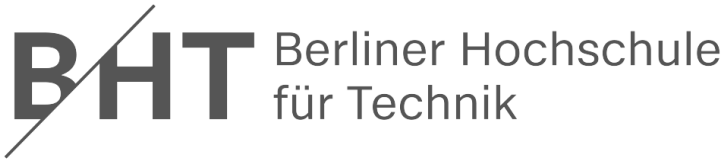
Evaluierung und Optimierung von Large Language Models für die Entwicklung von Webanwendungen

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 18. November 2024

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 18. November 2024

Unterschrift

Abstract

Abstract in Englisch.

Zusammenfassung

Zusammenfassung in Deutsch.

Inhaltsverzeichnis

Abstract i

Abbildungsverzeichnis v

Tabellenverzeichnis vi

Listings vii

Abkürzungsverzeichnis viii

1 Einleitung 1
- 1.1 Hintergrund und Kontext . . . 1
- 1.2 Problemstellung . . . 2
- 1.3 Stand der Forschung . . . 2
- 1.4 Zielsetzung und Forschungsfragen . . . 3
  - 1.4.1 Auswahl der LLMs . . . 4
  - 1.4.2 Prompt-Engineering . . . 4
- 1.5 Aufbau der Arbeit . . . 5
- 1.6 Abgrenzung . . . 5

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 . . . 11
- 2.2 Natural Language Processing . . . 11
- 2.3 Large Language Model . . . 12
  - 2.3.1 Grundlagen . . . 12
  - 2.3.2 Grenzen und Probleme bei LLMs . . . 13
  - 2.3.3 Verständnis für die LLMs . . . 13
- 2.4 Koordinationsstrategien für LLMs . . . 14
  - 2.4.1 Orchestrierung von LLMs . . . 14

2.4.2 Multi-Agenten-Systeme . . . 15
2.5 Prompt Engineering . . . 16
2.5.1 Prompt-Techniken . . . 16
2.5.2 Grenzen beim Prompt-Engineering für LLMs . . . 16
2.6 Grundlagen der Webentwicklung . . . 17
2.6.1 Programmiersprachen . . . 17
2.6.2 Entwicklung . . . 17

3 Implementierung **19**
3.1 Modelle lokal aufsetzen . . . 19
3.1.1 Install Ollama . . . 19
3.1.2 Open WebUI . . . 20
3.1.3 Python Client . . . 20

4 Evaluation **21**
4.1 Bewertung der Modelle . . . 22
4.2 Einfache HTML Seite . . . 24
4.2.1 ChatGPT 3.5 . . . 24
4.3 Einfache HTML Seite . . . 29
4.3.1 ChatGPT 3.5 . . . 29

5 Lessons Learned **33**

6 Anwendungsszenarien **35**

7 Diskussion und Ausblick **37**

8 Fazit **39**

Literatur **41**

Glossar **42**

Anhang **43**

Abbildungsverzeichnis

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

Tabellenverzeichnis

Listings

3.1 Ollama Hostanpasssng für Netzwerkbetrieb 19
4.1 JavaScript Ergebnise der Modelle (gekürzt) 22
4.2 JavaScript: Unit Test der Ergebnise 22
4.3 HTML: Einbindes des Testframeworks 23

Einleitung

1

1.1 Hintergrund und Kontext

Durch die zunehmende Globalisierung und Digitalisierung wird die Gesellschaft 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 stellen ihre Geschäftsprozesse um. 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 (Beweis fehlt). Ein weiteres Problem ist der wachsende Fachkräftemangel in der Wirtschaft und die damit verbundenen steigenden Gehälter der Entwickler (Beweis fehlt).

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. Tiefe Kenntnisse der verwendeten Programmiersprache sind nicht mehr in dem Maße erforderlich, wie ohne LLMs.

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 Mensch muss die generierten Ergebnisse überprüfen, ehe erstellte Programmcodestücke in vorhandene Programme eingefügt und in Produktionsumgebungen implementiert werden.

Viele Entwickler setzen auf Chatbots, wie ChatGPT oder Gemini zur Generierung von Code, wie eine Umfrage von *stackoverflow* vom Mai 2024 zeigt [3]. Gleichzeitig wachsen auch die technischen Schulden bei Softwareprojekten, da diese Modelle nicht für die Entwicklung von Software optimiert sind (Beweis fehlt).

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 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 wissenschaftlicher 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.

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 Softwareentwicklung 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, ob und wie weit sich der Prozess der Codegenerierung automatisieren lässt und ob einige Programmiersprachen, die in der Webentwicklung Verwendung finden, besser unterstützt und geeignet sind als andere und somit zu bevorzugen sind.

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 großen Sprachmodellen, für die Erstellung von Webanwendungen automatisieren.

Z4 Sind einige Programmiersprachen für die Codegenerierung besser geeignet als andere.

1.4.1 Auswahl der LLMs

Wird noch im Verlauf der Arbeit geändert

Als Referenzen kommen ChatGPT3.5 und das aktuelle Google Sprachmodell der Gemini-Familie zum Einsatz.

Als lokale Modelle werden zurzeit deepseek-coder-v2, llama3.1-claude und llama3.2 verwendet. Diese Modelle kommen auch zum Einsatz, für die Orchestrierung und die Multi-Agenten-Systeme.

Das neue Model Mistral Lage 2 (Links: Mistral Large 2 | Mistral AI und Mistral Large 2 | the-decoder.de [gelsen am: 03.11.2024]) soll in Sachen Coding mit Modellen wie GPT-4o, Claude 3 Opus und Llama 3 mithalten können. (Gibt es auch von mistral-lage | ollama Models)

Evtl. werden die Granite Code Models aus [6] auch getestet.

1.4.2 Prompt-Engineering

Wird noch im Verlauf der Arbeit geändert

Ob alle in Kap. 2.5.1 vorgestellten Prompt-Techniken Verwendung finden, steht zurzeit noch nicht fest. Evtl. kommen andere hinzu.

Evaluierung der Ergebnisse

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.

HumanEval-Benchmark and pass@k-Metrik

Einstieg

Modell

Huggenface

Github

Einführendes Paper

1.5 Aufbau der Arbeit

Ein paar Worte zum Aufbau dieser Arbeit. Um ein grundlegendes Verständnis für diese Arbeit zubekommen, werden im Kapitel 2 die Grundlagen besprochen.

Im Kapitel **??** wird der aktuelle Stand der Forschung vorgestellt und Erkenntnisse anderer Arbeiten diskutiert. Die Implementierung der Test LLMs wird in Kapitel 3 besprochen und in Kapitel 4 die Ergebnisse evaluiert.

Die negativen und positiven Erfahrungen und Herausforderungen werden in Kapitel 5 aufgegriffen und Lösungsansätze vorgeschlagen, die in den nachfolgenden Kapiteln vorgestellt werden.

Bevor in Kapitel 8 auf mögliche Folgearbeiten eingegangen wird, gibt es in Kapitel 6 Anwendungsszenarien, die auf den zuvor gewonnen Ergebnissen aufbauen und vorgestellt werden.

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.

Mein roter Faden: noch was zum Testen

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

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.

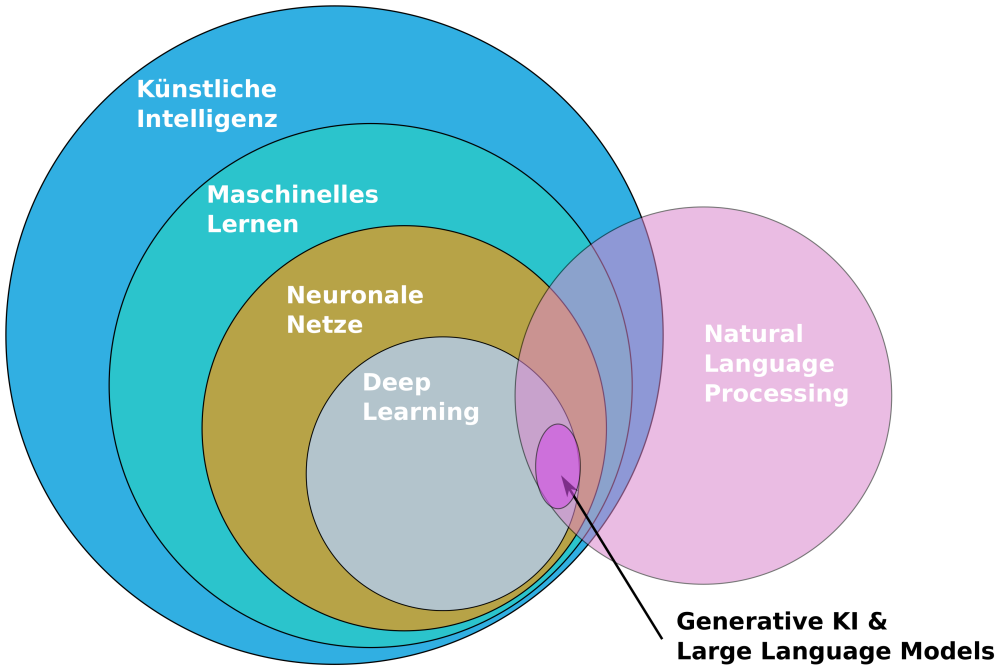


Abbildung 2.1: LLMs im Kontext der Forschungsbereiche von KI

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 [9, 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 [10] 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

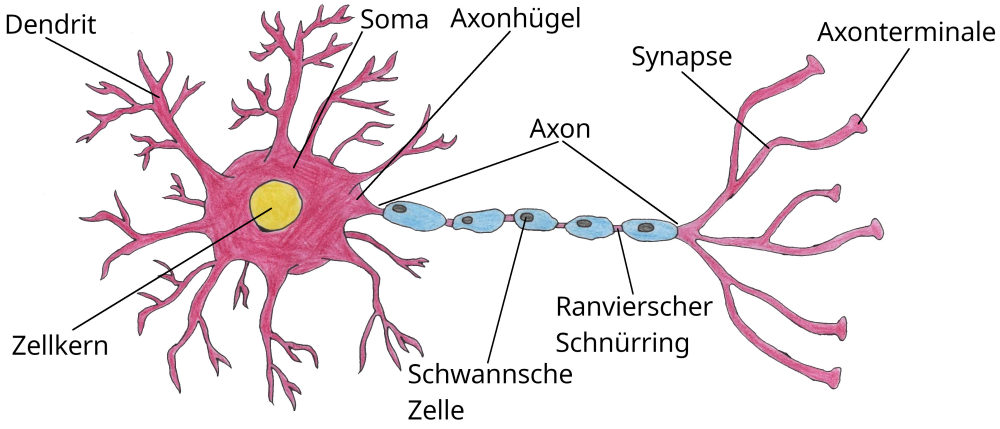


Abbildung 2.2: Biologische Nervenzelle

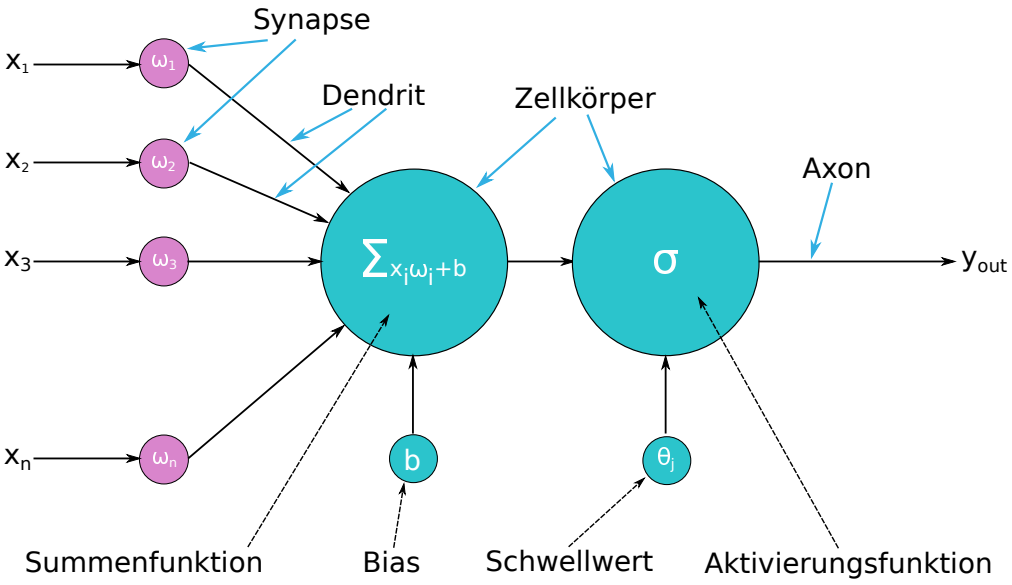


Abbildung 2.3: Künstliche Nervenzelle

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 [11] 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.

2.4 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 [12] 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 *Deplomen-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 [13] beschrieben, in Form von einer einfachen Anweisung bis hin zu detaillierten Beschreibungen oder spezifischen Aufgaben erfolgen.

[Hier Beispiel von ChatGPT oder Gemini einfügen, kann als Bild]

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.

2.5.2 Grenzen beim Prompt-Engineering für LLMs

Trotz der bemerkenswerten linguistischen Leistung, stoßen große Sprachmodelle an ihre Grenzen, unter anderem wie in [13] beschrieben,

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.

[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 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.

Implementierung

3

3.1 Modelle lokal aufsetzen

Als Server dient ein Debian 12 System.

3.1.1 Install Ollama

Ein Skript ausführen

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

Ein Model laden und im Anschluss starten, Beispiel

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

Config Ollama

Set correct IP and Post in /etc/systemd/system/ollama.service

```
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"
-
+   Environment="OLLAMA_HOST=0.0.0.0"
+   Environment="OLLAMA_MODELS=/home/ai/models"
+
```

Listing 3.1: Ollama Hostanpasssng für Netzwerkbetrieb

3.1.2 Open WebUI

Optional kann ein grafisches Tool, zum Testen und verwalten vom Ollama-Server im Netzwerk installiert werden. Der Aufruf der UI, kann mittel Browser erfolgen. Hier wird die IP und der Port 8080 angegeben. Beispiel `http://192.168.2.45:8080`.

```
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
-o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io \
docker-buildx-plugin docker-compose-plugin
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
```

3.1.3 Python Client

```
pip3 install langchain
pip3 install ollama
pip3 install mistral
```

Evaluation

4

Mein roter Faden

Alle Prompts und Ausgaben der LLMs kommen noch in den Anhang.

Auswertung des erzeugten Codes:

- Korrektheit
  - Syntax: Regeln nach Programmiersprache
  - Semantik: Funktionalität erfüllen, Aufgabe korrekt lösen
  - Fehlerfreiheit: keine Laufzeitfehler/unerwartete Ausgaben
- Qualität
  - Lesbarkeit: Struktur, einfach
  - Effizienz: Ressourcen schonen
  - Wartbarkeit: leicht verständlich
  - Best Practices: Coding Standards für den Kontext (z.B. WordPress)
- Relevanz für Aufgabenstellung
  - Vollständig: alle Aspekte der Aufgabenstellung
  - Präzision: genaue Lösung d. Aufgabe
  - Elemente: keine ungenutzten Elemente
- Anpassbarkeit
  - Parametrisierung: möglichst flexible, leichte Änderbarkeit für Parameter
  - Erweiterbarkeit: leicht erweiterbar
- Innovation

- alternative Lösungen: LLMs sollte mehrere anbieten können
- neue Ideen: hat der Code so was?

Wie können automatische Test umgesetzt und Code Metriken erstellt werden? Evtl. anderen Entwicklern zeigen und bewerten lassen!

4.1 Bewertung der Modelle

Für die Bewertung wird das Vorgehen gewählt, welches in [14] beschrieben wir. Dies wird exemplarisch für JavaScript und PHP angepasst. Anfragen an die LLMs und deren Ergebnisse werden in Funktionen gefasst und automatische mit Unit Test geprüft.

```
function gemini_create_solution(numbers) {
  let sum = 0;
  for (let i = 0; i < numbers.length; i += 2) {
    if (numbers[i] % 2 !== 0) {
      sum += numbers[i];
    }
  }
  return sum;
}

function gpt_create_solution(numbers) {
  let sum = 0;
  for (let i = 0; i < numbers.length; i += 2) {  // Only check even
     positions (0, 2, 4, ...)
    if (numbers[i] % 2 !== 0) {  // Check if the element is odd
      sum += numbers[i];
    }
  }
  return sum;
}
```

Listing 4.1: JavaScript Ergebnise der Modelle (gekürzt)

Nachdem die Ergebnisse von den Modellen vorliegen, können die Tests in JavaScript erstellt werden. Hierfür wird das JavaScript Framework Jasmin verwendet.

```
describe(
```

```
  'HumanEval: Given a non-empty list of integers, return the sum of all of
      the odd elements that are in even positions.',
  () => {
    it('Gemini create test 1', () => {
      expect(gemini_create_solution([5, 8, 7, 1])).toBe(12);
    })
    it('Gemini create test 2', () => {
      expect(gemini_create_solution([3, 3, 3, 3, 3])).toBe(9);
    })
    it('Gemini create test 3', () => {
      expect(gemini_create_solution([30, 13, 24, 321])).toBe(0);
    })

    it('ChatGPT3 create test 1', () => {
      expect(gpt_create_solution([5, 8, 7, 1])).toBe(12);
    })
    it('ChatGPT3 create test 2', () => {
      expect(gpt_create_solution([3, 3, 3, 3, 3])).toBe(9);
    })
    it('ChatGPT3 create test 3', () => {
      expect(gpt_create_solution([30, 13, 24, 321])).toBe(0);
    })
  })
```

Listing 4.2: JavaScript: Unit Test der Ergebnise

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Testing with Jasmine</title>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
    jasmine/2.8.0/jasmine.min.css">
<script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.8.0/
    jasmine.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.8.0/
    jasmine-html.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.8.0/boot.
    min.js"></script>
<script src="solution.js"></script>
```

```
11    <script src="solutionSpec.js"></script>
      </head>

      <body>
      </body>
16
      </html>
```

Listing 4.3: HTML: Einbindes des Testframeworks

4.2 Einfache HTML Seite

4.2.1 ChatGPT 3.5

Python Funktion

Schreibe eine Python Funktion die FPDF Dokument als Eingabe erhält und eine Zeichenkette zurückliefert.
(engl.: Write a Python function that takes FPDF document as input and returns a string.)

Antwort ChatGPT

Antwort.

Copilot

Klar, ich kann dir dabei helfen! Hier ist ein Beispiel für eine Python-Funktion, die ein FPDF-Dokument als Eingabe erhält und den Textinhalt als Zeichenkette zurückliefert. Dafür verwenden wir die Bibliothek PyMuPDF (auch bekannt als fitz), um das PDF zu lesen:

```
import fitz  # PyMuPDF

def pdf_to_string(pdf_path):
    # Öffne das PDF-Dokument
    document = fitz.open(pdf_path)
    text = ""

    # Iteriere durch jede Seite und extrahiere den Text
    for page_num in range(len(document)):
        page = document.load_page(page_num)
        text += page.get_text()

    return text

# Beispielverwendung
pdf_path = "dein_dokument.pdf"
text_inhalt = pdf_to_string(pdf_path)
print(text_inhalt)
```

Stelle sicher, dass du die PyMuPDF-Bibliothek installiert hast. Du kannst sie mit folgendem Befehl installieren:

```
pip install pymupdf
```

Diese Funktion öffnet das PDF-Dokument, iteriert durch jede Seite und extrahiert den Text. Wenn du Fragen hast oder weitere Anpassungen benötigst, lass es mich wissen!

Python Funktion

Schreibe eine Python Funktion die FPDF Dokument als Eingabe erhält und eine Zeichenkette zurückliefert. Verwende dafür die Python Bibliothek FPDF.
(engl.: Write a Python function that takes FPDF document as input and returns a string.)

Copilot

Die Bibliothek FPDF wird hauptsächlich zum Erstellen von PDF-Dokumenten verwendet und bietet keine direkte Methode zum Extrahieren von Text aus einem PDF. Um Text aus einem PDF-Dokument zu extrahieren, könntest du eine andere Bibliothek wie PyMuPDF oder PyPDF2 verwenden.
Falls du jedoch darauf bestehst, FPDF zu verwenden, um ein PDF zu erstellen und dann den Text aus diesem PDF zu extrahieren, könntest du den Text während des Erstellungsprozesses speichern und später zurückgeben. Hier ist ein Beispiel, wie du das machen könntest:

```
from fpdf import FPDF

class PDF(FPDF):
    def __init__(self):
        super().__init__()
        self.text_content = ""

    def add_text(self, text):
        self.text_content += text + "\n"
        self.set_xy(10, 10)
        self.set_font('Arial', 'B', 12)
        self.multi_cell(0, 10, text)

    def get_text(self):
        return self.text_content

    def create_pdf_with_text(text):
        pdf = PDF()
            pdf.add_page()
        pdf.add_text(text)
```

```
        pdf.output("output.pdf")
        return pdf.get_text()

# Beispielverwendung
text = "Dies ist ein Beispieltext für das PDF."
text_inhalt = create_pdf_with_text(text)
print(text_inhalt)
```

In diesem Beispiel wird der Text während des Erstellungsprozesses des PDFs in einer Zeichenkette gespeichert und kann später abgerufen werden. Wenn du Fragen hast oder weitere Anpassungen benötigst, lass es mich wissen!

Python Funktion

Schreibe eine Python Funktion die FPDF Dokument als Eingabe erhält und eine Zeichenkette zurückliefert. Verwende dafür die Python Bibliothek FPDF und speicher das Ergebnis nicht als Datei zwischendurch ab.

Copilot

Die Bibliothek FPDF ist hauptsächlich zum Erstellen von PDF-Dokumenten gedacht und bietet keine direkte Methode zum Extrahieren von Text aus einem PDF. Um Text aus einem PDF-Dokument zu extrahieren, könntest du eine andere Bibliothek wie PyMuPDF oder PyPDF2 verwenden.
Falls du jedoch darauf bestehst, FPDF zu verwenden, um ein PDF zu erstellen und dann den Text aus diesem PDF zu extrahieren, könntest du den Text während des Erstellungsprozesses speichern und später zurückgeben. Hier ist ein Beispiel, wie du das machen könntest:

```
from fpdf import FPDF

class PDF(FPDF):
    def __init__(self):
        super().__init__()
        self.text_content = ""

    def add_text(self, text):
        self.text_content += text + "\n"
        self.set_xy(10, 10)
        self.set_font('Arial', 'B', 12)
        self.multi_cell(0, 10, text)

    def get_text(self):
        return self.text_content

    def create_pdf_with_text(text):
        pdf = PDF()
        pdf.add_page()
        pdf.add_text(text)
        return pdf.get_text()
```

```
# Beispielverwendung
text = "Dies ist ein Beispieltext für das PDF."
text_inhalt = create_pdf_with_text(text)
print(text_inhalt)
```

In diesem Beispiel wird der Text während des Erstellungsprozesses des PDFs in einer Zeichenkette gespeichert und kann später abgerufen werden. Wenn du Fragen hast oder weitere Anpassungen benötigst, lass es mich wissen!

4.3 Einfache HTML Seite

4.3.1 ChatGPT 3.5

HTML Startseite

Schreibe eine einfache HTML Seite für die Startseite eines Blogs.

ChatGPT 3.5

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mein Blog</title>
  <style>
    // styles.
  </style>
</head>
<body>
  <header>
    <h1>Willkommen auf meinem Blog</h1>
  </header>
  <nav>
    <a href="#">Startseite</a>
    <a href="#">Über mich</a>
    <a href="#">Blog</a>
    <a href="#">Kontakt</a>
  </nav>
  <div class="container">
    <article>
      <h2>Erster Blogbeitrag</h2>
      <p>Willkommen zu meinem ersten Blogbeitrag!</p>
    </article>
  <footer>
    <p>&copy; 2024 Mein Blog. Alle Rechte vorbehalten.</p>
  </footer>
</body>
</html>
```

ChatGPT 3.5

```
<!DOCTYPE html>
<html lang="de">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Mein Blog</title>
<style>
// styles.
</style>
</head>
<body>
<header>
<h1>Willkommen auf meinem Blog</h1>
</header>
<nav>
<a href="#">Startseite</a>
<a href="#">Über mich</a>
<a href="#">Blog</a>
<a href="#">Kontakt</a>
</nav>
<div class="container">
<article>
<h2>Erster Blogbeitrag</h2>
<p>Willkommen zu meinem ersten Blogbeitrag!</p>
</article>
<footer>
<p>&copy; 2024 Mein Blog. Alle Rechte vorbehalten.</p>
</footer>
</body>
</html>
```

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.

Anwendungsszenarien

6

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

Diskussion und Ausblick

7

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.

Wie in [15] beschrieben,

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 [16] 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.

Fazit

8

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] 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).

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

[11] 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.

[12] 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).

[13] 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).

[14] Mark Chen u. a. *Evaluating Large Language Models Trained on Code*. 7. Juli 2021. URL: https://arxiv.org/abs/2107.03374 (besucht am 12. 11. 2024).

[15] 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).

[16] 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