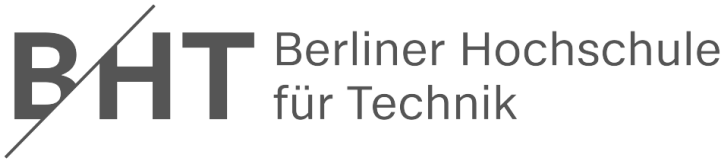
Evaluierung und Optimierung von Large Language Models für die Entwicklung von Webanwendungen

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 2. 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, den 2. Januar 2025

Unterschrift

Abstract

Abstract in Englisch.

Zusammenfassung

Zusammenfassung in Deutsch.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Einleitung

1

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 noch einmal beschleunigt. Immer mehr Unternehmen erkennen die Potenziale der Digitalisierung und passen ihre Geschäftsprozesse an und nutzen die Möglichkeiten. 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 wachsende Anzahl von Web-Anwendungen wächst auch der Druck für die Entwicklungsfirmen, ihre Anwendungen den schnell und oft wechselnden Kundenanforderungen anzupassen.

Durch diesen Prozess getrieben, müssen Entwicklungsfirmen in immer kürzeren Release-Zyklen Softwarekomponenten hinzufügen und 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 Menschen der Meinung, dass tiefere Kenntnisse in den jeweiligen Programmiersprachen nicht mehr so relevant wären, wie diese vor den LLMs waren.

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 kommt es zu Laufzeitfehlern oder einem unerwarteten Verhalten der Software.

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 werden, kann dies dazu führen, dass o.a. technische Schulden angehäuft werden und Softwareprojekte langfristig einen hohen Wartungsaufwand benötigen.

Gerade bei der Entwicklung im Bereich der Webanwendungen, welche mit PHP und JavaScript erstellt werden, wurde die Modelle nicht hinreicht getestet. Oft werden die Modelle für in Python- oder Javaproblemen evaluiert.

1.3 Stand der Forschung

In [4] wird eine bis dato fehlende Literaturrecherche zum Thema „Codegenerierung durch große Sprachmodelle“ bemängelt, was in dieser Arbeit nachgeholt wird und haben im Juni 2024 Literatur zusammengetragen, welche sich mit Codegenerierung befasst.

Um die Prompts im Ingenieurswesen zu optimieren, wird in [5] 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 Iterationen.

Es gibt Bestrebungen kleinere Modelle die auf Codegenerierung spezialisiert sind, mit den großen Sprachmodellen zu testen, so auch in [6]. Hier werden die Modelle als „Granite Code Models"-Familie zusammengefasst. Eine weitere Arbeit die sich mit kleinen Modellen, die besonders für das Generieren von Code trainiert wurden, befasst sich die Arbeit [7] mit StarCoder 2 betrachtet.

Der wissenschaftliche Artikel [8] 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 [9]. 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. 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. In dieser Arbeit wird, wie auch in [4, vgl. Seite 2] Language-to-Code, kurz NL2Code 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 auf die Modelle eine Verbesserung der Antworten ersichtlich ist. Des Weiteren soll gezeigt werden, inwieweit sich der Prozess der Codegenerierung automatisieren lässt.

Einen letzten Punkt soll der allgemeine Zustand des Codes evaluieren. Ist der Code erst einmal erstellt und muss von anderen Programmierer überarbeitet und verstanden werden, kostet dies wesentlich mehr Zeit als Code zu schreiben. Aus diesem Grund sollte Code unter anderem Kommentare enthalten und strukturiert sein.

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

- Z1 Welche Modelle eigenen sich für die Softwareentwicklung?
- Z2 Welche Methodiken helfen die Qualität der Antworten von Modellen zu verbessern?
- Z3 Wie weit lässt sich die Verwendung von Sprachmodellen, für die Codegenerierung automatisieren?

Z4 Wie gut sind die Ergebnisse, hinsichtlich Coding-Standards?

1.5 Aufbau der Arbeit

Um ein grundlegendes Verständnis zubekommen, werden im Kapitel 2 die Grundlagen für diese Arbeit vorgestellt.

Auf die Implementierung wird in Kapitel 3 eingegangen, welche für die Codegenerierung und Evaluierung erforderlich sind. Die daraus resultierenden Ergebnisse werden in Kapitel 4 diskutiert.

Die negativen und positiven Erfahrungen, sowie die Herausforderungen dieser Arbeit werden in Kapitel 5 aufgegriffen und Lösungsansätze vorgeschlagen.

Bevor in Kapitel 7 die Arbeit zusammengefasst und ein Fazit gezogen wird, werden im Kapitel 6 die Ergebnisse erläutert, diskutiert und mögliche Impulse für künftige Arbeiten und den praktischen Einsatz angesprochen.

1.6 Abgrenzung

In dieser Arbeit fokussiert sich die Betrachtung auf den Bereich der Webanwendungsentwicklung und deren verwendete Programmiersprachen. Parallelen zu anderen Anwendungsbereichen, wie beispielsweise Desktop-Anwendungsentwicklung werden hier nicht expliziert betrachtet können aber durchaus vorkommen.

Auch wenn rechtliche und ethische Überlegungen einen wichtigen Aspekt in Umgang mit Künstlicher Intelligenz darstellt, wird dies in dieser Arbeit nicht betrachtet. Es gibt hinreichend Literatur zu diesen Themen, die in dieser Arbeit Beachtung finden, es wird aber nicht explizit darauf eingegangen.

Grundlagen

2

In diesem Kapitel werden Grundlagen besprochen die eine Relevanz für diese Arbeit haben. Die angesprochenen Bereiche können nur oberflächlich einen kleinen Einstieg in die jeweiligen Teilgebiete geben.

Die Forschungsbereiche der großen Sprachmodelle, kurz LLM [eng. Large Language Model], ist ein Teilgebiet von Deep Learning und der Forschung von der Verarbeitung natürlicher Sprache, kurz NLP [eng. Natural Language Processing]. Die Grafik 2.1 zeigt die Einordnung der Bereiche.

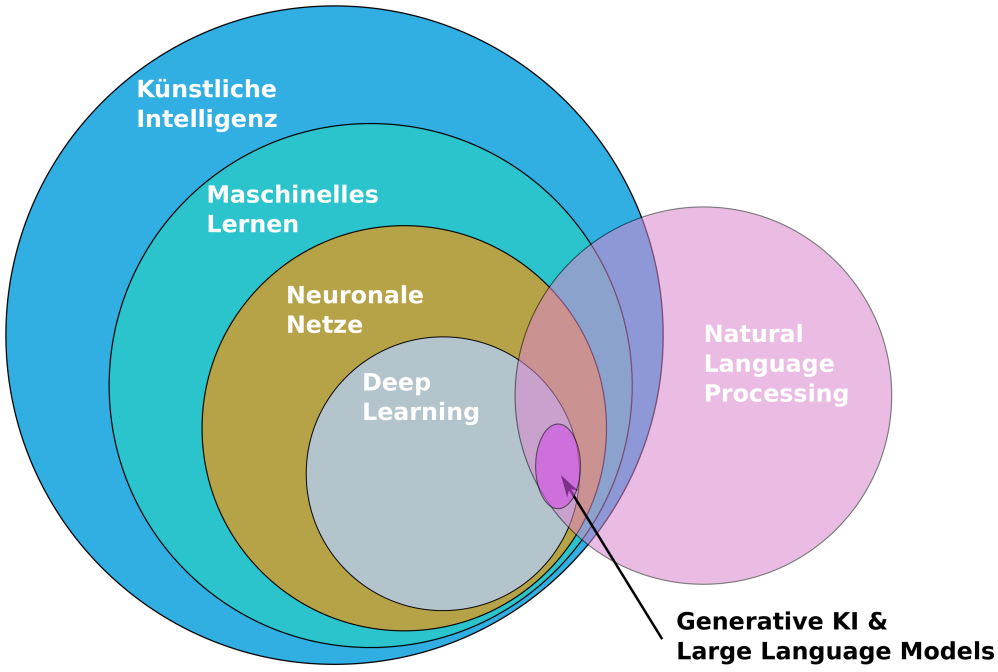


Abbildung 2.1: LLMs im Kontext der Forschungsbereiche von KI

2.1 Künstliche Intelligenz

Die künstliche Intelligenz hat bereits in viele Unternehmensprozesse Einzug gehalten. 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.

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 als Definition für die Künstliche Intelligenz, die aus [10, 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 dem Forschungsgebiet der künstlichen Intelligenz ist für die großen Sprachmodelle der Bereich des „Deep Learning“ besonders interessant. Hier findet die Überschneidung mit dem Bereich der NLP statt, welche massiv dazu betrug, dass die großen Sprachmodelle diesen Erfolg erfahren.

2.1.1 Maschinelles Lernen

Als Teilgebiet der künstlichen Intelligenz befasst es sich 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 aber 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 Leraning) werden die Systeme mit Belohnung un 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.

Eine weitere Art ist das *Semi-überwachte Lernen* die eine Kombination aus unüberwachten und überwachten Lernens ist. Bei diesem Lernen steuern kleine gelabelte Datensätze eine große Menge an ungelabelten Datensätzen. Die verwendeten Technologien von GANs (Generative Adversarial Networks) bis zu Diffusionsmodellen sind in der Lage neue Inhalte zu schaffen und sind Voraussetzungen für heutige generative KI.

2.1.2 Neuronale Netze

Neuronale Netze oder auch künstliche neuronale Netze (KNN) sind spezifische Typen des maschinellen Lernens. Sie sollen die biologischen Neuronen des Gehirns nachempfinden. Die Abbildung 2.2 von [11] zeigt eine stark vereinfachte biologische Nervenzelle.
Bei Nervenzellen werden elektrische Eingangssignale über Dendriten aufgenommen und in den Zellkern geleitet. Dort werden die eingehenden Signale zusammen geführt und es bildet sich das Aktionspotential. Übersteigt es das Schwellenpotential der Zelle, so wird das Signal über das Axon abgeleitet, die Nervenzelle „*feuert*“.

Die kleinste Einheit in künstlichen neuronalen Netzen sind die Neuronen. Sie sind den biologischen Nervenzellen nachempfunden.
Sie haben als Eingangswert einen Vektor und als Ausgangssignal ein Skalar. Außer in der Eingabe Schicht ist jedes Eingangssignal x_n ein Ausgangssignal y_{out} eines anderen Neuron. Die Wichtungen der

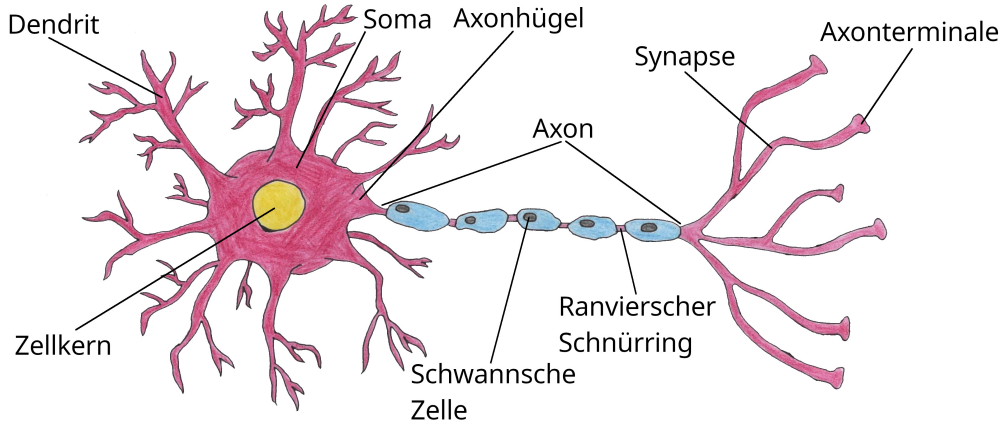


Abbildung 2.2: Biologische Nervenzelle

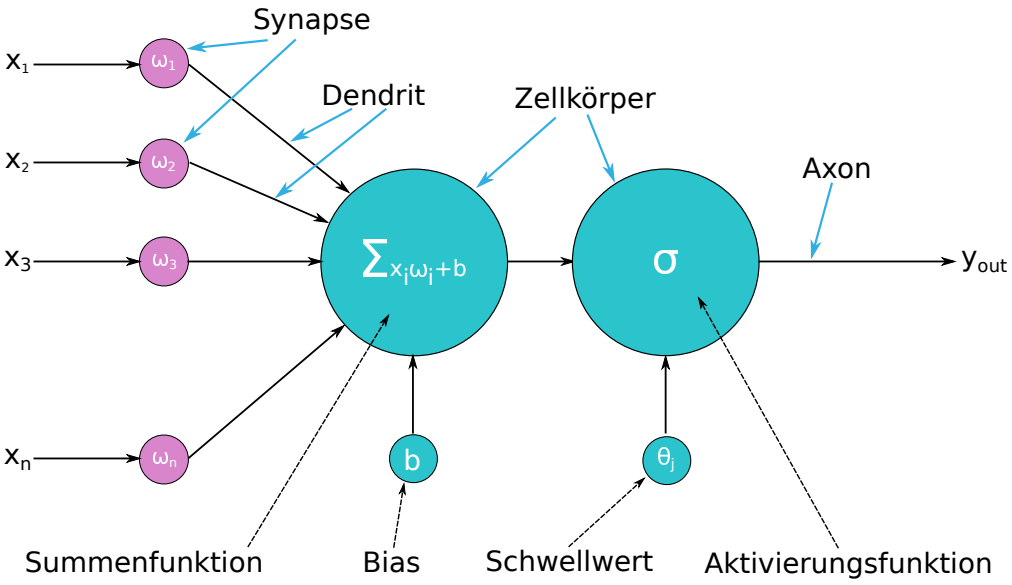


Abbildung 2.3: Künstliche Nervenzelle

Eingangssignale modellieren den synaptischen Spalt zwischen zwei biologischen Nervenzellen. Dieser kann ebenfalls verstärkten oder hemmend wirken. Alle Eingangssignale zusammen mit den Wichtungen, werden durch die Summenfunktion aufaddiert. Im Anschluss wird das Bias mit eingerechnet. Die Formel 2.1 zeigt die Summenfunktion für n Eingangssignale mit Beachtung des Bias Wert.

$$y_{sum} = x_1 + x_2 + \cdots + x_n + b \tag{2.1}$$

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.

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

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 große Fortschritte gemacht, mit immer besser werdender Technik und Verständnis der biologischen 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 vielen Hidden Layern zwischen der Ein- und Ausgabeschicht. Diese Strukturen erlauben die Verarbeitung und Analyse komplexer Datenmuster.

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 wie *Feedforward Neural Networks* und *Convolutional Neural Networks*, wie [12] 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.

Die heutigen
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 heutigen Sprachmodelle sind Modelle, welche die Transformer Technik verwenden.
Die grundlegende Funktionsweise der Large Language Models kann in vier Hauptkomponenten 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. Dekodierung: Auswahl der Ausgabestrategie

2.3.2 Grenzen und Probleme bei LLMs

Auch wenn Künstliche Intelligenz mit ihren großen Sprachmodellen in vielen Bereichen der privaten Nutzer und in den Prozessen von Unternehmen immer präsenter wird, haben die diese auch Grenzen. Im folgen 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 benötigen.

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 und es erforderlich ist, die Ergebnisse zu validieren. Nur so kann die ein Vertrauensverlust und eine stetige Weiterentwicklung der Modelle erfolgen.

2.3.4 Auswahl der LLMs

Die folgenden Modelle werden getestet und die Ergebnisse anschließend evaluiert.

Modell	Parameter	Größe	Sprache	offen	Ausführung	Benchmark
Qwen2.5-coder	32b	19 GB	German	X	lokal	HumenEval-XL
Deepseek-coder-V2	16B	8,9 GB	German	X	lokal	HumenEval-XL
Llama3.1-Claude	8b	4,7 GB	German	X	lokal	HumenEval-XL
Llama3.2	3b	2,0 GB	German	X	lokal	HumenEval-XL
Codellama	13b	7,4 GB	German	X	lokal	HumanEval-XL
Gemini Flash 1.5	k.A.	k.A.	German	-	online	HumenEval-XL
In Planung, wenn ich es schaffe						
Llama3.3	70b	43 GB	German	X	lokal	HumanEval-XL

2.4 Koordinationsstrategien für LLMs

Die Large Languarge Models haben große Leistungen auf dem Gebiet der Verarbeitung natürlicher Sprache gezeigt. Zunehmend arbeiten mehrere LLMs für diese Aufgaben zusammen. In diesem Fall spricht man von Agenten, die jeweils eine LLM darstellen können.

Werden für unterschiedliche Aufgaben verschiedene Modelle verwendet, spricht man von Agenten. Ein Agent ist eine autonome Einheit. Sie ist in der Lage ihre Umwelt wahr zunehmen, Entscheidungen zu treffen und führt ihre Handlungen aus, um ein definiertes Ziel zu erreichen. Dies kann beispielsweise durch die BDI-Architektur umgesetzt werden. Jeder Agent ist auf unterschiedliche Aufgaben spezialisiert. In [13] werden Multi-Agenten-System mit Team aus der Softwareentwicklung verglichen und gleich gesetzt.

Es gibt einige Methoden Large Language Model miteinander zu kombinieren, beispielsweise „Pipeline-Architektur“ und „Modular Approaches“. Im Folgen Kapiteln werden die zwei Ansätze für die Zusammenarbeit von mehreren LLMs, *Orchestrierung* und *Multi-Agenten-System (MAS)* kurz erläutert.

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. Gleichzeit 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-, Backenend- oder Datenbank-Agent umsetzt 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 ebenfalls aus mehreren Agenten. 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 als kollaborativ oder als 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 er Frameworks, wie React verwenden und auf externe Designer Tool zugreifen. Ein weiterer Agent ist der *Backend-Agent*, der 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 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 *Deplоment-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 Bereich 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

Siehe Prompting Techniques Hinweise für die Optimierung von Prompts. 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.

Kontextuelle Prompts haben neben der eigentlichen Eingabeaufforderung zusätzliche Informationen für die LLMs. 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:

```php
function sortiere_und_gebe_aus($zahlen) {
  // Die Liste wird nach aufsteigender Reihenfolge sortiert
  sort($zahlen);

  // Die sortierte Liste wird ausgegeben
  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.

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,

Prompttemplates

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

```
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:
[Liste der Bedingungen: z.B. Fehlerbehandlung, bestimmte Algorithmen]
. Die Ausgabe der Funktion soll in folgendem Format sein:
[Erwartetes Ausgabeformat]
```

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 () 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 GitLab[1] und Jenkins[2] Anwendung. Gerade in der Entwicklung von cloudbasierten Anwendungen kommen Containertools wie Docker[3] in Verbindung mit Kubernetes[4] 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.

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 steck hier noch in den Anfängen, bietet aber sehr gute Ansätze viele Aufgaben zu automatisieren oder als Werkzeug um die Entwicklung effizienter zu gestalten.

Die Codegenerierung für Designelemente kann ebenso mittels NL2Code erfolgen wie komplexe Backendfunktionalitäten. 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 großen Sprachmodelle wird der HumanEval-XL verwendet, welche unter https://github.com/FloatAI/humaneval-xl/tree/main heruntergeladen werden können.

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

[2] Jenkins 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.

[3] Durch 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.

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

Der Benchmark besteht aus Prompt, 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

Durch diesen Aufbau werden neben dem eigentlichen Prompt ein Test, eine vorgeschlagene Lösung mitgeliefert, sowie weitere Metadaten mit geliefert. Die Abbildung 2.4 zeigt den Aufbau und damit alle wichtigen Bereiche des Benchmark-Tests.

HumanEval-Xl Benchmark

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

Deklaration (1)

```php
<?php
```

Kommentar (2)

```php
/**
 * Sie sind ein erfahrener PHP-Programmierer und hier ist Ihre Aufgabe.
 * ...
 *
 */
function belowZero($operations) {
```

Ergebnis (3)

```php
  // 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;
}
```

Test (4)

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

$arg00 = [];
$x0 = belowZero($arg00);
$v0 = false;
if (!compare($x0, $v0)) {
  throw new Exception(\"Error at 1th assert statement.\");
  ...
```

PHP (Problem 0)

Abbildung 2.4: Codegeneration

Implementierung

3

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.

3.1 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.

3.1.1 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 Toll 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.

3.1.2 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 7.3 und 7.4 zu sehen. Hierbei ist zu beachten das genügend freier RAM zur Verfügung steht, um die Modelle abzuspeichern.

3.1.3 Orchestrierung von Modellen

Die Orchestrierung der Modelle erfolgt mithilfe des Python-Frameworks Longchain. Hierbei werden an die Modelle verschiedene Anforderungen gestellt. Zum einen müssen die Modelle Code generieren, zum anderen ist die Anforderung Text zu erstellen oder zu überarbeiten. Die Abbildung 3.1 zeigt schematisch den Aufbau der orchestrierten Modelle. Der Textfilter sucht in der Ausgabe des ersten Modells den Prompt und eliminiert die Anweisungen und Erklärungen.

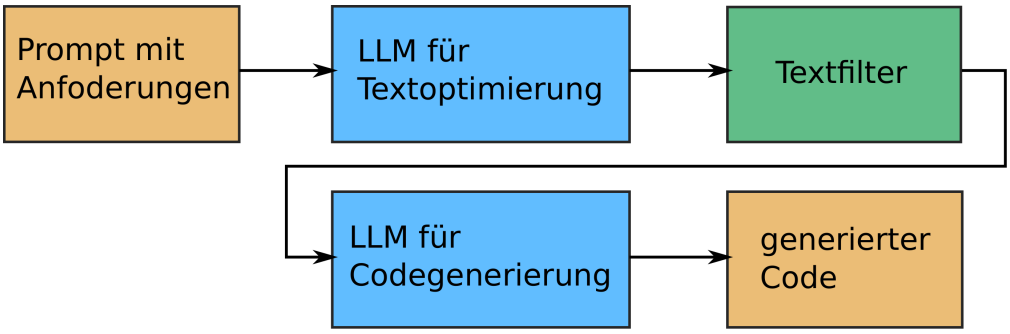


Abbildung 3.1: Orchestrierte LLM's für die Codegenerierung

3.2 Online Modelle

Text.

3.3 Benchmark Codeevaluation

Ausführen des Benchmarks.

3.3.1 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. Pro Problem erfolgen zehn Abfragen an jedes Modell.

3.3.2 Auswertung der Modellantworten

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 currect samples.
: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 3.1: Berechnung der pass@k Metrik in Python

3.4 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 3.3 vorgestellt wurde.

3.4.1 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.

3.4.2 JavaScript Codeevaluation

Text.

Evaluation

4

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.

4.1 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 4.1 zeigt die Berechnung der $pass@k$-Metrik.

$$\text{pass@k} = 1 - \frac{\prod_{i=0}^{n-k}(n-i-c)}{\prod_{i=0}^{n-k}(n-i)} \tag{4.1}$$

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 Formal 3.1 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 4.1 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 (Auswertung fehlt)	0,0	0,0	0,0
Gemini Flash 1.5 (Auswertung fehlt)	0,0	0,0	0,0
Qwen 2.5-Coder (Auswertung fehlt)	0,0	0,0	0,0
Mistral-small	0,0125	0,346329	0,65

Das Modell `codellama` 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 4.1 bis 4.4 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.`

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

Listing 4.1: Gemini Ergebnis für das PHP-3 Problem Version 1

Text.

```php
function greatestCommonDivisorBinary($a, b) {
  if ($a == 0) return $b;
  if ($b == 0) return $a;

  // Ensure a is odd
  if ($a % 2 == 0 && $b % 2 == 0)
    return 2 * greatestCommonDivisorBinary($a / 2, $b / 2);
  if ($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 - $b, b);
  return greatestCommonDivisorBinary($b - a, a);
}
```

Listing 4.2: Gemini Ergebnisse für das PHP-3 Problem Version 2

Text.

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

Listing 4.3: Gemini Ergebnisse für das PHP-3 Problem Version 3

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

Listing 4.4: 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 de ie Ausgabe davor beziehen, sodass der Code mit einer Kommentar-Zeile, wie beispielsweise
`// ... (Rest der Funktion bleibt ähnlich)`
ersetzt wurde.

4.1.1 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.

4.2 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

5

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.

5.1 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 5.1 dargestellt, aussehen.

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

```
    "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,
    "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,
    "Information": lambda x: int(x) > 0,
}
```

Listing 5.1: 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

5.1.1 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.

5.1.2 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.

5.1.3 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 vergleichen. Als Schlussfolgerung aus dem Ergebnis dieser Arbeit, wird auch hier die Codeanalyse durch eine LLM nicht erfolgen, sondern ebenfalls durch SonarQube.

Diskussion und Ausblick

6

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,

6.1 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: langsameres, 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 | Karrierrewelt.

6.2 Praktische Anwendung

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

6.2.1 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.

Fazit

7

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 Schwachen 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] 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).

[5] Juan David Velásquez-Henao, Carlos Jaime Franco-Cardona und Lorena Cadavid-Higuita. „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.

[6] 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).

[7] 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).

[8] 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).

[9] 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).

[10] 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).

[11] Johanna Pahl. *Zeichnung einer biologische Zelle*. 26. Sep. 2024.

[12] 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.

[13] Zhuoyun Du u. a. *Multi-Agent Software Development through Cross-Team Collaboration*. 13. Juni 2024. URL: https://arxiv.org/abs/2406.08979 (besucht am 04. 10. 2024).

[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.

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

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

Listing 7.1: Ollama Hostanpasssng 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.
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 7.2: 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 Hugging Face Modelle

Ein Hugging Face Modell kann wie im Listing 7.3 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

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

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

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

    generated_ids = model.generate(**model_inputs, max_new_tokens=512)
    generated_ids = [
        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)

# 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",
    prompt="Write a quick sort algorithm.",
)
```

Listing 7.3: 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.

```
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.

    :param root_name (str): Root name of model.
    :param vendor_name (str): Vendor name of model.
    """
    # Personal access token from Hugging Face.
    access_token = "hf_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

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

```
    tokenizer = AutoTokenizer.from_pretrained(
        f"{root_name}/{vendor_name}"
    )
    model = AutoModelForCausalLM.from_pretrained(
        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)

# 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"
)
```

Listing 7.4: Laden der Modelle von Hugging Face und lokal speichern

Um die Modelle abzufragen, kommt der Code, welcher in Listing 7.5 zu sehen ist, zum Einsatz.

```
"""Create answers."""

import os
import json
from transformers import AutoTokenizer, pipeline, AutoModelForCausalLM
from langchain_ollama.llms import OllamaLLM
from langchain.prompts import PromptTemplate

NUMBERS_OF_PROBLEMS: int = 80
START_NUMBER_OF_PROBLEMS: int = 0
STOP_NUMBER_OF_PROBLEMS: int = 80

PROBLEM_FILE_NAME: str = "PHP_German.jsonl"
PROBLEM_LANGUAGE: str = "php"

MODEL_NAME: str = "qwen2.5-coder:32b"
SUBFOLDER_NAME: str = "qwen25-coder"
MODEL_PATH = f"/huggingface/models/{MODEL_NAME.replace('-', '_')}"

PROMPT_REPETITIONS: int = 10
```

```python
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.
    :param language (str): Coding/Programming language.
    :returns: dict | None: Prompt and problem.
    """
    jsonl = open(
        f"{os.path.dirname(os.path.realpath(__file__))}/problems/{
            PROBLEM_FILE_NAME}",
        "r",
        encoding="utf-8",
    )
    lines = jsonl.readlines()
    problem: dict = None
    for line in lines:
    problem = json.loads(line)
    if problem.get("task_id", None) == f"{language}/{task_id}":
        break
        jsonl.close()
    return problem

def run_server_model(problem: str):
    """
    Connect to ollama model server.
    :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",
        model=MODEL_NAME,
    )
    promt_template: PromptTemplate = PromptTemplate(
    input_variables=["user_prompt"],
        template="{user_prompt}",
    )
    prompt = promt_template.format(user_prompt=problem)
    return ollama.invoke(prompt)
```

```
59
def run_local_model(problem: str):
    """
    Connect to local model.
    :param problem (str): Prompt for model.
64  :returns Result from model.
    """
    tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
    model = AutoModelForCausalLM.from_pretrained(
        MODEL_PATH, torch_dtype="auto", trust_remote_code=True
69  )
    pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
    promt_template: PromptTemplate = PromptTemplate(
        input_variables=["user_prompt"],
        template="{user_prompt}",
74  )
    prompt = promt_template.format(user_prompt=problem)
    generation_args = {
      "max_new_tokens": 600,
      "return_full_text": False,
79    "do_sample": False,
    }
    output = pipe(
        prompt,
        **generation_args,
84      pad_token_id=tokenizer.eos_token_id
    )
    if len(output) > 0:
        return output[0].get("generated_text", "")
    return ""
89

def write_log(answer: str, key: str, task_id: str) -> None:
    """
    Write the proompt result in JSON file.
    :param answer (str): _description_
94  :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('/', '-')
        }.json"
    answer = answer.replace("\n", r"\n")
    answer = answer.replace('"', r"\"")
    answer = f"\"{key}\":\"{answer}\""
    with open(file_name, "a", encoding="utf-8") as file:
        file.write("{" + answer + "}\n")

def execute_prompt(task_id: int, model_type: str) -> bool:
    """
    Create prompts and save local.
    :param task_id (int): Problem id
    :param model_type (str): Type of model.
    :returns bool: True prompt create.
    """
    language: str = PROBLEM_LANGUAGE
    problem: dict = read_problem(task_id=task_id, language=language)
    prompt: str = problem.get("prompt", None)
    if prompt is None:
        return False
    current_count: int = 0
    max_count: int = PROMPT_REPETITIONS
    while current_count < max_count:
        answer: str = ""
        if model_type == "local":
            answer = run_local_model(problem=prompt)
        elif model_type == "server":
            answer = run_server_model(problem=prompt)
        write_log(
            answer=answer,
            key=f"result_{current_count}",
            task_id=f"{language}/{task_id}",
        )
        current_count += 1
    return True
```

Listing 7.5: Laden der Modelle von Hugging Face und lokal speichern

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'':
{ßtreet": ''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