Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Praxistransferbericht

Vereinfachung von Entwicklungsprozessen durch Large Language Models

vorgelegt am 10. August 2024

Name:	Justin Becker
Matrikelnummer:	77204368351
Ausbildungsbetrieb:	SAP SE
Fachbereich:	FB2: Duales Studium — Technik
Studienjahrgang:	2023
Studiengang:	Informatik
Betreuerin Unternehmen:	Jenny Peter
Betreuerin Hochschule:	Carl Dolling
Wortanzahl:	2132

Sperrvermerk

Der vorliegende Bericht enthält vertrauliche Daten des Unternehmens SAP SE. Auf Wunsch des Unternehmens SAP SE ist der vorliegende Bericht für die öffentliche Nutzung zu sperren. Veröffentlichung, Vervielfältigung und Einsichtnahme sind ohne ausdrückliche Genehmigung des Unternehmen SAP SE, in 14469 Potsdam und des Verfassers Justin Becker nicht gestattet. Der Bericht ist nur den Gutachtern und des Mitgliedern des Prüfungsausschlusses zugänglich zu machen.

Ort, Datum	Justin Becker

Von der betrieblichen Betreuung zur Kenntnis genommen:

Ort, Datum	Jenny Peter

Zusammenfassung

Mit dem erneuten Aufschwung von KI Technologien findet Sie sich in vielen Anwendungsfälle wieder. So auch in der Software Entwicklung. Dieser Bericht wird sich damit auseinander setzen, wie mit Hilfe von Large Language Models (LLMs) Codebasen durchsucht werden können. Dafür wird der Ansatz verfolgt, einen Index aus sogenannte Vektor Embeddings zu erstellen, welcher dann mit Fragen natürlicher Sprache dursucht werden kann. Für das Erstellen der Vektor Embeddings werden LLMs genutzt. Es wird ein Test Projekt und Test Fragen herangezogen, um die im folgenen Bericht erarbeiteten Verfahren dafür zu evaluieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Hinweis

Aus Gründen der besseren Lesbarkeit wird im Text verallgemeinernd die männliche Form verwendet. Diese Formulierungen umfassen gleichermaßen weibliche und männliche Personen.

Akronyme

LLM Large Language Model

1 Einleitung

Bereits als 2017 das Paper „Attention is all you need“ von Mitarbeitern bei Google veröffentlicht wurde, gab es Optimimus gegenüber dem vorgestellten Aufmerksamkeitsmechanismus und Transformer Architektur. Nach erfolgreichen Tests dieser Prinzipien mit Englisch-Deutsch Übersetzungen wurde der Schluss gezogen: „We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text[...]“[16]. Doch allein diese Textverarbeitung durch KI Modelle erlebte einen Boom in der Öffentlichkeit mit dem Erscheinen von ChatGPT im November 2022. Ein ChatBot, der die Technologie zur Textverarbeitung von natürlicher Sprache, sogenannte LLMs, leicht zugänglich präsentiert. [3]

Durch die hohe Publizität wurde schnell nach Anwendung und Nutzung der Möglichkeiten in allen Bereichen des Lebens gesucht. So wurde die Technologie im Bereich der Bildung genutzt, um mit ihr zu Lernen, Ergnisse abzugleichen und Antworten zu verifizieren. Auch in der Medizin wurde Experimente dazu durchgefürt Patienten mit dem ChatBot einen Ansprechpartner zu geben oder sogar Krankheiten durch diesen zu identifizieren. [7]

So kam es ebenfalls, dass LLMs Einzug in die Entwicklungsprozesse von Software gefunden haben und dort eine Reihe von Aufgaben erledigen. Dazu gehört unter anderem die Generierung von Code, das Beheben von Fehlern im Code oder das Erklären von Code. Empirisch konnte in diesen Anwendungsbereichen das Potential der Technologien belegt werden. [14]

Vorallem in größeren Software Projekten kann es zu einem Problem werden einen Überblick über die Codebasis zu gewinnen und zu behalten. Deshalb wird dieser Praxistransferbericht untersuchen, wie man LLMs Nutzen kann, um eine Codebasis mit Fragen in natürlicher Sprache zu durchsuchen.

2 Theoretischen Grundlagen

2.1 Large Language Models

LLMs sind KI Systeme, die dadruch Charakterisiert sind, dass sie Information über natütürliche Sprachen kapseln. Dazu gehören nebst Sprachmuster und Syntax auch die Erfassung von Bedeutung und Zusammenhängen. [19]

Einen Ausdruck dieser Verständnissfähigkiet bringen LLMs durch sogenannte Completions (dt. Vervollständigungen). Dabei wird ein Textabschnitt vorgegeben und das LLM generiert auf dieser Grundlage sukzessiv Folgetext. In diesem Rahmen werden durch das LLM auch sogenannte Vektor Embeddings erzeugt. [10]

Außerdem können LLMs so konfiguriert werden, dass Sie nicht deterministisch arbeiten, d. h., dass selbst bei einer gleichen Eingabe unterschiedliche Ausgaben entsehen können. Diese Konfiguration ist bei der Arbeit mit LLMs in diesem Bericht vorgenommen.

Die genaue teschnische und mathematische Umsetzung zur Funktionsweise dieser Modelle spielt dabei in diesem Bericht keine Rolle.

2.2 Vektor Embeddings

Embeddings sind hochdimensionale Vektoren, die Syntax, Semantik und Zusammenhänge zwischen semantischen Objekten (im folgenden meist als Bedeutung bezeichnet) dartstellen können. Diese Darstellung kann für unterschiedlichste Datenformen wie Text, Bild oder auch Audio gelten. [9] In diesem Bericht wird sich auf die Darstellung von Text fokusiert und die anderen Formen werden in den Beschreibungen vernachlässigt, doch sind die Grundprinzipien Datenformen unabhänig.

Um Bedeutungen in Embedding Darstellung zuerfassen, bildet nebst einest Textteils selbst, wie einem einzeln Wort, der Kontext von diesem Textteil die Grundlage. [13] Das heißt, dass für die Qualität der Embeddings die Einarbeitung des Kontextes in das Embedding relevanz hat. [5]

2.3 Vektor Raum

Die Vektor Embeddings werden selten alleine betrachtet, sondern als Teil eines Vektor Raums. In diesem Raum wird mit den Embeddings Richtung und Entfernung von Vektoren Bedeutung zugewiesen. So liegt Text, der ähnliche Semantik hat, nahe aneinander in seiner Embedding Darstellung in diesem Vektor Raum. Außerdem stellt das Verhältnis zwischen zwei Vektoren semantische Konzepte dar und es könne "Konzept Achsen"gebildet werden, die verallgemeinernd Konzepte erfassen wie in Abbildung 1 zu erkennen. [4, 9]

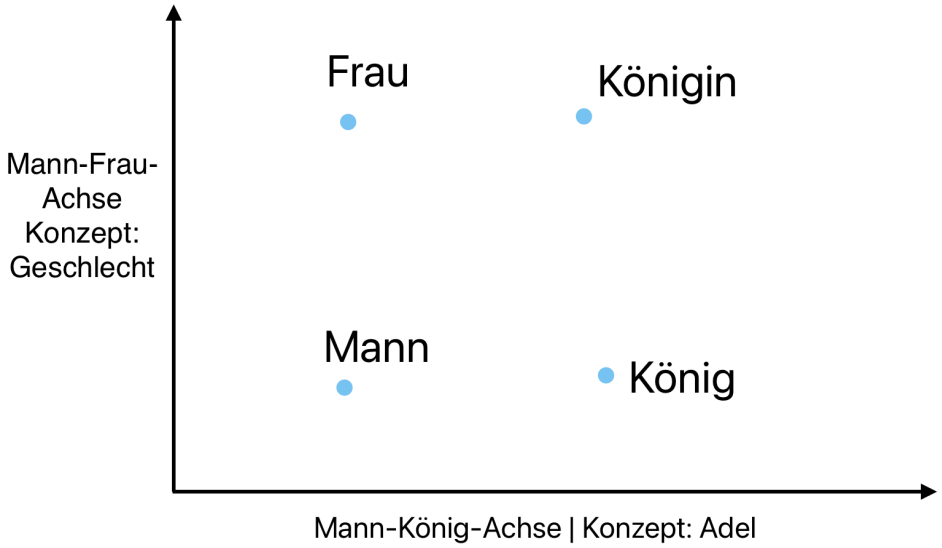

Abbildung 1: Darstellung von einem ausgedachten Emmedding Vektor Raum über zwei Konzept Achsen (Selbst genzeichnet)

Um einen Vektor Embedding Raum sinnvoll aufzubauen, ist eine einehitliche Methode zur Embedding Generierung nötig, in der Bedeutungszuweisung von Richtungen kohärent zwischen den Embeddings ist. Wie bereits erwähnt, wurden LLMs für Textverständniss trainiert und können deshalb genutzt werden, um aus einer Text Eingabe ein Ausgabe Vektor Embedding zu erzeugen. [18]

3 Indexing

Das Wort „Index“ hat in unterschiedlichsten Anwendungsbereichen unterschiedliche Bedeutung. Gemein haben diese, dass ein Index eine Form von Daten hält, dessen Sinn eine Schlussfolgerung aus eben diesen Daten ist. [2, 8, 17]

Unter dieser Definition stellt auch ein Vektor Embedding Raum, so wie hier vorgestellt, ein Index dar. In den Vektor Embedding Raum werden Daten als Ausprägung von Information über Text gehalten und es kann aus diesen Daten geschlussfolgert werden, wie bedeutungsnah z. B. zwei Textabschnitte sind.

Der Ablauf des Indexing, der Prozess des erstellen eines Index, mit LLMs läuft im allgemeinen so ab, dass Text vorverarbeitet und in Abschnitte geteilt wird, aus denen für jeden Abschnitt durch ein LLM sein Vektor Embedding erstellt wird, dass dann im idealfall die Bedeutung dieses Abschnitts repräsentiert. Die resultiernden Vektor Embeddings werden dann in einer Datenbank o.Ä. gespeichert, welche damit den Index beziehungweise den Vektor Embedding Raum hält. [6]

3.1 Suche

Um den in der Einleitung erläuterten Sinn dieser Arbeit zu erfüllen, das Durchsuchen von Text beziehungweise hier Code, muss sich mit der Umsetzung der Suche beschäftigt werden. Nachdem eine Frage formuliert ist, wird aus dem Fragetext ein Vektor Embedding erzeugt. Dieses Embedding wird dann mit den anderen Embeddings im Index abgeglichen und die zu den ähnlichsten Embeddings gehörenden Textabschnitte werden als Suchergebnis zurückgegeben. FÜr den Abgleich wird die Kosinus-Ähnlichkeit genutzt, die ein Maß für die Ähnlichkeit von zwei Vektoren bildet [12].

3.2 Indexing von Code

Der erste Schritt zum Indexing ist das Einteilen der Codebasis in Abschnitte. Dies wird mit der Tree-Sitter Libary durchgeführt, die den Code aufteilt. [15] Als nächstes wird aus diesen Code Abschnitten ein Index erzeugt, welcher dann durscut werden kann..

Zum Erfassen der Funktionstüchtigkeit wird eine Metrik eingeführt. Es gibt ein im Anhang enthaltends Test Projekt und folgende zehn Test Fragen. Jede dieser Fragen zielt auf ein Codeabschnitt des Projektes ab und es sind alles „Where“-Fragen.

Tabelle 1: Test Fragen für bespielhafte Suchen im Index

1.	„Where is the logger implemented?“
2.	„Where are the setting changes for the logger handled?“
3.	„Where is the linting configured?“
4.	„Where is typescript configured?“
5.	„Where are the styles for my app?“
6.	„Where is my app created?“
7.	„Where do I create the IDs for my web view context?“
8.	„Where do I manage the state of my web view panels?“
9.	„Where do I configure what commands will be available in my vs code extension?“
10.	„Where do I retrieve information about vs code settings?“

Für einen ersten Test 1 wird für das Projekt nach vorgestelltem Verfahren ein Index erstellt, welcher dann mit den Fragen druchsucht wird. Enthalten die drei besten Ergnisse, also jene dessen Embedding am meisten Kosinus-Ähnlichkeit mit dem Embedding der Frage habe, den Textabschnitt, der durch die Frage gemeint ist, wird das Ergebniss als richtig gewertet. Für richtige Ergebnisse wird die Kosinus-Ähnlichkeit multipliziert mit 100 notiert, sonst eine 0.

Vollständigkeitshalber: Alle Ergebniss wurden mit Hilfe der gpt-35-turbo [1] und text-embedding-ada-002 [11] Modelle erzeugt. Für Completions wurde eine Temprature von 0.2 und ein Top-P-Wert von 0.3 gewählt. Alle anderen Einstellungen blieben unverändert.

Tabelle 2: Ergebnisse der Index und Such Tests

Frage	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7
1.	79,93	0	79,93	79,37	80,08	79,25	80,62
2.	81,72	79,3	81,72	79,56	83,07	80,49	82,29
3.	0	0	0	0	84,36	0	0
4.	80,36	80,39	86,06	86,06	85,56	85,2	84,28
5.	0	0	0	0	81,38	75,76	79,12
6.	78,16	78,96	77,7	77,7	0	78,95	79,12
7.	0	0	80,87	0	80,87	78,44	0
8.	0	0	0	0	85,96	0	75,33
9.	80,04	0	85,4	89,28	85,16	0	0
10.	75,58	75,89	84,34	83,36	84,34	83,37	82,27

3.2.1 Problem der Code Semantik

In Test 1 konnten sechs von zehn Fragen richtig beantwortet werden. Zur weiteren Untersuchungen wurde Test 2 durchgeführt. In diesem Test war der Prozess des Indexing und der Suche gleich, doch wurde das Test Projekt insofern angepasst, dass sämtlich Dokumentation gelöscht wurde und Namen von Variablen nicht deskreptiv, z.B. „logger“ zu „l“, waren. Die vorgestellte Indexing und Such Metohde konnte dieser Änderung nicht stand halten und nur noch vier von zehn Fragen richtig beantworten. Außerdem ist anzumerken, dass die Fragen, die richtig beantwortet wurden, jene sind, bei dem der dazugehöriges Codeabschnitt weiterhin viele descreptive Namen enthält, da dort externe Biblotheken genutzt werden, dessen Namen nicht angepasst werden können.

Aus diesen beiden Punkt lässt sich vermuten, dass bei der Erzeugung der Embeddings vorallem aus der Namensgebung Information gezogen wird und nicht aus der Logik des Code.

3.2.2 Seperate Bedeutungserfassen via Completions

Um dieses Problem der Erfassug der Semantik von Code anzugehen, wird der Indexing Prozess für folgende Tests angepasst.

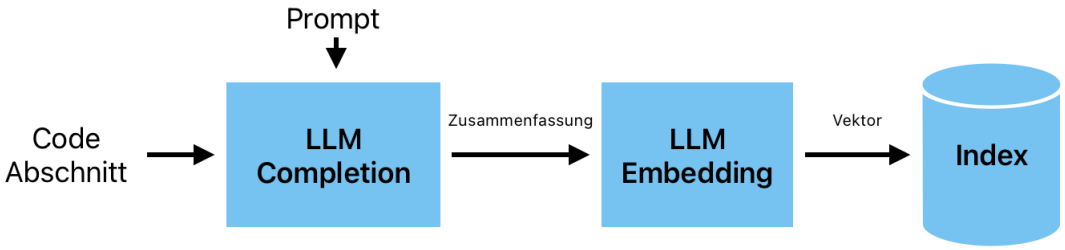

Abbildung 2: Indexing Prozess aus einem Code Abschnitt(Selbst genzeichnet)

Dafür wird, wie in Abbildung 2 ilustriert, vor der Embedding Generierung ein weiters LLM aufgerufen, welches eine Completion durchführt, die als Starttext ein Prompt und den Code Abschnitt erhält. Das Prompt fordert das LLM auf, den erhaltenden Code zusammenzufassem und diese Zusammenfassung wird dann für das Embedding genutzt. Einerseits kann dem ersten LLM durch diese Prompt mehr Kontext gegeben werden und anderseits ist das Ergebniss dieser ersten Analyse durch das LLM ein Text in natürlicher Sprache, der besser von dem Embedding LLM erfasst werden kann.

Als Prompt wird folgender Text genutzt:

```
You are a senior software developer. I provide you with a fenced code snippet
    that is part of a larger codebase.
Do the following things:
1. Summarize what the code does in non technical terms, in 2 sentences.(summary)

2. Briefly describe how the code works. (analysis)
3. Imagine a developer needs to find this code based on its functionality in the
    codebase. List 3-5 "Where" questions a developer might ask to locate this
    code, based on its functionality/intent. (questions)
Take your time to do a thorough review and identify meaningful questions.
Return nothing but JSON,
Here is an example:
{
    "summary": "This code snippet sorts an array of numbers. It supports
        sorting in ascending or descending order.",
    "analysis": "The code implements the quick sort algorithm to recursively
        esort an array in ascending or descending order.",
    "questions": [
        "Where is the code used for sorting data?",
        "Where can I find the array sorting logic?",
        "Where is the code that implements quicksort?"
    ]
}
If the code snippet is empty, just return the text ["NOTHING"].
Do not start with "Here is the JSON object" or similar, just output plain JSON no
     markdown!


<Code Abschnitt wird hier eingefuegt>
```

Im Test 3 wird dieses neue Verfahren genutzt und sieben von zehn Fragen können richtig beantwortet werden. Beim Test 4 wird mit dem selben Verfahren das Test Projekt im Zustand aus Test 2 untersucht und durch die Verbesserungen werden noch sechs von zehn Fragen richtig beantwortet. Dabei ist anzumerken, dass die Frage, die beim Vergleich von Test 3 und Test 4 wegfällt, selbst von einem Menschen nur durch die Dokumentation zu beantworten wäre, da der Codeabschnitt auch für andere Zwecke eingesetzt werden könnte als in der Dokumentation dargelet und durch die Frage gemeint.

3.2.3 Datei spezifische Promtps

Um den Kontext für die LLMs weiter zu erhöhen, wird dem Prompt in Test 5 der relative Dateipfad für den Code Abschnitt mitgegeben, indem das Prompt mit folgendem Satz ergänzt wird:

```
The file path for the snippet is "<Pfad zur Datei des Codeabschnittes>".
```

Dadurch wird auch der Dateiname übergeben, welcher Information darüber hält, was der Codeabschnitt in dieser Datei bedeuten kann. Deutlich wird dies zum Beispiel mit Frage 3 die nach der ESLint Konfiguration fragt. Alle vorherigen Tests mit sind an dieser Frage gescheitert. Die Antwort des Completion LLM war stets das vorgegebene „NOTHING“, welches nur zurückgegeben werden soll, wenn kein Code übergeben wurde. Mit dieser Anpassung war es dem LLM allerdings möglich Konfigurationen richtig einzuordnen und es würden neun von zehn Fragen richtig beantwortet.

Dabei ist auffällig, dass die Frage 6, die fehlgeschlagen ist, bei allen anderen Tests richtig beantwortet wurde. Erklärbar ist dieses verhalten dadurch, dass der Codeabschnitt selber, sehr nah an der Frage dran ist (wie sage ich das richtig???) und deshalb in den meisten Tests erkannt wird. Doch bei Test 5 wird dieser Abschnitt so abstrakt beschrieben, dass die Embedding zu bedeutungsfern für eine erfolgreiche Suche sind.

3.2.4 Embeddings aus erweiterten Erklärungen

Ein weiterer Ansatz die Ergenisse zu verbessern war es neben einer Zusammenfassung auch Beispielfragen zu generieren, die dann zusammen in einen Embedding Vektor gewandlet werden. Dafür wurde mit den Completions stand Test 3 und Test 4 gearbeitet. Die Ergebnisse von diesem Test 6 zeigen allerdings keine Verbesserung im Vergleich zu den vorherigen Tests auf.

In einem abschlienßden Versuch wird eine Analyse des Code übergeben. In diesem Test 7 kann allerdings auch keine Verbesserung festgestellt werden.

4 Bewertung

Bei der Bewertung der Ergebnisse muss hier zuerst die genutzte Metrik bewertet werden. Sowohl das ausgeählte Test Projekt als auch die zehn Test Fragen wurden nur beispielhaft gewählt und haben geringen repräsentativen Aussagewert, da die Ergebnisse für andere Fragen an einem anderem Projekt komplett anders ausfallen könnten. Die Metrik ist dementsprechend nur bedingt aussagekräftig.

Außerdem wurden alle Tests genau einmal durchgeführt. Da die LLMs allerdings nicht deterministisch Ergebnisse erzeugen, könnten auch bei einem zweiten Testlauf andere Ergebnisse entstehen. Zudem wurde vernachlässigt, dass durch z.B. Netzwerkprobleme oder eine Limitierung der Aufrufrate von den LLMs abgefälschte Ergebnisse entstanden sein können. Die Ergebnisse wurden auch nur mit den zwei vorgestellten Modellen ermittelt und es lässt sich nicht verallgemeinernd auf die funktionsweise bei anderen Modellen schließen.

Insgesamt sind die Ergebnisse deshalb skeptisch zu betrachten. Trotzdem kann unter diesen Bedingungen vermutet werden, dass das weitere Analysieren und Zusammenfassen von Code mit einem weitern Completion LLM, welches mehr Kontext verarbeiten kann, vorteilhaft ist. Dabei sind die Nachteile anzumerken, dass der weitere LLM Aufruf viel Zeit verbraucht und auch zusätzliche Kosten verursacht.

5 Fazit

Abschließend lässt sich festhalten, dass empirisch das Potential der erarbeiten Indexing Techniken dargelegt werden konnte. In weiterer Arbeit besteht dabei die absolute Notwendigkeit, die Tests zu weiter zu vertiefen. Das heißt, auf mehre Test Codebasen ausweiten und mehr Test Fragen zu erstellen, welche nicht nur von einer einzeln Person, sondern von einer vielzahl von Entwicklern aufgestellt wurde.

In diesem Rahmen können außerdem noch weitere Techniken für das Indexing erkundet werden. Dazu gehört zum Beispiel, dass für unterschiedliche Dateitypen noch mehr Datei spezifischer Kontext gegeben wird, damit die LLMs den Code noch besser einordnen können.

Literatur

[1] *Azure OpenAI Service-Modelle*. URL: https://learn.microsoft.com/de-de/azure/ai-services/openai/concepts/models#gpt-35 (besucht am 08. 08. 2024).

[2] A. Chatterjee. „Index and Indexing“. In: (2017), S. 125–134. DOI: 10.1016/B978-0-08-102025-8.00009-0.

[3] Will Douglas Heavenarchive. *The inside story of how ChatGPT was built from the people who made it*. MIT Technolog Review. 2023. URL: https://www.technologyreview.com/2023/03/03/1069311/inside-story-oral-history-how-chatgpt-built-openai/ (besucht am 01. 08. 2024).

[4] Florian Heimerl und Michael Gleicher. „Interactive analysis of word vector embeddings“. In: *Computer Graphics Forum*. Bd. 37. 3. Wiley Online Library. 2018, S. 253–265.

[5] Eric Huang u. a. „Improving Word Representations via Global Context and Multiple Word Prototypes“. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Hrsg. von Haizhou Li u. a. Jeju Island, Korea: Association for Computational Linguistics, Juli 2012, S. 873–882. URL: https://aclanthology.org/P12-1092.

[6] Xiang Ji u. a. „Speeding Up Question Answering Task of Language Models via Inverted Index“. In: *arXiv preprint arXiv:2210.13578* (2022).

[7] Yiheng Liu u. a. „Summary of chatgpt-related research and perspective towards the future of large language models“. In: *Meta-Radiology* (2023), S. 100017.

[8] A. Lo. „What Is an Index?“ In: *The Journal of Portfolio Management* 42 (2016), S. 21–36. DOI: 10.3905/jpm.2016.42.2.021.

[9] Tomas Mikolov u. a. „Efficient estimation of word representations in vector space“. In: *arXiv preprint arXiv:1301.3781* (2013).

[10] Humza Naveed u. a. „A comprehensive overview of large language models“. In: *arXiv preprint arXiv:2307.06435* (2023).

[11] *New and improved embedding model*. URL: https://openai.com/index/new-and-improved-embedding-model/ (besucht am 08. 08. 2024).

[12] Faisal Rahutomo, Teruaki Kitasuka, Masayoshi Aritsugi u. a. „Semantic cosine similarity“. In: *The 7th international student conference on advanced science and technology ICAST*. Bd. 4. 1. University of Seoul South Korea. 2012, S. 1.

[13] Herbert Rubenstein und John B Goodenough. „Contextual correlates of synonymy“. In: *Communications of the ACM* 8.10 (1965), S. 627–633.

[14] Haoye Tian u. a. „Is ChatGPT the ultimate programming assistant–how far is it?“ In: *arXiv preprint arXiv:2304.11938* (2023).

[15] *Using Parsers*. URL: https://tree-sitter.github.io/tree-sitter/using-parsers (besucht am 08. 08. 2024).

[16] Ashish Vaswani u. a. „Attention is all you need“. In: *Advances in neural information processing systems* 30 (2017).

[17] B. Vickery. „THE STRUCTURE OF A CONNECTIVE INDEX“. In: *Journal of Documentation* 6 (1950), S. 140–151. DOI: 10.1108/EB026158.

[18] Xin Zhang u. a. „Language Models are Universal Embedders“. In: *arXiv preprint arXiv:2310.08232* (2023).

[19] Yizhen Zheng u. a. „Large language models for scientific synthesis, inference and explanation“. In: *arXiv preprint arXiv:2310.07984* (2023).

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich:

1. dass ich meinen Praxistransferbericht selbstständig verfasst habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe,

3. dass ich meinen Praxistransferbericht bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum	Justin Becker