gemini-Modellreihe und bietet weiterentwickelte Fähigkeiten in der Codegenerierung. Durch die Kombination von fortschrittlichen Sprachverarbeitungstechniken und tiefen neuronalen Netzwerken ermöglicht es die Erstellung von hochwertigem und optimiertem Code

für komplexe Softwareprojekte. Gemini 2 hebt sich durch seine besonders hohe Präzision und Flexibilität in der Codegenerierung hervor.

Die Tabelle 3.1 zeigt zusammenfassend die ausgewählten Modelle.

Modell	Parameter	Größe	Sprache	offen	Ausführung	Benchmark
Llama3.2	3b	2,0 GB	German	X	lokal/Ollama	HE-XL
Mistral Small	22b	12 GB	German	X	lokal/Ollama	HE-XL
In Vorbereitung						
ChatGPT3.5-turbo	k.A.	k.A.	German	-	online	HE-XL
ChatGPT4-turbo	k.A.	k.A.	German	-	online	HE-XL
Gemini 1.5 Pro	k.A.	k.A.	German	-	online	HE-XL
Gemini 2 Pro	k.A.	k.A.	German	-	online	HE-XL
HF: Hugging Face						

Tabelle 3.1: Auswahl der LLMs für die Evaluierung

Die Einstellung für die Abfragen der Probleme wurden bei allen Modellen identisch gewählt.

Für die Abfragen der Testprobleme wurde eine *temperature* von 0.2 gewählt. Ein niedriger Wert veranlasst die Modelle deterministischere und standardisierte Antworten zu geben und verhindert Kreativität und Zufälligkeit. Die Generierung von Programmcode soll konsistenten und präzisen Code liefern.

Ein hoher *top_p* Wert verlangt von den Modellen eine Antwort die mit hoher Wahrscheinlichkeit richtig ist. Für die Codegenerierung sollten die wahrscheinlichsten und syntaktisch korrekten Token angewandt werden. Für die Abfragen wird hier ein Wert von 0.95 angesetzt.

Die maximale Anzahl Token sollte bei der Generierung von Code zwischen 20 und 1000 Token eingestellt werden, je nach Umfang der Antworten. Da hier nicht nur die Funktionsfähigkeit geprüft wird, sondern auch Struktur und Coding-Standards wird ein hoher Wert ausgewählt, sodass beispielsweise Kommentare ebenfalls enthalten sind können. Somit wird *max_token* auf sechshundert festgelegt.

In der Tabelle 3.2 sind die Werte in übersichtlicher kurzer Form noch einmal dargestellt.

Hinzu kommen weitere Parameter für die Modelle. Unter anderem wird der Parameter *do_sample* auf *False* gesetzt, was die Modelle veranlasst den wahrscheinlichsten folgenden Token zu wählen und ein deterministisches Verhalten fördert. Ein weiterer Parameter ist *return_full_text* der ebenfalls auf *False* gesetzt wird. Dadurch werden nur die neu generierten Tokens zurückgegeben, was die Relevanz der Antworten fördert.

Alle Prompts welche die Probleme enthalten, werden mit Python Skripten abgefragt. Dies gilt für die offenen lokalen wie auch für die kommerziellen Modelle. Die Abfragen werden jeweils an die entsprechenden

Modell	Temperatur	max. Tokenanzahl	Top-p Sampling
Qwen2.5-coder	0.2	600	0.95
Deepseek-Coder-v2	0.2	600	0.95
Llama3.1-Claude	0.2	600	0.95
Llama3.2	0.2	600	0.95
Codellama	0.2	600	0.95
Mistral Small	0.2	600	0.95
ChatGPT3.5-turbo	0.2	600	0.95
ChatGPT4-turbo	0.2	600	0.95
Gemini 1.5 Pro	0.2	600	0.95
Gemini 2 Pro	0.2	600	0.95

Tabelle 3.2: Einstellungen der Modellparameter

APIs abgesetzt. Als Framework für die Abfragen der lokalen Modelle, kommt das Python `langchain` Framework zu Einsatz. Für die Modelle der Gemini-Reihe wird die Google eigene Bibliothek, `google.genai` verwendet. Dasselbe trifft für die Modelle von OpenAI zu. Hier kommt die Python Bibliothek `openai` zum Einsatz.

3.3 Design der Experimente

Das Experiment wird mit dem HumanEval-XL Benchmark durchgeführt. Dieser Benchmark besteht aus einer Reihe von 80 Tests in verschiedenen Programmiersprachen, die wie folgt aufgebaut sind,

- **task_id**: Kennung der Datenprobe
- **prompt**: Anfrage für das Modell, Funktionsheader und Docstring
- **entry_point**: Einstiegspunkt für den Test
- **test**: Test für die geforderte Funktion
- **description**: Beschreibung der Aufgabe
- **language**: Kennung der Programmiersprache
- **canonical_solution**: Lösung für das Problem
- **natural_language**: Ländersprache in der Datei

Die Abbildung 3.1 zeigt den Aufbau und damit alle wichtigen Bereiche des Benchmark-Tests.

Um die Modelle zu evaluieren und ihre Fähigkeiten hinsichtlich der im Web vorherrschenden Programmiersprachen zu untersuchen, werden die Tests in den Programmiersprache(n) PHP (und JavaScript) vorgenommen. Dafür sollen die Modelle mehrfach einfache Funktionen generieren.

HumanEval-XL Benchmark

Tasks: Generation 1 2 >> 3 (Test: 1 2 3 4)

<?php
Deklaration ①

```
/**
 * Sie sind ein erfahrener PHP-Programmierer und hier ist Ihre Aufgabe.
 * ...
 */
```

Kommentar ②

```
function belowZero($operations) {

    // Initialize the current balance to zero
    $balance = 0;

    // Iterate over each operation in the list
    foreach ($operations as $operation) {
        // Add or subtract the operation amount from the balance
        $balance += $operation;

        // If the balance goes below zero, return True immediately
        if ($balance < 0) {
            return true;
        }
    }

    // If we've iterated over all operations and never gone below zero,
    // then return False
    return false;
}
```

Ergebnis ③

```
function compare($x, $y) {
    return $x == $y;
}

$args00 = [];
$x0 = belowZero($args00);
$v0 = false;
if (!compare($x0, $v0)) {
    throw new Exception("Error at 1th assert statement.");
    ...
}
```

Test ④

PHP (Problem 0)

Abbildung 3.1: Codegeneration

Der Benchmark liefert die Tests mit. Dazu werden bereits in den Prompts die Namen der Methoden und die zu übergebenen Parameter angegeben, welche zu erstellen ist. Der jeweilige Test verwendet dann diesen Namen und übergibt die geforderten Parameter. Das Listing 3.1 zeigt ein Beispiel für einen mitgelieferten HumanEval-XL Test.

```
function compare($x, $y) {  
2   return $x == $y;  
}  
$arg00 = [3, 1, 2, 4, 5];  
$x0 = median($arg00);  
$v0 = 3;  
7 if (!compare($x0, $v0)) {  
    throw new Exception("\nError at 1th assert statement.\n");  
}
```

Listing 3.1: Beispiel für einen Test aus dem HumanEval-XL Benchmark

Um die Modelle untereinander zu vergleichen, bekommen alle Modelle dieselben Prompts. Von jedem Prompt werden pro Modelle zehn Varianten erstellt. Die Ergebnisse werden in einer Liste chronologisch gespeichert. Diese Ergebnisse werden dann mittels der `pass@k` Metrik geprüft.

3.4 Konzeption des Prompt-Engineerings

Die Prompts im HunamEval-XL Benchmark sind als Few-Shot-Prompts verfasst. Sie neben der eigentlichen Aufgabe sind noch Beispiel für die Eingabedaten und erwarteten Ergebnisse angegeben. Das Listing 3.2 zeigt ein Beispiel für einen Prompt.

```
1 <?php  
  
/**  
 * Sie sind ein erfahrener PHP-Programmierer und hier ist Ihre Aufgabe.  
 * Gib den Median der Elemente in der Liste l zurück.  
6 * >>> median([3, 1, 2, 4, 5])  
 * 3  
 * >>> median([-10, 4, 6, 1000, 10, 20])  
 * 15.0  
 *  
11 */  
function median($l){
```

Listing 3.2: Prompt Beispiel für eine Aufgabe aus dem HumanEval-XL Benchmark

Alle Prompts im Benchmark sind als Code-Kommentare aufgebaut. Als letzte Zeile ist der Methodenname angegeben. Somit soll sichergestellt werden, dass die erstellten Tests funktionieren.

Mein roter Faden

Hier kommen noch Optimierungsangaben, die stehen zurzeit nicht fest.

3.5 Evaluationsumgebung

Die freien Modelle laufen auf einem Debian 12 System, welches mit 16 CPUs und 32 GB RAM ausgestattet ist. Um zusätzlichen Speicher zu erhalten, wurde eine 100 GB Swap Partition genutzt. Für die Bereitstellung ist das freie Framework Ollama zum Einsatz gekommen.

Auf die kommerziellen Modelle kann kein Einfluss auf die Systeme genommen werden.

IMPLEMENTIERUNG

Mein roter Faden

Wichtigsten Aspekte und Schritte der Implementierung. Aufbau und Struktur evtl. Programme, wichtige techn. Entscheidungen (Nicht den gesamten Quellcode abbilden, wenn überhaupt dann im Anhang). Einbinden, nur wenn,

1. Nur erklärungsbedürftigen Code ca. 5-20 Zeilen.
2. Für Nachvollziehbarkeit.
3. Kommentare für Erklärung.
4. Keine Standards oder Bibliotheken.
5. Komplexe Klassen/Strukturen besser beschreiben.
6. Alternativen:
 - Pseudocode.
 - Diagramme.
 - Erklären der Logik durch Text.

Max. 10-12% des Kapitels, Rest in Anhang.

4.1 Lokale Modelle

Für das Ausführen von Modellen zum Testen werden in dieser Arbeit zwei Techniken angewandt. Zum einen mittels Ollama Framework, das mit einer Web-GUI erweitert werden kann, zum anderen durch Dateien, welche beispielsweise mit dem Python Framework Langchain abgefragt werden können.

Die Modelle werden auf einem Debian 12 Server mit 32 GB RAM und einem 16 Kernprozessor ausgeführt. Um große Modelle zu testen wurde mit einer Swap-Partition von 100 GB gearbeitet, um die Ausführung größerer Modelle zu ermöglichen.

4.1.1 Modellbereitstellung mit Ollama

Für das Testen der lokalen Modelle wird das Ollama Framework angewandt. Dies ermöglicht eine neben einer Benutzeroberfläche im Browser eine Anbindung an einer API. Diese lässt sich beispielsweise mittels Python abfragen. Auf dieser Weise lassen sich Modelle von der Ollama Modell Seite testen. Dazu wird Ollama auf dem Server installiert und konfiguriert, siehe Anhang A.2. Nach dem Download stehen die Modelle zur Verfügung und es können Interaktionen mit dem Modell erfolgen.

Zusätzlich kann ein grafisches Tool zum Testen installiert werden. Mit deren Hilfe können die Modelle leicht getestet werden. Mit Open WebUI wird ein Browser basierendes Tool eingesetzt, dass auf dem Ollama-Server installiert wird. Nach der Installation ist das Tool einsatzbereit und im lokalen Netzwerk, unter `http://«server-ip»:«webui-port»` erreichbar. Die Installation wird im Anhang A.3 beschrieben.

4.1.2 Modellbereitstellung als Datei

Eine zweite Methode zur Bereitstellung von Modellen die für diese Arbeit Verwendung findet, ist die direkte Nutzung als lokale Datei. Diese können dann direkt angesprochen werden, in dieser Arbeit wird Python verwendet. Hierbei wurden die Modelle von Hugging Face fokussiert. Diese lassen sich unter anderem mit dem Python Framework Longchain orchestrieren.

Nachdem die Modelle von Hugging Face heruntergeladen und lokal abgespeichert wurden, sind diese ohne größeren Aufwand anwendbar. Ein Beispiel für ein mögliches Download-Skript ist in Anhang A.4 im Listing 8.3 und 8.4 zu sehen. Hierbei ist zu beachten das genügend freier RAM zur Verfügung steht, um die Modelle abzuspeichern.

4.2 Benchmark Codeevaluation

Ausführen des Benchmarks.

4.2.1 Problemabfragen an die Modelle

Nachdem die Modelle bereitstehen, erfolgt Erstellung der Antworten von den Modellen. Mit Prompts welche die Probleme aus dem HumanEval-XL Benchmark enthalten, werden nun die Modelle abgefragt. Die vollständigen Antworten werden, für eine spätere Auswertung als JSONL-Dateiformat gespeichert. Für jedes Problem erfolgen zehn Abfragen an jedes Modell. Für die Evaluierung der lokalen Modelle wurden sowohl Modelle von Hugging Face als auch Modelle von Ollama verwendet, siehe Kapitel ?? für die Zuordnung der Modelle.

4.2.2 Auswertung der Modellantworten

Die Auswertung der Antworten wird für jedes Modell angepasst, da die Antworten von Modell zu Modell variieren. Hier werden unterschiedliche Möglichkeiten für Codesnippets angewendet, beispielsweise (““php) oder (““php \n <?php), welche für die Ausführung anzupassen sind. Somit wird immer nur die Methode zur Extrahierung des Codes aus den Antworten geändert, siehe Beispielcode im Listing 8.6.

Mit den vorliegenden Codesnippets und den Tests aus dem jeweiligen Problem, wird ein ausführbarer PHP-Code erstellt und mittel Python ausgeführt. Die Tests werfen Exceptions, wenn der Code die Anforderungen nicht erfüllt oder Laufzeitfehler auftreten. Dann gibt der Code als nicht verwertbar. Der Code ?? zeigt den Ausschnitt im Code zur Ausführung des PHP Interpreters.

```
3 for answer in answers:
    answer = answer.replace(r"\n", "\n")
    try:
        result = subprocess.run(
            ["php", "-r", f"{test}{answer}"],
            capture_output=True,
            text=True,
            8 check=False,
            timeout=5,
        )
    except subprocess.TimeoutExpired:
        13 pass
```

```
if result.stderr.strip() == "":  
    answer_result_fault = True
```

Listing 4.1: Codesnippet zur Ausführung des PHP Interpreters label

Die unbehandelte Exception führt lediglich dazu, dass die Programmausführung nicht unterbrochen wird. Der hier abgefangene Timeout wird in der nachfolgenden `if` Klausel behandelt. Aus den erhaltenen Ergebnissen, berechnet die `pass@k` Metrik die Zuverlässigkeit des jeweiligen vorgegebenen Problems, hinsichtlich Codegenerierung. Des Weiteren wird anschließend der Durchschnitt für die Zuverlässigkeit des Modells errechnet.

Umsetzung der `pass@k` Metric

In Python steht, hierfür die Bibliothek `pass_at_k` zur Verfügung. Der PHP Code wurde auf einen Debian 11 mit PHP in der Version 8.2.26 evaluiert.

```
def custom_pass_at_k(n: int, c: int, k: int) -> float:  
    """  
    :param n (int): numbers of total samples.  
    :param c (int): number of correct samples.  
5    :param k (int): number of consider samples.  
    """  
    if n - c < k:  
        return 1.0  
    return 1.0 - np.prod(1.0 - k / np.arange(n - c + 1, n + 1))
```

Listing 4.2: Berechnung der `pass@k` Metrik in Python

4.3 Codeevaluation mit Frameworks

Neben dem bekannten Evaluationen mit beispielsweise dem HumanEval Benchmark, wird hier eine weitere Testmethodik überprüft, die mit verschiedenen Validierungstools der jeweiligen Programmiersprache ausgeführt wird. Für die Erstellung der Abfragen wird das Python-Skript verwendet, was schon im Kapitel 4.2 vorgestellt wurde.

4.3.1 PHP Codeevaluation

Der Test wird bei den erweiterten Problemen durchgeführt und beginnt mit den Unit-Tests die mit *PHPUnit* durchgeführt werden. Im Anschluss wird *PHPMetrics* ausgeführt. Hierbei wird geprüft, ob die

Codekomplexität und Wartbarkeit überprüft. Sind diese Tests bestanden, wird der Code noch gegen eine *SonarQube* Server validiert. Die Ausführung der Tests wird mithilfe eines Python-Skripts durchgeführt. Es wird eine PHP Datei erstellt, die mit den Frameworks geprüft wird.

4.3.2 JavaScript Codeevaluation

Text.

EVALUATION

Die Evaluation der Ergebnisse erfolgt im ersten Schritt anhand des HumanEval-XL Benchmarks. Dieser Benchmark wird in [16] vorgestellt und erweitert den HumanEval [17]. Der HumanEval-Benchmark evaluiert nur Python während der HumanEval-XL weitere Programmiersprachen und in verschiedenen Landessprachen unterstützt, darunter auch die deutsche Sprache. Neben Python sind auch Prompts für PHP und JavaScript enthalten, welche für die Webentwicklung wichtig sind. Die Datensätze des HumanEval-XL sind unter <https://github.com/FloatAI/humaneval-xl> einsehbar und bestehen jeweils aus 80 Tests. Für jedes Problem werden zehn Lösungsvorschläge generiert, die im Anschluss auf die Aspekte der Syntaktik und Semantik evaluiert werden.

~~Diese Tests fordern LLM's auf kleine Problem zu lösen. Aus diesem Grund werden weitere Tests erstellt mit umfangreicheren Anforderungen aus dem Bereich der Webentwicklung. Zu jedem Problem wird eine Musterlösung und ein Unittest erstellt. Der Aufbau für diese Bereitstellung orientiert sich an dem Format aus dem HunamEval-Benchmark.~~

Ein Versuch größere und komplexere Probleme zu lösen, hatte nicht den erwarteten Erfolg. Es sind viele Iterationen notwendig, um ein funktionierendes Ergebnis zu erhalten. Im Laufe der Iterationen sind die Prompts für die Modelle immer größer geworden und haben viele Missverständnisse bei den Modellen erzeugt. So das eine Zerlegung in kleine Probleme sich als sinnvoller erwies.

Des Weiteren ist die Bewertung der Coding-Standards der jeweiligen Programmiersprache vorgesehen. Für die Prüfung der Standards wird ein SonarQube-Server verwendet, der sowohl PHP als JavaScript unterstützt. Ebenfalls wird die Qualität des Codes evaluiert. Das Augenmerk liegt auf die Lesbarkeit, Effizienz und Wartbarkeit des generierten Codes.

5.1 Modellbewertung mit HumanEval Benchmark

Für die Bewertung wird das Vorgehen gewählt, welches in [17] und [16] beschrieben ist. Die Tests werden exemplarisch, mit den für die Webentwicklung relevanten Sprachen PHP und JavaScript durchgeführt. Die Evaluierung der Modelle wird auf den Ebenen „einfache Fragen“ und „komplexe Aufgaben“ erfolgen. Die „einfachen Fragen“ werden bereits durch den zuvor genannten Benchmarks abgedeckt, sodass der entwickelte Fragenkatalog sich auf die Ebenen mit den „komplexen Aufgaben“ konzentriert.

Aus Ergebnisse der Tests, wird mithilfe der $pass@k$ -Metrik, die Zuverlässigkeit der jeweiligen Modelle berechnet. Dieser Wert gibt an, mit welcher Wahrscheinlichkeit mindestens eine richtige Lösung unter k ausgewählten Vorschlägen vorhanden ist. Die Formel 5.1 zeigt die Berechnung der $pass@k$ -Metrik.

$$pass@k = 1 - \frac{\prod_{i=0}^{n-k} (n - i - c)}{\prod_{i=0}^{n-k} (n - i)} \quad (5.1)$$

Dabei ist n die Gesamtanzahl der Versuche, c die Anzahl der korrekten Lösungen unter den n Versuchen und k gibt die Anzahl der Lösungen an die betrachtet wurden. Für die Berechnung der $pass@k$ Metrik wird die Formel 4.2 verwendet, welche in [17] vorgeschlagen wird.

Für alle Probleme wurden jeweils zehn Abfragen erstellt und bewertet. Welche Modelle getestet an der Evaluation beteiligt waren und welche Ergebnisse ermittelt wurden, wird in Tabelle 5.1 gezeigt.

| Model | pass@1 | pass@5 | pass@10 |
|--------------------|--------|----------|---------|
| Llama3.2 | 0,0325 | 0,427629 | 0,6625 |
| llama3.1-claude | 0,0325 | 0,508929 | 0,725 |
| codellama | 0,0 | 0,0125 | 0,05 |
| codeqwen | 0,0 | 0,0 | 0,0 |
| (Auswertung fehlt) | | | |
| Gemini Flash 1.5 | 0,0 | 0,0 | 0,0 |
| (Auswertung fehlt) | | | |
| Qwen 2.5-Coder | 0,0 | 0,0 | 0,0 |
| (Auswertung fehlt) | | | |
| Mistral-small | 0,0125 | 0,346329 | 0,65 |

Das Modell code1lama hat beim Generierung von den Lösungen der PHP Probleme nicht gut abgeschnitten. Viele der Anforderungen wurden in Python erstellt und viele Tests sind als nicht bestanden gewertet wurden.

Die Antworten von ChatGPT enthielten bei der ersten Abfrage Programmcode, alle weiteren Abfragen verwiesen auf den ersten Prompt. Eine Antwort ist in ?? dar gestellt, diese wurde in ähnlicher Weise immer

wieder generiert.

Der Code in deinem Kommentar ist identisch mit dem aktuellen Inhalt. Wenn du Änderungen vornehmen möchtest, präzisiere bitte, was angepasst werden soll, und ich werde es umsetzen!

Generiert von ChatGPT 3.5

Im Gegensatz zu ChatGPT bringt Gemini nicht immer nur eine Lösung des Problems, sondern bietet weitere Ansätze die ebenfalls für das jeweilige Problem interessant sein könnten. Im folgenden Beispiel sind die verschiedenen Vorschläge des Modells in den Listings 5.1 bis 5.4 dargestellt, die beim PHP Problem 3 gefordert wurden. Die Aufgabenstellung in Kurzform war: Gib den größten gemeinsamen Teiler von zwei ganzen Zahlen a und b zurück.

```
1 function greatestCommonDivisorRecursive($a, $b) {  
    if ($b === 0) {  
        return $a;  
    } else {  
        return greatestCommonDivisorRecursive($b, $a % $b);  
6    }  
}
```

Listing 5.1: Gemini Ergebnis für das PHP-3 Problem Version 1

Text.

```
function greatestCommonDivisorBinary($a, b) {  
    if ($a == 0) return $b;  
3    if ($b == 0) return $a;  
  
    // Ensure a is odd  
    if ($a % 2 == 0 && $b % 2 == 0)  
        return 2 * greatestCommonDivisorBinary($a / 2, $b / 2);  
8    if ($a % 2 == 0) return greatestCommonDivisorBinary($a / 2, $b);  
    if ($b % 2 == 0) return greatestCommonDivisorBinary($a, $b / 2);  
  
    // a and b are odd  
    if ($a > $b) return greatestCommonDivisorBinary($a - $b, b);  
13    return greatestCommonDivisorBinary($b - a, a);  
}
```

Listing 5.2: Gemini Ergebnisse für das PHP-3 Problem Version 2

Text.

```
1 function leastCommonMultiple(int $a, int $b): int {  
    // kgV(a, b) = (a * b) / ggT(a, b)  
    return abs($a * $b) / greatestCommonDivisor($a, $b);  
}
```

Listing 5.3: Gemini Ergebnisse für das PHP-3 Problem Version 3

```
1 function greatestCommonDivisorMultiple(int ...$numbers): int {  
    // ggT von mehreren Zahlen  
    return array_reduce($numbers, 'greatestCommonDivisor');  
}
```

Listing 5.4: Gemini Ergebnisse für das PHP-3 Problem Version 4

Einige Aufgaben wurden von Gemini aber wie ein Chat behandelt. So sind Ausgaben generiert wurden, die sich auf die Ausgabe davor beziehen, sodass der Code mit einer Kommentar-Zeile, wie beispielsweise `// ... (Rest der Funktion bleibt ähnlich)` ersetzt wurde.

5.1.1 Nachteile der Evaluierung

Es wird geprüft, ob der Code ohne Fehler ausführbar ist und die richtigen Ergebnisse liefert. Was dieser Test nicht evaluiert sind unter anderem vorhandene Codeerklärungen, Doc-Strings oder Code-Smells werden bei diesem Test nicht beachtet. Ebenso wird auch nicht der Codestandard geprüft.

5.2 Optimierung der Ergebnisse

Als Ziel der Optimierung gilt das die LLMs effizienten, präzisen und korrekten Code zu generieren. Ein Ansatz dies zu erreichen ist die Prompts zur Codegenerierung mithilfe einer LLMs zu erstellen oder zu verbessern.

LESSONS LEARNED

Mein roter Faden

Dieses Kapitel wird die positiven und negativen Erfahrungen der Kapitel Implementierung und Evaluation auffassen. Die weiteren Kapitel bauen auf die hier gewonnenen Erkenntnisse auf.

6.1 Erweiterte Codeevaluation

Bei den vordefinierten Prüfungen der HumanEval Benchmarks, wird geprüft, ob der Code lauffähig ist, nicht aber die Codestruktur oder Kommentare. Ein Problem bei der Nutzung des von der LLM generiertem Code ist, dass Entwickler diesen einfach kopieren und in ihre Programme implementieren. Es wird also nur die Funktionalität des Codes geprüft, nicht aber Strukturen und Kommentare um die Lesbarkeit und Verständlichkeit zu erhöhen. Dieses Vorgehen mag zu schnellen Erfolgen in der Programmentwicklung führen, wird aber beim Refactoring oder Fehlersuche erhebliche Defizite mit sich bringen.

Aus diesem Grund sollte der erstellte Code nicht nur auf die Funktionalität geprüft werden. Dafür sollten weitere Test-Frameworks der jeweiligen Programmiersprache zur Anwendung kommen. Es gibt mehrere Frameworks zur Prüfung der Codequalität unter PHP. Zwei bekannte Frameworks die auch in dieser Arbeit Anwendung finden, sind die Frameworks `phpunit` und `phpmetrics`. Mit ihnen wird der, durch die LLMs generierten Codes geprüft.

Um PHPUnit und PHPMetrics für die Evaluierung zu verwenden, müssen weitere Angaben und Einträge im Benchmark erfolgen. So muss ein PHP-Unittest enthalten sein, dieser kann den einfachen benutzerdefinierten Test ersetzen. Des Weiteren sind die Kriterien für die Metrik Messung, für jeden Test erforderlich. Die Kriterien können wie in Listing 6.1 dargestellt, aussehen.

```
1  criteria = {  
    "Lines of code": lambda x: int(x) > 12,
```

```
6      "Logical lines of code by method": lambda x: float(x) > 7,  
      "Lack of cohesion of methods": lambda x: float(x) > 3,  
      "Average Cyclomatic complexity by class": lambda x: float(x) > 10,  
11     "Average Weighted method count by class": lambda x: float(x) > 20,  
      "Average bugs by class": lambda x: float(x) > 0.1,  
      "Critical": lambda x: int(x) > 0,  
      "Error": lambda x: int(x) > 0,  
      "Warning": lambda x: int(x) > 0,  
11     "Information": lambda x: int(x) > 0,  
      }
```

Listing 6.1: Beispiel für Bewertungskriterien

Mit den erweiterten Tests werden die Benchmarks, um die folgenden Punkte erweitert.

- **unittest**: Unittests für die geforderte Funktion, unterschied zu den einfachen Tests
- **metrics**: Kriterien für den Metriktest

6.1.1 PHPUnit

Eines der bekanntesten spezielles Framework für Unit-Tests in PHP, was als Industriestandard gilt. Mit diesem Framework können neben der Prüfung auf funktionsfähigen Code auch Randfälle betrachtet und Fehlerbehandlungen im Code getestet werden. Als Grundlage für die Auswahl des Tools wird auf Studie [18] verwiesen.

6.1.2 PHPMetrics

Ein PHP Framework für die Codeanalyse, welches detaillierte Berichte über die Codequalität, Komplexität des Codes und über dessen Wartbarkeit erzeugt. PHPMetrics wird in verschiedenen Arbeiten eingesetzt, um die Codequalität zu ermitteln. So auch in [19], bei der verschiedene Open Source LMS verglichen werden.

6.1.3 SonarQube

Als letztes Tool soll SonarQube zur statischen Codeanalyse und Codeprüfung zum Einsatz kommen. Es werden verschiedene Programmiersprachen unterstützt, darunter auch PHP und JavaScript. In der Arbeit [20] wird die Prüfung der Codequalität mit SonarQube, ChatGPT3.5 und ChatGPT4 verglichen. Als Schlussfolgerung aus dem Ergebnis dieser Arbeit, wird auch hier die Codeanalyse durch eine LLM nicht erfolgen, sondern ebenfalls durch SonarQube.

DISKUSSION UND AUSBLICK

Mein roter Faden

Struktur des Kapitels

1. **Einleitung:** Eine kurze Einführung in die Diskussion und den Ausblick.
2. **Zusammenfassung der Ergebnisse:** Eine kurze Übersicht über die wichtigsten Ergebnisse und in Relation mit den Forschungsfragen stellen.
3. **Diskussion der Ergebnisse:** Eine Analyse und Interpretation der Ergebnisse. Vergleich mit Stand der Forschung und früherer Arbeiten.
4. **Grenzen und Einschränkungen:** Eine Diskussion der Limitationen der Studie. Z.B. begrenzte Datenbasis, Grenzen der eingesetzter Tools und Technik.
5. **Impulse für zukünftige Forschung:** Vorschläge für weitere Studien. Verbesserungsmöglichkeiten der Methoden usw. und Zukunft des Forschungsfeldes und evtl. Trends.
6. **Praktische Anwendung:** Eine Diskussion der möglichen Anwendungen der Ergebnisse. In welchen Unternehmen und welche realen Anwendungen können die Ergebnisse eingesetzt werden.

Mein roter Faden

Unterschied Diskussion/Ausblick und Fazit

| Aspekt | Diskussion und Ausblick | Fazit |
|--------------------|---|-------------------------------|
| Funktion | Kritische Analyse und Zukunftsperspektive | Zusammenfassung und Abschluss |
| Zeitperspektive | Zukunftsorientiert | Rückblickend |
| Detaillierungsgrad | Detailreichere Auseinandersetzung mit Ergebnissen | Knapp und prägnant |

Während „Diskussion und Ausblick“ die Ergebnisse kritisch reflektiert und auf zukünftige Entwicklungen verweist, fasst das „Fazit“ die Arbeit kompakt zusammen und beantwortet die Forschungsfrage. Beide Kapitel sind komplementär, aber klar voneinander zu unterscheiden.

Wie in [21] beschrieben,

7.1 Impulse für zukünftige Forschungen

Ein interessantes Feld für die Forschung ist die Nutzung generativer KI und welche Auswirkungen dies auf das menschliche Denken und Handeln hat. In der Studie [22] wird von einem System 0 gesprochen, welches neben den bekannten

1. System 1: schnelles, intuitives und automatisches Denken
2. System 2: langsames, analytisches und reflektierteres Denken

eingeführt wird. Hierbei handelt es sich um ein Denken, welches die KI für den Menschen übernimmt. Entscheidungen und Daten werden durch die KI übernommen. Ein externes System, ähnlich wie eine USB-Festplatte eines PCs.

Inwieweit können auch *Small Language Models* für Programmieraufgaben eingesetzt werden. Könnte der enorme Energiebedarf und Ressourcen der LLMs durch SLMs ersetzt werden? Siehe Small Language Models (SLMs) oder Small but Powerful: A Deep Dive into Small Language Models (SLMs). Eine weitere Forschung kann die Evaluation sein, ob Finetuned SLMs, wie Phi-2, Google Gemini Nano oder Metas Llama-2-13b bessere Ergebnisse liefern, als die LLMs.

Ein weiteres Feld kann sich mit der Einführung einer KI in Firmen befassen und Fragen wie,

- Wie können Entwickler bestmöglich vorbereitet werden, um die Einführung von KI reibungslos zu ermöglichen?

- Wie kann Datensicherheit und Datenqualität sichergestellt werden?
- Evaluierung von Kosten/Nutzen für die Einführung von KI in Softwareunternehmen.

evaluieren.

Mein roter Faden: noch was zum Testen

Ein Tool zur Orchestrierung von Multi-Agenten-Systemen OpenAI Swarm, gefunden auf Golem | Karrierewelt.

7.2 Praktische Anwendung

Blaupause für Prompting Das Geheimnis hinter LLM-Halluzinationen [S. 16 ff.] noch testen und evaluieren.

7.2.1 Anwendung für Entwickler

Zur Optimierung des generierten Codes kann auch die freie Wahl der Softwarekomponenten durch die LLMs betragen. Wie in [17] beschrieben können Nutzer, anstatt in Suchmaschinen beispielsweise die Vorteilen und Nachteile von PyTorch und Tensorflow zu vergleichen, kann das die LLM übernehmen und als Prompt wird nur `# import machine learning package` angegeben.

Wie in [le-2024] beschrieben nimmt das Lesen von Programm zehn mal mehr Zeit in Anspruch, als Code zu schreiben. Diese Arbeit kann ebenfalls durch eine LLM übernommen werden.

Mein roter Faden

Struktur des Kapitels

1. **Zusammenfassung:** kurze Wiederholung der Zielsetzung d. Arbeit, Überblick der wichtigsten Ergebnisse aus Eval. und Optimierung, Fragestellung beantwortet?
2. **Reflexion:** Stärken und Schwächen d. Arbeit, Diskussion über mögliche Fehlerquellen, Einschätzung Optimierungsansätze oder Benchmarks

Mein roter Faden

Unterschied Diskussion/Ausblick und Fazit

| Aspekt | Diskussion und Ausblick | Fazit |
|---------------------------|---|-------------------------------|
| Funktion | Kritische Analyse und Zukunftsperspektive | Zusammenfassung und Abschluss |
| Zeitperspektive | Zukunftsorientiert | Rückblickend |
| Detaillierungsgrad | Detailreichere Auseinandersetzung mit Ergebnissen | Knapp und prägnant |

Während „Diskussion und Ausblick“ die Ergebnisse kritisch reflektiert und auf zukünftige Entwicklungen verweist, fasst das „Fazit“ die Arbeit kompakt zusammen und beantwortet die Forschungsfrage. Beide Kapitel sind komplementär, aber klar voneinander zu unterscheiden.

LITERATUR

- [1] Volker M. Banholzer. *Künstliche Intelligenz als Treiber der Veränderung in der Unternehmenskommunikation 4.0?* Bd. 1/2020. Technische Hochschule Nürnberg Georg-Simon-Ohm, 2020. URL: https://www.th-nuernberg.de/fileadmin/fakultaeten/amp/amp_docs/K%C3%BCnstliche_Intelligenz_und_die_Rolle_n_von_Unternehmenskommunikation_Banholzer_IKOM_WP_1_2020__fin-1.pdf.
- [2] *Digitale Transformation: Fallbeispiele und Branchenanalysen*. 2022. URL: https://library.oapen.org/bitstream/handle/20.500.12657/57358/978-3-658-37571-3.pdf?sequence=1&utm_source=textcortex&utm_medium=zenochat#page=70 (besucht am 19.10.2024).
- [3] Erin Yepis. *Developers want more, more, more: the 2024 results from Stack Overflow's Annual Developer Survey*. 24. Juli 2024. URL: <https://stackoverflow.blog/2024/07/24/developers-want-more-more-more-the-2024-results-from-stack-overflow-s-annual-developer-survey/> (besucht am 09.08.2024).
- [4] Rebeka Tóth, Tamas Bisztray und László Erdodi. *LLMs in Web Development: Evaluating LLM-Generated PHP Code Unveiling Vulnerabilities and Limitations*. 21. Apr. 2024. URL: <https://arxiv.org/abs/2404.14459v2> (besucht am 05.01.2025).
- [5] Juyong Jiang u. a. *A Survey on Large Language Models for Code Generation*. 1. Juni 2024. URL: <https://arxiv.org/abs/2406.00515> (besucht am 07.11.2024).
- [6] Juan David Velásquez-Henao, Carlos Jaime Franco-Cardona und Lorena Cadavid-Higuaita. „Prompt Engineering: a methodology for optimizing interactions with AI-Language Models in the field of engineering“. In: *DYNA* 90.230 (3. Nov. 2023), S. 9–17. DOI: 10.15446/dyna.v90n230.111700. URL: <https://doi.org/10.15446/dyna.v90n230.111700>.

- [7] Mayank Mishra u. a. *Granite Code Models: A Family of Open Foundation Models for Code Intelligence*. 7. Mai 2024. URL: <https://arxiv.org/abs/2405.04324> (besucht am 08.11.2024).
- [8] Anton Lozhkov u. a. *StarCoder 2 and The Stack v2: The Next Generation*. 29. Feb. 2024. URL: <https://arxiv.org/abs/2402.19173> (besucht am 08.11.2024).
- [9] Sasikala C 1 Dr.M.Kalpana Devi 2,Tholhappiyan T 3, Sasikala Nataraj. „REVOLUTIONIZING WEB DEVELOPMENT WITH AN INTELLIGENT CHATBOT: a NOVEL APPROACH UTILIZING OPENAI'S GPT-3 AND ADVANCED NLP STRATEGIES“. In: *Machine Intennigence Research* 18.1 (17. Aug. 2024), S. 1098–1109. URL: <http://machineintelligenceresearchs.com/index.php/mir/article/view/90> (besucht am 08.11.2024).
- [10] Daoguang Zan u. a. *Large language models meet NL2Code: a survey*. 19. Dez. 2022. URL: <https://arxiv.org/abs/2212.09420> (besucht am 26.12.2024).
- [11] Pekka Ala-Pietilä u. a. *Eine Definition der KI: Wichtigste Fähigkeiten und Wissenschaftsgebiete*. 5. März 2019. URL: https://elektro.at/wp-content/uploads/2019/10/EU_Definition-KI.pdf (besucht am 10.09.2024).
- [12] Johanna Pahl. *Zeichnung einer biologische Zelle*. 26. Sep. 2024.
- [13] Yoav Goldberg. „A Primer on Neural Network Models for Natural Language Processing“. In: *Journal of Artificial Intelligence Research* 57 (20. Nov. 2016), S. 345–420. DOI: 10.1613/jair.4992. URL: <https://jair.org/index.php/jair/article/view/11030>.
- [14] Xavier Amatriain. *Prompt Design and Engineering: Introduction and Advanced Methods*. 24. Jan. 2024. URL: <https://arxiv.org/abs/2401.14423v3> (besucht am 12.10.2024).
- [15] Banghao Chen u. a. *Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review*. 23. Okt. 2023. URL: <https://arxiv.org/abs/2310.14735v5> (besucht am 26.12.2024).
- [16] Qiwei Peng, Yekun Chai und Xuhong Li. *HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization*. 26. Feb. 2024. URL: <https://arxiv.org/abs/2402.16694> (besucht am 15.11.2024).
- [17] Mark Chen u. a. *Evaluating Large Language Models Trained on Code*. 7. Juli 2021. URL: <https://arxiv.org/abs/2107.03374> (besucht am 28.10.2024).

- [18] Radziah Mohamad, Noraniah Yassin und Easter Sandin. „Comparative Evaluation of Automated Unit Testing Tool for PHP“. In: *International Journal of Software Engineering and Technology* 2 (Dez. 2016), S. 7–11.
- [19] Rini Anggrainingsih u. a. „Comparison of maintainability and flexibility on open source LMS“. In: Aug. 2016, S. 273–277. DOI: 10.1109/ISEMANTIC.2016.7873850.
- [20] Igor Regis Da Silva Simões und Elaine Venson. *Evaluating Source Code Quality with Large Language Models: a comparative study*. 7. Aug. 2024. URL: <https://arxiv.org/abs/2408.07082> (besucht am 30. 12. 2024).
- [21] Sandro Hartenstein und Andreas Schmietendorf. „KI-gestützte Modernisierung von Altanwendungen: Anwendungsfelder von LLMs im Software Reengineering“. In: *Softwaretechnik-Trends* Band 44, Heft 2. Gesellschaft für Informatik e.V., 2024. URL: <https://dl.gi.de/handle/20.500.12116/44181> (besucht am 15. 08. 2024).
- [22] Massimo Chiriatti u. a. „The case for human–AI interaction as system 0 thinking“. In: *Nature Human Behaviour* 8.10 (22. Okt. 2024), S. 1829–1830. DOI: 10.1038/s41562-024-01995-5. URL: <https://www.nature.com/articles/s41562-024-01995-5>.

ANHANG

A Installationshinweise

A.1 Python

Da in dieser Arbeit Python verwendet wird, sollte zu den grundlegenden Paketen, für die Arbeit mit großen Sprachmodellen folgende Zusatzpakete installiert sein.

```
pip3 install langchain
pip3 install ollama
pip3 install transformer
```

Im Weiteren wird kein Hinweis auf verwendete Pakete gegeben. Diese sind evtl. den Fehlermeldungen während und nach der Programmausführung zu entnehmen.

A.2 Installation und Konfiguration von Ollama

Für die Installation von Ollama wird bei Linux folgendes Skript ausgeführt,

```
curl -fsSL https://ollama.com/install.sh | sh
```

Ollama kann in seiner Konfiguration angepasst werden, im Folgenden wurde der Pfad zur den Modellen geändert und die Erreichbarkeit von Ollama über Netzwerk eingestellt. Dazu wird die Datei `/etc/systemd/system/ollama.service` angepasst und der korrekte Host und IP-Adresse gesetzt.

3

```
diff --git a/ollama.service b/ollama.service
--- a/ollama.service
+++ b/ollama.service
```



```
@@ -10,3 +10,4 @@  
RestartSec=3  
Environment="PATH=/usr/local/bin:/usr/bin"  
-  
8 +   Environment="OLLAMA_HOST=0.0.0.0"  
+   Environment="OLLAMA_MODELS=/home/ai/models"  
+
```

Listing 8.1: Ollama Hostanpassng für Netzwerkbetrieb

Nach der Installation kann die Funktionsfähigkeit geprüft werden, der folgenden Beispielaufruf lädt ein Modell von Ollama und startet dieses.

```
ollama run deepseek-coder-v2:16b
```

Im Anschluss kann über die Konsole mit dem Modell interagiert werden.

A.3 Open WebUI Installationshinweise

Hier wird Open WebUI als docker Container verwendet, es ist also erforderlich vorher docker zu installieren. Die Installation von Open WebUI, unter Debian kann mit folgendem Skript erfolgen.

```
# Pull Open WebUI container.
docker pull ghcr.io/open-webui/open-webui:main

# Run container.
5 docker run -d --network=host -v open-webui:/app/backend/data \
  -e OLLAMA_BASE_URL=http://127.0.0.1:11434 --name open-webui \
  --restart always ghcr.io/open-webui/open-webui:main
```

Listing 8.2: Open WebUI installieren

Der Aufruf der UI, kann mittel Browser erfolgen. Hier wird die IP und der Port 8080 angegeben. Beispiel <http://192.168.2.45:8080>.

A.4 Download der Hugging Face Modelle

Ein Hugging Face Modell kann wie im Listing 8.3 gezeigt, heruntergeladen werden. Das Modell wird dann im Cache von Hugging Face gehalten und steht bis zum Löschen des Cache zur Verfügung.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

3 def call_model_by_huggingface(root_name: str, vendor_name: str, prompt:
  str) -> list:
    """
    Load model from Hugging Face and call the model with prompt.

    :param root_name (str): Model root name.
    8 :param vendor_name (str): Model vendor name.
    :param prompt (str): Prompt for model.

    Returns:
        list: Answer from model.
    13 """

    model_name: str = f"{root_name}/{vendor_name}"
    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        torch_dtype="auto",
        18 device_map="auto",
        low_cpu_mem_usage=True,
    )
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    23 messages = [
        {"role": "user", "content": prompt},
    ]
    text = tokenizer.apply_chat_template(
        messages, tokenize=False, add_generation_prompt=True
    )
    28 model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

    generated_ids = model.generate(**model_inputs, max_new_tokens=512)
    generated_ids = [
    33     output_ids[len(input_ids) :]
     for input_ids, output_ids in zip(model_inputs.input_ids,
```

```

        generated_ids)
    ]

    return tokenizer.batch_decode(generated_ids, skip_special_tokens=True)
38
# Example from https://huggingface.co/Qwen/Qwen2.5-Coder-32B-Instruct.
call_model_by_huggingface(
    root_name="Qwen",
    vendor_name="Qwen2.5-Coder-32B-Instruct",
43    prompt="Write a quick sort algorithm.",
)

```

Listing 8.3: Laden der Modelle von Hugging Face und lokal speichern

Um den Cache abzufragen oder zu löschen werden folgende Befehle angewandt.

```

huggingface-cli scan-cache
huggingface-cli remove-cache

```

Soll ein Modell auch nach dem Löschen des Caches zur Verfügung stehen, sollte das Modell separat abgespeichert werden. Für den Download und speichern der Modelle kommt folgendes Python-Skript zur Anwendung. Hier ist zu erwähnen, dass ausreichend RAM zur Verfügung stehen muss, um die Modelle zu speichern. Mit diesem Script kann ein Modell an einem angegebenen Pfad abgespeichert werden und ist nach dem Cache löschen immer noch lokal vorhanden.

```

1  from transformers import AutoModelForCausalLM, AutoTokenizer

def load_model_from_huggingface(root_name: str, vendor_name: str) -> None:
    """
    Load model from Hugging Face and save local.
6
    :param root_name (str): Root name of model.
    :param vendor_name (str): Vendor name of model.
    """
    # Personal access token from Hugging Face.
11    access_token = "hf_XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

    model_path_to_save = (
        f"/huggingface/models/{vendor_name.replace('-', '_')}"
    )

```

```
16     tokenizer = AutoTokenizer.from_pretrained(  
        f"{root_name}/{vendor_name}"  
    )  
    model = AutoModelForCausalLM.from_pretrained(  
21         f"{root_name}/{vendor_name}", is_decoder=True, token=access_token  
    )  
  
    tokenizer.save_pretrained(model_path_to_save)  
    model.save_pretrained(model_path_to_save)  
26  
# Example from https://huggingface.co/Qwen/Qwen2.5-Coder-32B-Instruct.  
load_model_from_huggingface(  
    root_name="Qwen",  
    vendor_name="Qwen2.5-Coder-32B-Instruct"  
31 )
```

Listing 8.4: Laden der Modelle von Hugging Face und lokal speichern

A.5 Abfragen lokaler Modelle

Um die Modelle abzufragen, kommt der Code, welcher in Listing 8.5 zu sehen ist, zum Einsatz.

```
import os  
import json  
from transformers import AutoTokenizer, pipeline, AutoModelForCausalLM  
4 from langchain_ollama.llms import OllamaLLM  
from langchain.prompts import PromptTemplate  
  
NUMBERS_OF_PROBLEMS: int = 80  
START_NUMBER_OF_PROBLEMS: int = 0  
9 STOP_NUMBER_OF_PROBLEMS: int = 80  
  
PROBLEM_FILE_NAME: str = "PHP_German.jsonl"  
PROBLEM_LANGUAGE: str = "php"  
  
14 MODEL_NAME: str = "qwen2.5-coder:32b"  
SUBFOLDER_NAME: str = "qwen25-coder"  
MODEL_PATH = f"/huggingface/models/{MODEL_NAME.replace('-', '_')}"
```

```
19 PROMPT_REPETITIONS: int = 10

def read_problem(task_id: int, language: str) -> dict | None:
    """
    Read file with problem description and get the problem by task_id.
    :param task_id (int): Task id.
    24 :param language (str): Coding/Programming language.
    :returns: dict | None: Prompt and problem.
    """

    jsonl = open(
        f"{os.path.dirname(os.path.realpath(__file__))}/problems/{
        29 PROBLEM_FILE_NAME}",
        "r",
        encoding="utf-8",
    )
    lines = jsonl.readlines()
    problem: dict = None
    34 for line in lines:
        problem = json.loads(line)
        if problem.get("task_id", None) == f"{language}/{task_id}":
            break
        jsonl.close()
    39 return problem

def run_server_model(problem: str):
    """
    Connect to ollama model server.
    44 :param problem (str): Prompt there is send to ollama server.
    :returns: Result from ollama server by prompt.
    """

    ollama = OllamaLLM(
        base_url="192.168.178.140:11434",
        49 model=MODEL_NAME,
    )
    prompt_template: PromptTemplate = PromptTemplate(
        input_variables=["user_prompt"],
        template="{user_prompt}",
    54 )
    prompt = prompt_template.format(user_prompt=problem)
```

```
    return ollama.invoke(prompt)

def run_local_model(problem: str):
    """
    Connect to local model.
    :param problem (str): Prompt for model.
    :returns Result from model.
    """
    tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
    model = AutoModelForCausalLM.from_pretrained(
        MODEL_PATH, torch_dtype="auto", trust_remote_code=True
    )
    pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
    prompt_template: PromptTemplate = PromptTemplate(
        input_variables=["user_prompt"],
        template="{user_prompt}",
    )
    prompt = prompt_template.format(user_prompt=problem)
    generation_args = {
        "max_new_tokens": 600,
        "return_full_text": False,
        "do_sample": False,
    }
    output = pipe(
        prompt,
        **generation_args,
        pad_token_id=tokenizer.eos_token_id
    )
    if len(output) > 0:
        return output[0].get("generated_text", "")
    return ""

def write_log(answer: str, key: str, task_id: str) -> None:
    """
    Write the prompt result in JSON file.
    :param answer (str): _description_
    :param key (str): _description_
    :param task_id (str): _description_
    """
```

```

file_name: str = f"{os.path.dirname(os.path.realpath(__file__))}/
    answer/"
file_name = f"{file_name}/{SUBFOLDER_NAME}/{task_id.replace('/', '-')}
    }.jsonl"
answer = answer.replace("\n", r"\n")
answer = answer.replace("'", r"\'")
99 answer = f"\n{{key}}\n:{{answer}}\n"
with open(file_name, "a", encoding="utf-8") as file:
    file.write("{ " + answer + " }\n")

def execute_prompt(task_id: int, model_type: str) -> bool:
104     """
    Create prompts and save local.
    :param task_id (int): Problem id
    :param model_type (str): Type of model.
    :returns bool: True prompt create.
109     """

    language: str = PROBLEM_LANGUAGE
    problem: dict = read_problem(task_id=task_id, language=language)
    prompt: str = problem.get("prompt", None)
    if prompt is None:
114         return False

    current_count: int = 0
    max_count: int = PROMPT_REPETITIONS
    while current_count < max_count:
        answer: str = ""
119         if model_type == "local":
            answer = run_local_model(problem=prompt)
        elif model_type == "server":
            answer = run_server_model(problem=prompt)
        write_log(
124             answer=answer,
            key=f"result_{current_count}",
            task_id=f"{language}/{task_id}",
        )
        current_count += 1
129     return True

```

Listing 8.5: Abfragen der Modelle von Hugging Face und Ollama

A.6 Evaluation der Antworten von den Modellen

Das Listing 8.6 zeigt beispielhaft eine Suche nach Codezeilen in einer Antwort eines Modells. Die Methode `search_generated_code` muss auf jedes Modell angepasst sein.

```

1 def search_generated_code(content: str) -> str:
    """
    Search the generated code in string, it is one line of file.
    :param content (str): Answer from LLM.
    :returns str: Generated code.
6    """
    for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
        if content.startswith('{ "result_' + str(i) + '": '):
            generated_code: str = content.split('{ "result_' + str(i) + '": '
11            "": '')[1]
            generated_code = generated_code[:-2]
            generated_code = generated_code.strip()

            if len(generated_code.split("'''php")) > 1:
                generated_code = generated_code.split("'''php")[1]
                generated_code = generated_code.split("'''")[0]

16            if generated_code.startswith("<?php"):
                generated_code = generated_code.split("<?php")[1]
                generated_code = generated_code.split(">")[0]

21            if len(generated_code.split(r"\n}\n")) > 0:
                generated_code = generated_code.split(r"\n}\n")[0] + "\n}\n"

    return generated_code
return ""

```

Listing 8.6: Evaluation der Modellantworten: Suche nach Codeausschnitten

B Fragenkataloge

Die Fragenkataloge enthalten Aufgaben welche an die großen Sprachmodelle gesendet und dessen Antwort evaluiert wurde. Die Struktur der Fragen ähnelt dem Benchmark aus [16].

B.1 PHP

Alle Fragen in diesem Katalog sind in der natürlichen Sprache Deutsch verfasst und als Programmiersprache in PHP.

php/f0 - JSON Informationen in einer MySQL speichern

Ziel:

Hierbei wird die Fähigkeit geprüft, ob LLM die Konvertierung der JSON Daten in SQL Format korrekt erledigt wird, insbesondere das Datumsformat. Den Verbindungsaufbau zu einer Datenbank, sowie das erstellen und füllen von Tabellen. Ebenso wird ein Test erwartet, der den generierten Code testet.

Aufgabe:

Du bist ein PHP Entwickler und hast folgende Aufgabe:

Erstelle einen Backendservice der Daten aus einem JSON String in einer SQL Datenbank speichert. Die Methode `convAndSaveData`, die die Aufgabe erfüllt, soll sich in der Klasse `RestService` befinden. Die Datenbankparameter sollen als Klassenattribute vorliegen und die Kommunikation mit der Datenbank in einer privaten Methode gekapselt werden. Die Daten sollen in den Tabellen „user“ und „address“ gespeichert werden. Die Felder in der Tabelle entsprechen denen aus dem JSON. Prüfe ob die Tabellen vorhanden sind, wenn nicht legt diese an. Verwende kein Framework wie Laravel oder Symfony, sondern nur PHP Funktionen.

Die Daten im JSON haben folgendes Format:

```
{'firstname': 'Max', 'surname': 'Musterman','birthday': '20.3.1990','address':  
{'street': 'Straße','streetnumber': 3,'streetaddon': 'A','postcode':  
'12345','town': 'Berlin'}}
```

Erwartetes Ergebnis:

Am Testende wurden in der MySQL Datenbank zwei Tabellen „user“ und „address“ angelegt, welche mit den Beispieldaten aus der Formatbeschreibung gefüllt wurden.

B.2 JavaScript

ANLAGEN

Erklärung zur Verwendung von KI-Systemen

Ich erkläre, dass ich

- mich aktiv über die Leistungsfähigkeit und Beschränkungen der in meiner Arbeit eingesetzten KI-Systeme informiert habe;
- alle Inhalte aus wissenschaftlich anerkannten Quellen entnommen und entsprechend gekennzeichnet habe; alle Inhalte unter Anwendung wissenschaftlicher Methoden im Rahmen der vorliegenden Arbeit von mir selbst entwickelt wurden;
- mir bewusst bin, dass ich als Autor*in dieser Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.

Bei der Erstellung der Arbeit habe ich die folgenden auf künstlicher Intelligenz (KI) basierten Systeme in der im Folgenden dargestellten Weise benutzt:

| Arbeitsschritt | Eingesetzte(s) KI-System(e) | Beschreibung der Verwendungsweise |
|---|-----------------------------|-----------------------------------|
| Generierung von Ideen und Konzeption der Arbeit | - - - - | - - - - |
| Literatursuche | - - - - | - - - - |
| Literaturanalyse | - - - - | - - - - |
| Literaturverwaltung und Zitationsmanagement | - - - - | - - - - |
| Auswahl der Methoden und Modelle | - - - - | - - - - |

KAPITEL 8. ANHANG

| | | |
|---|--------------------------|---|
| Datensammlung und -analyse | - - - - | - - - - |
| Generierung von Programmcodes | ChatGPT und Gemini | Erstellen grundlegender Anweisungen und Coderecherche für Python |
| Erstellung von Visualisierungen | - - - - | - - - - |
| Interpretation und Validierung | - - - - | - - - - |
| Strukturierung des Texts der Arbeit | ChatGPT | Generierung von Vorschlägen für den Aufbau der Kapitel |
| Formulierung des Texts der Arbeit | ChatGPT | Umformulieren von Absätzen, mit denen ich nicht zufrieden war |
| Übersetzung des Texts der Arbeit | DeepL | Übersetzung von Text für eigenen Code |
| Redigieren des Texts | ChatGPT und Duden Online | Korrektur von Rechtschreibfehlern und Grammatikfehlern und Umstrukturierung von Satzbau |
| Vorbereitung der Präsentation des Texts | - - - - | - - - - |
| Sonstiges | - - - - | - - - - |

Temmen-Ringenwalde, 8. Januar 2025

Ort, Datum

Unterschrift