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

Praxistransferbericht

Vereinfachung von Entwicklungsprozessen durch Large Language Models

vorgelegt am 30. Juni 2025

Name:	Justin Becker
Matrikelnummer:	77204368351
Ausbildungsbetrieb:	SAP SE
Fachbereich:	FB2: Duales Studium — Technik
Studienjahrgang:	2023
Studiengang:	Informatik
Betreuer Unternehmen:	Jonathan Busshoff
Betreuer Hochschule:	Carl Dolling
Wortanzahl:	2199

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 den Mitgliedern des Prüfungsausschusses zugänglich zu machen.

Ort, Datum	Justin Becker

Von der betrieblichen Betreuung zur Kenntnis genommen:

Ort, Datum	Jonathan Busshoff
Ort, Datum	Peter, Jenny

Zusammenfassung

Mit einem Aufschwung von Technologien, die basierend auf Künstlicher Intelligenz (KI) arbeiten, findet diese sich in vielen Anwendungsfällen wieder. So auch in der Software-Entwicklung. Dieser Bericht wird sich damit auseinandersetzen, wie mithilfe von Large Language Models (LLMs) Codebasen durchsucht werden können, um dadurch einen Überblick über Projekte zu bekommen und zu behalten, um letztendlich Entwicklungsprozesse zu vereinfachen. Dafür wird der Ansatz verfolgt, einen Index aus sogenannten Vektor Embeddings zu erstellen, welcher dann mit Fragen natürlicher Sprache dursucht werden kann. Für das Erstellen der Vektor Embeddings werden die LLMs genutzt. Es werden ein Testprojekt und Testfragen herangezogen, um die im folgenden Bericht erarbeiteten Verfahren dafür zu evaluieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Hinweis

In der vorliegenden Arbeit wird auf die Verwendung gendergerechter Sprachformen verzichtet, um eine bessere Lesbarkeit zu gewährleisten. Es wird in der Regel das generische Maskulinum verwendet, wobei alle Geschlechter gleichermaßen adressiert werden

1 Einleitung

Wissenschaftliches Schreiben ist ein essentieller Teil des Schüler- und Studentendaseins. Die Relevanz, in Schulen Wert auf die Entwicklung der Schreibfertigkeiten der Schüler zu legen betonte bereits 2003 die National Commission on Writing in America's Schools & Colleges. Dabei stellt der Prozess des wissenschaftlichen Schreibens variable Anforderungen an einen Schüler. Die Fähigkeit, sich selber zu organisieren und eigenes oder durch Recherche erlangtes Wissen zu strukturieren, sind dabei essentiell. Durch das Schreiben wissenschaftlicher Arbeiten trainieren Schüler und Studenten ihre Fähigkeit, Ideen und Konzepte verständlich auszuformulieren. Dies fördert ebenfalls die Fähigkeit klar zu kommunizieren, wodurch auch bessere Zusammenarbeit im Team gewährleistet wird.

Zudem wird während der Recherche zu einem wissenschaftlichen Thema auch die Medienkompetenz gestärkt. Zu den häufig verwendeten Medien gehören seit kurzem nicht nur bekannte akademische Suchmaschinen wie Google Scholar, sondern auch diverse KI Tools. Spätestens seit der Veröffentlichung des verbesserten ChatGPT von openAI im Jahr 2022, welches in der Lage war, Texte zu generieren, die menschengeschriebenen sehr ähnlich sind, wird die Verwendung künstlicher Intelligenz im wissenschaftlichen Schreiben viel diskutiert und untersucht. [[Paper] From human writing to artificial intelligence generated text: examining the prospects and potential threats of ChatGPT in academic writing]

Dabei zeigen sich sowohl Vor- und Nachteile, welche durch den Einsatz von künstlicher Intelligenz an verschiedensten Punkten des Schreibprozesses entstehen. Dieses Paper befasst sich sowohl mit diesen Vor- und Nachteilen, als auch mit möglichem Lösungsvorschlägen zum Umgang mit den Nachteilen der KI-Nutzung. Es wird ein neues Schreibassistenten-Tool vorgestellt, welches die Risiken von KI minimieren und den Einsatz von künstlicher Intelligenz im Schreibprozess vereinfachen soll.

2 Theoretische Grundlagen

2.1 Large Language Models

LLMs sind KI-Systeme, die darüber charakterisiert sind, dass sie Informationen über natürliche Sprachen erfassen können. Dazu gehören nebst Sprachmustern und Syntax auch die Erfassung von Bedeutung und Zusammenhängen. [16]

Einen Ausdruck dieser Verständnisfähigkeit bringen LLMs durch sogenannte Completions (dt. Vervollständigungen). Dabei wird ein Textabschnitt vorgegeben und das LLM generiert auf dessen Grundlage sukzessiv Folgetext. [8]

Außerdem sind LLMs in der Regel so konfiguriert, dass sie nicht deterministisch arbeiten, d. h., dass selbst bei einer gleichen Eingabe unterschiedliche Ausgaben entstehen können. Diese Konfiguration ist bei der Arbeit mit LLMs in diesem Bericht vorgenommen. [10]

Da die technische und mathematische Funktionsweise dieser Modelle keine Relevanz für diese Arbeit hat, wird sie hier nicht weiter erläutert.

2.2 Vektor Embeddings

Embeddings (dt. Einbettungen) sind hochdimensionale Vektoren, die Syntax, Semantik und Zusammenhänge zwischen semantischen Objekten (im Folgenden als Bedeutung bezeichnet) darstellen können. Diese Darstellung kann für unterschiedliche Datenformen wie Text, Bild oder auch Audio gelten. [7]

Dieser Bericht konzentriert sich auf die Darstellung von Text und vernachlässigt andere Formen in den Beschreibungen, aber die Grundprinzipien sind unabhängig von der Form der Daten.

Für das Erfassen von Bedeutung bildet nebst einem Textteil selbst der Kontext eine entscheidende Rolle [12]. Da das Ziel der Generierung von einem Embedding ist, möglichst viel von dieser Bedeutung zu beschreiben, ist die Qualität der Embeddings

auch bestimmt durch die Einarbeitung des Kontextes in das Embedding. [4]

2.3 Vektor Embedding Raum

Die Vektor Embeddings werden selten isoliert betrachtet, sondern als Teil eines Vektorraums. In diesem Raum wird durch die Embeddings Richtung und Entfernung Bedeutung zugewiesen. So liegt Text, der ähnliche Semantik hat, nahe aneinander in seiner Embedding Darstellung in solch einem Vektorraum. Außerdem stellt das Verhältnis zwischen zwei Vektoren semantische Konzepte dar und es können Achsen gebildet werden, auf die die Embeddings projeziert werden und damit verallgemeinernd Konzepte erfassen, wie in Abbildung 1 zu erkennen. [3, 7]

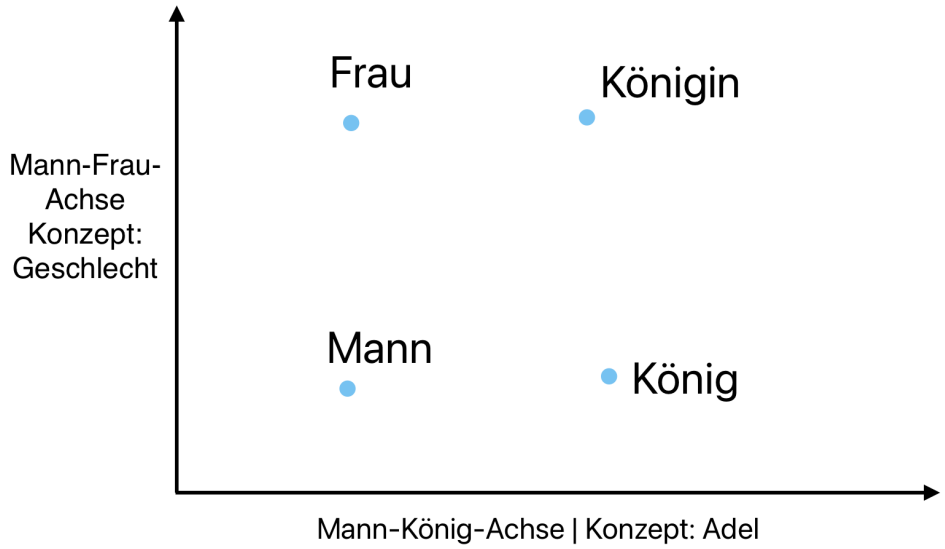

Abbildung 1: Darstellung von einem ausgedachten Vektor Embedding Raum über zwei Konzeptachsen (eigene Darstellung)

Um einen Vektor Embedding Raum sinnvoll aufzubauen, ist eine einheitliche Methode zur Generierung von Embeddings nötig, in der die Bedeutungszuweisung von Richtungen kohärent zwischen den Embeddings ist. Dafür können LLMs durch ihr Textverständnis genutzt werden, um aus einer Texteingabe ein Ausgabe-Vektor Embedding zu erzeugen. [15]

3 Indizieren

Das Wort „Index“ hat in unterschiedlichen Anwendungsbereichen jeweils verschiedene Bedeutungen. Gemein haben diese, dass ein Index eine Form von Daten hält, dessen Sinn eine Schlussfolgerung aus eben diesen Daten ist. [2, 6, 14]

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

Der Ablauf der Indizierung mit LLMs wird im Allgemeinen so funktionieren, dass Text vorverarbeitet und in Abschnitte geteilt wird, aus denen für jeden Abschnitt durch ein LLM sein dazugehöriges Vektor Embedding erstellt wird, welches dann die Bedeutung dieses Abschnitts repräsentiert. Die resultierenden Vektor Embeddings werden dann in einer Datenbank o. Ä. gespeichert, welche damit den Index beziehungweise den Vektor Embedding Raum hält. [5]

3.1 Suche

Zur Suche ist eine Frage in natürlicher Sprache gegeben. Aus dem Fragetext wird dann mit dem selben LLM, das für das Indizieren genutzt wurde, ein Vektor Embedding erzeugt. Dieses Embedding wird mit den anderen Embeddings abgeglichen und die zu den ähnlichsten Embeddings gehörenden Textabschnitte werden als Suchergebnis zurückgegeben. Für den Abgleich wird die Kosinus-Ähnlichkeit[11] zwischen allen Vektoren im Index und dem Frage-Vektor berechnet.

3.2 Indizierung von Code

Der erste Schritt zum Indizieren ist das Einteilen der Codebasis in Abschnitte. Dies wird hier mit der Tree-Sitter Bibliothek durchgeführt, die das Aufteilen von Code übernimmt [13]. Als Nächstes werden aus diesen Codeabschnitten Vektor Embeddings erzeugt und im Index gespeichert.

Zum Erfassen der Funktionstüchtigkeit wird eine Metrik eingeführt. Es gibt ein im Anhang enthaltendes Testprojekt und folgende zehn Testfragen. Jede dieser Fragen, die „Where“-Formulierungen aufweisen, zielt auf einen Codeabschnitt des Projektes ab.

Tabelle 1: Testfragen für die Suchen im Index des Testprojekts

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 durchsucht wird. Enthalten die drei besten Ergebnisse, also jene, dessen Embedding am meisten Kosinus-Ähnlichkeit mit dem Embedding der Frage hat, den Textabschnitt, der durch die Frage gemeint ist, wird das Ergebnis als richtig gewertet. Für richtige Ergebnisse wird die Kosinus-Ähnlichkeit notiert, sonst eine 0.

Alle Ergebnisse wurden mithilfe der gpt-35-turbo [1] und text-embedding-ada-002 [9] Modelle erzeugt. Für Completions wurde eine Temperature von 0.2 und ein Top-P-Wert von 0.3 gewählt. Alle anderen Einstellungen blieben unverändert.

Tabelle 2: Ergebnisse der Index- und Suchtests

Frage	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7
1.	0,7993	0	0,7993	0,7937	0,8008	0,7925	0,8062
2.	0,8172	0,793	0,8172	0,7956	0,8307	0,8049	0,8229
3.	0	0	0	0	0,8436	0	0
4.	0,8036	0,8039	0,8606	0,8606	0,8556	0,852	0,8428
5.	0	0	0	0	0,8138	0,7576	0,7912
6.	0,7816	0,7896	0,777	0,777	0	0,7895	0,7912
7.	0	0	0,8087	0	0,8087	0,7844	0
8.	0	0	0	0	0,8596	0	0,7533
9.	0,8004	0	0,854	0,8928	0,8516	0	0
10.	0,7558	0,7589	0,8434	0,8336	0,8434	0,8337	0,8227

3.2.1 Problem der Code Semantik

In Test 1 konnten sechs von zehn Fragen richtig beantwortet werden. Zur weiteren Untersuchung wurde Test 2 durchgeführt. In diesem Test war der Prozess der Indizierung und der Suche gleich, doch wurde das Testprojekt insofern angepasst, als dass sämtliche Dokumentation gelöscht wurde und Namen von Variablen nicht deskriptiv, z. B. „logger“ zu „l“, waren. Die vorgestellte Indizierungs- und Suchmethode konnte dieser Änderung nicht standhalten und nur noch vier von zehn Fragen richtig beantworten. Außerdem ist anzumerken, dass die Fragen, die richtig beantwortet wurden, jene sind, bei denen der dazugehörige Codeabschnitt weiterhin viele deskriptive Namen enthält, da dort externe Bibliotheken genutzt werden, deren Namen nicht angepasst werden können.

Aus diesen beiden Punkten lässt sich vermuten, dass bei der Erzeugung der Embeddings vor allem aus der Namensgebung Information gezogen wird und nicht aus der Logik des Codes.

3.2.2 Seperate Bedeutungserfassen via Completions

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

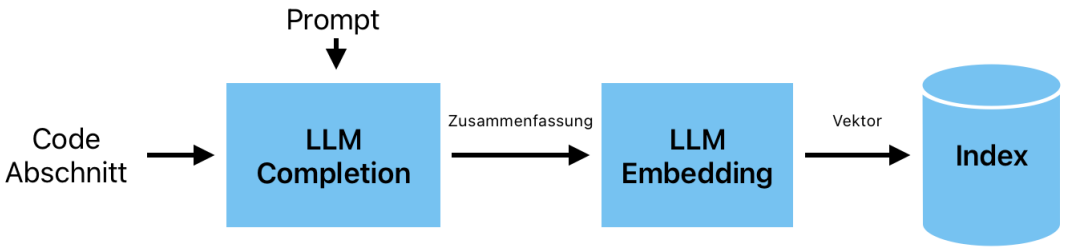

Abbildung 2: Indizierungs Prozess aus einem Code Abschnitt (eigene Darstellung)

Dafür wird, wie in Abbildung 2 illustriert, vor der Embedding Generierung ein weiteres LLM aufgerufen, welches eine Completion durchführt, die als Starttext einen Prompt und den Codeabschnitt erhält. Das Prompt fordert das LLM auf, den erhaltenden Code zusammenzufassen. Diese Zusammenfassung wird wiederum für das Embedding genutzt. Einerseits kann dem ersten LLM durch dieses Prompt mehr Kontext gegeben werden und andererseits ist das Ergebnis 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 demselben Verfahren das Testprojekt 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 dargelegt und durch das Suchembedding erfragt.

3.2.3 Datei spezifische Promtps

Um den Kontext für die LLMs weiter zu erhöhen, wird dem Prompt in Test 5 der relative Dateipfad zu den Codeabschnitt mitgegeben, indem der Prompt durch den folgenden Satz ergänzt wird:

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

Aufgrund dieser Änderung wird nun auch der Dateiname übergeben, der mehr Kontext über den Zweck der Datei bereitstellen kann. Deutlich wird dies am Beispiel von Frage 3, die nach der ESLint Konfiguration fragt. In den vorheringen Tests konnte diese Frage nicht beantwortet werden. Die Antwort des Completion LLM war 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 wurden neun von zehn Fragen richtig beantwortet.

Dabei ist auffällig, dass Frage 6 nicht beantwortet werden konnte, obwohl dies bei allen vorherigen Tests möglich war. Erklärbar ist dieses Verhalten dadurch, dass der Codeabschnitt selber im Wortlaut ähnlich zu der Frage ist und deshalb in den meisten Tests erkannt wird. Doch bei Test 5 wird dieser Abschnitt so abstrakt beschrieben, dass die Embeddings zu fern für eine erfolgreiche Suche sind.

3.2.4 Embeddings aus erweiterten Erklärungen

Ein weiterer Ansatz die Ergebnisse zu verbessern, war es neben einer Zusammenfassung auch Beispielfragen zu generieren, die dann zusammen in einen Embedding Vektor gewandelt 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 dem Test 3. Gründe dafür könnten sein, dass die erzeugten Fragen teilweise nur einzelnen Detailaspekte des Codes erfassen (1.) oder sehr allgemein formuliert sind (2.). Beispiele für solche Fragen sind:

```
1. "Where is the code that defines the configuration properties for logging
    level and source location tracking?"
2. "What is the purpose of the JSON object?"
```

In einem abschließenden Versuch wird nur das übergeben, was im Prompt Analyse bezeichnet ist und meist etwas ausführlicher als die Zusammenfassung war. In diesem Test 7 kann allerdings auch keine Verbesserung festgestellt werden.

4 Bewertung

Für die Bewertung der Ergebnisse muss zunächst die genutzte Metrik bewertet werden. Sowohl das ausgewählte Testprojekt, als auch die zehn Testfragen wurden nur beispielhaft gewählt und haben einen geringen repräsentativen Aussagewert. Die Ergebnisse für andere Fragen an einem anderen Projekt könnten anders ausfallen. Z. B. hätte Test 7 bei detailreicheren Fragen eventuell besser abschneiden können. Die Metrik ist dementsprechend nur bedingt aussagekräftig.

Außerdem wurden alle Tests nur genau einmal durchgeführt. Da die LLMs allerdings nicht deterministisch Ergebnisse erzeugen, könnten auch bei einem zweiten Testlauf andere Ergebnisse entstehen. Zudem wurde vernachlässigt, dass beispielsweise durch Netzwerkprobleme oder eine Limitierung der Aufrufrate von den LLMs abgefälschte Ergebnisse entstehen können. Die Ergebnisse wurden außerdem 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 kritisch zu betrachten. Trotzdem kann die Hypothese aufgestellt werden, dass das weitere Analysieren und Zusammenfassen von Code mit einem weiteren Completion-LLM, welches mehr Kontext verarbeiten kann, vorteilhaft ist und das Durchsuchen von Programmcode verbessert. Dabei sind die Nachteile anzumerken, dass der weitere LLM-Aufruf viel Zeit in Anspruch nimmt und auch zusätzliche Kosten verursacht.

5 Fazit

Abschließend lässt sich festhalten, dass empirisch das Potenzial der erarbeiteten Indizierungs-Techniken dargelegt werden konnte. Bei Folgearbeiten besteht die absolute Notwendigkeit, die Tests weiter zu vertiefen, d. h. auf mehrere Testcodebasen auszuweiten und mehr Testfragen zu erstellen, welche nicht nur von einer einzelnen Person, sondern von einer Vielzahl von Entwicklern aufgestellt werden sollten.

Außerdem können noch weitere Techniken für das Indizieren 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 oder die Verwendung von anderen Suchemethoden.

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] 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.

[4] 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.

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

[6] 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.

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

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

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

[10] Shuyin Ouyang u. a. „LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation“. In: *arXiv preprint arXiv:2308.02828* (2023).

[11] 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.

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

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

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

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

[16] 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