technische universität dortmund

IT in Produktion und Logistik

Entwicklung eines Empfehlungssystems zur Erstellung der Rückverfolgbarkeit in technischen Spezifikationen

Masterarbeit zur Erlangung des Grades M. Sc

Vorgelegt von:	Vladimir Kupershtein
Matrikelnummer:	237836
Studiengang:	Master Maschinenbau

Ausgegeben am:	20.03.2024
Abgegeben am:	27.08.2024

Erstprüfer:	Dr.-Ing- Dipl.-Inform. Anne Antonia Scheidler
Zweitprüfer:	Katharina Langenbach, M.Sc.

Technische Universität Dortmund

Fakultät Maschinenbau

Fachgebiet IT in Produktion und Logistik

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
BERT	Bidirectional encoder Representations from Transformer
BPE	Byte-Pair Encoding
CLS	Klassifizierungstoken
CSV	Komma getrennte Werte (*engl. Comma separated values*)
FTLR	Feingranulare Wiederherstellung von Trace-Links (*engl. Fine-grained traceability links recovery*)
GPT	Generative Pretrained Transformer
HW	Hardware
ID	Identifikationsnummer
IDF	inverse Dokumenthäufigkeit (*engl. Inverse Document Frequency*)
IR	Informationsrückgewinnung (*engl. Information Retrieval*)
ISO	Internationale Organisation für Normung
JSON	Javascript Object Notation
KS	Stichwort- oder Volltextsuche (*engl. Keyword Search or Full Text Search*)
LSTM	Long Short-Term Memory
MAP	Mittlere Durchschnittliche Genauigkeit (*engl. Mean Average Precision*)
MLM	Maskiertes Sprachmodellieren (*engl. Masked Language Modelling*)
MRR	Mean Reprocical Rank
MSE	Mittlere quadratische Abweichung (*engl. Mean Squared Error*)
NDCG	Normalized Discounted cumulative gain
NER	Eigennamenerkennung (*engl. Named Entity Recognition*)
NLI	Textuelles Schließen (*engl. Natural Language Inference*)
NLP	Verarbeitung natürlicher Spracher (*engl. Natural Language Processing*)
POS	Wortart (*engl. Part of speech*)
REST	Representational State Transfer
RNN	Rekurrentes neuronales Netz
RR	Reranking
SEP	Trennungstoken (*engl. Separation Token*)

SNLI	Stanford Natural Language Inference
STS	Semantische textuelle Ähnlichkeit (*engl. Semantic Textual Similarity*)
SW	Software
TF	Vorkommenshäufigkeit (*engl. Term Frequency*)
WMD	Word Mover‘s Distance
XLM	Interlinguales Sprachmodell (*engl. Cross-lingual language model*)

1 Einleitung

Im Anforderungsmanagement bei der Entwicklung eines komplexen Produkts spielt die Automatisierung eine Schlüsselrolle (Umar & Lano, 2024). Anforderungen, die in natürlicher Sprache verfasst sind, bilden zahlreiche Textdokumente und erzeugen dadurch große Informationsmengen. Die Analyse und Verwaltung von Zehntausenden von Anforderungen ohne den Einsatz von Automatisierungswerkzeugen sind fehleranfällig und zeitaufwändig. Gleichzeitig zwingt der Wettbewerb auf dem modernen Markt dazu, den Entwicklungszyklus zu verkürzen und den Bedürfnissen des Kunden gerecht zu werden. (Dick, Hull, & Jackson, 2017).

Bereits in den 1980er Jahren wurde das Potenzial von Verarbeitung natürlicher Sprache (*engl. Natural Language Processing*, NLP) zur Durchführung von Requirements-Engineering-Aufgaben wie Anforderungserhebung, Klassifizierung, Priorisierung, Nachverfolgung und Verknüpfung festgestellt (Zhao, et al., 2021). Die bis vor kurzem durchgeführten Forschungsarbeiten gingen jedoch nicht über den akademischen Bereich hinaus und führten nicht zu einer breiten Anwendung in der Industrie. Das Hauptproblem ist die mangelnde Effizienz der Methoden, weshalb die Unternehmen nicht in der Lage waren, die oben genannten Aufgaben weitgehend zu automatisieren (Hey, 2023).

Die wachsende Leistungsfähigkeit von NLP, die durch die Einführung von Deep-Learning-Technologien und großen Sprachmodellen sowie die zunehmende Verfügbarkeit von Rechenressourcen vorangetrieben wird, verspricht, die Situation in der Branche zu ändern und die praktische Anwendung von NLP zur Automatisierung des Anforderungsmanagementprozesses zu ermöglichen.

Bei sicherheitskritischen Systemen ist die Rückverfolgbarkeit ein zentraler Aspekt des Anforderungsmanagements (Pachiana, Grunwald, Markwirth, & Sohrmann, 2021). Die Rückverfolgbarkeit von Anforderungen beschreibt den Zustand, in dem einerseits jede Domänenanforderung (z.B. Software-, Hardware- oder mechanische Anforderung) einen eindeutigen Ursprung in den Kunden- und/oder Systemspezifikationen, Normen oder anderen grundlegenden Dokumenten hat. Andererseits muss jede Kunden- und Systemanforderung durch eine oder mehrere Domänenanforderungen abgedeckt sein (*ISO/IEC/IEEE 29148-2018*, 2018).

Werden die Kundenanforderungen in den Spezifikationen der unteren Ebene nicht vollständig abgedeckt, kann dies zu einer unvollständigen Umsetzung und infolgedessen zu einer Nichtabnahme des Produkts führen. Im schlimmsten Fall kann ein Produkt mit Mängeln in der Umsetzung der Anforderungen eine Bedrohung für das Leben und die Gesundheit seiner Nutzer

darstellen. Die Notwendigkeit der Rückverfolgbarkeit von Anforderungen ist in Normen für Produkte, die die Sicherheit von Menschen betreffen, wie z. B. Automobilkomponenten, ausdrücklich vorgeschrieben (*ISO 26262-8,* 2011).

Die Aufgabe der vorliegenden Masterarbeit besteht darin, die Erstellung der Rückverfolgbarkeit von Anforderungen, die in den Spezifikationen für ein in der Entwicklung befindliches Produkt enthalten sind, zu automatisieren. Zur Ermittlung der zugehörigen Anforderungen ist geplant, ein Empfehlungssystem zu entwickeln, das auf Daten aus bestehenden oder bereits durchgeführten Projekten zur Entwicklung mechatronischer und leistungselektronischer Systeme für Kraftfahrzeuge, wie z. B. Laderegler, Lenksäulenmodule und andere, beruht. Das Empfehlungssystem soll die neuesten Technologien zur Verarbeitung natürlicher Sprache nutzen, die auf neuronalen Netzen mit Transformer-Architektur basieren, und den Produktentwicklern bei der Identifizierung verwandter Anforderungen als Assistent dienen. Das Hauptziel der Masterarbeit ist die Verbesserung des Anforderungsmanagementprozesses, der die Rückverfolgbarkeit von Anforderungen in Spezifikationen sicherstellt und gleichzeitig den Arbeitsaufwand für Entwickler reduziert.

Die spezifischen Ziele der Masterarbeit lauten wie folgt:

1) Aufbau eines neuronalen Netzes auf der Grundlage der Transformer-Architektur, das in der Lage ist, verwandte Artefakte in der Spezifikationsdatenbank zu erkennen.

2) Anwendung des erstellten Modells als Kern eines Empfehlungssystems und dessen Evaluierung anhand von Daten aus einer breiten Palette von Projekten.

Bei der Entwicklung des Empfehlungssystems werden fortgeschrittene Deep-Learning-Techniken eingesetzt. Diese Methoden sind in der Lage, komplexe Muster in Textdaten zu verstehen und zu interpretieren (Zhang, Yao, Sun, & Tay, 2018), wodurch sie sich am besten für die Ermittlung von Beziehungen zwischen Anforderungen in technischen Spezifikationen eignen. Da das System Empfehlungen mit minimaler Verzögerung erstellen muss, werden gleichzeitig semantische Suchtechniken eingesetzt, um relevante Informationen aus einer großen Menge technischer Daten effizient zu extrahieren.

Der Entwicklungsprozess umfasst sowohl eine separate Evaluierung der trainierten Sprachmodelle als auch das Testen der Funktionsweise des Empfehlungssystems als Ganzes. Die quantitative Bewertung erfolgt je nach Aufgabe des Modells anhand verschiedener Metriken, darunter Genauigkeit, Sensitivität, F1-Maß und fortgeschrittene Metriken, die für Information Retrieval-Systeme geeignet sind, zu denen das Empfehlungssystem gehört.

Diese Masterarbeit, abgesehen von der Einleitung, ist in vier bedeutende Teile gegliedert. Im Kapitel 2 wird der Stand der Technik in zwei Hauptbereichen vorgestellt, an deren Schnittstelle

diese Forschung durchgeführt wird - Anforderungsmanagement und Verarbeitung natürlicher Sprache.

Als Erstes werden die Begriffe und die grundlegenden Prozesse im Anforderungsmanagement erläutert. Der Fokus in diesem Teil liegt auf der Rückverfolgbarkeit. Es wird dargestellt, wie die Dekomposition von Anforderungen auf verschiedenen Betrachtungsebenen erfolgt - von Kundenspezifikationen bis hin zu untergeordneten Spezifikationen für Software und Hardware. Zusätzlich wird auf die Besonderheiten der technischen Sprache hingewiesen, die die Suche nach verwandten Anforderungen erschweren.

Weiterhin wird der Bereich der Verarbeitung natürlicher Sprache untersucht, mit Schwerpunkt auf Techniken des Tiefen Lernens (*engl. Deep Learning*). In diesem Kapitel wird erläutert, welche Arten von Sprachmodellen existieren und für welche typischen Aufgaben sie verwendet werden. Es wird die Frage der Vektorrepräsentation von Sätzen und einzelnen Tokens, die von Sprachmodellen ausgegeben werden, behandelt. Es werden auch Methoden zur Feinabstimmung von neuronalen Netzen für spezifische Anwendungsfälle betrachtet. Weiterhin wird in diesem Kapitel diskutiert, wie die vorgestellten Techniken zur Erstellung der Rückverfolgbarkeit im Anforderungsmanagement angewendet werden können.

Kapitel 3 beschreibt die Methodik und die konkreten Schritte, die bei der Entwicklung des Empfehlungssystems unternommen werden. Dies beinhaltet die Datenerfassung und Vorverarbeitung, das Training von Deep-Learning-Modellen sowie die Auswahl von geeigneten Empfehlungsalgorithmen auf Basis der Trainingsergebnisse.

In Kapitel 4 wird die Leistungsfähigkeit des Empfehlungssystems bewertet. Die Evaluierung basiert auf technischen Spezifikationen realer Projekte, deren Daten nicht in den Trainingsprozess einbezogen sind. Bewertet wird die Fähigkeit, die Rückverfolgbarkeit zwischen verschiedenen Anforderungsebenen herzustellen und auch Verweise auf Normen, Standards und andere externe Dokumente in den Texten zu finden. Bei der Evaluierung wird auch untersucht, wie sich die Kombination verschiedener Hyperparameter auf die Funktionsweise des Systems auswirkt.

In der Schlussfolgerung werden die Ergebnisse zusammengefasst und eine Aussage über den Grad der Effektivität der erarbeiteten Lösung gemacht und die Möglichkeiten für die weitere Entwicklung des Empfehlungssystems betrachtet.

2 Grundlagen von Anforderungsmanagement und natürlicher Sprachverarbeitung

In diesem Kapitel wird eine Beschreibung der Grundlagen gegeben, auf denen die weitere Entwicklung des Empfehlungssystems aufbaut. Im Unterkapitel 2.1 werden Konzepte und Prinzipien erläutert, die dem Prozess des Anforderungsmanagements zugrunde liegen. Anschließend werden im Unterkapitel 2.2 die wichtigsten Methoden zur Verarbeitung natürlicher Sprache und für Information Retrieval vorgestellt, die bei der Entwicklung des Empfehlungssystems eine Rolle spielen. Im Abschnitt 2.3 wird ein Überblick über verwandte wissenschaftliche Arbeiten gegeben, die mit der automatischen Auffindung von Trace-Links (oder Rückverfolgbarkeitsverbindungen) zusammenhängen.

2.1 Grundlagen des Anforderungsmanagements

2.1.1 Grundlegende Begriffe

Für die Zwecke dieser Arbeit ist es vor allem wichtig, drei Begriffe zu definieren: Anforderung, Anforderungsmanagement und Requirements Engineering. Im weitesten Sinne wird unter einer Anforderung gewöhnlich jede Aussage über einen Wunsch oder Bedarf verstanden. Einige Quellen gehen weiter und geben eine ausführlichere Definition: „Eine Anforderung ist eine detaillierte Darstellung spezifischer Aspekte weniger detaillierter Anfragen von beteiligten Parteien (Stakeholdern), die durchgearbeitete grundlegende Ziele dieser beinhalten“ (Kossmann, 2016, S. XXII). Hier wird betont, dass die Anforderungen die Beziehung zwischen dem Kunden und dem Auftragnehmer formalisieren und als gemeinsames Verständnis zur Aufteilung der Verantwortlichkeiten zwischen ihnen dienen. Für diese Arbeit ist es wichtig zu wissen, dass Anforderungen in der Regel in natürlicher Sprache verfasst werden, da sie allen beteiligten Parteien verständlich sein müssen. Es ist jedoch nicht ausgeschlossen, dass auch Modelle, Formeln und Bilder verwendet werden, um den Inhalt der Anforderungen eindeutig zu gestalten.

Die Norm IEEE 1220-2005 definiert den Begriff "Anforderung" strenger: „Eine Aussage, die eine operative, funktionale oder konstruktive Eigenschaft oder Einschränkung des Produkts oder des Prozesses definiert, die eindeutig, überprüfbar oder messbar ist und für die Freigabe des Produkts oder Prozesses (von Kunden oder internen Qualitätsmanagement-Stellen) notwendig ist“ (*IEEE STD 1220-2005*, 2005, S. 9). Hierbei ist einerseits wichtig, dass nur überprüfbare Anforderungen als solche gelten, andererseits zeigt diese Definition jedoch auch,

dass es verschiedene Arten von Anforderungen gibt: funktionale und nicht-funktionale. Funktionale Anforderungen beantworten die Frage, was das System können muss, nicht-funktionale Anforderungen definieren die Qualität der Ausführung ihrer Aufgaben. Außerdem sind Einschränkungen, die etwa durch Gesetze oder branchenspezifische Standards auferlegt werden, auch Anforderungen und Bestandteil eines Projekts.

Die Tätigkeit im Zusammenhang mit der Verarbeitung von Anforderungen wird oft als Anforderungsmanagement oder Requirements Engineering bezeichnet. Obwohl diese Begriffe manchmal synonym verwendet werden, ist es wichtig, ihre Unterschiede hervorzuheben. Laut der Norm IEEE 29148:2018-11 ist das „Requirements Engineering ein Teilbereich des Systems Engineerings, der die Erkennung, Identifizierung, Entwicklung, Analyse, Verifikation (einschließlich Verifikationsmethoden und -strategien), Bestätigung, Übertragung, Dokumentation und das Management von Anforderungen umfasst“ (*ISO/IEC/IEEE 29148-2018*, 2018, S. 9). Das Anforderungsmanagement ist hingegen Teil des Requirements Engineering und „beinhaltet Aktivitäten zur Erfassung und Aufrechterhaltung sich ändernder Anforderungen sowie der mit ihnen verbundenen Kontext- und Verlaufsdaten, die im Prozess der Anforderungsentwicklung gesammelt wurden“ (*ISO/IEC/IEEE 29148-2018*, 2018, S. 45). Das Anforderungsmanagement konzentriert sich also auf die Verfolgung von Änderungen in den Anforderungen, die Verwaltung von Informationen dazu und deren Umsetzungsgrad. Normalerweise wird das Anforderungsmanagement mit Hilfe von spezialisierten Software-Tools durchgeführt. Tools für das Anforderungsmanagement unterstützen Ingenieure bei der Entwicklung und Analyse von Anforderungen und kontrollieren den Fortschritt des Projekts, um sicherzustellen, dass es den geltenden Rahmenbedingungen entspricht.

Wie später gezeigt wird, ist es von entscheidender Bedeutung, Trace-Links im Anforderungsmanagement zu identifizieren oder wiederherzustellen. Ohne diese Links ist es nicht möglich, die Abdeckung von Anforderungen zu analysieren (d. h. den Grad, in dem Anforderungen implementiert oder verifiziert wurden, *engl. Coverage Analysis*), oder die Auswirkungen von Änderungen an spezifischen Anforderungen zu untersuchen (*engl. Impact Analysis*) (Kossmann, 2016).

2.1.2 Requirements-Engineering-Prozess

Der Prozess der Anforderungsentwicklung beginnt mit der Identifizierung von Bedürfnissen der Stakeholder. Zu den Stakeholdern gehören in der Regel Kunden des Projekts und dessen Benutzer sowie Regulierungsbehörden. Anschließend werden die Bedürfnisse der Stakeholder in konkrete Anforderungen umgewandelt, die später verifiziert werden können. Die Anforderungen des Kunden spiegeln die Perspektive des Benutzers zu den gewünschten Systemmerkmalen wider. Um sie jedoch als Ausgangspunkt für die Entwicklung nutzen zu können,

müssen sie in Systemanforderungen umgewandelt werden, d. h. in eine Menge von technischen Merkmalen, die das System ausführen sollte, aus der Sicht des Auftragnehmers (*ISO/IEC/IEEE 29148-2018*, 2018). Die Systemanforderungen sind der Ausgangspunkt für die Systemarchitektur, durch die das System in Teilsysteme und Komponenten aufgeteilt wird. Der Entwicklungsprozess des Systems wird rekursiv für jede nachfolgend Ebene wiederholt. Abbildung 2.1 zeigt das oben beschriebene allgemeine Prozessdiagramm.

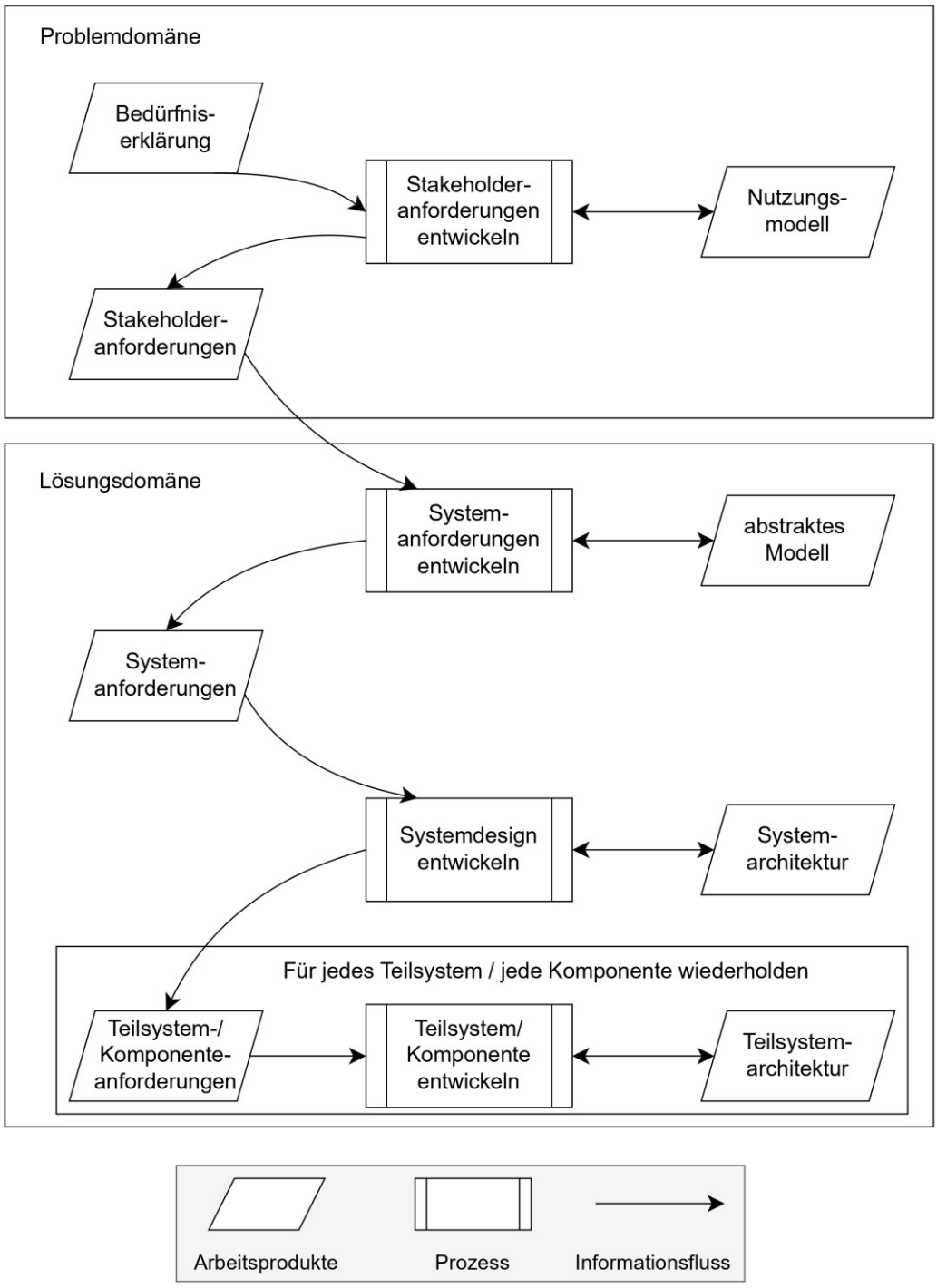

Abbildung 2-1 Prozessdiagram für die Entwicklung des Systems. Veränderte Darstellung nach: Dick, Hull, & Jackson, 2017, S. 35

Die Anforderungen, die auf einer Ebene oder innerhalb eines Teilsystems/einer Komponente erstellt werden, dienen als Eingangsdaten für die darunterliegende Ebene oder das angrenzende Teilsystem/Komponente. Die auf dieser Ebene entwickelten Anforderungen sind wiederum abgeleitet von den Eingangsdaten. Außerdem wird für jede Anforderung eine Qualifizierungsstrategie festgelegt, z.B. Testen oder Review. Abbildung 2.2 zeigt den Prozess des Requirements Engineerings im Kontext einer Einzelebene oder eines Teilsystems/einer Komponente ohne Berücksichtigung von kontinuierlicher Projektentwicklung und Änderungen in der Dokumentation (Dick, Hull, & Jackson, 2017). Der Hauptpunkt, der aus dem obigen Text gezogen werden kann, ist, dass es Verbindungen zwischen Anforderungen und anderen Artefakten wie Testfällen oder Quellcode gibt, die diese Anforderungen implementieren. Bereits in einer frühen Phase des Projekts besteht die Notwendigkeit, diese Beziehungen zu verfolgen.

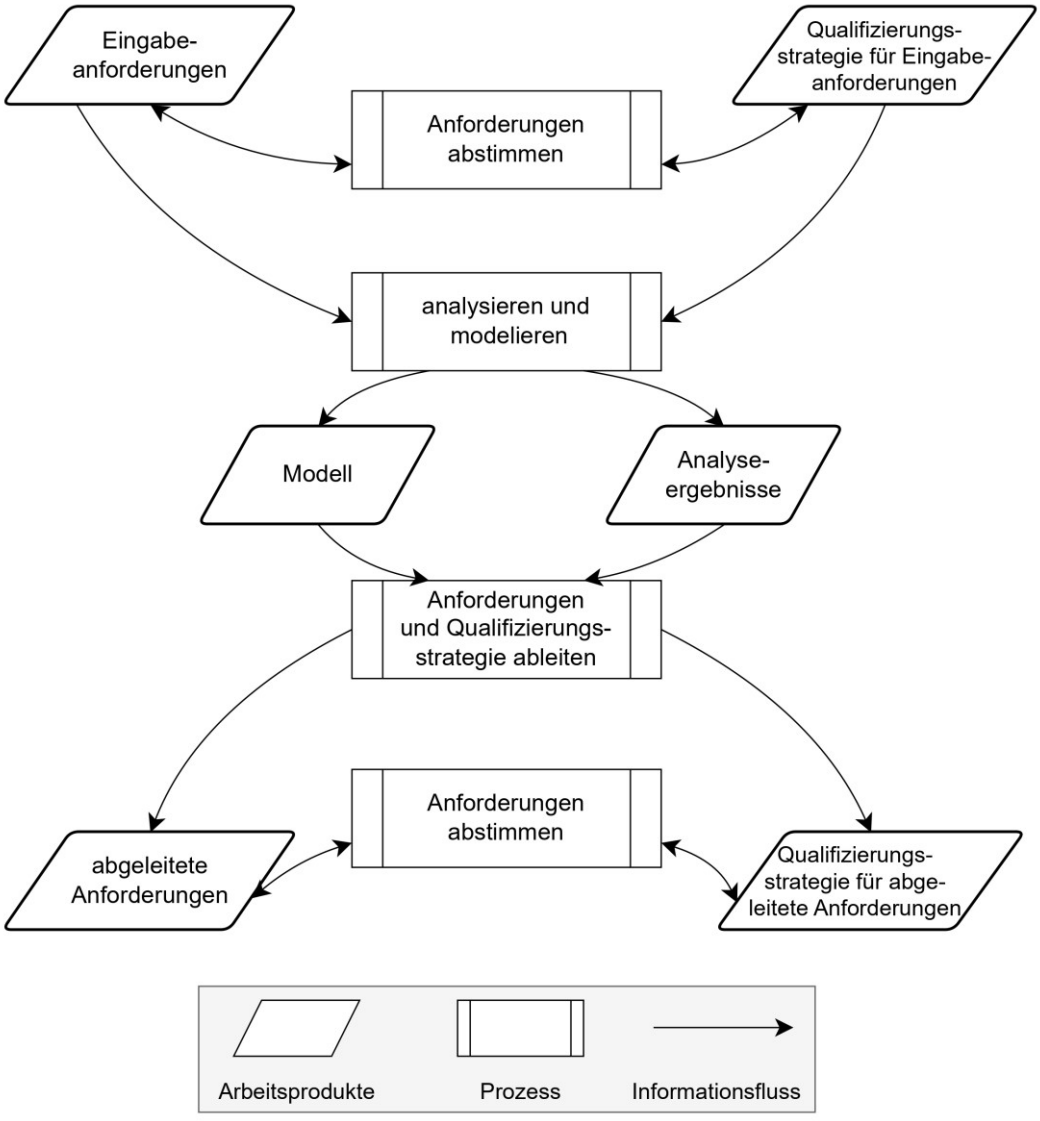

Abbildung 2-2 Requirements-Engineering-Prozess. Veränderte Darstellung nach: Dick, Hull, & Jackson, 2017, S. 40

2.1.3 Rückverfolgbarkeit

Rückverfolgbarkeit (*engl. Traceability*) bezieht sich auf die „Fähigkeit, die Verbindungen zwischen Informationsobjekten wie Anforderungsquellen (Stakeholder-Bedürfnisse), Anforderungen, Quellcode, der die Anforderungen umsetzt, Verifizierungs- und Validierungsdaten zu verfolgen“ (Kossmann, 2016, S. XXV). Die Bedeutung der Rückverfolgbarkeit im Verlauf eines Projekts wurde von verschiedenen Autoren (Dick, Hull, & Jackson, 2017; Kossmann, 2016) hervorgehoben und durch:

- die Erhöhung der Zuversichtlichkeit hinsichtlich der Zielerreichung,
- die Möglichkeit zur Bewertung der Auswirkungen von Änderungen,
- die Verbesserung der Rechenschaftspflicht der Anbieter sowie
- die Möglichkeit, Fortschritt zu verfolgen und genau zu messen,

begründet. Die Etablierung und Formalisierung von Beziehungen hilft bei dem Verständnis, wie Ziele erreicht werden, und verschiedene Analyseformen von Einflüssen werden durch die Verfolgung möglich. Abbildung 2.3 zeigt die Rolle der Rückverfolgbarkeit bei der Änderungsverwaltung innerhalb des V-Modells der Entwicklungsprozesse, wo jedem Systemniveau eine bestimmte Art des Tests entspricht.

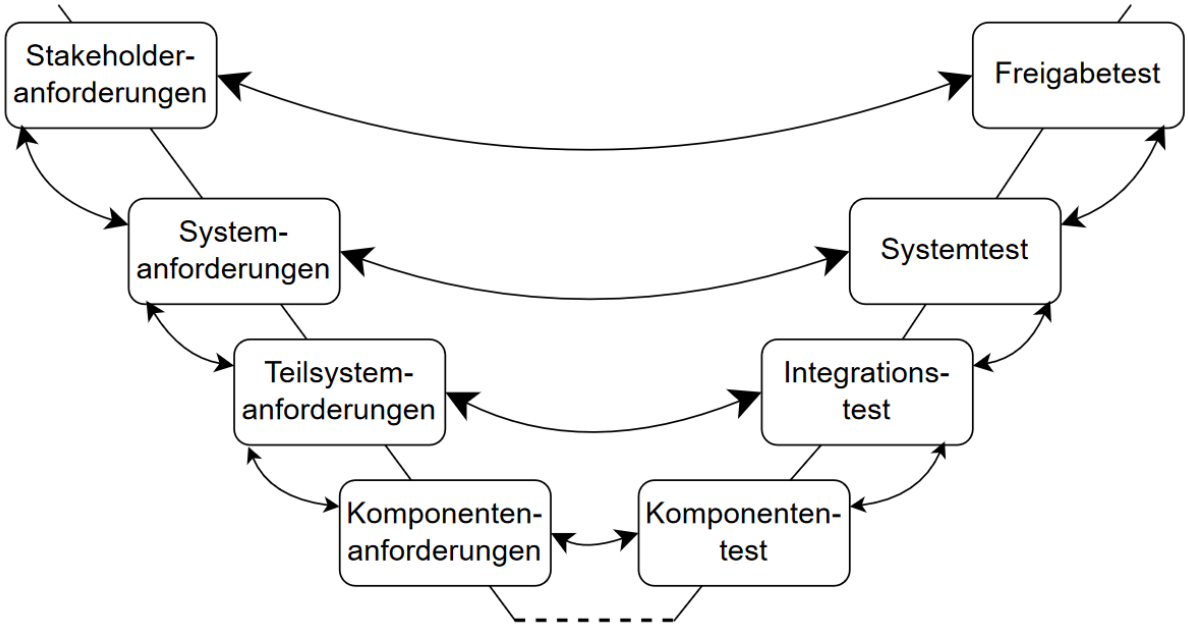

Abbildung 2-3 Die Rolle der Rückverfolgbarkeit bei der Änderungsverwaltung. Veränderte Darstellung nach: Dick, Hull, & Jackson, 2017, S. 14

In der Automobilindustrie, innerhalb derer diese Arbeit durchgeführt wird, gelten spezifische Standards – ASPICE (VDA Working Group, 2023) und (*ISO 26262-8,* 2011) - für alle Organisationen, die in der Entwicklung von Fahrzeugkomponenten beteiligt sind, welche auch eine

bidirektionale (d. h. vorwärts und rückwärts) Verfolgbarkeit für jeden Projektabschnitt vorschreiben. Vorwärts-Verfolgbarkeit (oder manchmal Nachverfolgbarkeit) der Anforderungen bedeutet die Verfolgbarkeit von Anforderungsquellen bis hin zu abgeleiteten Anforderungen. Rückwärtige Verfolgbarkeit der Anforderungen bedeutet die Verfolgbarkeit von Anforderungen zurück zu ihren Quellen.

Die Natur der Anforderungen bedeutet, dass die Beziehungen zwischen ihnen viele-zu-viele sind (Dick, Hull, & Jackson, 2017). Obwohl der Standardfall die Verfeinerung und Dekomposition von Anforderungen auf niedrigerer Ebene ist, wodurch beispielsweise eine Kundenanforderung auf mehrere Systemanforderungen verweisen kann, kann es auch Situationen geben, in denen eine niedrigere Anforderung mehr als einen Elternteil hat. Abbildung 2.4 zeigt, wie die Rückverfolgbarkeit zwischen verschiedenen Ebenen von Spezifikationen und Testartefakten in einem beispielhaften Fall aussehen kann.

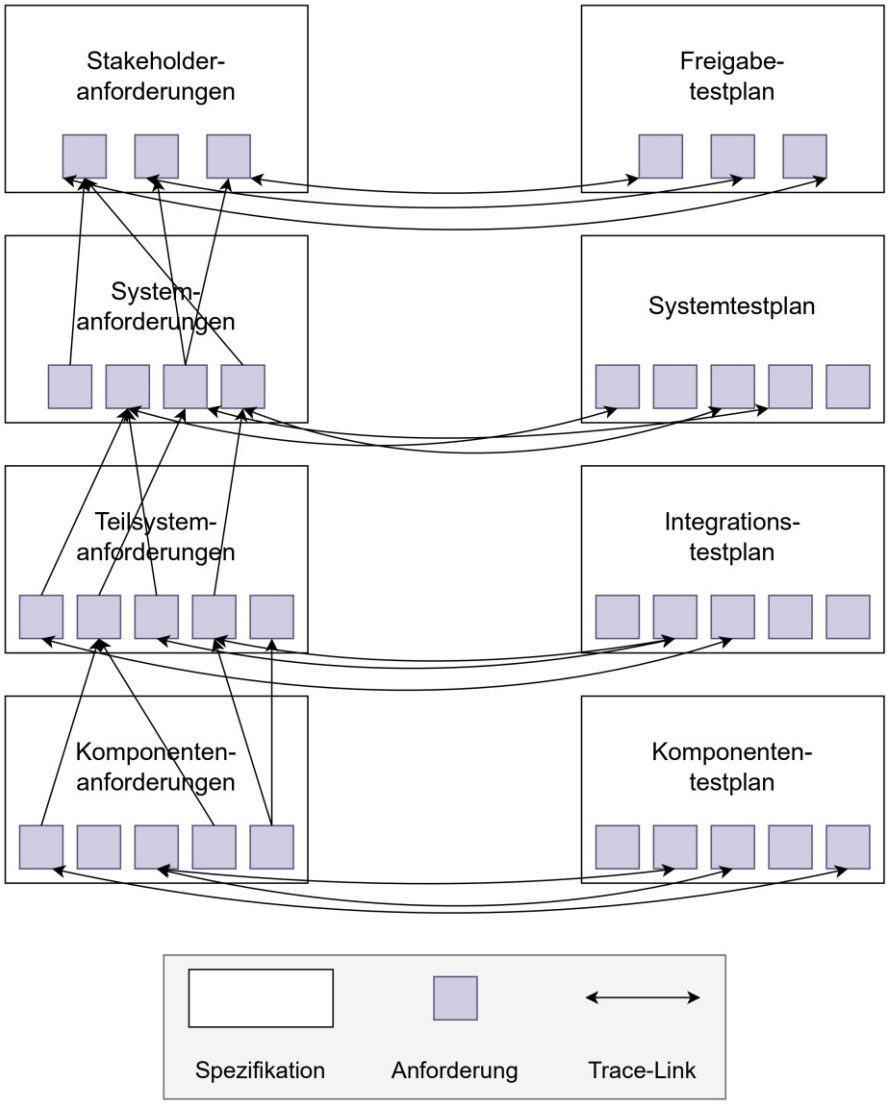

Abbildung 2-4 Beispielhafte Rückverfolgbarkeit der Anforderungen. Veränderte Darstellung nach: Dick, Hull, & Jackson, 2017, S. 16

Um die Rückverfolgbarkeit sicherzustellen, müssen Informationen zu den Anforderungen in einer spezialisierten Datenbank gespeichert werden, die das Erstellen von Links zwischen Artefakten ermöglicht. Zusammen mit dem Text der Anforderung und seiner eindeutigen ID werden Metainformationen zu ihm in der Datenbank in Form von Werten verschiedener Attribute gespeichert. Die Zusammensetzung von Attributen wird durch das Business Object Model für verschiedene Arten von Artefakten definiert (Kossmann, 2016). Abbildung 2.5 zeigt ein vereinfachtes Datenmodell, das für die Anforderungen in der Automobilindustrie typisch ist.

Nur Anforderungen, die bestimmte Regeln erfüllen, d. h. über bestimmte Attributwerte verfügen, können miteinander verknüpft werden. Zum Beispiel kann nur eine Kundenanforderung mit einem "Responsible"-Attributwert von "HW" mit Hardware-Anforderungen verbunden werden. Basierend auf diesen Regeln werden regelmäßige (tägliche oder wöchentliche) Überprüfungen des Anforderungszustands durchgeführt, die als Metriken bezeichnet werden. Die Hauptmetrik für die Rückverfolgbarkeit ist die Rückverfolgbarkeitsmatrix. Ein Beispiel für eine solche Matrix ist in Abbildung 2.6 gezeigt.

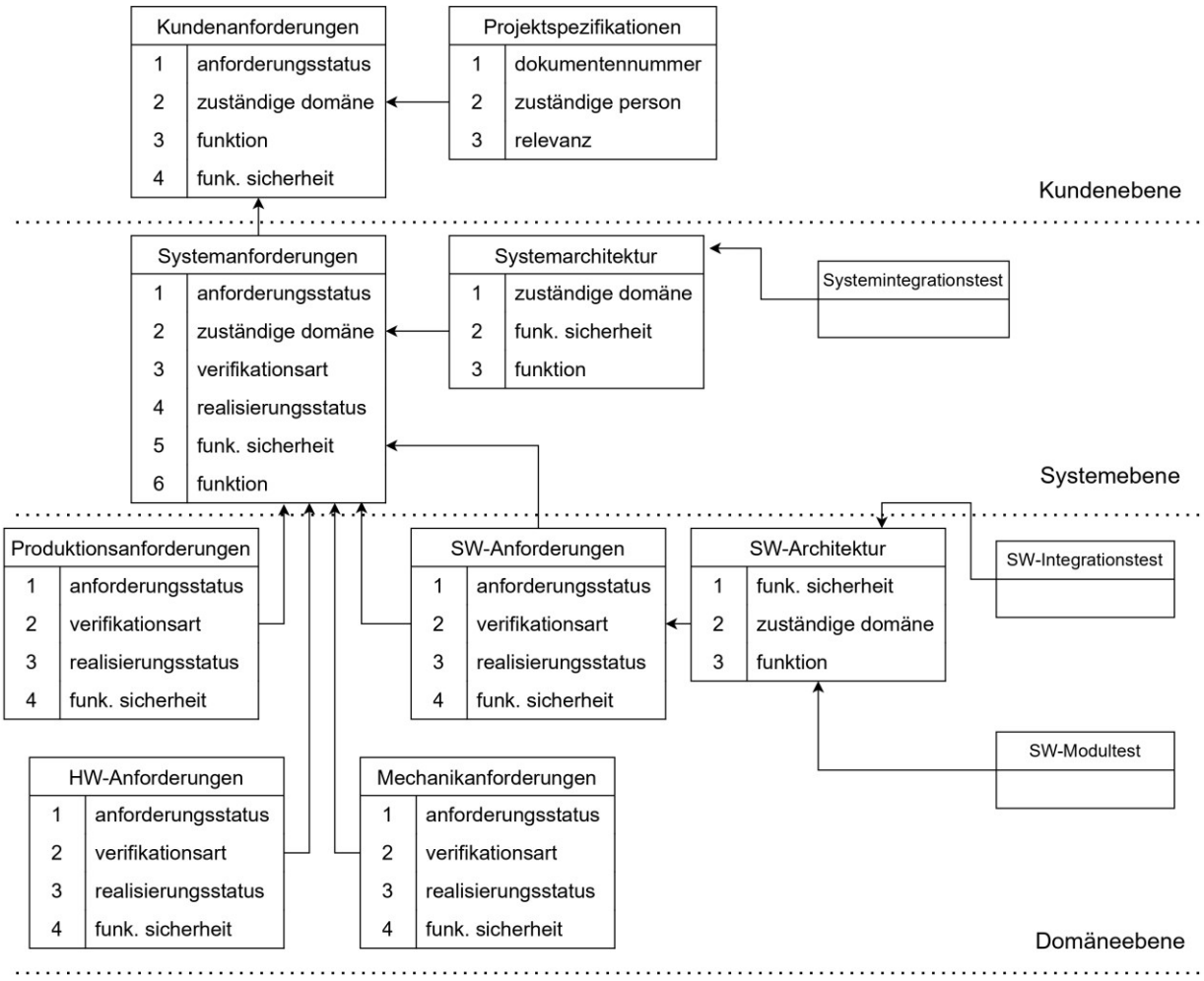

Abbildung 2-5 Datenmodell der Anforderungen

Domäne	System			Software		
	Relevante Objekte	Verlinkte Objekte	Missing Links	Relevante Objekte	Verlinkte Objekte	Missing Links
Kunde	1480	1480		924	840	1
Hardware						
Mechanik						
System				1119	1082	17

Abbildung 2-6 Rückverfolgbarkeitsmatrix

In der ersten Spalte und der ersten Zeile der Matrix werden Domänen (oder Klassen) von Anforderungen aufgelistet. Die Zelle in der Tabelle, die eine bestimmte Domäne und Spalte schneidet, zeigt die Anzahl der Anforderungen an, die einem der drei möglichen Zustände entsprechen:

- Die Gesamtzahl relevanter Anforderungen von der Domäne in der entsprechenden Zeile, die einen eingehenden Link von zumindest einer Anforderung in der darunterliegenden Domäne in der entsprechenden Spalte bekommen sollen (Relevancy-Check).
- Die Anzahl der Anforderungen von der Domäne in der Zeile, die einen korrekten Link von zumindest einer darunterliegenden Anforderung, die zur Domäne in der Spalte gehört, haben (Linked-Check).
- Die Anzahl der Anforderungen von der Domäne in der Zeile, die keinen korrekten Link von untergeordneten Anforderungen haben, die zur Domäne in der entsprechenden Spalte oder zu einer anderen zulässigen Domäne gehören (Missing Links-Check).

Zusätzlich kann ein Check eingeführt wird, der die Anzahl der Anforderungen anzeigt, die falsche Links aufweisen.

Letztendlich soll sichergestellt werden, dass keine Anforderung unverlinkt oder mit falschen Verlinkungen zurückgelassen wird. Das in dieser Arbeit entwickelte System zielt darauf ab, die Suche nach Verlinkungen zu automatisieren oder zumindest den Aufwand für die Analyse der Beziehungen zwischen den Anforderungen zu reduzieren, indem es die wahrscheinlichsten Kandidaten für die Verknüpfung empfiehlt.

2.2 Grundlagen der Verarbeitung natürlicher Sprache

2.2.1 Grundlegende Begriffe

Wie bereits erwähnt, werden die meisten Anforderungen in Spezifikationen in natürlicher Sprache formuliert. In diesem Kapitel werden daher grundlegende Konzepte aus dem Bereich der natürlichen Sprachverarbeitung erläutert, auf die im weiteren Verlauf des Entwicklungsprozesses des Empfehlungssystems Bezug genommen wird.

Eine natürliche Sprache ist jede Sprache, die nicht absichtlich geschaffen wurde, sondern auf natürliche Weise durch menschliche Interaktion entstanden ist (Lyons, 1991). Englisch und Deutsch sind also natürliche Sprachen, während alle Programmiersprachen oder zum Beispiel Esperanto konstruierte Sprachen sind. Obwohl die Sprache in technischen Spezifikationen einen eher speziellen Charakter haben kann, voll von Fachbegriffen und Abkürzungen, handelt es sich dennoch um eine natürliche Sprache (wie Englisch und Deutsch), die den Regeln der Grammatik und Rechtschreibung entspricht.

Jede Sprache besteht aus zwei fundamentalen Systemen - Syntax und Semantik - die zusammenarbeiten, um die Sprache als zuverlässiges Mittel zur Übertragung von Informationen zu ermöglichen. Syntax bezieht sich auf die grammatische Struktur der Sprache. Sie umfasst Regeln, die bestimmen, wie Wörter sowie Satzzeichen miteinander kombiniert werden, um Sätze, Phrasen und Absätze zu bilden (Bussmann, 2008). Im Wesentlichen bietet Syntax die Grundlage für die Organisation von Wörtern in sinnvolle Strukturen. Semantik hingegen befasst sich mit der Bedeutung, die mit der Sprache verbunden ist. Sie untersucht, wie Wörter und Sätze Informationen vermitteln (Bussmann, 2008). Semantik hilft uns zu verstehen, was ein Satz bedeutet, abgesehen von seiner einfachen Struktur.

Die Verarbeitung natürlicher Sprache (engl. *Natural Language Processing, NLP*) ist ein Bereich an der Schnittstelle von Linguistik und Informatik, die statistischen Methoden sowie die Verfahren des maschinellen und tiefen Lernens einsetzt. Das Ziel der natürlichen Sprachverarbeitung ist es, den Computer in die Lage zu versetzen, Texte in natürlicher Sprache zu verstehen und/oder zu generieren, ähnlich wie ein Mensch (Jurafsky & Martin, 2000). Mit den Methoden der NLP können verschiedene Arten von linguistischer Analyse durchgeführt werden, die die Eigenschaften von einzelnen Wörtern, Sätzen und größeren abgeschlossenen Texten bestimmen. Die Analyse kann sich auf die Erkennung von Wort- oder Satzstruktur (d. h. Syntax) oder auf die Bestimmung der semantischen Bedeutung von Wörtern und Sätzen konzentrieren. Letztendlich trägt die Anwendung verschiedener Arten von linguistischer Analyse zur Entwicklung spezifischer Anwendungen wie Textklassifizierung nach Thema, Sentimentanalyse, maschinelle Übersetzung, Information Retrieval, Textgenerierung, Speech-zu-Text oder Text-zu-Speech-Umwandlung bei (Tunstall, von Werra, & Wolf, 2022). Die wichtigsten Arten der Analyse, die auch für die vorliegende Arbeit relevant sind, sind Tokenisierung, Part-of-Speech-Tagging und Named Entity Recognition.

2.2.2 Grundlegende Arten der Sprachanalyse in NLP

Tokenisierung

Bei der Tokenisierung wird der Text in einzelne Einheiten, sogenannte Token, zerlegt, z. B. in Wörter oder Teilwörter, die dann analysiert und weiterverarbeitet werden können (Tunstall, von Werra, & Wolf, 2022). Wichtig ist, dass jedem Token ein eindeutiger numerischer Identifikator in einem Wörterbuch zugewiesen werden kann und die Wortfolge in eine Zahlenreihe umgewandelt wird, die von einem Computer verstanden werden kann.

Das einfachste Beispiel für einen Tokenizer ist der Algorithmus zum Parsen nach Leerzeichen, Zeilenumbrüchen oder Tabulatorzeichen. Bei Sprachen, die Satzzeichen verwenden, wird dieser Tokenizer immer falsche Ergebnisse liefern. Der wortbasierte Tokenizer verwendet komplexere Regeln und berücksichtigt die Zeichensetzung bei der Aufteilung des Textes in Wörter. Normalerweise verwendet dieser Tokenizer reguläre Ausdrücke, um Wortgrenzen zu definieren. Ein wortbasierter Tokenizer ist deshalb problematisch, weil er ein großes Vokabular benötigt, um alle Wörter einer Sprache abzubilden (Tunstall, von Werra, & Wolf, 2022). Zum Beispiel wird die Anzahl der Wörter allein im Deutschen auf 350.000 bis 500.000 Wörter geschätzt. In Sprachen mit komplexer Wortmorphologie werden ähnliche Wörter, die sich nur durch Endungen, Präfixe oder Suffixe unterscheiden, als separate Token dargestellt, z. B. Wörter wie "Katze" und "Katzen", was zu Informationsverlusten führt. Wenn der Text einen Rechtschreibfehler enthält, kann es sein, dass der Algorithmus das Token mit keinem der Token im Tokenizer-Wörterbuch abgleichen kann und es dann als unbekanntes Token gekennzeichnet wird.

Am anderen Ende des Spektrums der Tokenisierungsmethoden steht die symbolische Tokenisierung, mit der die Möglichkeit unbekannter Token vollständig eliminiert werden kann. Bei der symbolischen Tokenisierung wird jedes Zeichen als eigenes Token behandelt, wodurch ein kleinerer Wortschatz entsteht und der Informationsverlust verringert wird (Tunstall, von Werra, & Wolf, 2022). Dieser Ansatz hat aber auch Nachteile, denn ein Symbol selbst bedeutet natürlich viel weniger als ein Wort, d. h. es ist keine semantische Einheit, es sei denn, es wird eine Hieroglyphensprache betrachtet, was für diese Arbeit nicht relevant ist.

Moderne Tokenisierungsalgorithmen nehmen das Beste aus den beiden vorherigen Ansätzen und basieren auf Teilwörtern. Das Prinzip eines solchen Algorithmus ist es, die häufigsten Zeichenkombinationen innerhalb von Wörtern im Korpus zu finden. Solche Kombinationen, auch wenn sie das ganze Wort umfassen, sollten nicht unterteilt werden. Seltene Wörter hingegen sollten in die häufigsten Unterwörter unterteilt werden (Tunstall, von Werra, & Wolf,

2022). Tabelle 2.1 zeigt einige der typischsten Beispiele für die Aufteilung von Wörtern in Teilwörter, z. B. die Trennung von Suffixen, Präfixen, Endungen oder die Aufteilung eines zusammengesetzten Wortes.

Tabelle 2-1 Beispiele für die Aufteilung vom Wort in Teilwörter

Beispielwort	Tokenisiertes Wort	Anmerkung
gehen	['▁gehen']	Keine Trennung
Lehrerin	['▁Lehrer', 'in']	Suffixabtrennung
konsumieren	['▁konsum', 'ieren']	Abtrennung der Wortendung
abnehmen	['▁ab', 'nehmen']	Trennung der Präfixe
Zusammensetzung	['▁Zusammen', 'setzung']	Aufteilung vom zusammengesetzten Wort
Übersiedlungskarton	['▁Uber', 'si', 'ed', 'lung', 'skar', 'ton']	Aufteilung in mehrere Teile

Jedes Teilwort hat also eine bestimmte Semantik, und die Größe des Wörterbuchs zur Abdeckung aller möglichen sprachlichen Konstruktionen bleibt relativ klein. Die am häufigsten verwendeten Algorithmen zur Tokenisierung von Teilwörtern sind Byte-Pair Encoding (BPE), WordPiece und Unigram.

Die Sprachmodelle, die in dieser Arbeit trainiert werden, basieren auf dem XLM-Roberta-Mehrsprachenmodell (Conneau, Khandelwal, Goyal, Chaudhary, & Wenzek, 2020), das den BPE-Tokenizer (Gage, 1994) verwendet, auf den weiter unten näher eingegangen wird. Der Tokenisierungsprozess besteht aus vier Hauptphasen: Normalisierung, Vor-Tokenisierung, Modellierung und Nachbearbeitung. Die Normalisierung ist optional und besteht aus dem Entfernen von zusätzlichen Leerzeichen, diakritischen Zeichen und der Kleinschreibung von Wörtern. Bei der Pretokenisierung wird der Text einfach in einzelne Wörter zerlegt. Bei der Modellierung werden Wörter auf der Grundlage des Wörterbuchs und der Zusammenführungsregeln des Tokenizers in einzelne Unterwörter zerlegt. Bei der Nachbearbeitung werden spezielle Token hinzugefügt, wie z. B. ein Textanfangs- oder Trennzeichen, wenn zwei Texte in das Modell eingegeben werden (Hugging Face, 2023).

Das Tokenizer-Wörterbuch und die Zusammenführungsregeln werden durch unüberwachtes Lernen gewonnen, das wie folgt abläuft:

1) Zunächst wird ein großer Textkorpus genommen, der die Zielsprache oder mehrere Sprachen gut repräsentiert.

2) Ein Array aller Zeichen im Korpus wird extrahiert. Dies ist das anfängliche Token-Wörterbuch.

3) Dann wird das häufigste Tokenpaar gefunden. Dieses Paar bildet ein neues Token und wird dem Wörterbuch hinzugefügt. Außerdem wird das verschmolzene Tokenpaar dem Array der erlernten Zusammenführungsregeln hinzugefügt.

4) Der Prozess wird so lange wiederholt, bis die Größe des Wörterbuchs eine bestimmte Größe erreicht hat.

Bei der Tokenisierung eines neuen Satzes besteht die dritte Stufe der Tokenisierung, die Modellierung, darin, jedes Wort in Symbole zu zerlegen und dann die gelernten Zusammenführungsregeln in der Reihenfolge der Symbole im Wort anzuwenden. Die Tokenisierung ist die Grundlage für alle anderen Arten der linguistischen Analyse, die im NLP durchgeführt werden (Hugging Face, 2023).

POS-Tagging

Das Part-of-Speech-Tagging gehört ebenfalls zu den grundlegenden Aufgaben der Verarbeitung natürlicher Sprache. Es besteht darin, jedes Wort in einem Satz zu kennzeichnen, um welchen Teil der Sprache es sich handelt, z.B. Substantiv, Verb oder Adjektiv (Bird, Klein, & Loper, 2009). Die sich daraus ergebende Folge von Kennzeichnungen wird dann für andere Arten von Analysen verwendet, z.B. für das Parsen der Abhängigkeiten in einem Satz und die Erkennung von benannten Entitäten. In dieser Arbeit wird das Part-of-Speech-Tagging verwendet, um die Qualität des Anforderungstextes zu bewerten, insbesondere die Vollständigkeit der Sätze. Auf der Grundlage dieser Bewertung können einige Elemente des Datensatzes aus dem Trainingssatz ausgeschlossen werden.

Named Entity Recognition

Named Entity Recognition (NER) ist der Prozess der automatischen Identifizierung und Klassifizierung von benannten Entitäten wie Personen, Organisationen, Orten und anderen relevanten Entitäten im Text (Bird, Klein, & Loper, 2009). Abbildung 2.7 zeigt ein Beispiel für eine solche Klassifizierung.

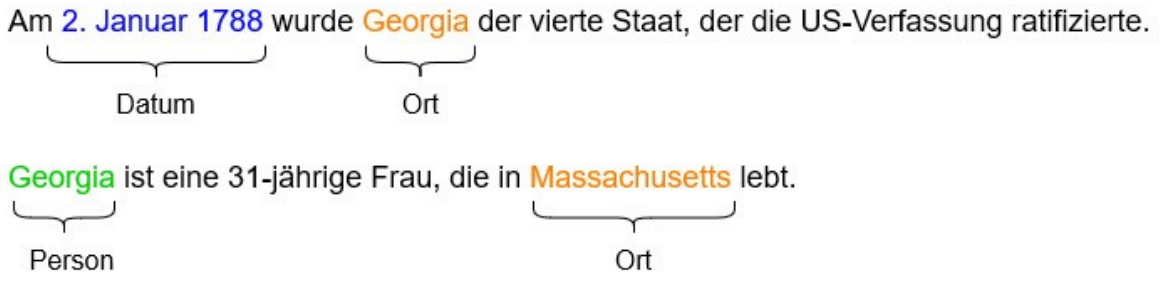

Abbildung 2-7 Beispiel für Named Entity Recognition

NER dient also der Extraktion von Informationen aus Text auf der Grundlage von vordefinierten Bezeichnungen. Der Prozess der Entitätserkennung besteht aus zwei aufeinanderfolgenden Phasen: 1) der Segmentierung des Textes in Entitäten, wobei eine einzelne Entität aus mehreren Token bestehen kann, und 2) der Entitätsklassifizierung, d. h. der Bestimmung der Wahrscheinlichkeit, dass eine bestimmte Entität einem Label entspricht (Pakhale, 2023). Im Rahmen der vorliegenden Arbeit ermöglicht die NER die Extraktion bestimmter Schlüsselwörter aus den Anforderungstexten, die bei der Suche im Korpus mit einem höheren Gewicht als die anderen berücksichtigt werden. In dem Kapitel 3 wird die Rolle dieses Prozesses näher erläutert.

2.2.3 Vektordarstellung von Texten

Ein zentrales Problem bei der Verarbeitung natürlicher Sprache ist das Problem der Kodierung der Bedeutung eines einzelnen Wortes oder Textes in einer Form, die Computeralgorithmen möglichst effizient und korrekt verarbeiten können. Im klassischen NLP erfolgt die Kodierung eines Wortes mit Hilfe eines dünnbesetzten Vektors, der die Größe eines Wörterbuchs hat. Das Element mit dem Index des zu kodierenden Wortes hat den Wert 1, alle anderen Elemente des Vektors sind Nullen. Auf die gleiche Weise können Sätze oder große Textabschnitte kodiert werden, indem die Anzahl der Erwähnungen eines Wortes oder Tokens gezählt und der entsprechende Wert einem Element im Vektor zugewiesen wird (Lavrač, Podpečan, & Robnik-Šikonja, 2021).

Das Problem bei diesem Ansatz ist, dass es unmöglich ist, die Ähnlichkeit bestimmter Wörter zueinander zu schätzen. Das heißt, wenn Wörter im Wörterbuch nahe beieinanderstehen, bedeutet das nicht, dass sie auch nahe Bedeutungen haben. Die Lösung für das Problem der dünnbesetzten Vektoren ist die dichte Vektordarstellung (Gillick, Presta, & Tomar, 2018). Wenn die Größe eines dünnbesetztes Vektors Hunderttausende beträgt (Größe des Wörterbuchs), liegt die Größe eines dichten Vektors zwischen 100 (Word2Vec) und mehreren Tausend (OpenAI Ada-Modell, 3072) Elementen. Und der Wortbedeutung ist auf alle Elemente des Vektors verteilt. In diesem Fall kann die Ähnlichkeit der Wörter anhand des Skalarprodukts der Vektoren oder des Kosinus-Ähnlichkeit gemessen werden.

In den frühen 2010er Jahren wurden mehrere Algorithmen zum Lernen von Vektordarstellungen von Wörtern (sogenannten Worteinbettungen) aus einem bestehenden Korpus entwickelt. Die bekanntesten sind Word2Vec (Mikolov, Chen, Corrado, & Dean, 2013), GloVe (Pennington, Socher, & Manning, 2014) und fastText (Bojanowski, Piotr, Grave, Edouard, Joulin, & Mikolov, 2016). Das Training basiert auf der Annahme, dass die Bedeutung eines Wortes durch den Kontext der Wörter bestimmt wird, die im Text neben dem Wort stehen

(siehe Abbildung 2.8). Während des Trainings optimiert der Algorithmus die Elemente des Vektors so, dass die Häufigkeit der Kontextvorhersage für ein bestimmtes Wort maximiert wird.

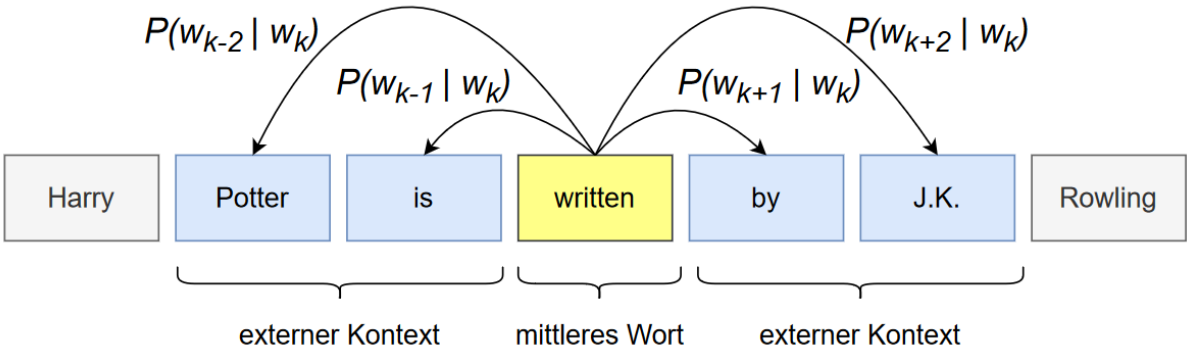

Abbildung 2-8 Kontext des Wortes im Satz. Veränderte Darstellung nach: Zhao, Liu, Li, Li, & Du, 2017

Die oben genannten Modelle gelten inzwischen als veraltet und sind den Sprachmodellen unterlegen, die mit Deep Learning und der Transformer-Architektur erstellt wurden und im nächsten Kapitel behandelt werden.

2.2.4 Deep-Learning-Modelle

Bevor es zu Deep Learning-Modellen kam, entwickelte sich NLP von regelbasierten symbolischen Ansätzen zu statistischem NLP, bei dem maschinelle Lerntechniken und große Mengen an linguistischen Daten (Textkorpora) verwendet werden, um ungefähre, probabilistische Modelle der Sprache zu entwickeln (Zhao, et al., 2021). Bei symbolischen Methoden wird die Sprache eingehend analysiert, um zahlreiche Muster zu erkennen, die dann in einer Vielzahl von Anwendungen wie der Textklassifizierung verwendet werden können. Unabhängig davon, wie präzise und ausgefeilt die Regeln sind, können sie die Vielfalt der Sprache nicht erfassen und müssen ständig aktualisiert werden, um neue sprachliche Phänomene zu berücksichtigen. Seit den 1990er Jahren ist es dank bahnbrechender Fortschritte in der Computertechnologie möglich, große Rechenressourcen für das Training statistischer Modelle zu nutzen (Hirschberg & Manning, 2015). Mit Hilfe von etikettierten Daten, die eine große Anzahl von Beispielen des realen Sprachgebrauchs enthalten, können maschinelle Lernmethoden automatisch Regeln generieren, die den Fehler bei einer bestimmten Aufgabe minimieren.

Deep Learning-Modelle sind eine Weiterentwicklung der Methoden des maschinellen Lernens und basieren auf neuronalen Netzwerken (Goodfellow, Bengio, & Courville, 2016). Ein neuronales Netzwerk ist in der Lage, tokenisierten Rohtext ohne weitere Vorverarbeitung als Eingabe zu nehmen und dessen Vektordarstellung zu erzeugen. Auf diese Weise muss der Mensch keine Muster und Merkmale in den Daten selbst entdecken.

Die elementare Einheit eines neuronalen Netzwerks ist ein Operator (das so genannte künstliche Neuron), der einen Vektor als Eingabe erhält und diesen Skalar mit einem Vektor von Gewichten multipliziert und eine Verschiebung (oder Bias) hinzufügt (siehe Abbildung 2.9). Der resultierende Wert wird dann durch eine nichtlineare, sogenannte Aktivierungsfunktion geleitet (Zhou, 2021). Früher wurde die Sigmoid- oder logistische Funktion als Aktivierungsfunktion verwendet, heute sind die beliebtesten Funktionen Tanh, ReLu und Leaky ReLu, da sie mit weniger Rechenaufwand numerisch abgeleitet werden können (Godoy, 2021, S. 354-358).

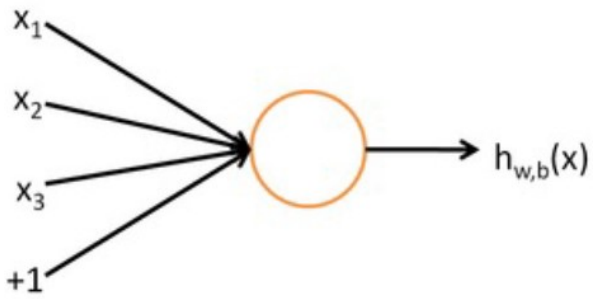

Abbildung 2-9 Ein künstlicher Neuron. Erschienen in: Cui, 2018, S. 2

Eine Schicht des neuronalen Netzes besteht aus mehreren solcher Neuronen und erzeugt einen Vektor der entsprechenden Dimensionalität als Ausgabe. Das neuronale Netzwerk besteht aus mehreren Schichten (siehe Abbildung 2.10), was mehrere Transformationen und Zusammensetzungen der Daten und das Training eines Klassifizierers ermöglicht, der in Bezug auf den Eingabevektor stark nichtlinear ist. Das klassische neuronale Netzwerk, das oben beschrieben wurde, heißt Feed-Forward Netzwerk (Zhou, 2021).

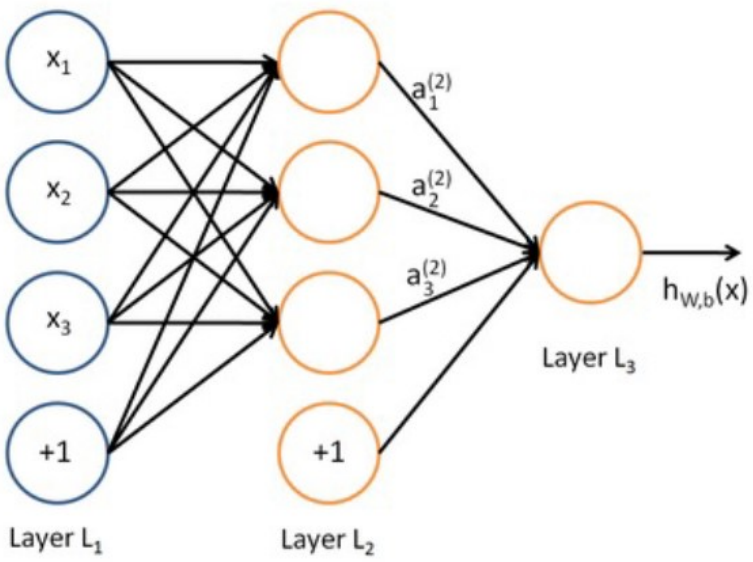

Abbildung 2-10 Feed-Forward neuronales Netzwerk. Erschienen in: Cui, 2018, S. 3

Unter der Annahme, dass das Modell den Eingabevektor in drei Klassen klassifizieren muss, muss die letzte Schicht des neuronalen Netzwerks aus drei Neuronen bestehen. Ganz am Ende des Netzwerks vergleicht die Verlustfunktion die Werte des Vektors mit dem Label. Die klassische Verlustfunktion für Klassifizierer ist eine Kreuzentropie-Funktion, die den negativen Logarithmus der Wahrscheinlichkeit einer "wahren" Klasse darstellt (Goodfellow, Bengio, & Courville, 2016).

Jedes Gewicht und jede Verschiebung im Neuron sind Parameter des Netzwerks. Die Aufgabe des Lernalgorithmus besteht darin, den Parameter so zu aktualisieren, dass die Verlustfunktion minimiert wird. Der Standardoptimierungsalgorithmus für ein neuronales Netzwerk ist der stochastische Gradientenabstieg (Goodfellow, Bengio, & Courville, 2016). Die Aktualisierung der Parameter durch diesen Algorithmus erfolgt nach der folgenden Formel:

$$\theta_{neu} = \theta_{alt} - \alpha \cdot grad(J(\theta))$$

wobei α – die Lernrate des Netzwerks ist.

Um das Netzwerk zu trainieren, müssen also die Gradienten der Verlustfunktion in Bezug auf die einzelnen Netzwerkparameter berechnet werden, was normalerweise mit dem Backpropagation-Algorithmus geschieht.

Beim rein stochastischen Gradientenabstieg werden die Parameter nach jedem Vorwärtsdurchlauf aktualisiert, d. h. nach jedem Datenpunkt im Trainingssatz. Ein solcher Ansatz führt zu einem instabilen Trainingsverhalten und einer insgesamt schlechteren Leistung. Ein anderer extremer Ansatz besteht darin, Fehler zu aggregieren, nachdem alle Daten des Satzes durch das Netzwerk gelaufen sind, d. h. nach einer einzigen Epoche. Bei großen Datenmengen führt dieser Ansatz zu einem übermäßigen Verbrauch von Rechenressourcen und ist praktisch unmöglich zu implementieren, so dass in der Praxis der Datensatz in Mini-Batches aufgeteilt wird. Die Verluste werden durch die Daten im Mini-Batch aggregiert und nach dem Passieren aller Punkte werden die Netzwerkparameter aktualisiert. Die Anzahl der Punkte in einem Mini-Batch sowie der Wert der Lernrate sind einige der wichtigsten Hyperparameter für das Training (Godoy, 2021).

Wenn es direkt um die neuronale Netzwerke für Sprachmodelle geht, ist zu beachten, dass die Anzahl der Token im Text erheblich variieren kann. Gleichzeitig ist die Anzahl der Parameter in der klassischen neuronalen Netzwerkarchitektur muss vor dem Start des Trainings festgelegt werden. Darüber hinaus ist ein neuronales Netzwerk erforderlich, das die Wortreihenfolge in einem Satz berücksichtigt, da diese in den meisten Sprachen (insbesondere im Englischen) die Bedeutung des Satzes direkt beeinflusst. Daher wurden zunächst andere Architekturen, die für Token-Sequenzen geeignet sind, insbesondere rekurrente neuronale Netze

(RNN), für NLP-Anwendungen verwendet (Goodfellow, Bengio, & Courville, 2016). Das Prinzip der rekurrenten neuronalen Netze ist in Abbildung 2.11 dargestellt.

Bei jedem Zeitschritt im RNN erhält das Netzwerk einen Eingabevektor *x*, der z.B. mit Word2Vec ermittelt wird und das aktuelle Wort oder Zeichen in der Sequenz darstellt, sowie einen verborgenen Zustandsvektor *h*, der den "Speicher" des Netzwerks für frühere Eingaben darstellt, die es gesehen hat. Die Eingabedaten und der verborgene Zustand werden von den Neuronen des Netzwerks kombiniert und verarbeitet, um einen Ausgabevektor *o* zu erzeugen, der für die Vorhersage des nächsten Worts oder Zeichens in der Sequenz verwendet wird. Das Ausgabesignal wird auch als neuer verborgener Zustand für den nächsten Zeitschritt an das Netzwerk zurückgegeben, so dass sich das Netzwerk Informationen aus früheren Schritten merken und sie für seine Vorhersagen in jedem nachfolgenden Schritt verwenden kann (Goodfellow, Bengio, & Courville, 2016).

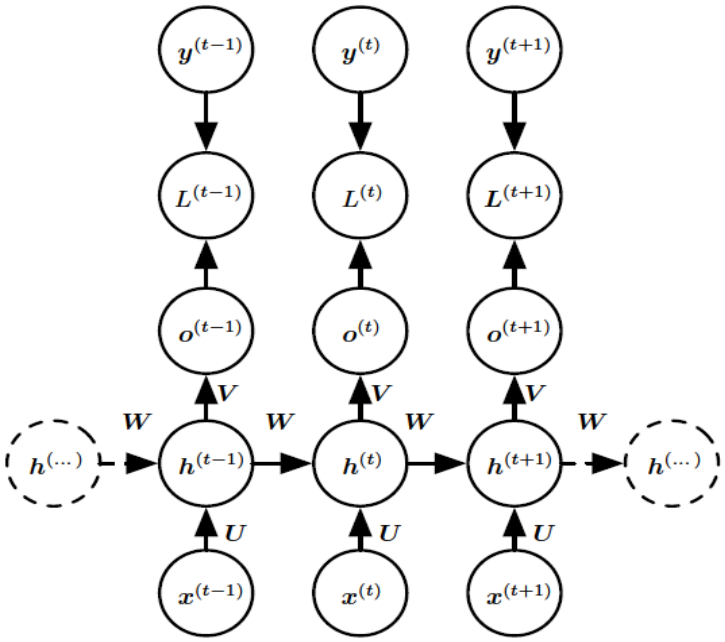

Abbildung 2-11 Rekurrentes neuronales Netzwerk. Erschienen in: Goodfellow, Bengio, & Courville, 2016, S. 369

Klassische RNNs haben einen Nachteil, der sie für ausreichend lange Sequenzen unwirksam macht, nämlich das Problem der verschwindenden Gradienten. Während des Trainings eines RNN wird der verborgene Zustand ständig überschrieben, was dazu führt, dass bei der Anpassung der Parameter durch den Backpropagation-Algorithmus keine Wörter mehr berücksichtigt werden, die weit vom Ende des Satzes entfernt sind (Goodfellow, Bengio, & Courville, 2016). Die LSTM-Architektur (*engl. Long Short-Term Memory*), die Anfang der 2000er Jahre

entwickelt wurde, geht dieses Problem an, indem sie mehrstufige "Tore" einführt, die den Informationsfluss in den und aus dem verborgenen Zustand regulieren. Diese Tore sollen es dem LSTM ermöglichen, sich selektiv an Informationen aus früheren Zuständen zu erinnern oder diese zu vergessen (Hochreiter & Schmidhuber, 1997). Dies hilft dem Netzwerk, langfristige Abhängigkeiten aufrechtzuerhalten, indem das Problem des verschwindenden Gradienten vermieden wird.

Das LSTM löst somit den größten Nachteil herkömmlicher rekurrenter Netzwerke, bleibt aber rechnerisch ineffizient, da jeder neue versteckte Zustand sequentiell berechnet wird. Der Vorteil von Grafikkarten, die mehrere Operationen parallel ausführen können, bleibt ungenutzt. Die Lösung des Problems liegt in der Anwendung des Attention-Mechanismus (Vaswani, et al., 2017), oder genauer gesagt der Self-Attention, der die rekurrenten Schichten in modernen neuronalen Netzen des NLP vollständig ersetzt hat.

Die Attention-Funktion kann als Abbildung einer Abfrage (*engl. query*) und eines Satzes von Schlüssel-Wert-Paaren auf eine Ausgabe beschrieben werden, wobei die Abfrage, die Schlüssel, die Werte und die Ausgabe Vektoren sind. Die Ausgabe wird als gewichtete Summe der Werte berechnet, wobei das jedem Wert zugewiesene Gewicht durch die Mapping-Funktion der Abfrage mit dem entsprechenden Schlüssel berechnet wird. Im Falle einer Textsequenz ist die Abfrage eine Vektordarstellung von einem Token. Die Schlüssel und Werte sind Vektordarstellungen von anderen Token im Satz, die verschiedene lineare Transformationen durchlaufen haben. In der Regel wird das Skalarprodukt von Vektoren als Mapping-Funktion gewählt (Vaswani, et al., 2017).

Abbildung 2.12 zeigt die mathematischen Operationen, die an dem Attention-Mechanismus beteiligt sind, und Abbildung 2.13 zeigt die Transformer-Architektur als Ganzes, die mehrere Ebenen von Attention umfasst. Die Architektur wurde ursprünglich zur Bewältigung eines maschinellen Übersetzungsproblems entworfen und setzt sich daher aus zwei Komponenten zusammen, einem Encoder und einem Decoder. Der Encoder erzeugt Vektorrepräsentationen für jedes Token aus dem Eingabetext. Der Decoder generiert Wahrscheinlichkeiten für jedes Wort aus dem Wörterbuch, um das nächste Wort im übersetzten Text zu werden (Vaswani, et al., 2017).

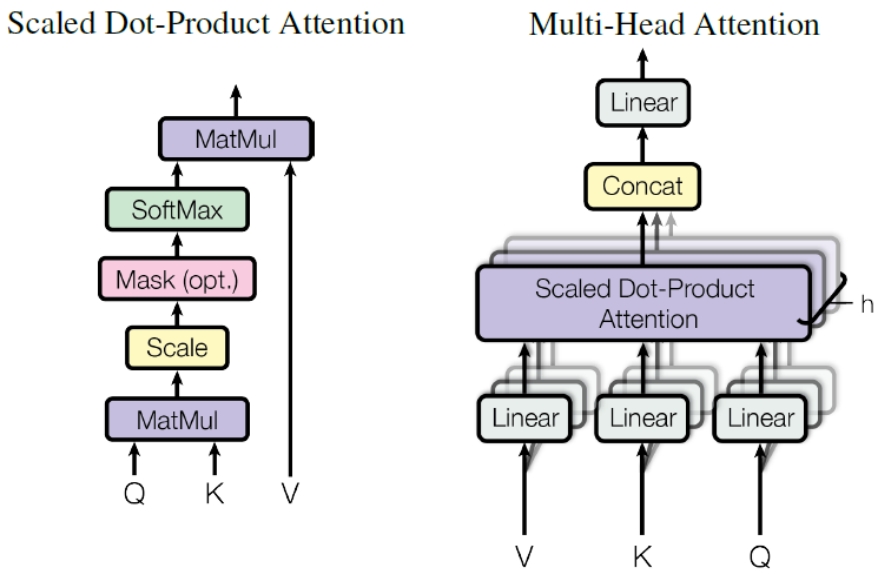

Abbildung 2-12 Attention-Mechanismus. Erschienen in: Vaswani, et al., 2017, S. 4

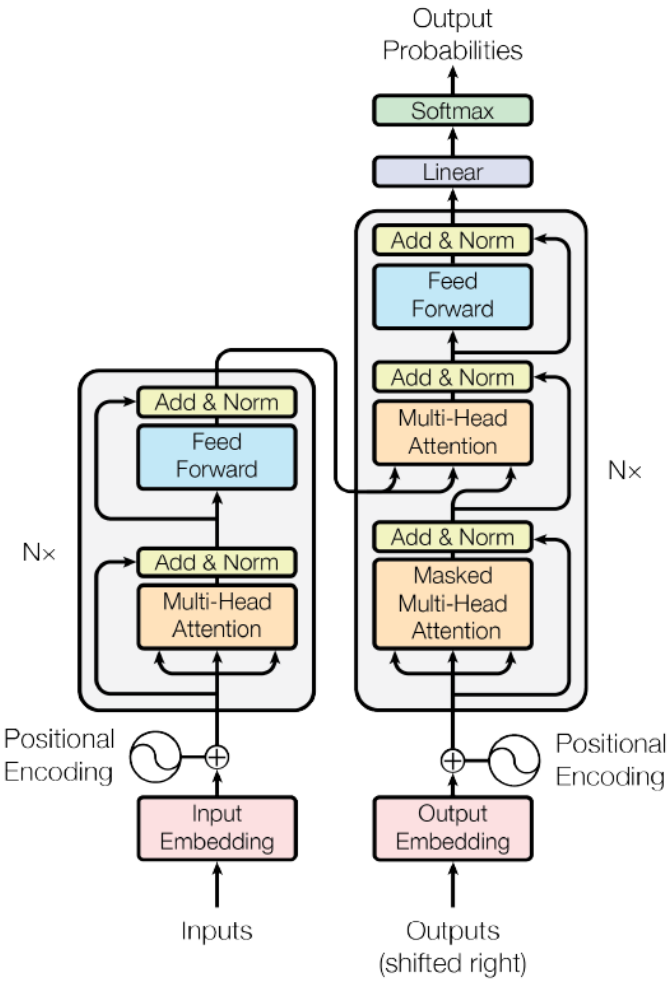

Abbildung 2-13 Transformer-Architektur. Erschienen in: Vaswani, et al., 2017, S. 3

Dank der Transformer-Architektur bleibt die Anzahl der sequenziellen Operationen unabhängig von der Länge des Eingabetextes konstant, was das Training eines neuronalen Netzwerks mit einer großen Anzahl von Parametern bei geringerem Ressourcenverbrauch ermöglicht. Bis heute basieren alle Sprachmodelle, die die besten Ergebnisse bei verschiedenen Benchmarks erzielen, sowohl Open Source (BERT, T5) als auch proprietär (GPT, Claude), auf der Transformer-Architektur. Die oben genannten Sprachmodelle wurden auf riesigen Datensätzen trainiert, was nach wie vor erhebliche Ressourcen - Zeit, Energie und Hardware – erfordert (Sanh, Debut, Chaumond, & Wolf, 2019), so dass das Training eines solchen Sprachmodells von Grund auf für die meisten Forschungs- und gewerblichen Organisationen unpraktisch ist. Gleichzeitig müssen dichte Vektordarstellungen oft für bestimmte Anwendungsfälle oder spezielle Wissensgebiete angepasst werden (Luan, Eisenstein, Toutanova, & Collins, 2020). Die Transfer-Learning-Technologie kann in dieser Situation helfen, indem sie die Parameter eines bereits trainierten Modells als Input für das Training auf einem eigenen, viel kleineren Datensatz für einen bestimmten Anwendungsfall verwendet.

Für Aufgaben im Zusammenhang mit dem Verstehen natürlicher Sprache, auf die sich diese Arbeit bezieht, ist das gängigste vortrainierte Modell BERT (Devlin, Chang, Lee, & Toutanova, 2018) und ähnliche Netzwerke (RoBERTa, SpanBERT). BERT kam 2018 auf den Markt und ist ein reiner Encoder, der als Eingabe einen tokenisierten Text (oder zwei Texte mit einem speziellen Token dazwischen) nimmt und eine Vektordarstellung jedes Worts erzeugt. Daran kann sich dann ein sogenannter Kopf anschließen, der ein reguläres Feed Forward-Netzwerk mit einer für die gewählte Aufgabe geeigneten Verlustfunktion ist. Solche Aufgaben können die Textklassifizierung, die Suche im Text nach einer Antwort auf eine Frage oder die Sprachmodellierung unter Verwendung einer Maske sein (Tunstall, von Werra, & Wolf, 2022).

BERT wurde ursprünglich für die Aufgabe trainiert, beschädigten oder maskierten Text wiederherzustellen. Konkret wurde BERT so trainiert, dass es 15 % der Token in den Eingabetexten vorhersagen konnte. 80 % davon wurden durch ein spezielles Maskentoken ersetzt, 10 % wurden durch ein anderes Wort ersetzt und 10 % wurden überhaupt nicht ersetzt. Das BERT-Basismodell umfasst 110 Millionen Parameter mit 12 Attention-Schichten und erzeugt einen latenten Zustandsvektor mit 768 Elementen (Devlin, Chang, Lee, & Toutanova, 2018). Die Datenquellen für das BERT-Training waren Wikipedia (2500 Millionen Wörter) und BookCorpus (800 Millionen Wörter), d. h. das Modell ist generisch und passt zunächst nur auf Englisch. Für die Behauptung, dass das Training eines großen Sprachmodells erhebliche Ressourcen erfordert, spricht die Tatsache, dass das BERT-Training auf 64 TPUs über 4 Tage hinweg durchgeführt wurde. Die Feinabstimmung kann jedoch problemlos auf einer GPU mit viel weniger Daten durchgeführt werden, wie im Kapitel 3 gezeigt wird. Als Grundlage für die

weitere Arbeit wurde das XLM-Roberta-Sprachmodell gewählt, das eine sehr ähnliche Architektur wie BERT hat, aber aufgrund seiner längeren Trainingszeit eine Verbesserung darstellt. Außerdem unterstützt das XLM-Roberta-Wörterbuch mehrere Dutzend Sprachen (Conneau, Khandelwal, Goyal, Chaudhary, & Wenzek, 2020).

Im nächsten Teil wird erörtert, wie Deep-Learning-Modelle bei der semantischen Suche helfen können, und es werden Algorithmen für die Stichwortsuche vorgestellt, die auf einer dünnbesetzten vektoriellen Textdarstellung beruhen und auch bei der Implementierung eines Empfehlungssystems weiter angewendet werden.

2.2.5 Information Retrieval

„Die Informationsrückgewinnung (*engl. Information Retrieval, IR*) ist das Auffinden von Material (in der Regel Dokumente) unstrukturierter Natur (in der Regel Text), das einen Informationsbedarf aus großen Sammlungen (in der Regel auf Computern gespeichert) erfüllt“ (Manning, Raghavan, & Schütze, 2008, S. 1). Beim IR stimmen die Abfrage und das gesuchte Dokument in der Regel nicht genau überein, und die Aufgabe besteht darin, ihre Darstellung zu finden, die es uns ermöglicht, ihre Relevanz im Verhältnis zueinander zu beurteilen. Im Folgenden werden zwei Haupttypen der IR betrachtet - die semantische Suche und die Stichwortsuche (*engl. Keyword Search, KS* oder *Full-Text Search*) - die auf unterschiedlichen Methoden zum Aufbau dieser Repräsentation, dichten und dünnbesetzten Indizes, basieren.

2.2.5.1 Stichwortsuche

Traditionell verwenden Suchmaschinen dünnbesetzte Indizes, um Dokumente in einem Korpus bei Bedarf zu durchsuchen. Wie bereits in Abschnitt 2.2.3 erwähnt, wird in diesem Fall jedes Dokument durch einen Vektor kodiert, der die Größe des gesamten Wörterbuchs hat. Die Werte eines Elements des Vektors sind die Anzahl der Vorkommen des entsprechenden Worts oder Tokens aus dem Wörterbuch in dem Dokument (Lavrač, Podpečan, & Robnik-Šikonja, 2021). Bevor der Index erstellt wird, werden die Dokumente im Korpus einer Vorverarbeitung unterzogen, die aus Stemming, d. h. der Extraktion von Wurzeln aus den im Dokument enthaltenen Wörtern, oder einem fortgeschritteneren Lemmatisierungsverfahren besteht, d. h. der Aufteilung der Wörter in morphologische Einheiten - Präfixe, Wurzeln und Suffixe. Außerdem können gesperrte Wörter (engl. stop words), die bei der Suche nicht berücksichtigt werden sollten, aus den Dokumenten entfernt werden. Bei den gesperrten Wörtern handelt es sich in der Regel um in der Sprache gebräuchliche Hilfswörter (Artikel, Präpositionen, Hilfsverben usw.), die keine spezifische Bedeutung haben und daher bei der Suche nicht berücksichtigt werden sollten (Bird, Klein, & Loper, 2009).

Nach der Vorverarbeitung und der Erstellung eines dünnbesetzten Index werden für jede Suchanfrage zwei Metriken berechnet: TF - die Häufigkeit eines Tokens/Wortes aus der Suchanfrage in einem Dokument und IDF - die inverse Häufigkeit der Dokumente, in denen das Token vorkommt (Spärck Jones, 1972). Um die endgültige Relevanzbewertung zu erhalten, werden diese Metriken multipliziert und dann über jedes Token in der Abfrage gemittelt. Das Dokument, in dem die Tokens der Abfrage häufiger vorkommen, wird somit als das relevanteste Dokument betrachtet, wobei die Punktzahl nach der Spezifität des Tokens gewichtet wird. Je häufiger ein Token in den Dokumenten des Korpus vorkommt, desto weniger spezifisch ist es und desto geringer sollte sein Beitrag zur Gesamtbewertung sein.

Moderne Suchmaschinen wie OpenSearch verwenden eine leicht modifizierte Metrik im Vergleich zu TF*IDF, die sich erfahrungsgemäß als stabiler erweist - BM25 (OpenSearch, 2024). BM25 berücksichtigt ebenfalls IDF und TF, aber die IDF jedes Tokens in einer Abfrage wird nach der folgenden Formel berechnet:

$$IDF(i) = \ln(\frac{N - n(i) + 0{,}5}{n(i) + 0{,}5} + 1)$$

wobei N die Gesamtzahl der Dokumente im Korpus und n(i) die Anzahl der Dokumente ist, in denen das Token vorkommt. Die endgültige Punktzahl wird als das Produkt aus der Summe der IDFs für jedes Token und der gewichteten TF errechnet:

$$score = \sum_i IDF(i) \cdot \frac{TF(i) \cdot (k1 + 1)}{TF(i) + k1 \cdot (1 - b + b \cdot \frac{Length(D)}{avgdl})}$$

wobei k1 und b freie Koeffizienten sind, deren Werte typischerweise 1,2 bzw. 0,75 betragen. Length(D) ist die Länge des Dokuments und avgdl ist die durchschnittliche Länge der Dokumente im Korpus, d. h. die Metrik unterschätzt künstlich die Relevanz langer Dokumente in der Ausgabe, um zu verhindern, dass diese gegenüber kürzeren Dokumenten bevorzugt werden (Robertson & Zaragoza, 2009).

Die Stichwortsuche oder Volltextsuche zielt also darauf ab, abfragespezifische Token in den Dokumenten des Korpus zu finden und sie auf der Grundlage des Vorkommens oder Nicht-Vorkommens dieser Stichwörter zu bewerten. Dabei wird das Vorkommen von Synonymen in den Dokumenten nicht berücksichtigt, und die Suche ist auf Dokumente beschränkt, die mit der Abfragesprache übereinstimmen. Das nächste Kapitel befasst sich mit der semantischen Suche, die versucht, die Mängel der Stichwortsuche zu überwinden.

2.2.5.2 Semantische Suche

Die semantische Suche setzt voraus, dass es eine Darstellung des Textes gibt, die seine Bedeutung in ihrer Gesamtheit erfasst. Die relevantesten Dokumente sollten daher so nah wie

möglich an der Bedeutung der Anfrage sein (Bast & Buchhold, 2016). In diesem Fall kann ein Dokument die Suchanfrage des Benutzers erfüllen und kein einziges Wort mit ihr gemeinsam haben oder in einer ganz anderen Sprache verfasst sein. Die semantische Suche sollte daher bei Abkürzungen, Synonymen oder Rechtschreibfehlern besser funktionieren.

Während Deep Learning-Sprachmodelle wie BERT genutzt werden können, um Text in eine universelle Vektordarstellung zu kodieren, sollte beachtet werden, dass jedes Transformer-Modell in seiner ursprünglichen Form einen mehrdimensionalen verborgenen Zustand erzeugt. Wenn ein Satz beispielsweise aus 10 Token besteht, wird für jedes Token eine Vektordarstellung mit 768 (für BERT) Elementen definiert. Darüber hinaus wird die Vektordarstellung durch spezielle anfängliche [CLS] und trennende [SEP] Token erhalten. Daher können Texte mit unterschiedlichen Dimensionen nicht direkt mit dem Skalarprodukt oder der Kosinus-Ähnlichkeit verglichen werden.

Eine Standardlösung bestand bisher darin, nur die Vektordarstellungen des anfänglichen Tokens [CLS] zu vergleichen, da sie sich auf den gesamten Text beziehen, oder den Durchschnitt der Werte aller Vektoren zu ermitteln. Solche Lösungen sind nicht optimal und zeigen bei Standard-Benchmarks, z.B. beim STS-Datensatz, keine akzeptablen Ergebnisse. Daher wurde die BERT-Architektur speziell für die semantische Suchaufgabe leicht angepasst. Insbesondere wurde eine Pooling-Schicht hinzugefügt, die die Token-Vektoren mittelt, während das neuronale Netzwerk weiter auf Datensätzen wie STS und SNLI trainiert werden muss, damit die resultierenden Satzeinbettungen semantisch sinnvoll sind und durch Kosinus-Ähnlichkeit verglichen werden können. Abbildung 2.14 zeigt die Architektur des neuronalen Netzwerks, genannt Sentence-Transformer (Reimers & Gurevych, 2019).

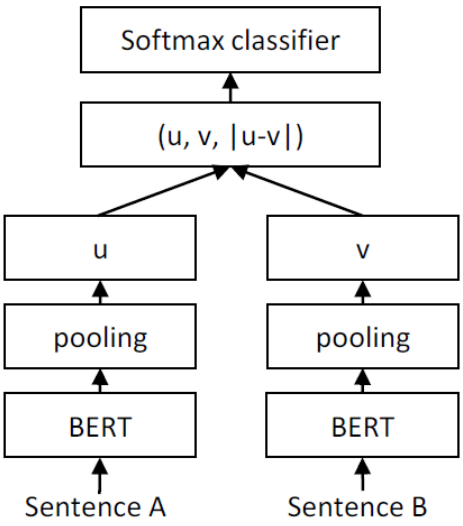

Abbildung 2-14 Architektur des Sentence-Transformers. Erschienen in: Reimers & Gurevych, 2019, S. 3

Das daraus resultierende Sprachmodell übertraf den Stand der Technik, und der vorgestellte Ansatz wurde zu einem generischen Ansatz für das Training von Vektordarstellungen von Sätzen. Auf dieser Grundlage wurde in der Folge eine Bibliothek von Sentence-Transformern erstellt, um neuronale Netzwerke dieser Art zu trainieren und einzusetzen (UKPLab, 2023).

Wenn der Transformer um eine Klassifizierungsschicht erweitert wird, die ein Feed-Forward-Netzwerk ist, kann es ebenfalls mit den oben genannten Datensätzen trainiert und so angepasst werden, dass es zwei Texte miteinander vergleicht. Ein solches Netzwerk erzeugt einen numerischen Wert (logit), der dann mithilfe der Sigmoid-Funktion in eine Wahrscheinlichkeit umgewandelt werden kann, dass die Sätze miteinander in Beziehung stehen. Ein solches Netzwerk wird im Folgenden als Cross-Encoder bezeichnet (Hofstätter, Althammer, Schröder, Sertkan, & Hanbury, 2020). Der Vorteil des Sentence-Transformer ist, dass die Berechnung der Einbettungen und der Indexaufbau in einer Suchmaschine offline erfolgen kann. Bei der Online-Suche muss man nur die Anfrage in einen Vektor umwandeln und dann die nächsten Nachbarn im Korpus finden. Gleichzeitig liefert der Cross-Encoder in der Regel genauere Ergebnisse, aber er ist zu langsam für die Aufgabe, beispielsweise die fünf relevantesten Dokumente in einer Datenbank mit vielen Tausend Dokumenten zu suchen. In der Praxis wird der Cross-Encoder verwendet, um Dokumente, die in einer Datenbank durch semantische Suche und/oder Stichwortsuche gefunden wurden, neu zu ranken (Luan, Eisenstein, Toutanova, & Collins, 2020).

2.2.5.3 Evaluierungsmetriken

Unabhängig von der verwendeten Suchmethode wird die Leistung eines Information-Retrieval-Systems beim Auffinden und Ranking relevanter Dokumente anhand mehrerer Standardmetriken gemessen. Alle Metriken lassen sich in zwei große Klassen einteilen - Online- und Offline-Metriken. Online-Metriken konzentrieren sich auf die Bewertung des Benutzerverhaltens bei der Interaktion mit dem System, z.B. die Anzahl der Benutzerklicks auf ein bestimmtes Dokument, die Zeit, die ein Benutzer mit dem Ansehen eines Dokuments verbringt, die Positionsbewertung eines ausgewählten Dokuments, die Zeit für die Auswahl eines Dokuments. Offline-Metriken bewerten die Qualität der Systemleistung vor dem Betriebsstart und setzen das Vorliegen von goldenen Labels voraus, d. h. eine Reihe wirklich relevanter Dokumente als Antwort auf bestimmte Abfragen, mit denen die von der Suchmaschine erzeugte Menge an Dokumenten verglichen wird (Herlocker, Konstan, Terveen, & Riedl, 2004). Es liegt außerhalb des Umfangs dieser Arbeit, die Leistung des Systems nach seiner Einführung zu untersuchen, daher werden im Folgenden die Offline-Metriken im Mittelpunkt stehen.

Offline-Metriken können auch in zwei Kategorien unterteilt werden - mit und ohne Berücksichtigung der Reihenfolge. Zu den nicht-reihenfolgenbezogenen Metriken gehören Genauigkeit,

Sensitivität und F1-Maß. Zu den fortschrittlicheren Metriken, bei denen die Reihenfolge in der Ausgabe berücksichtigt wird, gehören die durchschnittliche mittlere Genauigkeit (MAP), der mittlere umgekehrte Rang (MRR) und der normalisierte diskontierte kumulative Gewinn (NDCG).

Jedes einzelne Ergebnis, das das System als Antwort auf eine Anfrage liefert, wird als positives Ergebnis bezeichnet. Wenn das Ergebnis nach den festgelegten („goldenen“) Etiketten relevant ist, wird es als wahres positives Ergebnis betrachtet, andernfalls als falsches positives Ergebnis. Ein nicht ausgegebenes Ergebnis wird als negativ bezeichnet, wenn es dabei relevant ist, dann gilt es als falsches negatives Ergebnis, andernfalls als wahres negatives.

Genauigkeit (oder *engl. Precision*) ist die Anzahl der wahren positiven Ergebnisse im Verhältnis zur Gesamtzahl der erzeugten Ergebnisse, d. h.:

$$Genauigkeit = \frac{wahrePositive}{wahrePositive + falschePositive}$$

Sensitivität (oder *engl. Recall*) ist die Anzahl der richtig positiven Ergebnisse im Verhältnis zur Gesamtzahl der relevanten Ergebnisse für die betreffende Anfrage im goldenen Datensatz, d. h.:

$$Sensitivität = \frac{wahrePositive}{wahrePositive + falscheNegative}$$

Der F1-Maß ist das harmonische Mittel aus Genauigkeit und Sensitivität, das durch die folgende Formel ausgedrückt wird:

$$F1\text{-}Maß = \frac{2 \cdot (Genauigkeit \cdot Sensitivität)}{Genauigkeit + Sensitivität}$$

Diese Punktzahl kombiniert die beiden vorherigen Metriken und soll das System ausgewogen machen, so dass die Verbesserung der einen Metrik nicht ausschließlich zu Lasten der anderen geht (Zhou, 2021, S. 33-35).

Eine Abfrage kann eine unterschiedliche Anzahl von Antworten ausgeben, weiter als Niveau k bezeichnet, von dem die Werte der oben genannten Metriken abhängen. Daher wird im Folgenden immer nicht nur von Genauigkeit oder Sensitivität gesprochen, sondern auf Genauigkeit und Sensitivität bei einem bestimmten Niveau k. Darüber hinaus wird jede Metrik über die Anzahl der Abfragen aus dem goldenen Datensatz gemittelt. Recall@5 ist zum Beispiel die durchschnittliche Sensitivität des Systems, wenn es dem Benutzer 5 Ergebnisse für alle betrachteten Abfragen liefert.

Die beliebteste fortgeschrittene Metrik, die die Reihenfolge der Ergebnisse in der Ausgabe berücksichtigt, ist MAP (*engl. Mean Average Precision*). Um MAP zu berechnen, wird zunächst

die durchschnittliche Genauigkeit einer einzelnen Abfrage (*engl. Average Precision*, AP) ermittelt. Die durchschnittliche Genauigkeit ist die Summe der Genauigkeiten auf Niveau k, bei denen eine relevante Antwort erhalten wurde, geteilt durch die Anzahl der relevanten Ergebnisse in der Ausgabe (Herlocker, Konstan, Terveen, & Riedl, 2004). Wenn zum Beispiel insgesamt fünf Antworten gegeben wurden und die relevanten Ergebnisse an den Positionen 2 und 5 liegen, dann ist AP@5 = (P@2 + P@5) / 2 = (1/2 + 2/5) / 2 = 0,45. Die MAP@k ist die durchschnittliche AP@k für alle Abfragen.

MRR berücksichtigt auch die Reihenfolge der Ergebnisse in der Ausgabe, aber nur die erste relevante Antwort ist von Bedeutung. Zunächst wird die Summe der inversen Ränge der ersten relevanten Antworten für jede Abfrage berechnet und dann durch die Anzahl der Abfragen geteilt (Voorhees, 2000), d. h.:

$$MRR = \frac{\sum_i 1/_i}{Q}$$

NDCG ist eine fortschrittlichere Metrik als die vorherigen Metriken, da sie den Grad der Relevanz der Antworten auf die Abfrage berücksichtigt (Järvelin & Kekäläinen, 2002). Zunächst wird

$$DCG@k = \sum_k {rel_k}/{\log(1+k)}$$

für jede Abfrage berechnet, wobei relk ein gewisser Grad an Relevanz ist, je höher der Wert, desto höher die Relevanz. Dann wird der ideale Koeffizient

$$IDCG@k = \sum_k {\max(rel_k)}\Big/{\log(1+k)}$$

berechnet, d. h. es wird angenommen, dass alle Ausgaben maximale Relevanz haben. Der normalisierte diskontierte kumulative Gewinn ist einfach das Verhältnis von DCG zu IDCG. Der letzte Schritt: der NDCG@k wird über alle Abfragen gemittelt. Leider verfügt der goldene Datensatz oft nicht über eine Kennzeichnung, die für den Grad der Relevanz einer bestimmten Antwort verantwortlich ist, aber selbst in diesem Fall ist die NDCG-Metrik nützlich, da sie die Reihenfolge von Ergebnissen berücksichtigt.

2.3 Verwandte Arbeiten

Das Auffinden oder Wiederherstellen von Trace-Links ist ein seit Langem bekanntes Problem vor allem im Bereich der Softwareentwicklung. Zwei damit zusammenhängende Hauptaufgaben sind für Forscher innerhalb dieses Problems von Interesse - das Auffinden von Verbindungen zwischen Anforderungen und Anwendungsquellcode und das Auffinden von Verbindungen zwischen verschiedenen Ebenen von Anforderungen. Das erste Problem sprengt den

Rahmen dieser Arbeit und wird auch als schwieriger angesehen, weil Anforderungen und Programmcode eine unterschiedliche Syntax haben. Da es jedoch ganz oder teilweise dieselben Methoden und Prinzipien verwendet, sind die Artikel, die ihm gewidmet sind, ebenfalls in dieser Übersicht enthalten.

Bis in die späten 2010er Jahre waren die wichtigsten Techniken für diese Aufgabe verschiedene IR-Techniken, die auf der dünnbesetzten Vektordarstellung von Dokumenten basieren. In dem Paper "Comparison of Information Retrieval Techniques for Traceability Link Recovery" (Rodriguez & Carver, 2019) werden drei Methoden zur Berechnung der Ähnlichkeit zwischen Anforderungstext und Quellcode - probabilistisches Modell, sparse vector (in dem Papier Vektorraummodell genannt) und dichte semantische Vektordarstellung - am Beispiel von zwei Coest-Datensätzen (COEST Community, 2017) - EBT und eTour - verglichen. Die auf dem dünnbesetzten Vektor basierende Ähnlichkeitsfindung verwendet dieselben TF- und IDF-Metriken, die zuvor mit kleinen Abweichungen diskutiert wurden. Die dichte semantische Repräsentation wurde mit Hilfe der latenten semantischen Indexierungstechnik erzielt, die darin besteht, die Matrix der am häufigsten vorkommenden Wörter zu zerlegen und ihre Dimensionalität zu reduzieren. Diese Technik ist im Vergleich zu Word2Vec eine noch ältere Technik und hat erhebliche Einschränkungen, was auch in dieser Arbeit bestätigt wurde. Die in dieser Arbeit erzielten Ergebnisse zeigen, dass das Vektorraummodell eine recht hohe Genauigkeit von etwa 90% bei einer geringen Sensitivität von 26% aufweist. Die latente semantische Indizierung zeigt sowohl eine geringe Genauigkeit als auch eine geringe Sensitivität. Es ist anzumerken, dass die in dieser Arbeit verwendeten Datensätze aus weniger als 200 Artefakten und etwa 300 Trace-Links bestehen, was eine Übertragung der Ergebnisse auf andere Bereiche nicht zulässt. Der Mangel an Daten wird auch von den Autoren der Arbeit betont. Dennoch zeigt die Arbeit, dass das Vorkommen bestimmter Begriffe einen größeren Einfluss auf die Verlinkung von Artefakten haben kann als deren enge Semantik.

Die Arbeit "Automatische Wiederherstellung von Nachverfolgbarkeit zwischen Anforderungen und Quelltext" (Hey, 2023) ist eine Doktorarbeit und befasst sich, wie der Titel vermuten lässt, ebenfalls mit demselben Problem wie die erste besprochene Arbeit. In dieser Arbeit wird die so genannte FTLR (fine-grained traceability links recovery) entwickelt, die auf Vektordarstellungen von fastText basiert und als Ähnlichkeitsfunktion die Word Mover‘s Distance (WMD) verwendet. Im Gegensatz zu Sentence-Transformer-Modellen fasst FTLR die Einbettungen einzelner Text-Token nicht zu einem einzigen Vektor zusammen, sondern verwendet vollständige mehrdimensionale Einbettungen. Als zusätzliche Erweiterung wird ein BERT-basierter Klassifikator verwendet, der darauf abzielt, die im Quellcode enthaltenen Methoden und Klassen in die Umsetzung funktionaler und nicht-funktionaler Anforderungen zu klassifizieren und

somit irrelevante Teile des Programms im Voraus zu verwerfen. FTLR wird auch an frei verfügbaren Datensätzen getestet, die aus einer relativ kleinen Anzahl von Artefakten bestehen. Der Autor dieser Arbeit hat sich zum Ziel gesetzt, das Tool nutzen zu können, ohne es für jedes Projekt anpassen zu müssen, d. h. ohne die in jedem einzelnen Projekt bereits vorhandenen Rückverfolgbarkeitsinformationen zu nutzen. Die in der Arbeit erzielten Ergebnisse zeigen ein gemischtes Bild und variieren je nach Testdatensatz stark. Der Wert der wichtigsten MAP-Metrik für die fünf Hauptdatensätze liegt zwischen 0,3 und 0,6, was einerseits eine Verbesserung für die meisten Tests im Vergleich zu früheren Ansätzen darstellt, andererseits aber auch nicht erlaubt, von einer vollständigen Automatisierung der Aufgabe zu sprechen.

Ein weiterer beliebter Ansatz zum Auffinden von Trace-Links ist die Verwendung von Techniken des maschinellen Lernens wie Random Forest oder logistische Regression, um Anforderungspaare und Anforderungs-Quellcode-Paare in gültig und ungültig zu klassifizieren. Diese Möglichkeit wird in den Artikeln "Automatic Traceability Maintenance via Machine Learning Classification" (Mills, Escobar-Avila, & Haiduc, 2018) und "A Machine Learning Approach for Determining the Validity of Traceability Links" (Mills & Haiduc, 2017) erforscht. Die Autoren schlagen vor, ein maschinelles Lernmodell auf Daten aus Softwareentwicklungsprojekten zu trainieren, das auf verschiedenen Ähnlichkeitsmetriken sowie auf Metriken zur Abfragequalität basiert. Das heißt, es wird im Wesentlichen ein Ensemble von IR-Modellen erstellt. Die Tests wurden auch mit Datensätzen aus der Coest-Bibliothek (COEST Community, 2017) durchgeführt. Wie bereits erwähnt, haben diese Datensätze nur eine geringe Anzahl positiver Beispiele, so dass die Trainingsdaten neu ausbalanciert werden müssen. Nichtsdestotrotz erreichen die maschinellen Lernmodelle etwa 0,75 auf dem F1-Maß, was ein hohes Ergebnis für eine solche Aufgabe ist. Es wurde vor allem durch die Feinabstimmung der Hyperparameter des Modells auf der Grundlage der verwendeten Datensätze erreicht. Die Arbeit zeigt die Notwendigkeit einer umfassenden Studie über den Einfluss und die Auswahl bestimmter Merkmale (*engl. Features*) für das Training eines maschinellen Lernmodells.

3 Entwicklung des Empfehlungssystems

In diesem Kapitel wird der Prozess der Entwicklung des Empfehlungssystems beschrieben, von der Erhebung der Systemanforderungen über den Entwurf der Anwendungsarchitektur bis hin zur Softwareimplementierung. Das Kapitel enthält Informationen über die Entwicklung der Sprachmodelle, die dem Empfehlungssystem zugrunde liegen, sowie über die Server-Software des Empfehlungssystems. Ein Empfehlungssystem durchläuft, wie jede andere Software auch, während der Entwicklung Standardphasen wie Analyse, Design, Implementierung, Test und Betrieb. In dieser Arbeit werden diese Phasen nacheinander beschrieben, obwohl die Softwareentwicklung in Wirklichkeit ein iterativer Prozess ist, bei dem in jedem neuen Schritt die Anforderungen an das System spezifiziert, Änderungen am Quellcode vorgenommen und der Betrieb des Systems getestet wird. Die aus den Tests gewonnenen Informationen werden analysiert, und dann wird der Entwicklungszyklus wiederholt, wobei die Software schrittweise auf den optimalen Zustand gebracht wird. Die Bewertung der Funktionsweise des Systems unter realen Bedingungen würde den Rahmen dieser Arbeit sprengen, da sie eine lange Beobachtung der Funktionsweise erfordert.

3.1 Ermittlung der Anforderungen an das Empfehlungssystem

Ausgehend von den Informationen in Kapitel 2.1 werden in diesem Abschnitt die grundlegenden Anforderungen an ein Empfehlungssystem erörtert. Die funktionalen Anforderungen, die an das System gestellt werden, ergeben sich aus den Anwendungsfällen, die im Folgenden aufgeführt sind:

1) Das System ist in der Lage, die Rückverfolgbarkeit zwischen zwei beliebigen Mengen von Anforderungen herzustellen.

Die Aufgabe, die Rückverfolgbarkeit herzustellen, reduziert sich darauf, eine Menge von Anforderungspaaren zu finden, von denen eines die Quelle des anderen ist. Die Rückverfolgbarkeit wird durch eine oder mehrere Spezifikationen zwischen zwei verschiedenen Domänen hergestellt. Beispiele für Domänen, die miteinander verbunden werden müssen, sind Kunden- und System-, System- und Software-, System- und Hardwareanforderungen. Um diese Aufgabe zu erfüllen, muss das Empfehlungssystem in der Lage sein, Artefakte, die zu der einen oder anderen Domäne gehören, anhand der Werte ihrer Attribute auszuwählen. Die Artefakte aus einer der Domänen bilden dann die Abfragemenge, während die Artefakte aus der anderen Domäne den zu durchsuchenden Textkorpus darstellen. Das Empfehlungssystem sollte für jede Abfrage eine Liste der wahrscheinlichsten Trace-Links mit einer Wahrscheinlichkeitsbewertung erstellen. Außerdem sollte das System anzeigen, ob eine Verbindung zwischen

Artefakten bereits in der Anforderungsdatenbank existiert oder nicht. Dieser Anwendungsfall wird im Folgenden als Basisfall betrachtet. Das Funktionieren des Systems in diesem Fall wird weiter als Basismodus oder Mapper-Modus bezeichnet. Abbildung 3.1 zeigt die erwarteten Eingaben und Ausgaben des Systems im Basisfall.

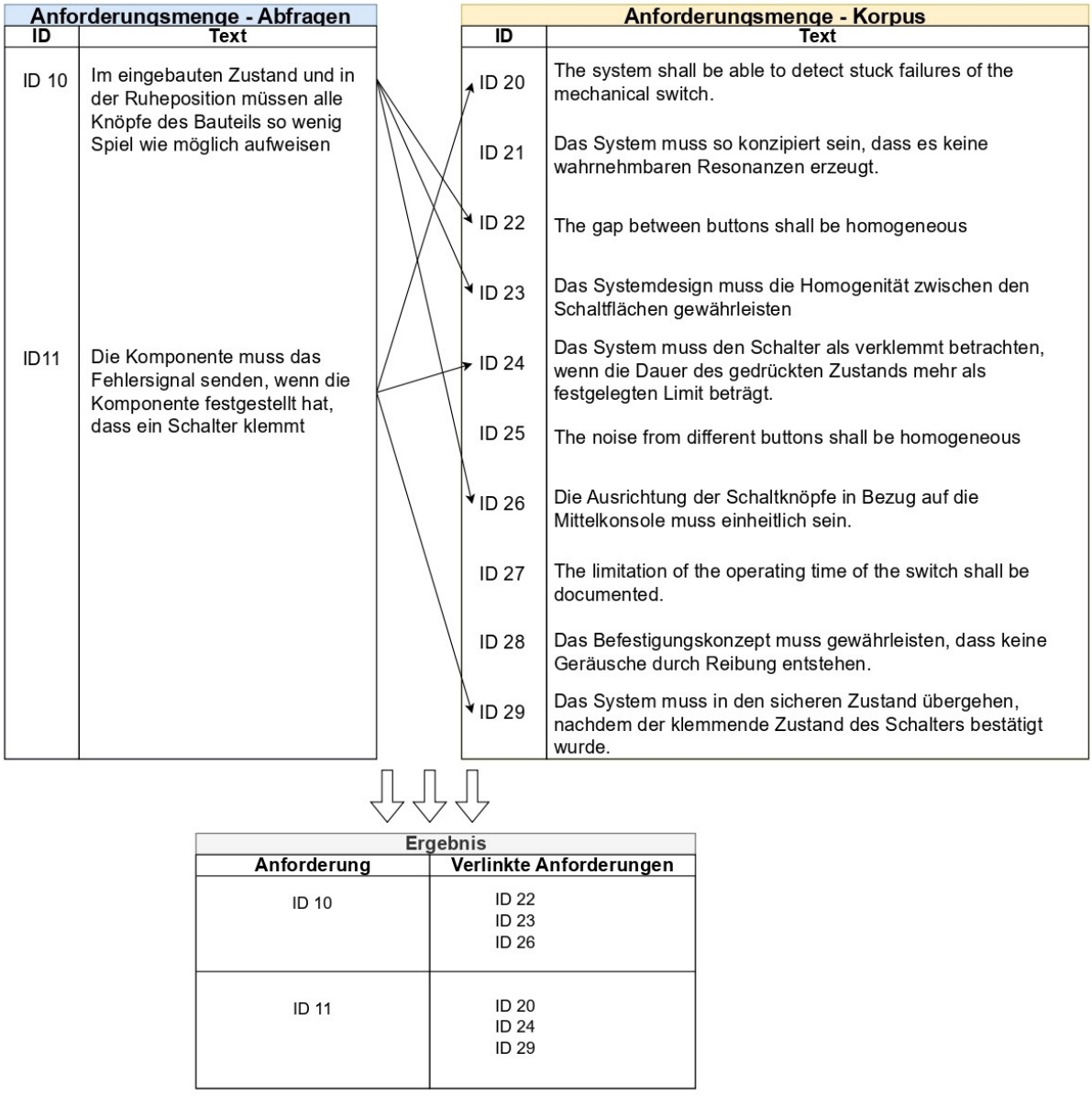

Abbildung 3-1 Beispiel für Eingabe- und Ausgabedaten im Basisfall

2) Das System ist in der Lage, in einer Menge von Anforderungen Verweise auf externe Standards, Normen und Spezifikationen zu finden und diese in der so genannten Projektspezifikationsliste zusammenzufassen.

Die Besonderheit von Anforderungen besteht darin, dass sie eine große Anzahl von Verweisen auf andere Anforderungen enthalten, die in bestehenden Normen, Standards und anderen externen Dokumenten festgelegt sind. Diese Anforderungen sind ebenfalls Teil des Projekts,

aber sie sind zunächst in impliziter Form in das Projekt eingebunden. Häufig kann eine Liste von Projektspezifikationen aus bereits bekannten Dokumenten zusammengestellt werden, die in der Branche gelten oder für einen bestimmten Kunden spezifisch sind. In diesem Fall können die Referenzen nach der Methode 1 (Basisfall) gesucht werden. Bestehen jedoch Zweifel an der Vollständigkeit oder Relevanz dieser Liste für ein bestimmtes Projekt, müssen zunächst die Verweise auf externe Dokumente ermittelt und dann zusammengefasst werden, um die Rückverfolgbarkeit zwischen Projektspezifikationen (Normen, Standards usw.) und Kundenanforderungen herzustellen. Dieser Anwendungsfall wird als Alternative weiter betrachtet. Das Funktionieren des Systems in diesem Fall wird weiter als Alternativmodus oder Extractor-Modus bezeichnet. Abbildung 3.2 veranschaulicht die erwarteten Ein- und Ausgabedaten bei der Arbeit im Alternativfall.

Anforderungsmenge - Korpus

ID	Text
ID 30	Der Laderegler muss DIN70121 für die Kommunikation auf hoher Ebene unterstützen.
ID 31	Der Laderegler muss eine Schnittstelle gemäß GB/T 18487.1-2015 implementieren.
ID 32	Der Laderegler muss den Industriestandard IEEE-519 für leitungsgebundene Emissionen erfüllen.
ID 33	The On-Board Charger shall support the ISO 15118 and DIN 70121 based Charging Station interface
ID 34	The On-Board Charger shall limit its current draw from the supply line for single phase equipment per GBT 18487.1-2015
ID 35	Das Design des Ladereglers muss ISO 26262 entsprechen

⇩⇩⇩

Ergebnis

Projektspezifikationen	Verlinkte Anforderungen
DIN70121	ID 30 ID 33
GB/T 18487.1-2015	ID 31 ID 34
IEEE-519	ID 32
ISO 26262	ID 35

Abbildung 3-2 Beispiel für Eingabe- und Ausgabedaten im Alternativfall

3) Das System ist in der Lage, im Test- und Produktionsmodus zu laufen.

Während der Entwicklungsphase sollte es möglich sein, nach jedem Durchlauf automatisch Offline-Evaluierungsmetriken unter verschiedenen Parametern zu berechnen, wie z. B. verwendete Sprachmodelle, Suchmethoden und andere Merkmale, die später in Abschnitt 3.4 ausführlich beschrieben werden. In diesem Fall erfolgt die Bewertung durch den Vergleich der Systemergebnisse mit bereits bekannten Metriken. Die Offline-Auswertung wird nur im Testmodus durchgeführt und im Produktionsmodus deaktiviert.

4) Das System kann sowohl im Online- als auch im Batch-Modus arbeiten.

In einer typischen Situation versucht ein Benutzer, Trace-Links für eine Reihe von Objekten zu finden. In diesem Fall stellt er mehrere Suchanfragen auf einmal - von 2 bis zu mehreren zehntausend. Da die Bearbeitung einer solchen Massenabfrage bis zu mehreren Minuten dauern kann, wird sie auf dem Server asynchron mit Hilfe eines Befehlszeilenskripts auf der Grundlage der Konfigurationsdatei ausgeführt, d. h. im Batch-Modus, der einen verzögerten Abruf der Ergebnisse voraussetzt. Stellt der Benutzer eine einzige Abfrage, d. h. versucht er, Quellen oder umgekehrt Ableitungen einer einzigen Anforderung zu finden, dann kann eine solche Abfrage durchaus im Online-Modus mit minimaler Verzögerung bei der Ausgabe der Ergebnisse ausgeführt werden. In diesem Fall ist es auch möglich, dem Benutzer eine grafische Schnittstelle zur Verfügung zu stellen.

5) Das System kann Daten aus verschiedenen Quellen laden.

Das System sollte die Arbeit mit einer Datenbank ermöglichen, in der Texte, Vektordarstellungen sowie andere Attribute von Anforderungen gespeichert werden (Stateful-Modus). Es sollte auch möglich sein, Daten aus separaten Dateien im Tabellenformat (CSV, Excel) zu laden (Stateless-Modus), und auch im JSON-Format, das für die Interaktion zwischen Webdiensten geeignet ist.

3.2 Entwurf der Architektur des Empfehlungssystems

Architektonisch besteht jedes Empfehlungssystem aus drei Schlüsselelementen: einem Collector, einem Ranker und einem Server. Der Collector führt die anfängliche Suche nach Informationen auf Anfrage des Benutzers durch und grenzt die Kandidaten für die Empfehlung ein. Der Ranker weist jedem Kandidaten eine Bewertung zu und sortiert sie danach. Der Server gibt dem Benutzer Empfehlungen nach einem vordefinierten Datenschema (Bischof & Yee, 2023).

Um nach Anforderungen zu suchen, die für die Anfrage relevant sind, sollten diese in der Datenbank gespeichert werden, die die Funktion der Volltextsuche (oder Stichwortsuche) und der

semantischen Suche auf der Grundlage von Vektordarstellungen implementiert. Diese Funktionalität impliziert die Indizierung von Text für die weitere Suche mit dem BM25-Algorithmus sowie die Fähigkeit, Vektoren für die semantische Suche effizient zu speichern und zu indizieren. Einige moderne relationale Datenbanken, z.B. PostgreSQL, haben im Prinzip ähnliche Fähigkeiten, wenn ein Plugin für die Arbeit mit Vektoren pgvector installiert ist (GitHub, 2022). Es gibt auch Systeme wie ElasticSearch, die über eine komfortable REST-API verfügen und in der Lage sind, Massenabfragen durchzuführen, verschiedene Arten von Suchen zu kombinieren und Objekte nach Attributwerten zu filtern (Elastic, 2015). Des Weiteren wurde aufgrund von lizenzrechtlichen Besonderheiten OpenSearch als Suchmaschine gewählt, die ein Ableger von ElasticSearch ist und daher über eine weitgehend ähnliche Funktionalität und API verfügt.

Die semantische Suche setzt ein Sprachmodell voraus, das in der Lage ist, Text in eine eindimensionale Vektordarstellung zu kodieren. BERT-basierte Sentence-Transformer-Modelle sind in diesem Fall die Standardlösung. Die Sentence-Transformer-Bibliothek bietet Zugang zu bereits trainierten Modellen, einschließlich mehrsprachiger Modelle, die für einen bestimmten Anwendungsfall durch Training auf Ihrem eigenen Datensatz angepasst werden können. In Anbetracht der relativ geringen Größe des Sentence-Transformer-Modells und der entsprechend geringeren Hardware-Anforderungen sowohl im Trainings- als auch im Evaluierungsmodus wird sie als die Grundtechnologie zur Erzeugung der Satzeinbettungen eingesetzt.

Der Volltextsuchalgorithmus von OpenSearch und die semantische Suche unter Verwendung von Vektordarstellungen, die vom Sentence-Transformer-Sprachmodell generiert werden, arbeiten zusammen, um einen Empfehlungssystem-Collector zu bilden. Der Ranking-Ansatz kann variieren und entweder die Verwendung eines separaten Cross-Encoder-Modells, das auf den Vergleich zweier Texte zugeschnitten ist, oder die Verwendung von Keyword- und/oder semantischen Suchergebnissen beinhalten. Im Folgenden werden verschiedene Ranking-Strategien für die betrachteten Anwendungsfälle untersucht und bewertet.

Schließlich gibt der Server die relevantesten Objekte mit einer eindeutigen ID, einem Text, Metadaten und einer Relevanzbewertung aus, die dem Nutzer als Anhaltspunkt für seine eigene fachliche Bewertung der Ergebnisse des Empfehlungssystems dienen sollen. Es soll möglich sein, die Ergebnisse des Empfehlungssystems in Form einer Excel-Tabelle, einer JSON-API, die von anderen Softwaretools verarbeitet werden kann, oder als Datenexport direkt in das Anforderungsmanagementsystem zu generieren. Die Integration des Empfehlungssystems mit anderen Unternehmenstools liegt außerhalb des Rahmens dieser Arbeit, so dass zum Zeitpunkt der Fertigstellung nur der einfachste Export in separate Dateien implementiert wurde.

Die entsprechende Architektur des Empfehlungssystems ist in Abbildung 3.3 dargestellt. Ein neues Element, das noch nicht diskutiert wurde, ist ein Modell zur Erkennung von Verweisen auf externe Dokumente wie NER (Named Entity Recognition). Wenn das Empfehlungssystem im Basisfall verwendet wird, extrahiert dieses Modell beim Abgleich zweier Anforderungssätze von jeder Entität in der Abfragemenge die Anzahl der darin erwähnten externen Dokumente und fügt sie der Abfragemenge hinzu. Das Ranking aggregiert dann die Ergebnisse der Haupt- und Unterabfragen. Mit dieser Technik soll die Effizienz des Systems verbessert werden, indem das Gewicht der Dokumentennummern unter allen anderen Wörtern im Anforderungstext erhöht wird.

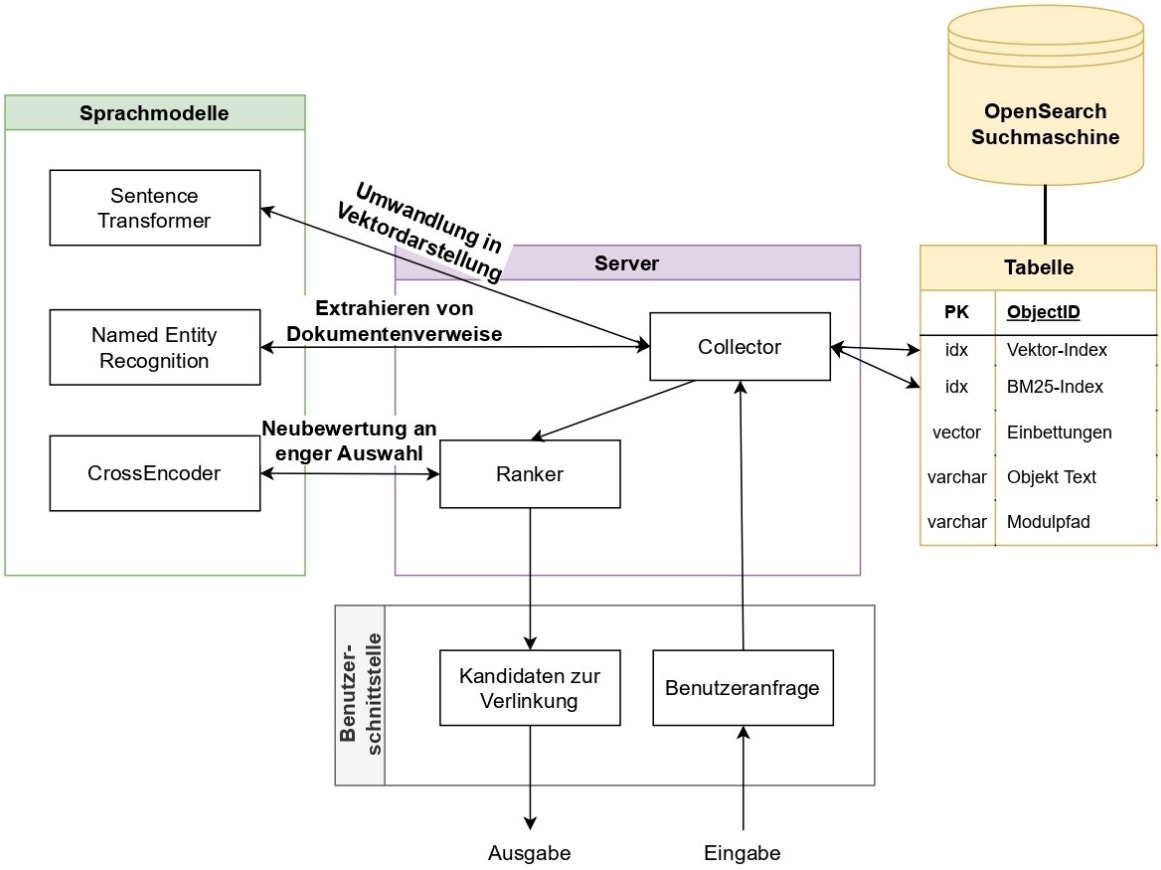

Abbildung 3-3 Architektur des Empfehlungssystems

Wenn das System in Modus 2 arbeitet (siehe Abschnitt 3.1), ist das NER-Modell das Hauptmodell. In diesem Fall existiert die Menge der Abfragen zunächst nicht und das NER-Modell bildet sie, indem es Dokumentennummern aus dem Text der Anforderungen extrahiert. Die Ergebnisse werden dann aggregiert, um die Anforderungen, die einen Verweis auf dieselbe Dokumentennummer enthalten, in einer einzigen Liste zusammenzufassen.

Wird der Textkorpus aus einer separaten Datei oder von einem anderen Webdienst geladen (siehe Abschnitt 3.1, Punkt 5), arbeitet das System im zustandlosen Modus (oder Stateless-

Modus), d. h. das Ergebnis seiner Arbeit hängt nur von den eingehenden Daten ab. In diesem Modus muss es „on-the-fly“ einen Index für die Stichwortsuche mit Hilfe der BM25-Methode erstellen und außerdem die Anforderungstexte in Vektordarstellungen kodieren, sowohl die Abfragen als auch die Menge, in der die Suche durchgeführt wird. Dieser Modus ist sehr praktisch, wenn die miteinander zu korrelierenden Mengen klein sind (nicht mehr als 10 000 Instanzen in jeder) und bereits im Voraus in einem anderen Dienst definiert wurden.

3.3 Entwicklung der Sprachmodelle

Wie aus dem vorangegangenen Abschnitt hervorgeht, besteht das Kernstück des Empfehlungssystems aus Sprachmodellen, die auf die Aufgaben der semantischen Informationsrückgewinnung, des Rerankings und der Erkennung von Referenzen in Dokumenten zugeschnitten sind und die auf proprietären Daten eines Unternehmens für Automobilelektronik trainiert werden müssen. Im Folgenden wird der Prozess der Datengenerierung und des Trainings von drei entsprechenden neuronalen Netzen auf diesen Daten dargestellt.

3.3.1 Datenerfassung

Die Anforderungsdaten werden zunächst in einem speziellen Anforderungsmanagementsystem in Form von so genannten Modulen gespeichert, die den vom Kunden gelieferten oder von der Organisation intern entwickelten Spezifikationen entsprechen. Ein Modul ist im System zunächst in Ordnern organisiert, die ähnlich wie in einem Dateisystem den Pfad zu ihm definieren. Der Pfad zu einem Modul besteht in der Regel aus einer Projektbezeichnung, einem Spezifikationstyp, ggf. einem oder mehreren Unterkategorien und einer Spezifikationsnummer. Die wichtigsten Spezifikationstypen sind: Kundenanforderungen, Systemanforderungen, Normen (Standards), Software, Hardware (Elektrotechnik) und Mechanik. Neben der ID und dem Text der Anforderung werden ihr auch verschiedene Attribute oder Metainformationen zugeordnet, die die Rückverfolgbarkeit der Anforderungen unterstützen (siehe Abschnitt 2.1.3). Der Benutzer des Anforderungsmanagementsystems hat auch die Möglichkeit, Verknüpfungen zwischen Artefakten innerhalb des Systems herzustellen.

Die Daten zu den Artefakten (Attribute und Referenzen) werden regelmäßig in einen Zwischenspeicher exportiert, wo die Anforderungen anhand verschiedener Metriken nach bestimmten Konfigurationen analysiert und weitere Berichte erstellt werden, um den Stand der Arbeit an den Projekten zu verfolgen. Speziell für diese Arbeit wurden Daten zu allen Anforderungen in 15 aktiven Projekten des Unternehmens exportiert. In Tabelle 3.1 sind die im Datensatz enthaltenen Artefakt-Attribute mit einer kurzen Beschreibung zusammengefasst.

Tabelle 3-1 Attribute des Datensatzes

Attributname	Attributbeschreibung
object_id	Identifikationsnummer der Anforderung
object_text	Text der Anforderung
module_path	Pfad zur Spezifikation im Anforderungsmanagementsystem im Format <Projekt>/<Kategorie>/<Unterkategorie 1>/.../<Unterkategorie N>/<Spezifikationsnummer>
responsible	Für die Anforderung verantwortlicher Bereich
inlink_direct	Array von Anforderungsbezeichnern, die direkt mit dieser Anforderung verknüpft sind
inlink_infinity	ein Array von Anforderungskennungen, die indirekt auf diese Anforderung verweisen
outlink_direct	Array der Anforderungskennungen, die von dieser Anforderung referenziert werden
outlink_infinity	ein Array von Anforderungskennungen, die indirekt von dieser Anforderung referenziert werden

Zusätzlich zu den einfachen String-Attributen enthält der Datensatz für jedes Artefakt auch Informationen darüber, mit welchen anderen Artefakten es verknüpft ist, sowohl direkt als auch durch Traversieren des Anforderungsgraphen in der Breite von jedem Artefakt aus.

In diesem Stadium liegen die Projektdaten in separaten CSV-Dateien vor und müssen zusammengeführt, analysiert, bereinigt und in eine für das Training neuronaler Netze geeignete Form gebracht werden.

3.3.2 Datenanalyse

Alle weiteren Arbeiten zur Datenvorverarbeitung und -analyse wurden mit dem Jupyter-Notebook-Tool durchgeführt, einer traditionellen Technologie für das maschinelle Lernen, die es ermöglicht, Code in der Programmiersprache Python interaktiv auszuführen und das Ergebnis der Berechnungen sofort im Format der so genannten Dataframes der Pandas-Bibliothek oder der von der Seaborn-Bibliothek generierten Grafiken zu sehen (Jupyter Team, 2022). Pandas ist eine spezielle Python-Bibliothek zur Manipulation von tabellarischen, d. h. zweidimensiona-

len Daten, die auf der Bibliothek für wissenschaftliche Berechnungen NumPy basiert. Mit Pandas lassen sich tabellarische Daten aus relationalen Datenbanken oder Dateien in verschiedenen Formaten in den Arbeitsspeicher laden, wobei die gängigsten Formate CSV (Comma Separated Values) und JSON (JavaScript Object Notation) sind. Die Daten können dann gefiltert, sortiert, aggregiert, kombiniert, aufgeteilt, transformiert und auf andere Weise verarbeitet und anschließend in Dateien oder Datenbanken exportiert werden (Pandas, 2020).

In der ersten Phase wurden Artefakte aus verschiedenen Projekten in einem Dataframe zusammengeführt. Die Gesamtzahl der Artefakte, die für die Untersuchung zur Verfügung standen, betrug 345185, und die Artefakte mit demselben ID, Text und Pfad zur Spezifikation wurden aus dem Dataframe ausgeschlossen. Die Anwesenheit solcher Objekte ist auf die technische Besonderheit des Tools zum Importieren von Daten aus dem Anforderungsmanagementsystem zurückzuführen. Anschließend wurde mit den Attributen „inlink_direct“, „inlink_infinity“, „outlink_direct“ und „outlink_infinity“ eine Tabelle erstellt, in der jede Zeile ein verknüpftes Paar mit seinen ID, Texten und der Art der Verknüpfung (direkt oder indirekt) darstellt. Doppelte Links sowie einige fehlerhafte Links, wie z. B. Self-Links, wurden ebenfalls ausgeschlossen. Insgesamt gibt es 267 717 direkte Links und etwa 1 Million (1 017 092 Zeilen) indirekte Links in der Link-Tabelle.

Ziel der Datenanalyse ist es, die Merkmale des verfügbaren Datensatzes zu ermitteln, die:
1) bei der Zusammenstellung von Trainings- und Testsätzen für das Training eines neuronalen Netzes helfen können
2) direkt im Suchalgorithmus verwendet werden können
3) die Interpretation der Ergebnisse des Empfehlungssystems verbessern.

Die im Datensatz dargestellten Anforderungen werden in mehrere Spezifikationskategorien eingeteilt. Die wichtigsten sind Kundenspezifikationen, Systemspezifikationen, Softwarespezifikationen, Hardwarespezifikationen, Spezifikationen für mechanische Anforderungen und Spezifikationen mit internen Standards. Abbildung 3.4 zeigt, wie sich die Anforderungen auf die genannten Kategorien verteilen. Mehr als 30 % der Anforderungen entfallen auf Kundenspezifikationen, wobei die Anzahl der Systemanforderungen mit etwa 25 % an zweiter Stelle steht, was möglicherweise darauf hinweist, dass einige Kundenanforderungen nicht akzeptiert wurden. Unter den Domänenspezifikationen übersteigt die Anzahl der Softwareanforderungen die der anderen um mehr als das Doppelte, was im Allgemeinen nicht überraschend ist. Die Dominanz der Software-Anforderungen wird noch deutlicher, wenn man die Verteilung aller Anforderungen nach zuständiger Domäne betrachtet, die in Abbildung 3.5 dargestellt ist. Sie zeigt, dass etwa 70 Prozent aller Anforderungen im Datensatz zu einem gewissen Grad softwarebezogen sind.

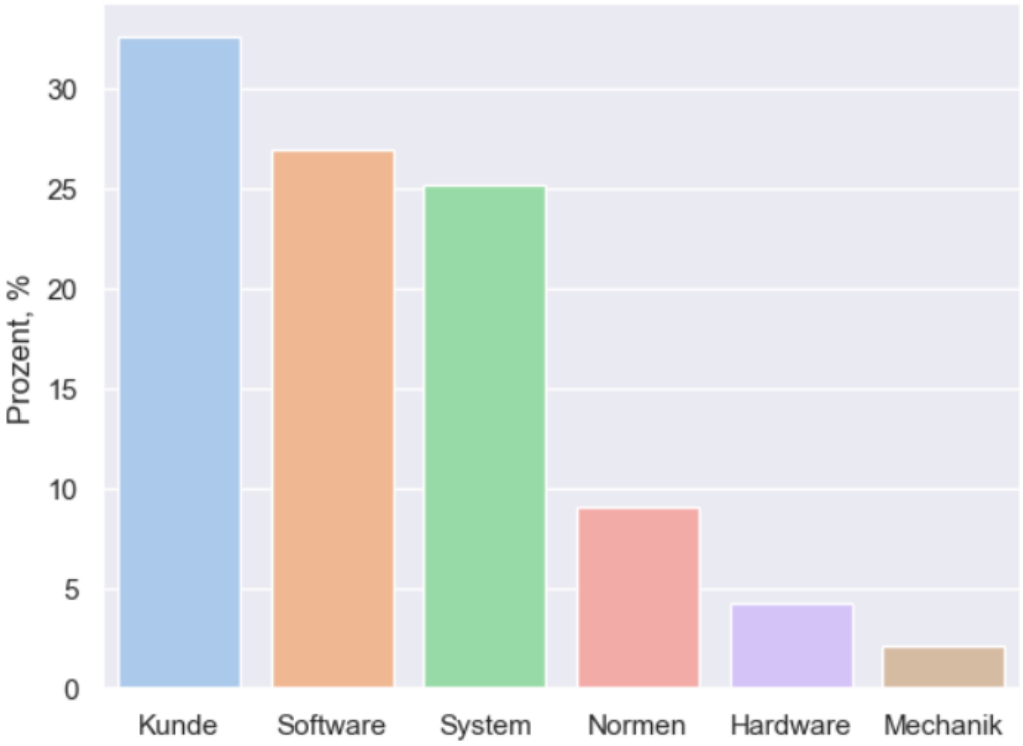

Abbildung 3-4 Aufteilung der Anforderungen im Datensatz auf die Spezifikationskategorien

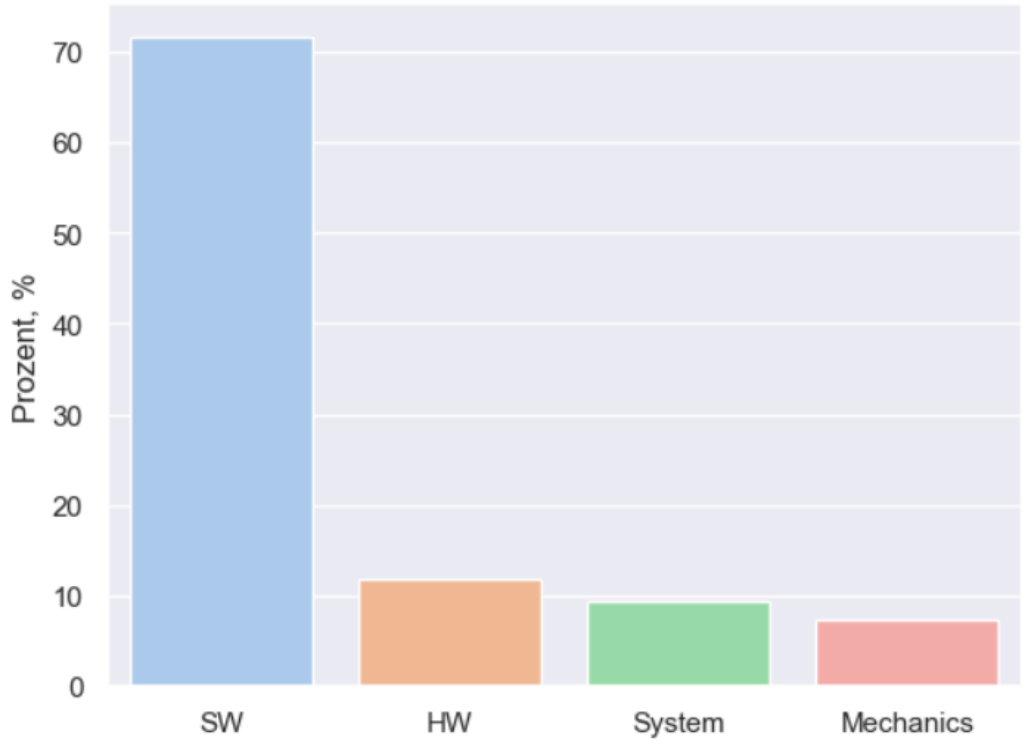

Abbildung 3-5 Aufteilung der Anforderungen im Datensatz auf die zuständigen Domänen

Der nächste wichtige Aspekt ist die Sprache, in der die Anforderungen verfasst sind. Da es im Datensatz kein Attribut gibt, das die Sprache des Artefakts angibt, wurde die Lingua-Bibliothek

(Stahl, 2022) verwendet, um die Sprache des Textes automatisch zu erkennen. Die Ergebnisse (siehe Abbildung 3.6) zeigen, dass 65 % aller Anforderungen auf Englisch und der Rest auf Deutsch verfasst sind. Die Sprache eines kleinen Teils der Anforderungen (etwa 1 %) konnte nicht automatisch bestimmt werden.

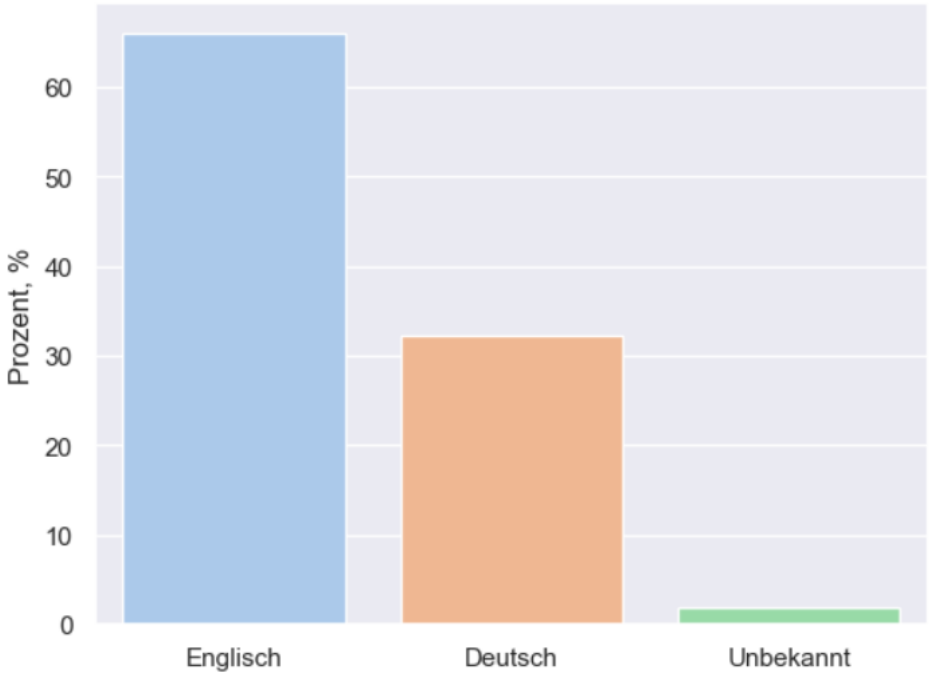

Abbildung 3-6 Aufteilung der Anforderungen nach Sprache

Betrachtet man die Verteilung der Sprache nach Spezifikationskategorien (siehe Abbildung 3.7) und nach verantwortlichen Domänen (Abb. 3.8), so lässt sich eine interessante Tendenz feststellen, dass bei System- und Domänenanforderungen die englische Sprache dominiert. Gleichzeitig spielt Deutsch bei den Kundenanforderungen eine bedeutende Rolle, und da die Kundenanforderungen der Ausgangspunkt für alle anderen sind, kann dieser Aspekt nicht vernachlässigt werden. Unter den verantwortlichen Domänen ist Deutsch am häufigsten bei den Systemanforderungen (ca. 30 Prozent) und am wenigsten häufig bei den mechanischen Anforderungen (ca. 18 Prozent) zu finden.

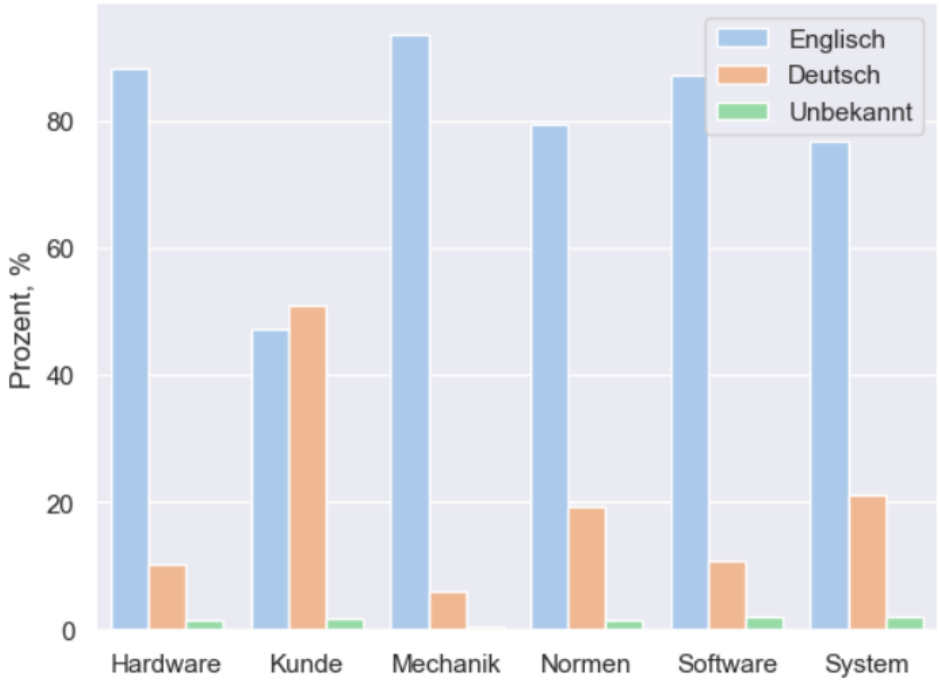

Abbildung 3-7 Aufteilung der Anforderungen nach Sprache und Spezifikationskategorien

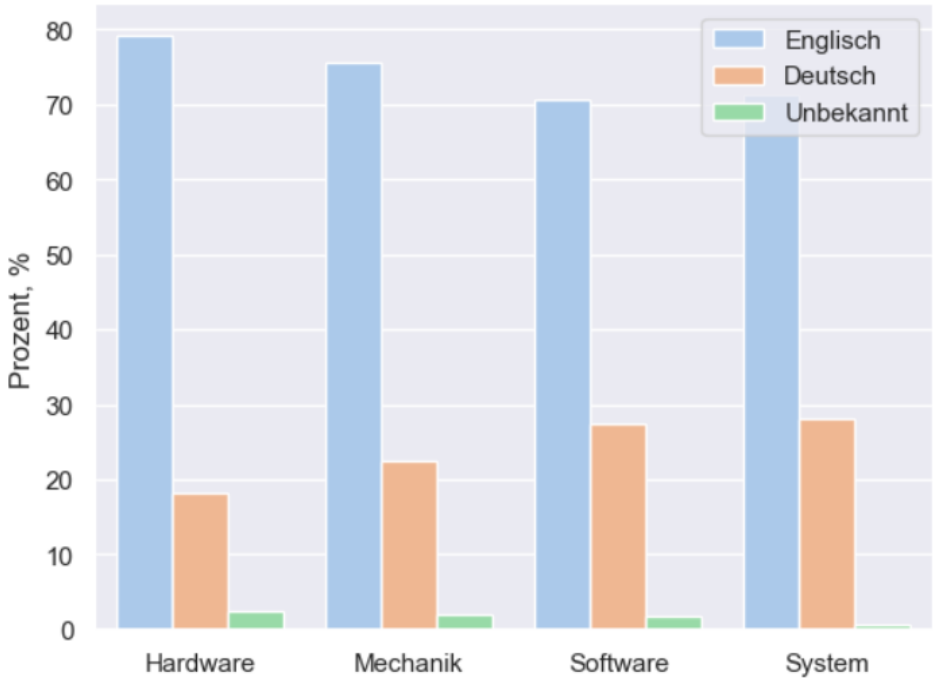

Abbildung 3-8 Aufteilung der Anforderungen nach Sprache und zuständigen Domänen

Die nächste Frage, die es zu klären gilt, ist, welche Arten von Anforderungen am häufigsten miteinander verbunden sind (siehe Abbildung 3.9). Es wird auch überprüft, ob es eine Diskrepanz gibt, wenn nur die Paare, in denen Anforderungen in verschiedenen Sprachen verfasst werden, betrachtet werden (siehe 3.10).

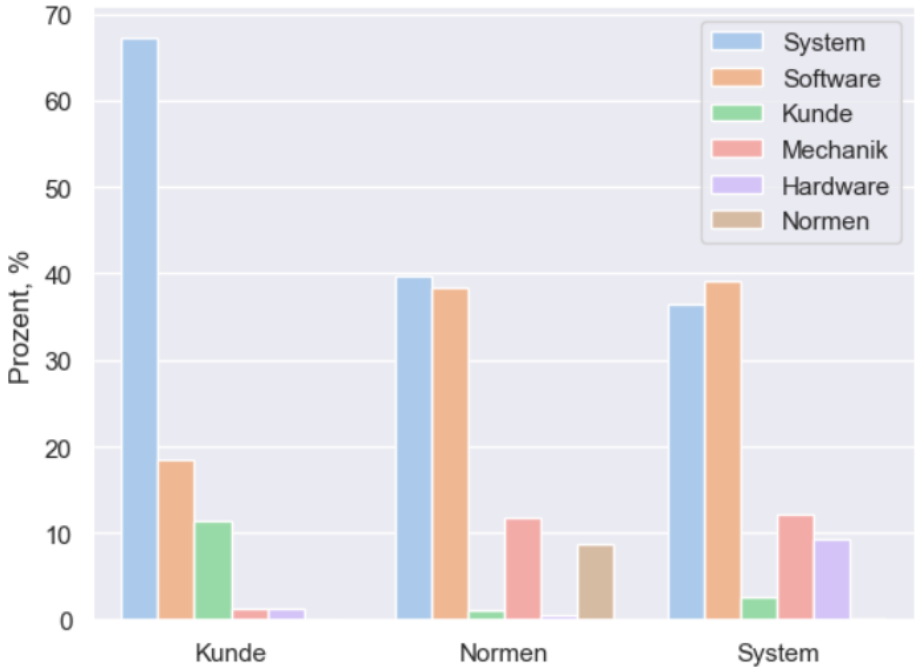

Abbildung 3-9 Aufteilung der verknüpften Paare nach Spezifikationskategorien der beteiligten Anforderungen

Aus den Abbildungen ist ersichtlich, dass Kundenanforderungen und Systemanforderungen erwartungsgemäß das häufigste Paar bilden. Einige Kundenanforderungen beziehen sich direkt auf die Domänenanforderungen und umgehen die Systemanforderungen. Es kommt auch häufig vor, dass sich Anforderungen in Systemspezifikationen gegenseitig referenzieren. Es überrascht nicht, dass die Systemanforderungen angesichts ihrer großen Anzahl am häufigsten einen Verweis von Softwareanforderungen erhalten. Es fällt auch auf, dass einige Systemanforderungen auf Kundenanforderungen verweisen, was eigentlich nicht korrekt ist und auf Verknüpfungsfehler im ursprünglichen Datensatz hindeutet. Aus Abbildung 3.10 kann geschlossen werden, dass Domänenanforderungen, insbesondere für Hardware und Mechanik, viel seltener an intersprachlichen Verknüpfungen beteiligt sind, d. h. solche Verknüpfungen sind auf einer höheren Anforderungsebene viel häufiger.

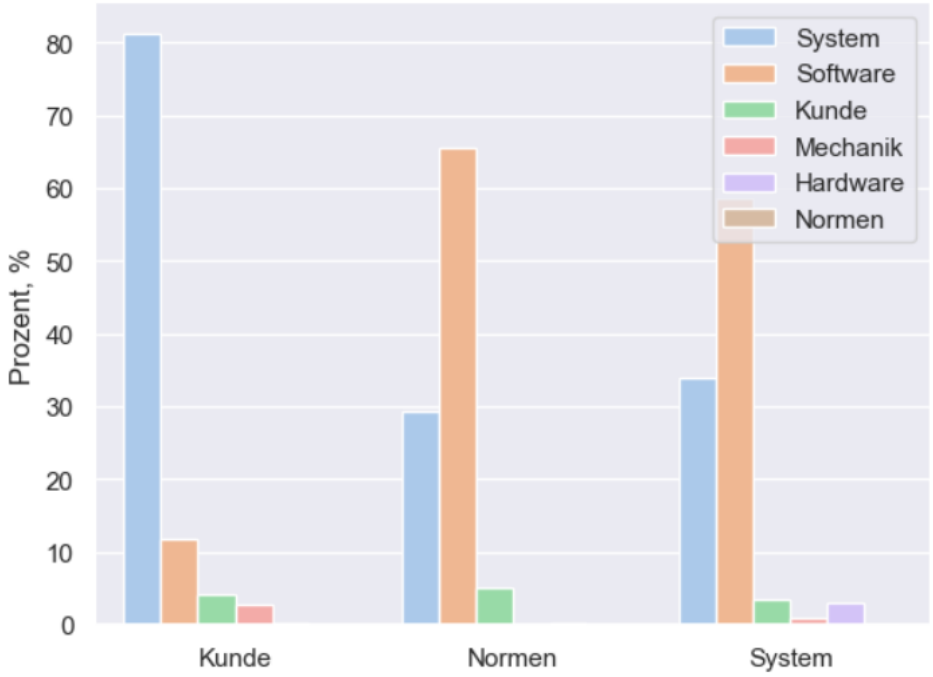

Abbildung 3-10 Aufteilung der verknüpften intersprachlichen Paare nach Spezifikationskategorien der beteiligten Anforderungen

Als Nächstes wird geprüft, ob die Anforderungen, die sich aufeinander beziehen, semantisch und lexikalisch ähnlich sind. Zu diesem Zweck wird ein vortrainiertes mehrsprachiges neuronales Netz vom Typ Sentence-Transformer "distiluse-base-multilingual-cased-v1" genommen (UKPLab, 2023). Damit werden Vektordarstellungen von Texten erzeugt und die Kosinus-Ähnlichkeit zwischen Paaren von Vektoren verwandter Anforderungen berechnet. Abbildung 3.11 zeigt ein Histogramm der Verteilung des Kosinus-Ähnlichkeitswertes über alle Anforderungen im Korpus.

In mehr als 28 % der Fälle weist ein Paar verwandter Anforderungen einen sehr hohen Ähnlichkeitskoeffizienten auf (mehr als 95 %), aber bei den übrigen Paaren liegt er sehr nahe an einer Normalverteilung. Es ist zu beachten, dass das Basismodell nicht auf dem betrachteten Datensatz trainiert wurde und eine Sicht auf die Daten aus der Perspektive eines Modells mit allgemeiner Gelehrsamkeit darstellt. Tabelle 3.2 zeigt Beispiele von Textpaaren, die einen bestimmten Grad an Ähnlichkeit aufweisen.

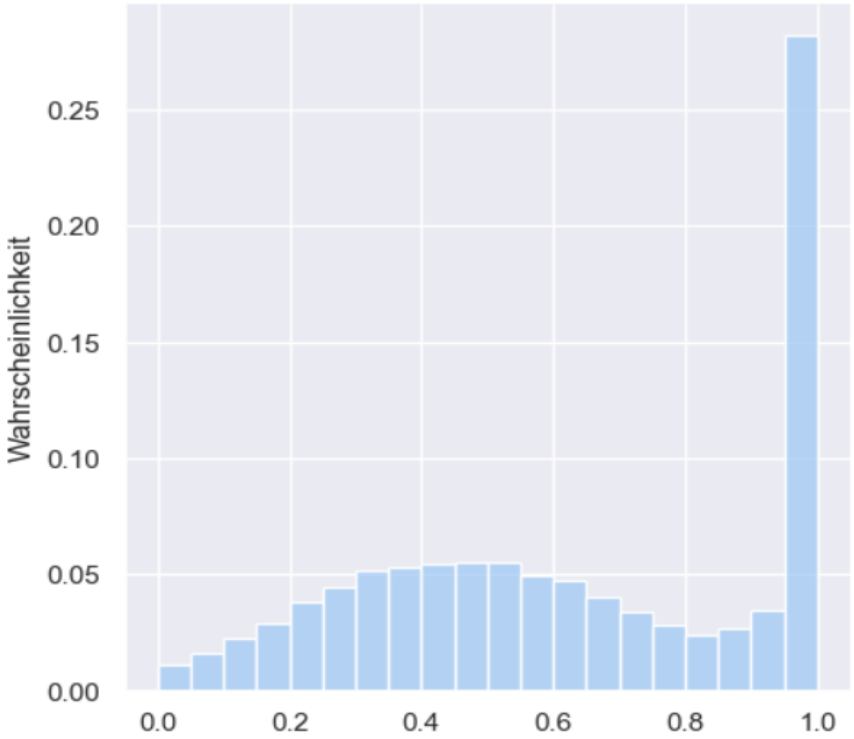

Abbildung 3-11 Verteilung der Kosinus-Ähnlichkeit zwischen verknüpften Anforderungen

Tabelle 3-2 Kosinus-Ähnlichkeit von einigen Textpaaren, berechnet vom vortrainierten Sprachmodell

Satz 1	Satz 2	Kosinus-Ähnlichkeit
In diesem Zustand sind im Modul keine Netzwerkparameter gespeichert.	While being in this state no network parameters shall be stored in the module.	0,94
Es muss die Innenraumtemperatur ermittelt werden	Es muss die Innenraumtemperatur über den Lüfter ermittelt werden	0,89
Im Fehlerfall muss für jeden Treiber ein Fehler abgelegt werden.	Im Fehlerfall muss für jeder Treiber einzeln ausgelesen werden.	0,7
Die Sensorkapseln sind an der Platine angeschlossen	Die Sensorkapseln werden über eine Steckverbindung kontaktiert	0,58

Tabelle 3-2 (Fortsetzung)

Satz 1	Satz 2	Kosinus-Ähnlichkeit
Alle Ein- und Ausgänge müssen gegen Überspannung geschützt werden.	Alle externen Endstufen sind kurzschlussfest gegen Stromversorgung und Masse.	0,45
Der Sensorwert muss durch einen eigensicheren Mechanismus entprellt werden.	Die Dauer der Entprellung muss minimiert werden.	0,33

Wenn die Kosinus-Ähnlichkeit kleiner als 0,8 ist, kann eine Person ohne fachliche Kenntnisse normalerweise nicht feststellen, ob Aussagen miteinander in Beziehung stehen oder nicht. Das resultierende Histogramm kann also darauf hindeuten, dass das zugrundeliegende Modell in den meisten Fällen zu schwach ist, um Ähnlichkeit zu erkennen. Das Ergebnis könnte darauf hindeuten, dass der Datensatz zahlreiche Fehler aufweist und viele Anforderungen in unangemessener Weise verknüpft sind. Es könnte auch der Fall sein, dass der Text einer Anforderung oft ohne Bezug zum Kontext des Dokuments steht und seine wahre Bedeutung nicht vermittelt.

Im Bereich der Verarbeitung natürlicher Sprache gibt es andere Arten von Modellen, die versuchen, den Zusammenhang von Texten zu erkennen. Ein solcher Typ sind Modelle, die anhand der Aufgabe "Rückschließen natürlicher Sprache" (*engl. Natural Language Inference, NLI)* trainiert wurden. Bei dieser Aufgabe wird ein Textpaar in drei Kategorien eingeteilt: "Implikation", "Neutralität" und "Widerspruch". Abbildung 3.12 zeigt das Ergebnis des Durchlaufs aller Paare des Datensatzes durch ein einfaches Modell vom Typ NLI. Das Ergebnis stimmt im Allgemeinen mit dem vorangegangenen Experiment überein. In mehr als der Hälfte der Fälle stellte das Modell fest, dass der eine Text den anderen nicht bedingt, und ging davon aus, dass die Bedeutung der Anforderungen in Bezug aufeinander neutral ist. In 13 % der Fälle stellt das Modell fest, dass die Texte eher im Widerspruch zueinander stehen.

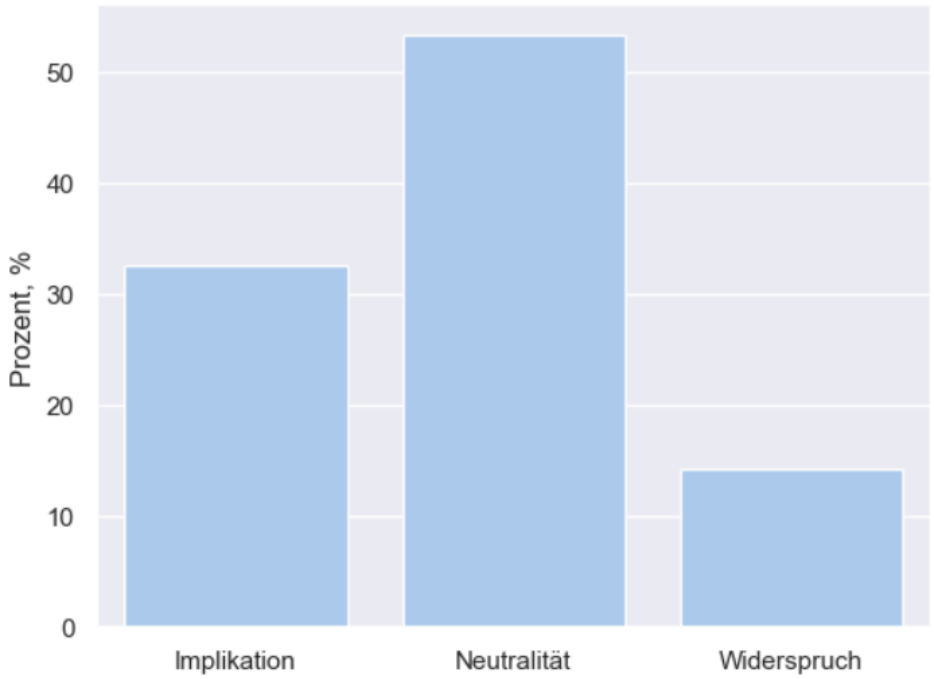

Abbildung 3-12 Aufteilung der Anforderungspaare nach dem Ergebnis vom NLI-Modell Lauf

Einer der Gründe, warum die vortrainierten Modelle bei dem betrachteten Datensatz fehlerhafte Ergebnisse liefern, kann die geringe Qualität der Texte sein. Unter geringer Qualität ist im Weiteren die Situation zu verstehen, wenn der Anforderungstext kein vollständiger Satz ist. Insbesondere wird geprüft, wie viele Anforderungstexte mindestens ein Substantiv und ein Verb enthalten. Dazu wird das POS-Tagger-Modell (siehe Abschnitt 2.2.2) aus der spacy-Bibliothek (ExplosionAI, 2015) eingesetzt. Die Ergebnisse dieser Prüfung zeigen, dass 35% der Anforderungstexte von schlechter Qualität sind. Abbildung 3.13 zeigt die Verteilung der Qualität nach Spezifikationstypen, und Abbildung 3.14 zeigt die Qualität von Anforderungspaaren, die nach dem Ergebnis vom NLI-Check aufgeteilt wurden. Ausgehend von Abbildung 3.13 weisen die Kundenanforderungen und die Systemanforderungen die schlechteste Qualität auf, während die mechanischen Anforderungen und die Hardwareanforderungen meist aus vollständigen Sätzen bestehen. Aus Abbildung 3.14 kann geschlossen werden, dass, wenn die zusammengehörigen Anforderungen eine unterschiedliche Qualität haben (eine ist schlecht und die andere gut), dies eher dazu führt, dass das Modell in die Irre geht und falsche Ergebnisse liefert. Allerdings wäre es auch falsch, das Modell aus Beispielen lernen zu lassen, bei denen eine der verwandten Anforderungen aus einem oder mehreren vollständigen Sätzen und die andere aus nur einem Wort besteht. Solche Daten von schlechter Qualität sind im Grunde genommen Ausreißer und müssen ausgeschlossen werden, da sonst die Interpretation der Ergebnisse des Modells schwierig wird.

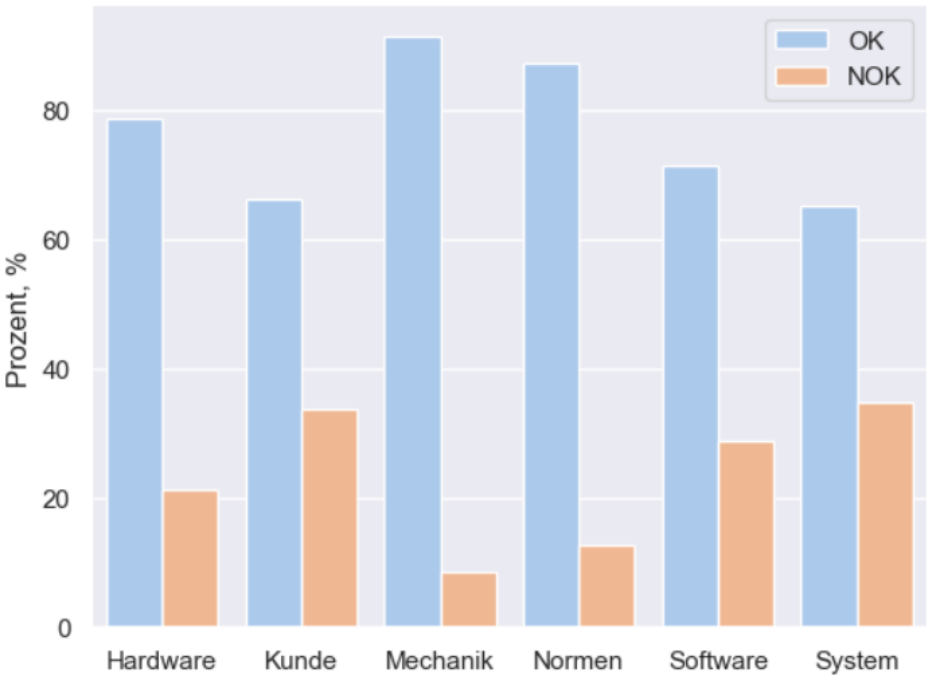

Abbildung 3-13 Aufteilung der Anforderungen nach Spezifikationskategorien und Textqualität

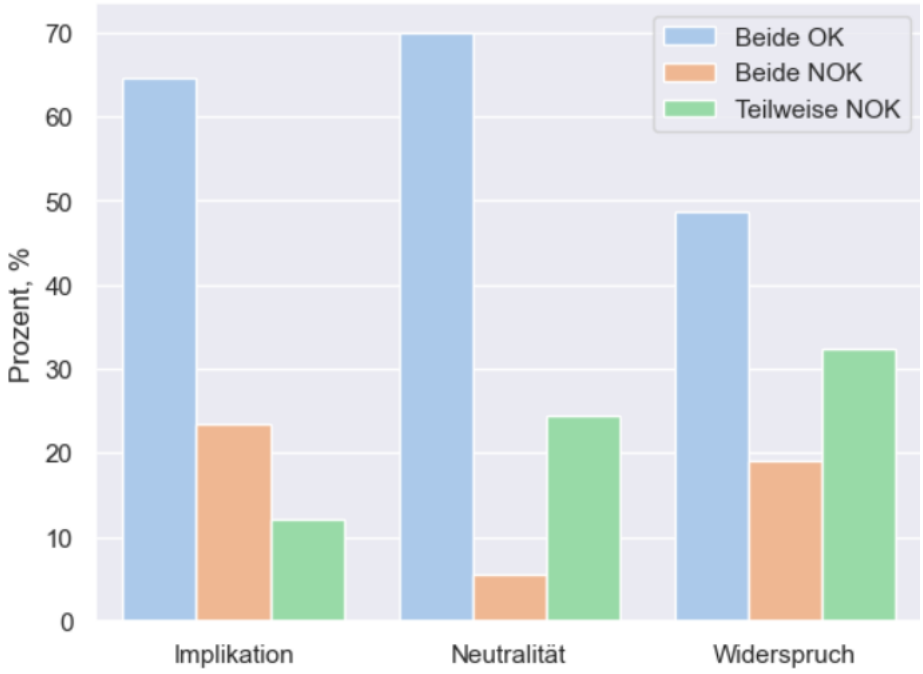

Abbildung 3-14 Aufteilung der Anforderungspaare nach Textqualität und Ergebnis von NLI-Lauf

Als Nächstes wird betrachtet, wie die Anzahl der Links, die jede Anforderung besitzt, verteilt ist. Abbildung 3.15 zeigt die Verteilung der Anzahl der eingehenden Links (zu höher Ebene) und Abbildung 3.16 zeigt die Verteilung der ausgehenden Links (von niedrieger Ebene).

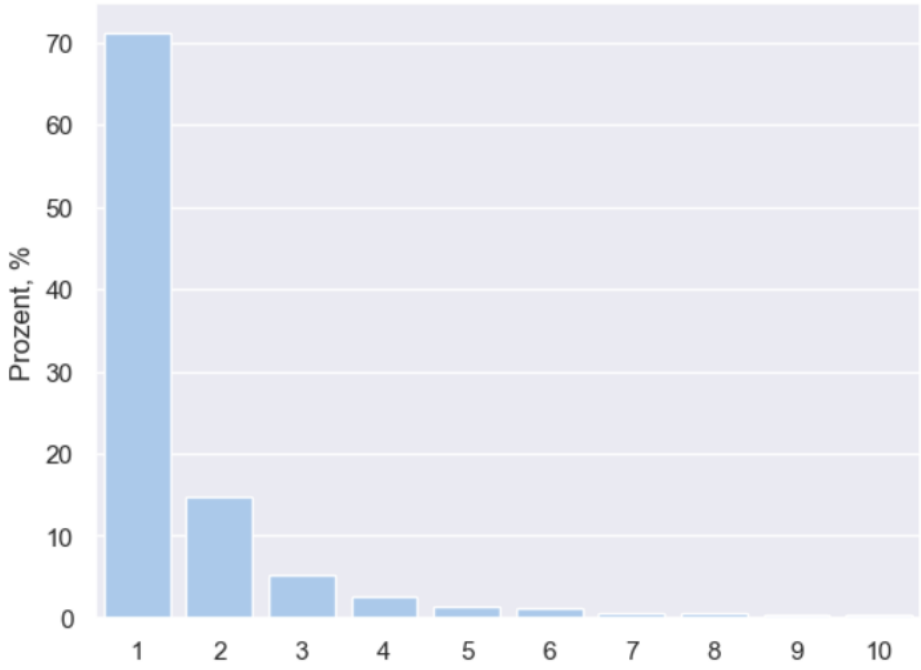

Abbildung 3-15 Verteilung von Anzahl der eingehenden Links

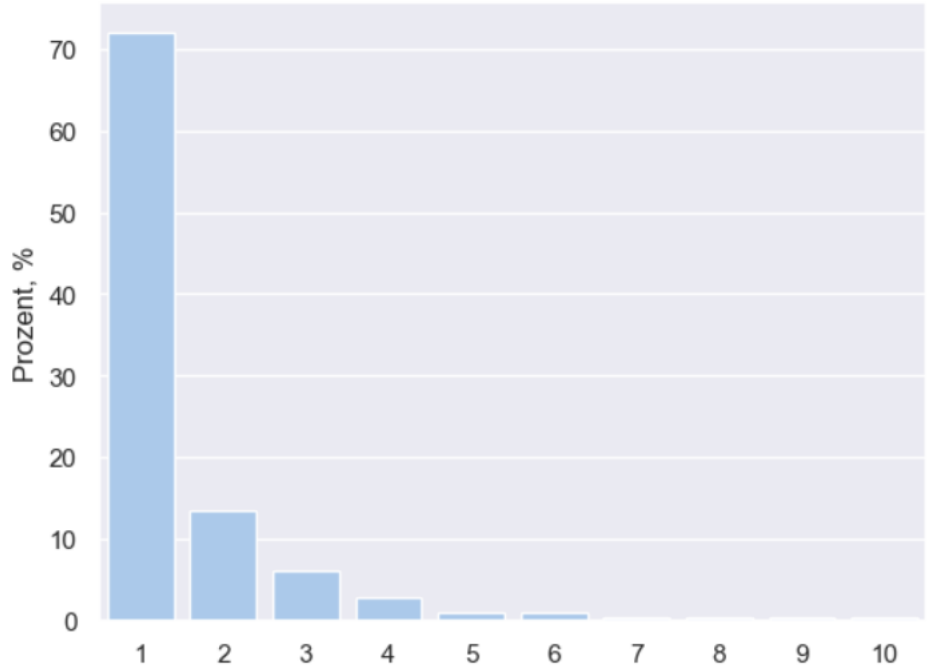

Abbildung 3-16 Verteilung der Anzahl der ausgehenden Links

Überraschenderweise sind die beiden Diagramme sehr ähnlich, obwohl man erwarten könnte, dass Anforderungen weniger ausgehende als eingehende Links haben. Insgesamt lässt sich feststellen, dass ungefähr 70 % der Anforderungen jeweils nur auf eine andere Anforderung verweisen. Dann nimmt die Zahl der Anforderungen mit mehreren Verweisen rasch ab, und es ist zu beachten, dass das Diagramm bei einem Wert von 10 abgeschnitten wird, obwohl es in Wirklichkeit einzelne Anforderungen gibt, die auf Hunderte von anderen verweisen. So gibt es

beispielsweise ein Objekt in der Datenbank, das auf 551 nachgeordnete Anforderungen verweist. Diese Situation führt dazu, dass einige Anforderungen im Datensatz überrepräsentiert sind. Daher sollte die Anzahl der Paare, die ein Objekt beinhalten, im Datensatz streng begrenzt werden.

Ein weiterer wichtiger Faktor ist die Verteilung der Anzahl von Wörtern und Sätzen in den Anforderungstexten. Mit dem Tokenizer der nltk-Bibliothek (Bird, Klein, & Loper, 2009) lassen sich die Texte leicht in Wörter und Sätze aufteilen, wobei Zeichensetzung und andere Merkmale berücksichtigt werden. Die Ergebnisse sind in den Abbildungen 3.17 und 3.18 - Boxplot der Verteilung der Wortanzahl für alle Objekte und unterteilt nach Spezifikationsarten - und 3.19 - Säulendiagramm der Verteilung nach der Anzahl der Sätze - dargestellt.

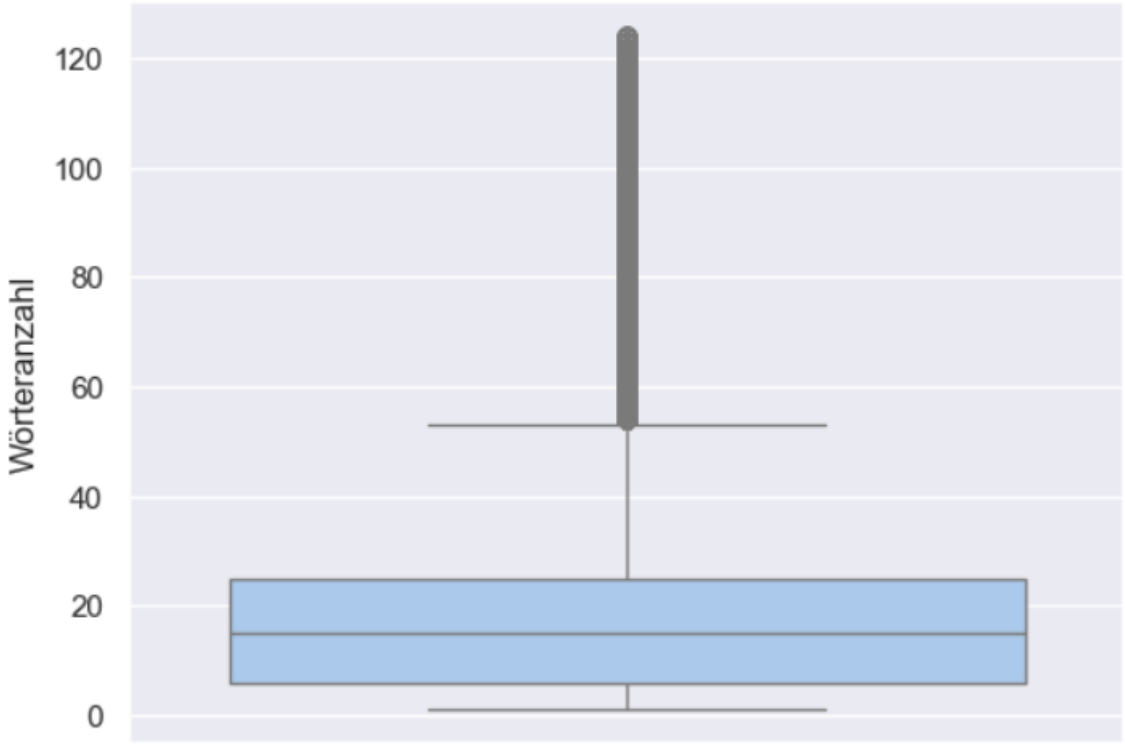

Abbildung 3-17 Verteilung der Wörteranzahl in Anforderungstexten

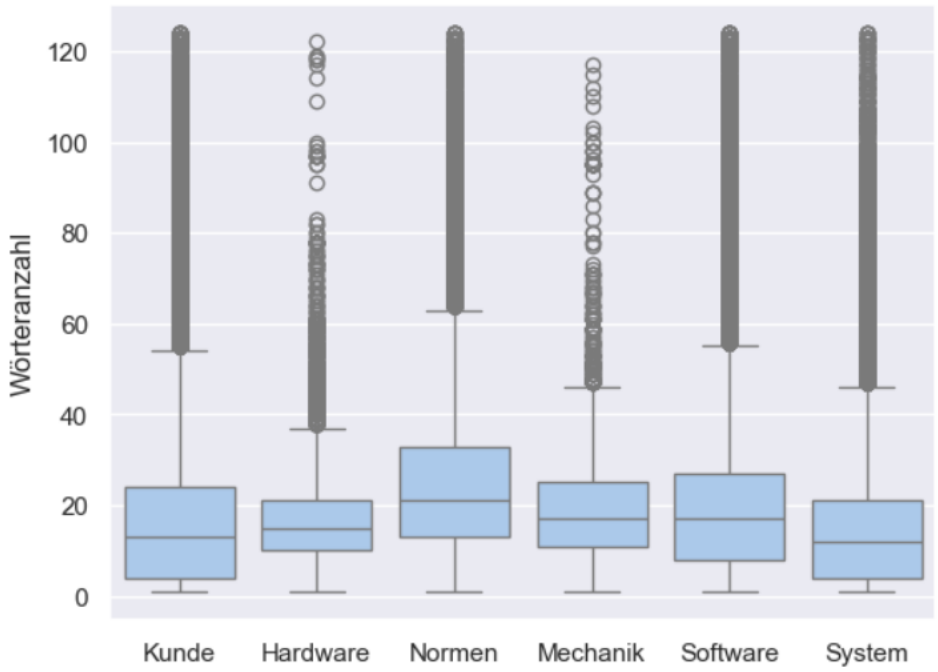

Abbildung 3-18 Verteilung der Wörteranzahl in Anforderungstexten nach Spezifikationskategorien

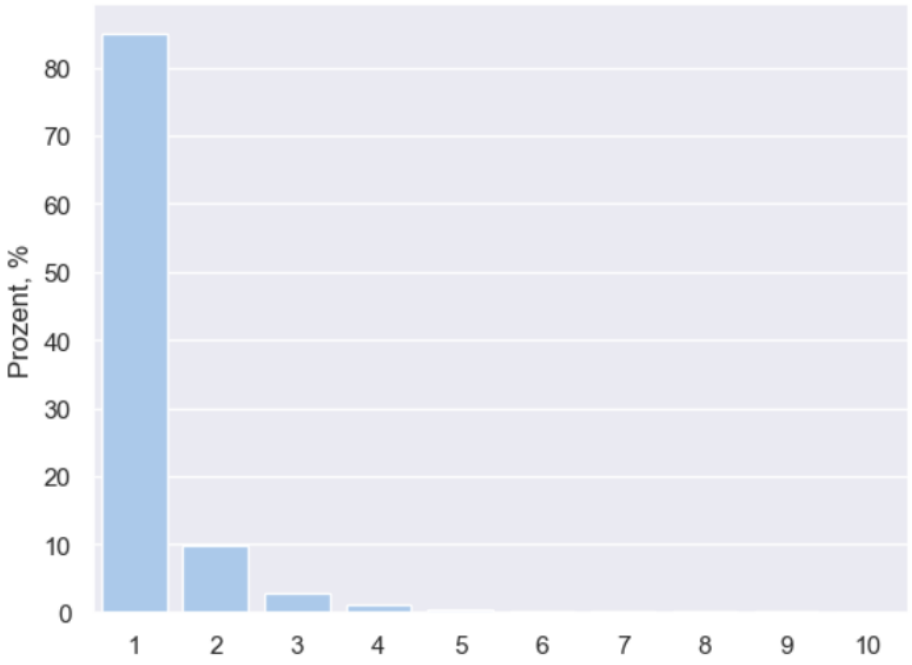

Abbildung 3-19 Verteilung der Anzahl der Sätze in Anforderungstexten

In 99% der Texte wurden weniger als 125 Wörter gefunden, deshalb wurde in den Diagrammen 3.17 und 3.18 dieser Wert als Obergrenze gewählt, da das Diagramm sonst unleserlich gewesen wäre. Im Durchschnitt enthält eine Anforderung im Median 18 Wörter, wobei 75 % der Anforderungen eine Länge zwischen 1 und 55 Wörtern haben. Die Mehrheit der Anforde-

rungen besteht aus nur einem Satz. In der Regel handelt es sich um kurze Texte, was vorteilhaft ist, da die neuronalen Netze, die in dieser Arbeit verwendet werden, nicht in der Lage sind, Texte mit mehr als 512 Token zu verarbeiten. Wenn es um den Vergleich von zwei Texten in Modellen geht, sollte die Gesamtlänge Ihrer Texte die angegebene Obergrenze nicht überschreiten. Dementsprechend sollten auch Texte mit mehr als 160 Wörtern aus der Trainingsmenge ausgeschlossen werden, wobei davon ausgegangen wird, dass ein Wort im Durchschnitt 1,6 Token entspricht (1,3 Token für Englisch und 2,1 Token für Deutsch).

Der letzte Aspekt, der in der Datenanalyse behandelt werden soll, ist die Rolle von Verweisen auf externe Dokumente (Standards, Normen, Spezifikationen) in den Anforderungstexten. Die Frage ist, wie häufig solche Entitäten in den Texten vorkommen und ob es sinnvoll ist, sie zu priorisieren. Um Verweise auf Dokumente für die Datenanalyse zu extrahieren, wird ein neuronales Netz „Gliner" vom Typ „Zeroshot“ eingesetzt, das in der Lage ist, eine Entität anhand ihres Kurznamens aus dem Text zu unterscheiden, ohne dass ein Training erforderlich ist. Unter Verwendung dieses Netzes sowie zusätzlich definierter regulärer Ausdrucksmuster wurde eine Beispielliste von Anforderungen erstellt, die in ihrem Text Verweise auf externe Dokumente enthalten.

Das Ergebnis zeigt, dass ungefähr 25 % der Anforderungen solche Verweise enthalten. Bei verwandten Anforderungen verweist in 30 % der Fälle mindestens ein Objekt des Paares auf ein externes Dokument und in 20 % der Fälle enthalten beide Texte einen Verweis auf dasselbe Dokument (siehe Abbildung 3.20).

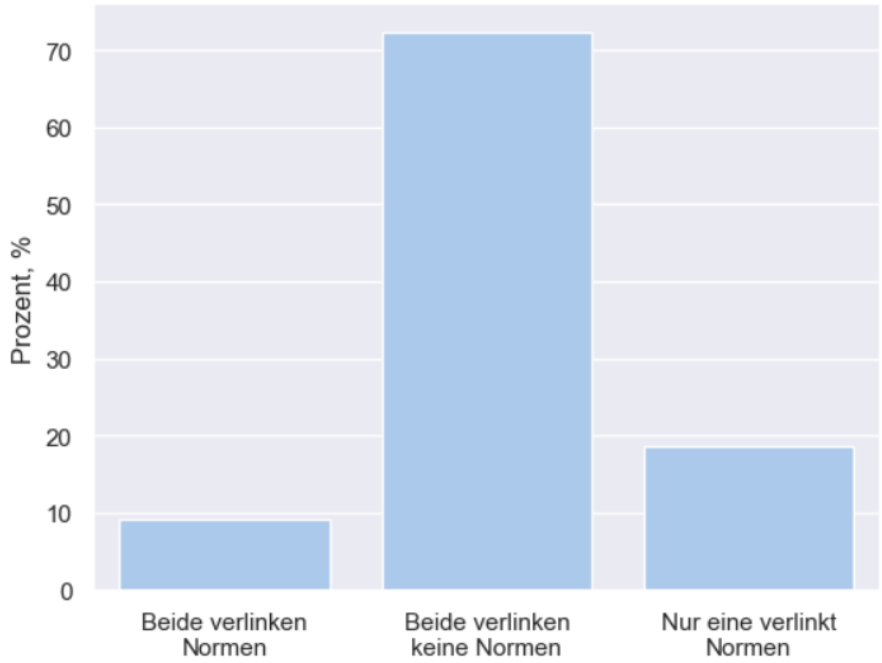

Abbildung 3-20 Vorkommen von Normenverweisen in den Texten der verknüpften Anforderungspaare

Wenn also zwei Anforderungen auf dasselbe externe Dokument verweisen, ist dies ein guter Indikator dafür, dass die Anforderungen miteinander in Beziehung stehen sollten. Als Nächstes wird in Abschnitt 3.3.5 das neuronale NER-Netz an den verfügbaren Datensatz angepasst und sowohl zur Verbesserung der Leistung des Empfehlungssystems im Suchmodus als auch direkt für Anwendungsfall 2 (siehe Abschnitt 3.1) verwendet, bei dem das Netz eine entscheidende Rolle spielt.

3.3.3 Training des Sprachmodells für Information Retrieval

Wie in den Abschnitten 2.2.4 und 2.2.5 dargestellt, werden Deep-Learning-Modelle auf Textkorpora mit allgemeinem Wissen wie Wikipedia und BookCorpus trainiert. Modelle für die semantische Suche, die dichte Vektordarstellungen von Text (Satzeinbettungen) erzeugen, bauen auf diesen auf und erfassen daher die Bedeutung von Texten außerhalb der Datendomäne, die beim Training in das Modell eingegeben wurden, nicht genau genug. Die Feinabstimmung des Sprachmodells, die später in diesem Abschnitt gezeigt wird, zielt darauf ab, es an die Domäne des betrachteten Anwendungsfalls anzupassen. Das heißt, das Modell sollte lernen, die Sprache von Entwicklern, die Spezifikationen für elektrotechnische Produkte im Automobilbereich schreiben, zu verstehen.

Die Methodik der Domänenanpassung umfasst mehrere Schritte. Im ersten Schritt wird unüberwacht mit der Maskenfüllmethode trainiert, mit der Modelle wie BERT ursprünglich trainiert wurden. Dieses Mal lernt das Modell jedoch mit den im vorherigen Abschnitt vorgestellten Spezialdaten. Der zweite Schritt besteht darin, das mehrsprachige Transformer-Modell in ein mehrsprachiges Sentence-Transformer-Modell, das Vektordarstellungen von Text als Ausgabe erzeugt, umzuwandeln. In diesem Fall wird das Training mit parallelen Daten durchgeführt, d. h. jeder englische Text entspricht einer deutschen Übersetzung. Das trainierte Modell versucht, solche Satzeinbettungen zu finden, die dem von einem anderen, bereits trainierten, aber einsprachigen Modell erzeugten Vektor so nahe wie möglich kommen, d. h. einen minimalen mittleren quadratischen Fehler (engl. mean squared error, MSE) aufweisen. Die dritte Stufe ist die eigentliche Feinabstimmung des mehrsprachigen Modells anhand von Daten über Textpaare, die miteinander verlinkt werden.

3.3.3.1 Maskierte Sprachmodellierung

Beim Maskierungstraining werden keine Etiketten benötigt, d. h. das Modell lernt ausschließlich aus dem Inhalt der Texte selbst und errät, welches Token sich hinter der Maske verbirgt. Das Training wird mit der Transformers-Bibliothek (Hugging Face, 2021) durchgeführt, einem Add-on zum beliebten Deep-Learning-Framework PyTorch (The PyTorch Foundation, 2018).

Das mehrsprachige XLM-RoBERTa-Modell, eine verbesserte Version des BERT-Modells, wird als Grundlage für das Training gewählt.

Der Prozess der Datenaufbereitung und des Trainings besteht aus den folgenden Schritten:

1) Auswahl von Daten aus dem Textkorpus, die das Qualitätskriterium erfüllen: Der Text muss mindestens ein Substantiv und ein Verb enthalten.
2) Tokenisierung des Textkorpus
3) Ersetzen von 15% Tokenisierung durch Masken unter Verwendung der Klasse DataCollatorForLanguageModeling
4) Aufteilung des Korpus in Trainingsmenge (80%) und Testmenge (20%)
5) Aufteilung in Mini-Batches
6) Training mit der Trainer-API der Transformers-Bibliothek für drei Epochen.

Das vollständige Skript für das Training ist in Anhang A.1 zu finden. Um zu beurteilen, inwieweit das Training zu einer Domänenanpassung geführt hat, wird die Perplexität-Metrik verwendet, die angibt, wie sehr das Modell "überrascht" ist, dieses oder jenes Token hinter der Maske zu sehen. Ein niedriger Perplexitätswert weist auf eine bessere Qualität des Modells hin (Tunstall, von Werra, & Wolf, 2022, S. 333). Speziell für den Testfall beträgt die Perplexität des XLM-RoBERTa-Basismodells etwa 12, nach der Anpassung sank der Wert auf 4. Tabelle 3.3 zeigt einige Sätze aus dem Testsatz, das maskierte Wort und die wahrscheinlichsten Kandidaten, die vom Basismodell und vom angepassten Modell vorhergesagt wurden.

Tabelle 3-3 Vergleich zwischen dem vortranierten und dem angepassten Modell bei der MLM-Aufgabe

Echter Satz	Maskiertes Wort	Basismodellvorschläge	Vorschläge vom angepassten Modell
Any error shall be sent to main controller maximum time of 300 ms between two messages	controller	"with", "server", "client", "message", "provider", "at", "command", "on"	"controller", "with", "function", "client", "interface", "on", "component", "system"
Nach dem Neustart muss die Anlage wieder in ihren spezifizierten Betriebszustand gehen.	Neustart	'Unfall', 'Bau', 'Krieg', 'Verkauf', 'Einsatz', 'Feuer', 'Winter', 'Kauf', 'Arbeiten', 'Jahr'	'Test', 'Betrieb', 'Start', 'Wechsel', 'Ablauf', 'Laden', 'Release', 'Austausch', 'Update', 'Service'

Anhand der vorgestellten Beispiele wird deutlich, dass das feinabgestimmte Modell den Kontext des Wortes jedes Mal besser versteht und eine viel genauere Vorhersage trifft.

3.3.3.2 Training des mehrsprachigen Sentence-Transformer-Modells

In dem untersuchten Datensatz sind etwa ein Drittel aller Anforderungen auf Deutsch, so dass das Basismodell für weitere Experimente beide Sprachen gleichermaßen gut verstehen sollte. Daher besteht der nächste Trainingsschritt darin, ein mehrsprachiges Modell zu trainieren, das Satzeinbettungen erzeugt. Wie bereits in Abschnitt 3.2.3 erwähnt, wird das Training von einem Lehrermodell überwacht, das nur mit englischen Texten arbeitet, und zwar nach der Methode, die in dem Artikel "Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation" beschrieben wurde (Reimers & Gurevich, 2020). Die Essenz des Trainingsprozesses ist in Abbildung 3.21 dargestellt.

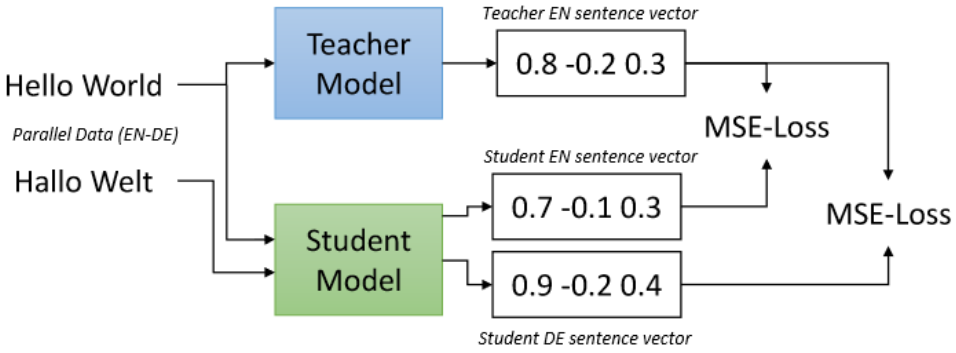

Abbildung 3-21 Umwandlung in ein mehrsprachiges Modell. Erschienen in: Reimers & Gurevych, 2020, S. 2

Das Modell paraphrase-distilroberta-base-v2 (UKPLab, 2023) wurde als Lehrermodell und das im vorherigen Schritt angepasste XLM-RoBERTa-Modell als Schülermodell verwendet. Die parallelen Daten stammten aus zwei unterschiedlichen Quellen. Zum einen wurden die Übersetzungen einiger ursprünglich auf Deutsch verfasster Anforderungen verwendet, die bereits in der Datenbank des Unternehmens vorhanden waren. Da jedoch nur etwa 35 000 solcher Instanzen gefunden wurden, was für ein qualitativ hochwertiges Modell nicht ausreicht, wurden weitere 80 000 Instanzen aus dem TED2020-en-de-Korpus hinzugefügt. Die Art der Daten in diesem Satz ist generisch, was für das Training eines mehrsprachigen Modells akzeptabel ist, da der Schwerpunkt genau auf der korrekten Übersetzung von einer Sprache in eine andere liegt und nicht auf der Anpassung an die Domäne.

Zum Trainieren des Modells wurde die Sentence-Transfomers-Bibliothek anstelle der üblichen Transformers-Bibliothek mit der Trainer-API verwendet, da sie bereits alle notwendigen Klassen und Methoden zum Trainieren mehrsprachiger Modelle enthält. Das Skript für das Training eines mehrsprachigen Modells ist in Anhang A.2 vollständig wiedergegeben und besteht aus folgenden Schritten:

1) Initialisierung des Lehrermodells und des Schülermodells
2) Hinzufügen einer Pooling-Schicht zum Schülermodell
3) Initialisierung des Klassenobjekts ParallelSentencesDataset durch beide Modelle und Laden der parallelen Daten in dieses Objekt. Diese Klasse ist verantwortlich für die Erstellung von Minibatches aus den von Lehrer und Schüler generierten Vektordarstellungen.
4) Initialisierung der Verlustfunktion MSE (mittlerer quadratischer Fehler)
5) Initialisierung der Evaluierungsobjekte mit Testdatensätzen. In diesem Fall wurden zwei Evaluatoren angewandt: Der eine basiert auf dem Testdatensatz des TED2020-en-de-Korpus und misst den MSE zwischen Vektordarstellungen desselben Textes in Englisch und Deutsch. Der andere Evaluator nutzt den klassischen STS-Datensatz mit sowohl ähnlichen als auch unterschiedlichen Satzpaaren und ihrer Kosinus-Ähnlichkeit. Der Evaluator zeigt, wie sehr der vom trainierten Modell erzeugte Kosinus-Ähnlichkeit mit den goldenen Etiketten korreliert.
6) Das Modell wird für 5 Epochen zum Training ausgeführt.

Infolgedessen erreichte die Korrelation der Kosinus-Ähnlichkeit nach Spearman während des Trainingsprozesses 80 % und der MSE-Fehler sank von 52 auf 11.

3.3.3.3 Feinabstimmung des Sentence-Transformer-Modells

Die Feinabstimmung von Vektordarstellungen (Satzeinbettungen) wurde auch mit den Klassen und Methoden der Sentence-Transformers-Bibliothek durchgeführt. Insbesondere bietet sie eine umfangreiche Liste von Verlustfunktionen für das Modelltraining. Es sind die Verlustfunktionen, die bestimmen, welche Art von Daten während des Trainings verfügbar sein sollten. Es gibt drei Hauptoptionen, die dem Lernalgorithmus als Eingabe dienen:

1) Eine Liste von positiven Textpaaren, also Textpaaren, die in Beziehung zueinander stehen und deren Ähnlichkeit gesteigert werden soll.
2) Eine Liste von Triplets, d. h. ein Ankertext, dem ein positives und ein negatives Beispiel zugeordnet sind.
3) Eine Liste von Paaren und der Wert des Ähnlichkeitsmerkmals (in der Regel eine Kosinus-Ähnlichkeit) von 0 bis 1, auf den der Optimierungslernalgorithmus abzielt.

Die erste Option ist die einfachste, da die Paare verwandter Texte bereits im Datensatz vorhanden sind und daher nur wenige oder keine zusätzlichen Schritte vor dem Training erforderlich sind. Die zweite Option erfordert die Auswahl negativer Paare, und solche Paare sind auf diese Weise standardmäßig schwer zu klassifizieren. Bei der dritten Option wird der Datensatz ebenfalls umgewandelt, es werden sowohl positive als auch negative Paare ausgewählt und der goldene Ähnlichkeitswert bestimmt, den der Lernalgorithmus anstrebt.

Für die erste und zweite Variante kann die Verlustfunktion "Multiple negatives ranking" als Verlustfunktion verwendet werden. Wenn es nur positive Paare gibt, wird die Liste der negativen Paare aus den verbleibenden Beispielen im Mini-Batch gebildet. Nach jeder Epoche werden die Beispiele in den Batches gemischt, daher ist es in der ersten Variante erforderlich, das neuronale Netz so viele Epochen wie möglich zu trainieren. Darüber hinaus sollte die Batchgröße so groß wie möglich sein, was einen Grafikkartenspeicher mit maximaler Kapazität erfordert. Beim Training der dritten Variante wird die Funktion CosineSimilarityLoss angewendet, die die Standardabweichung zwischen der Kosinus-Ähnlichkeit und dem Etikett-Wert minimiert.

Die Trainingsskripte sind für alle drei Optionen ähnlich und unterscheiden sich nur in der Art der Daten und der Verlustfunktion (siehe Anhang A.3) und bestehen aus den folgenden Schritten:

1) Initialisierung des im vorherigen Schritt trainierten Basismodells.
2) Aufteilung der Daten in einen Trainings- und einen Validierungssatz
3) Laden der eingegebenen Trainingsdaten in DataLoader, eine Standardklasse des PyTorch-Frameworks, die den Datensatz in Mini-Batches aufteilt.
4) Initialisierung der Verlustfunktion
5) Initialisierung des Schätzers mit dem Validierungsdatensatz
6) Training des Modells für 6 Epochen

Der Ansatz zum Testen von Modellen für die semantische Suche besteht darin, den ursprünglichen Anforderungskorpus in zwei Teile zu unterteilen - Training und Test. Die Trainingsmenge wird dann auf die Art von Daten reduziert, die für eine bestimmte Art von Training erforderlich sind. Vor dem Training werden 20 % des Trainingssatzes aus dem Trainingssatz in einen Validierungssatz extrahiert, der dazu dient, den Fortschritt des Trainingsprozesses zu verfolgen und gegebenenfalls Hyperparameter wie die Lernrate oder die Mini-Batch-Größe zu verfeinern. Der Testsatz ermöglicht einen unvoreingenommenen Vergleich der Modelle. Als Evaluator wird eine spezielle Klasse - InformationRetrievalEvaluator - aus der Sentence-Transformers-Bibliothek verwendet, die eine semantische Suche mit Anfragen, die den Anforderungen aus der Testmenge entsprechen, gegen das gesamte Korpus durchführt und dann mit bekannten relevanten Antworten auf jede Anfrage vergleicht. Die Bewertung erfolgt anhand der bereits aus Abschnitt 2.2.5.3 bekannten Metriken - Genauigkeit, Sensitivität, F1-Maß und MAP.

Die Ergebnisse für jedes der trainierten Modelle sind in Tabelle 3.4 zusammengefasst, wobei zu beachten ist, dass die absoluten Werte der Metriken nicht signifikant sind und nur im Vergleich zueinander von Interesse sind.

Tabelle 3-4 Ergebnisse der Feinabstimmung vom Sentence-Transformer-Modell

Variante	Genauigkeit@5	Recall@5	F1-maß@5	MAP @100
Basismodel	0,069	0,257	0,109	0,192
Feinabgestimmt auf MLM-Aufgabe	0,07	0,261	0,11	0,201
Feinabgestimmt nur mit positiven Paaren	0,073	0,264	0,114	0,202
Feinabgestimmt mit Tripletten	0,089	0,322	0,139	0,249
Feinabgestimmt mit der auf Kosinus-Ähnlichkeit basierten Verlustfunktion	0,072	0,272	0,114	0,216

Die niedrigen Werte der Metriken sind auf die Tatsache zurückzuführen, dass das Modell im gesamten Korpus, der aus verschiedenen Projekten besteht, nach relevanten Anforderungen suchen muss, wobei Paare relevanter Dokumente im Benchmark-Datensatz immer zum selben Projekt gehören. In dieser Situation sind hohe Werte der Metriken nicht zu erwarten. Im Abschnitt 4 wird die Beurteilung der Leistung des Empfehlungssystems innerhalb einzelner Projekte vorgestellt. Dabei sind die absoluten Werte von MAP und F1-Maß wichtig, um Rückschlüsse auf die Leistung des Empfehlungssystems unter realen Bedingungen ziehen zu können.

Zum jetzigen Zeitpunkt ist aus der Tabelle 3.4 zu schließen, dass das mit Tripletten trainierte Modell eindeutig bessere Ergebnisse liefert. Es ist auch anzumerken, dass das mehrsprachige Standardmodell aus der Sentence-Transformers-Bibliothek "distiluse-base-multilingual-cased-v1" nur geringfügig schlechtere Ergebnisse liefert als das angepasste Basismodell "xlm-roberta-en-de-reqs". Daher können die ersten beiden Trainingsschritte (Maskentraining und Training des mehrsprachigen Modells) als optional betrachtet werden, aber es ist möglich, dass das Ergebnis des Basismodells deutlich verbessert werden kann, wenn mehr und qualitativ bessere Beispiele in einem eigenen Datensatz verfügbar sind.

3.3.4 Training des Sprachmodells für Reranking

Die Aufgabe des Rerankings sieht die Verwendung eines Modells vom Typ Cross-Encoder vor, bei dem es sich im Wesentlichen um einen Klassifikator für ein Textpaar handelt, das durch ein spezielles Token getrennt ist. Ein solches Modell kann sowohl mit der Cross-Encoder-Klasse aus der Sentence-Transformers-Bibliothek als auch mit der Trainer-API der Transformers-Bibliothek trainiert werden. Die Klasse CrossEncoder ist eigentlich nur eine dünne Hülle (*engl. Wrapper*) der Klasse AutoModelForSequenceClassification aus der Bibliothek

Transformers, verfügt aber über eine benutzerfreundliche API, weshalb sie im Folgenden verwendet wird. Außerdem bietet die Sentence-Transformers-Bibliothek bereits vortrainierte Modelle vom Typ Cross-Encoder, einschließlich mehrsprachiger Modelle, z. B. "cross-encoder/msmarco-MiniLM-L12-en-de-v1". Der Vorteil dieses Modells gegenüber dem gleichen XLM-RoBERTa als Ausgangspunkt für das Vortraining ist die Tatsache, dass es auf realen Suchanfragen und Antworten der Suchmaschine Bing trainiert und entsprechend für die Aufgabe des Rerankings bei der Informationssuche optimiert wurde. Außerdem wurde das Training genau für das Zielpaar Englisch-Deutsch durchgeführt (UKPLab, 2023).

Um Cross-Encoder zu trainieren, kann der bereits für das Training des Sentence-Transformer-Modells nach der dritten Variante vorbereitete Datensatz eingesetzt werden, d. h. mit zwei Texten und einer Kosinus-Ähnlichkeit zwischen ihnen. Das Skript für das Training von Cross-Encoder ist vollständig in Anhang A.4 enthalten. Die Reihenfolge der Schritte im Skript unterscheidet sich grundsätzlich nicht vom Trainingsprozess des Sentence-Transformer-Modells. Der einzige Unterschied besteht in der Verwendung einer anderen Klasse für die Modellinitialisierung (CrossEncoder) und eines anderen Evaluatortyps (CECorrelationEvaluator). Das Training wurde für 6 Epochen durchgeführt, während derer die Spearman-Korrelation von 63% auf 87% anstieg. Der trainierte Reranker wird als Teil des Empfehlungssystems weiter getestet und seine Leistung in Abschnitt 4 demonstriert.

3.3.5 Training des Sprachmodells zur Erkennung von Dokumentenverweisen

Wie in Abschnitt 3.3.2 (Datenanalyse) gezeigt wurde, sind Verweise auf externe Dokumente im Datensatz allgegenwärtig und ein wichtiger Indikator dafür, dass zwei Anforderungen miteinander in Beziehung stehen könnten. Darüber hinaus hängt der zweite Hauptanwendungsfall des Empfehlungssystems (siehe Abschnitt 3.1) direkt mit der Erkennung von Verweisen auf externe Dokumente in den Anforderungstexten zusammen. Um Wörter aus dem Anforderungstext zu extrahieren, die eine bestimmte Entität repräsentieren, ist es notwendig, jedes Token des Textes zu klassifizieren und dann zu einer Phrase (d. h. einem kontinuierlichen Intervall im Text) zusammenzufassen, die der gesuchten Entität entspricht. Die beschriebene Aufgabe gehört zur Klasse der Named-Entity-Recognition (NER) im NLP-Bereich (siehe Abschnitt 2.2.2). In diesem Fall versucht das Modell die einzige Entität zu erkennen, nämlich die Dokumentennummer.

Im Gegensatz zu anderen bisher durchgeführten Trainingsverfahren fehlt zum Trainieren eines solchen Netzes zunächst ein etikettierter Datensatz. Um einen Trainingsdatensatz zu erhalten, müssen mindestens alle Kundenanforderungen aus dem gesamten Rohdatensatz, d. h. etwa 55 000 Anforderungen mit eindeutigem Text, etikettiert werden. Um den Aufwand der manuellen Etikettierung zu reduzieren, wird das bereits aus Abschnitt 3.3.2 bekannte neuronale

Zeroshot-Netzwerk für die NER-Aufgabe "Gliner" eingesetzt, um in erster Näherung Dokumentennummern aus den Anforderungen zu extrahieren. Aufgrund der niedrigen Wahrscheinlichkeitsschwelle für die Klassifizierung mit dem vortrainierten neuronalen Netz enthält die resultierende Liste mehr Phrasen als eigentlich vorhanden sein sollten. Nämlich erkannte das neuronale Netz von 55 000 Objekten Dokumentennummern in etwa 15 000. Nach der manuellen Etikettierung wurde die Liste auf 8000 Beispiele fast halbiert. Jede Zeile des Datensatzes enthält eine ID, den Text der Anforderung und einen Teilstring des Anforderungstextes, der die Dokumentennummer darstellt. Wenn also eine Anforderung mehrere Verweise auf ein Dokument enthält, wird sie in der Tabelle mehrfach aufgeführt.

Das Training wurde mit Hilfe der Trainer-API aus der Transformers-Bibliothek durchgeführt, die über eine spezielle AutoModelForTokenClassification-Klasse verfügt, die das Modell in eine für die Token-Klassifikation geeignete Struktur umwandelt. Das Skript für das Training des NER-Modells ist in Anhang A.5 vollständig wiedergegeben und besteht aus den folgenden Schritten.

1) Initialisierung des zugrundeliegenden Modells, das auf MLM-Aufgaben trainiert wurde (siehe Abschnitt 3.3.3.1).

2) Tokenisierung der Anforderungstexte

3) Aufbereitung der Datensatz-Labels in eine Form, die für das Training des Token-Klassifikators geeignet ist. Die Datasatz-Etiketten sollten in diesem Fall aus einer Liste mit 0, 1, 2 bestehen, deren Länge der Anzahl der Token im Text entspricht. Das Etikett "1" entspricht dem ersten Token in der Dokumentennummer, das Etikett "2" entspricht jedem nachfolgenden Token in der Dokumentennummer, das Etikett "0" bedeutet, dass das Token nicht zu der Dokumentennummer gehört.

4) Aufteilung der Daten in einen Trainings- und einen Validierungssatz

5) Initialisierung eines Objekts der Klasse DataCollatorForTokenClassification zum automatischen Abgleich von Eingabedaten und Etiketten im Batch.

6) Initialisierung einer Funktion zur Berechnung von Metriken (Genauigkeit, Sensitivität, F1-score) unter Verwendung der seqeval-Bibliothek

7) Bildung von Hyperparametern (Batchgröße, Lernrate).

8) Training des Modells für 9 Epochen

In der Validierungsmenge erreichte das neuronale System einen F1-Maß von 0,8 mit einer Genauigkeit von 0,77 und einer Sensitivität von 0,82. Es ist anzumerken, dass sich die oben genannten Metriken von den in Abschnitt 2.2.5.3 beschriebenen insofern unterscheiden, als die Bewertung auf der Relevanz jedes Tokens für das Etikett basiert. Selbst wenn einzelne Token falsch klassifiziert wurden, bedeutet dies nicht zwangsläufig, dass die Dokumenten-

nummer als Ganzes falsch erkannt worden ist. In Abschnitt 4 wird dann die Leistung des Gesamtsystems bewertet, wobei die Evaluierungsmetrik so angepasst wird, dass die Erkennung des Dokumentenverweises als Ganzes berücksichtigt wird. Um Verzerrungen zu vermeiden, wird die Evaluierung an Projekten durchgeführt, deren Daten nicht in das Training des neuronalen Netzes einbezogen wurden.

3.4 Entwicklung der Software für den Empfehlungssystem-Server

Basierend auf dem Entwurf der Systemarchitektur aus Abschnitt 3.2 wird nun die Serversoftware des Empfehlungssystems entwickelt. Der Server soll die Arbeit der OpenSearch-Suchmaschine, die auch als Anforderungsdatenbank fungiert, und der in Abschnitt 3.3 trainierten neuronalen Netze in einem System vereinen und das Ergebnis in Form der am besten geeigneten Anforderungspaare zur Verknüpfung ausgeben. Außerdem sollte das Empfehlungssystem über einen Testmodus verfügen (siehe Abschnitt 3.1, Punkt 3) und sowohl im Batch-Modus, in dem mehrere Abfragen verarbeitet werden, als auch im Online-Modus arbeiten können. Das schließt die Möglichkeit ein, dem System eine beliebige Abfrage zu stellen, die sich nicht auf eine Anforderung in der Datenbank bezieht (siehe Abschnitt 3.1, Punkt 4). Außerdem sollte es möglich sein, Daten aus einzelnen Dateien oder anderen Webdiensten zu laden; in diesem Fall ist keine Suchmaschine eines Drittanbieters erforderlich (siehe Abschnitt 3.1, Punkt 5). Die Software des Servers des Empfehlungssystems ist in Python geschrieben, was auf die reiche Auswahl an Werkzeugen in dieser Sprache für die Interaktion mit neuronalen Netzen, das bereits vorhandene Wissen des Autors dieser Arbeit und die Notwendigkeit einer schnellen Entwicklung des Systemprototyps zurückzuführen ist.

Die im OpenSearch-Suchindex gespeicherten Daten haben eine möglichst einfache Struktur und entsprechen dem in Abschnitt 2.1 vorgestellten Anforderungsdatenmodell (siehe Abbildung 2.5). Die Menge der Anforderungsobjekte hat die gleichen Attribute wie der Datensatz, der zum Training der neuronalen Netze verwendet wird (siehe Abschnitt 3.3.1). Zusätzlich werden die mit einem neuronalen Netzwerk vom Typ Sentence-Transformer vorberechneten Vektordarstellungen der Anforderungen in die Suchmaschine geladen und für die semantische Suche verwendet. Die Interaktion mit dem OpenSearch-Server erfolgt über dessen REST-API, die durch die Python-Bibliothek opensearch-py unterstützt wird.

Die Abbildung 3.22 zeigt das Ablaufdiagramm für den Betrieb des Empfehlungssystems in beiden Laufmodi.

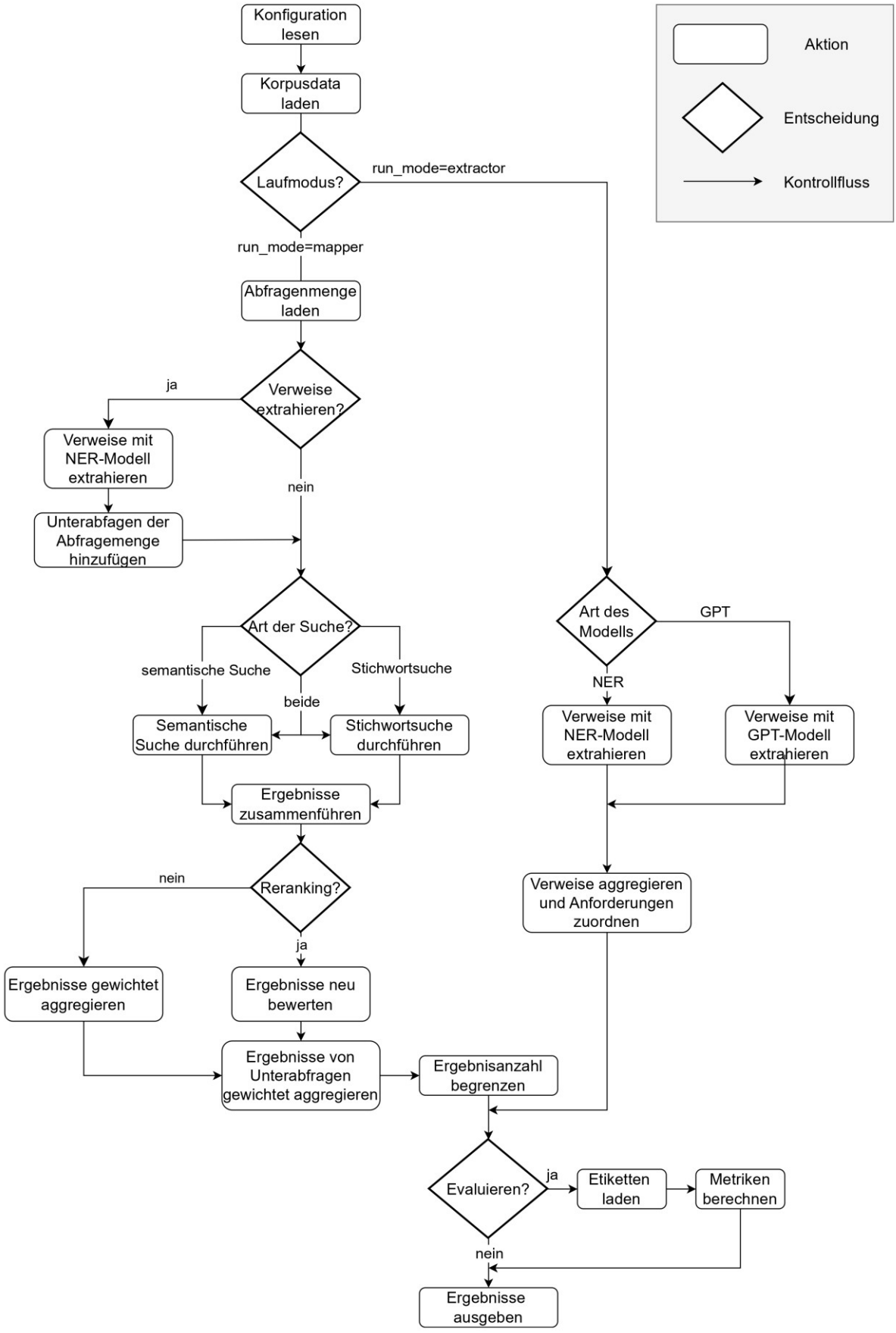

Abbildung 3-22 Ablaufdiagramm des Empfehlungssystems

Im Basismodus arbeitet das Empfehlungssystem im Batch-Modus und setzt das Laden der Konfigurationsdatei im YAML-Format voraus. Wenn das System im Online-Modus oder als Webservice arbeitet, wird diese Datei im JSON-Format vorliegen, das für den Datenaustausch zwischen Webanwendungen verwendet wird. Die Konfigurationsdatei definiert das weitere Verhalten des Programms und besteht aus den folgenden Parametern, die in Tabelle 3.5 aufgeführt sind.

Tabelle 3-5 Konfigurationsparameter des Empfehlungssystems

Parameter-name	Pflichtparameter	Erwarteter Datentyp	Beschreibung
run_mode	Ja	String	Als Eingabe werden die Werte "mapper" oder "extractor" erwartet. Der Modus "mapper" entspricht dem Anwendungsfall 1), der Modus "extractor" entspricht dem Anwendungsfall 2) (siehe Abschnitt 3.1).
index	Nein, wenn im Stateless-Modus gearbeitet wird, sonst ja	String	Der Name des zu durchsuchenden OpenSearch-Index.
query_filter	Ja im Mapper-Modus, nein im Extractor-Modus	Dictionary	Ein Abfragefilter für OpenSearch, der eine Menge von Anforderungen (Suchabfragen) definiert – Stateful-Modus
		Liste	Eine Liste von Objekten, die nicht an die Anforderungsdatenbank gebunden sind und eine Reihe von Suchabfragen definieren – Stateless-Modus
		String	Pfad zu der Datei, die den Anforderungsmenge (Suchabfragen) enthält – Stateless-Modus
corpus_filter	Ja	Dictionary	Ein Abfragefilter für OpenSearch, der eine Menge von Anforderungen in Korpus definiert, in der gesucht wird – Stateful-Modus
		Liste	Eine Liste von Objekten, die nicht mit einer Anforderungsdatenbank verknüpft ist, die die zu durchsuchende Anforderungsmenge definiert– Stateless-Modus
		String	Pfad zu der Datei, die die zu durchsuchende Anforderungsmenge enthält – Stateless-Modus

Tabelle 3-5 (Fortsetzung)

Parameter-name	Pflichtparameter	Erwarteter Datentyp	Beschreibung
retriever	Nein	String	Pfad zu einem Modell des Typs Sentence-Transformer
reranker	Nein	String	Pfad zu einem Modell des Typs Cross-Encoder
ner	Nein im Mapper-Modus, Ja im Extractor-Modus	String	Pfad zu einem Modell des Typs Named-Entity-Recognition
keyword_search	Nein	Boole'scher Wert	Boole'scher Parameter zum Aktivieren oder Deaktivieren der Stichwortsuchoption.
semantic_search	Nein	Boole'scher Wert	Boole'scher Parameter zum Aktivieren oder Deaktivieren der semantischen Suchoption.
rerank	Nein	Boole'scher Wert	Boole'scher Parameter zum Aktivieren oder Deaktivieren der Rerankings unter Verwendung des Cross-Encoder-Modells
extract_refs	Nein	Boole'scher Wert	Boole'scher Parameter zum Aktivieren oder Deaktivieren der Funktion zur Erkennung von Dokumentverweisen in Anforderungstexten unter Verwendung des NER-Modells.
tokenised	Nein	Boole'scher Wert	Wenn „Wahrheit“, wird die Stichwortsuche mit „Subword“-Tokenisierung durchgeführt, andernfalls wird der Standard-Wort-Tokenizer verwendet.
add_weight	Nein	Gleitkomma-Zahl	Ein Wert zwischen 0 und 1 bestimmt die Gewichtung der aus der Stichwortsuche gewonnenen Relevanzeinstufung im Rerankingverfahren.
sub-query_weight	Nein	Gleitkomma-zahl	Ein Wert zwischen 0 und 1 bestimmt die Relevanzgewichtung in Bezug auf die Unterabfrage (Verweise auf im Abfragetext erkannte Dokumente), wenn Ergebnisse kombiniert werden
max_retrieve_count	Nein	Ganzzahl	Maximale Anzahl von Objekten, die vom Retriever als Antwort auf jede Anfrage abgerufen werden

Tabelle 3-5 (Fortsetzung)

Parameter-name	Pflichtparameter	Erwarteter Datentyp	Beschreibung
min_out-put_count	Nein	Ganzzahl	Mindestanzahl von Objekten, die vom Retriever als Antwort auf jede Anfrage ausgegeben werden
min_score	Nein	Gleitkomma-zahl	Mindestschwellenwert für die Relevanz, der für jede Abfrage in die Ergebnisliste aufgenommen wird
evaluation	Nein	String	Pfad zu einem beschrifteten Datensatz zur Bewertung der Leistung des Systems.
output_mode	Nein	String	Datenformat für die Ausgabe, standardmäßig Excel

Anschließend lädt das System die Daten auf der Grundlage des Parameters „corpus_filter“ entweder aus der Datenbank, aus einer Datei oder direkt aus der Konfiguration. Je nach dem Wert des „run_mode“-Parameters arbeitet das System entweder im Mapper-Modus weiter, d. h. es setzt zwei Anforderungssätze miteinander in Beziehung, oder im Extractor-Modus, d. h. es erkennt Verweise auf externe Dokumente in den Texten der Anforderungen im Korpus.

Im Mapper-Modus führt das Empfehlungssystem die folgenden Schritte durch:

1) Es lädt die Liste der Abfrageobjekte, abhängig vom „query_filter“-Parameter, entweder aus der Datenbank, aus einer Datei oder direkt aus der Konfiguration.

2) Bei Vorhandensein des „extract_refs“-Parameters und des NER-Modells erkennt es Dokumentenreferenzen in den Abfragetexten und fügt sie der Liste der Abfrageobjekte als Unterabfrage hinzu.

3.1) Bei Vorhandensein des „semantic_search“-Flags und des Retriever-Modells erzeugt es Vektordarstellungen der Abfragetexte.

3.2) Beim Stateful-Modus führt das System eine semantische Suche in der OpenSearch-Datenbank durch. Wenn der Textkorpus aus einer Datei oder einem Webdienst stammt (Stateless-Modus), werden zunächst Vektordarstellungen der Suchtexte im Korpus erzeugt und dann eine semantische Suche mit der Funktion „semantic_search“ aus der Bibliothek Sentence-Transformers durchgeführt.

4) Wenn das „keyword_search“-Flag vorhanden ist, wird beim Stateful-Modus die OpenSearch-Datenbank mit der BM25-Methode durchsucht. Wenn der Textkorpus aus einer Datei oder einem Webdienst abgerufen wird (Stateless-Modus), wird zunächst ein Index auf der

Grundlage des Anforderungstextes erstellt und dann mit der BM25-Methode und der Bibliothek rank-bm25 gesucht.

5) Die mit den beiden Methoden erzielten Ergebnisse werden kombiniert, wobei Duplikate entfernt und die BM25-Ergebnisse so normalisiert werden, dass das Relevanzranking zwischen 0 und 1 liegt.

6.1) Wenn das „rerank"-Flag vorhanden ist, werden Abfrage-Ergebnis-Paare gebildet, und das Cross-Encoder-Modell erstellt dann für jedes dieser Paare eine neue Relevanzbewertung. Wenn der Wert von add_weight größer als Null ist, werden die neue Bewertung und die durch die bm25-Methode erhaltene Bewertung gewichtet aufsummiert:

$$joinScore = addWeight \cdot bm25Score + (1 - addWeight) \cdot \boldsymbol{rerankerScore}$$

6.2) Wenn der „rerank"-Parameter nicht gesetzt ist, ist die endgültige Relevanzbewertung eine gewichtete Summe der mit der BM25-Methode und der semantischen Suche erzielten Ergebnisse:

$$joinScore = addWeight \cdot bm25Score + (1 - addWeight) \cdot \boldsymbol{semanticScore}$$

7) Wurde das Abfragemenge zuvor um Verweise auf in den Anforderungstexten gefundene Dokumente erweitert (siehe Schritt 2), werden die durch die Abfrage selbst und die Unterabfrage erzielten Ergebnisse aggregiert. Die endgültige Punktzahl wird nach folgender Formel berechnet:

$$finalScore = \sum_{i}^{n} subWeight_i \cdot subScore_i + \frac{(1 - subWeight) * joinScore}{1 + (queryCount - 2) \cdot subWeight}$$

8) Überflüssige Ergebnisse werden auf der Grundlage der Parameter min_output_count und min_score aussortiert. Der Parameter min_output_count hat immer Vorrang vor min_score. Dies bedeutet, dass die Ausgabe stets mindestens die Anzahl von Ergebnissen enthält, die in min_output_count festgelegt ist, selbst wenn die Relevanzbewertung niedriger als min_score ausfällt.

Im Extractor-Modus führt das Empfehlungssystem die folgenden Schritte durch:

1) Entsprechend dem Parameter „ner" wählt es ein Sprachmodell vom Typ Named Entity Recognition

2) Das NER-Modell erkennt Dokumentenverweise in den Texten der Korpusanforderungen

3) Das Vorhersage-Array wird dem ursprünglichen Korpus-Datensatz hinzugefügt

4) Der Datensatz wird neu indiziert, sodass eine Dokumentennummer einem oder mehreren Anforderungsobjekten entspricht

Außerdem werden in beiden Modi bei Vorhandensein des Evaluierungsparameters der markierte Datensatz geladen und die Metriken Genauigkeit, Sensitivität und F1-Maß nach Abschnitt 2.2.5.3 berechnet. Bei der Arbeit im Mapper-Modus werden zusätzlich die erweiterten Metriken MRR, NDCG und MAP berechnet.

Die in Datasätzen gespeicherten Ergebnisse des Empfehlungssystems werden in Dateien ausgegeben, deren Format dem Wert des Parameters "output_mode" entspricht (CSV, Excel oder JSON). Im Testmodus werden neben den direkten Ergebnissen („predictions“-Datei) auch die „labels“-Datei, aus der hervorgeht, welche Anforderungspaare richtig klassifiziert wurden, und die Metrik-Datei, die die Werte der Evaluierungsmetriken anzeigt, ausgegeben.

Es ist zu beachten, dass die Werte der Parameter „retriever“ und „reranker“ auch als Modellnamen im Huggingface Hub Repository angegeben werden können, die dem Typ Sentence-Transformer bzw. Cross-Encoder entsprechen. Zusätzlich kann „gpt“ als Wert des „ner“-Parameters angegeben werden, was bedeutet, dass die Erkennung von Dokumentennummern im Anforderungstext unter Verwendung des GPT3.5-Modells und der OpenAI-API (OpenAI, 2023) durchgeführt wird. Die Möglichkeit, externe Modelle oder Dienste zu nutzen, ermöglicht es, die Leistung der feinabgestimmten Modelle im Vergleich zu den Grundmodellen zu analysieren und somit die Notwendigkeit des Trainings eigener Modelle zu beurteilen.

4 Evaluierung des Empfehlungssystems

In diesem Abschnitt wird die Leistung des Empfehlungssystems als Ganzes bewertet. Wie in Abschnitt 3.4 ersichtlich ist, wurde das Empfehlungssystem ursprünglich mit der Absicht entworfen, flexibel zu sein und es zu ermöglichen, verschiedene Algorithmen und Technologien für die Erfüllung der gleichen Aufgaben zu verwenden. Die Konfigurationsparameter des Empfehlungssystems ermöglichen es, es in verschiedenen Modi zu betreiben und seine Leistungsfähigkeit mit unterschiedlichen Einstellungen zu testen. Anhand der im Abschnitt 3.1 identifizierten Anwendungsfällen wird die Leistung des Empfehlungssystems im Basismodus und im alternativen Modus untersucht.

Der Basismodus, der in Abschnitt 3.4 auch als "Mapper" bezeichnet wird, sieht die Erstellung der Trace-Links zwischen zwei bereits definierten Anforderungssätzen vor. Der alternative Modus, der in Abschnitt 3.4 als "Extractor" bezeichnet wird, ist nur für einen spezifischen, aber das wichtigste Anwendungsfall konzipiert, nämlich die Identifizierung von Verweisen auf externe Dokumente, einschließlich Normen und Standards (alle diese Dokumente werden unter dem Begriff "Projektspezifikationen" zusammengefasst) in den Texten der Kundenanforderungen. Die Bewertung der Funktionsweise des Systems im Basismodus wird in Abschnitt 4.1 und im Alternativmodus in Abschnitt 4.2 gegeben.

Für die Evaluierung wurden Daten aus den Projekten des Unternehmens herangezogen, die nicht an dem Training teilgenommen haben. Tabelle 4.1 zeigt die Bezeichnung und die Größe der Projekte, ausgedrückt durch die Anzahl der Anforderungen in den jeweiligen Spezifikationen.

Tabelle 4-1 An der Evaluierung beteiligte Projekte

Projektbezechnung	Größe des Projekts
P17	~11000
P18	~8500
P20	~8500
P22	~60000
P25	~17000

4.1 Basismodus: Rückverfolgbarkeit zwischen zwei Mengen von Anforderungen

4.1.1 Beschreibung der Testfälle

Im Basismodus versucht das Empfehlungssystem unter Verwendung der Anforderungstexte aus einer Abfragemenge, die relevantesten Anforderungen aus der zweiten Menge zu finden (siehe Abb. 3-1). Die Anforderungstexte auf verschiedenen Ebenen können entweder sehr ähnlich sein, bis hin zur völligen Überlappung, oder völlig unterschiedlich, aber eine der verknüpften Anforderungen weist immer auf mindestens einen Aspekt der anderen hin. Wie im Abschnitt 2.1.3 bereits dargelegt, sind die Anforderungen auf den verschiedenen Ebenen in einer viele-zu-vielen-Verknüpfung miteinander verbunden. Daher kann nicht pauschal behauptet werden, dass eine Anforderung auf der obersten Ebene immer allgemeiner ist als eine Anforderung auf einer niedrigeren Ebene.

Im Basismodus werden vier Testfälle bewertet:

1) Erstellung der Rückverfolgbarkeit zwischen Kundenanforderungen und Projektspezifikationen. Diese Aufgabe ähnelt derjenigen, für die der Extractor-Modus konzipiert ist; der Unterschied im Basismodus besteht in der Voraussetzung, dass die Liste der Projektspezifikationen bereits vordefiniert ist.

2) Erstellung der Rückverfolgbarkeit zwischen Kunden- und Systemanforderungen

3) Erstellung der Rückverfolgbarkeit zwischen System- und Softwareanforderungen

4) Erstellung der Rückverfolgbarkeit zwischen System- und mechanischen Anforderungen

Die Testfälle 2-4 unterscheiden sich darin, dass die Kundenanforderungen und die Systemanforderungen häufig in verschiedenen Sprachen (Deutsch bzw. Englisch) vorliegen. In Gegensatz dazu sind die Systemanforderungen und die Domänenanforderungen (einschließlich der Software- und mechanischen Anforderungen) in der Regel beide auf Englisch verfasst. Die durchgeführte Datenanalyse zeigt, dass einerseits Softwareanforderungen im Datensatz viel stärker vertreten sind als andere Domänen, während andererseits die mechanischen Anforderungen in Bezug auf die Sprache eher qualitativ zu sein scheinen. Aus diesem Grund werden die beiden letztgenannten Anwendungsfälle auch in der Auswertung berücksichtigt.

Die folgenden Faktoren sind für die Bewertung der Systemleistung in der Baseline von Interesse und werden weiter untersucht.

1) Modus der Information Retrieval (IR). Die Leistung des Systems wird bei der Anwendung verglichen:

1.1) Nur semantische Suche („semantic“)

1.2) Nur Stichwortsuche („keyword“)

1.3) Zwei Methoden in Kombination („hybrid“).

2) Reranking-Modus. Die hybride Suche berücksichtigt:

2.1) Reines Reranking, d. h. die Relevanz wird neu bewertet, ohne die in den vorherigen Schritten erzielten Punktzahlen zu berücksichtigen.

2.2) Gewichtete Aggregation von Stichwort- und semantischen Suchergebnissen

2.3) Hybrides Reranking - gewichtete Kombination der Ergebnisse des reinen Rerankings und der Stichwortsuche.

3) Extraktion von Links zu externen Dokumenten als Unterabfrage. Die Anzahl der externen Dokumente, auf die in der Anforderung verwiesen wird, wird mit Hilfe eines neuronalen Netzes vom Typ NER aus den Anforderungstexten extrahiert. Die Verweise werden in Unterabfragen extrahiert, so dass ihr Gewicht bei der Relevanzbewertung im Vergleich zu den übrigen Wörtern in der Anforderung verstärkt wird.

4) Maximale Anzahl der abgerufenen Objekte für jede Abfrage.

5) Verwendung einer auf Teilwörtern („subword“) oder auf ganzen Wörtern („word“) basierenden Tokenisierung für die Stichwortsuche.

6) Anwendung des feinabgestimmten Cross-Encoder fürs Reranking gegenüber dem vortrainierten Sprachmodell

7) Anwendung des feinabgestimmten Sentence-Transformer gegenüber dem vortrainierten Sprachmodell

Die Evaluierung erfolgt anhand der in Abschnitt 2.2.5.3 vorgestellten Metriken - MAP, Genauigkeit, Sensitivität, NDCG und MRR - bei verschiedenen Stufen der Ausgabe k, wobei Sensitivität und MAP die wichtigsten Metriken für die endgültige Schlussfolgerung über die Systemleistung sind.

Für jeden Testfall wurden insgesamt 27 Varianten in Betracht gezogen, die sich durch eine Kombination der oben vorgestellten Faktoren auszeichnen. Tabelle 4.2 enthält eine Beschreibung der einzelnen Varianten des Empfehlungssystems.

Tabelle 4-2 Systemvarianten zur Evaluierung

Variante	IR-Modus	KS-Gewicht für Aggregation	KS-Gewicht für RR	Gewicht der Unterabfrage	Max. Anzahl der Objekte	Tokenisierung bei KS	IR-Sprachmodell	RR-Sprachmodell
hybrid_rerank_000	hybrid	n/a	0.0	n/a	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_base	hybrid	n/a	0.0	n/a	50	subword	feinabgestiimmt	vortrainiert
hybrid_rerank_000_extract_refs_025	hybrid	n/a	0.0	0.25	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_extract_refs_050	hybrid	n/a	0.0	0.5	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_extract_refs_075	hybrid	n/a	0.0	0.75	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_get_10	hybrid	n/a	0.0	n/a	10	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_get_25	hybrid	n/a	0.0	n/a	25	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_get_50	hybrid	n/a	0.0	n/a	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_get_75	hybrid	n/a	0.0	n/a	75	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_get_100	hybrid	n/a	0.0	n/a	100	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_000_word_tokenize	hybrid	n/a	0.0	n/a	50	word	feinabgestiimmt	feinabgestimmt
hybrid_rerank_025	hybrid	n/a	0.25	n/a	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_050	hybrid	n/a	0.5	n/a	50	subword	feinabgestiimmt	feinabgestimmt
hybrid_rerank_075	hybrid	n/a	0.75	n/a	50	subword	feinabgestiimmt	feinabgestimmt
keyword	keyword	n/a	n/a	n/a	50	subword	n/a	n/a
keyword_extract_refs_025	keyword	n/a	n/a	0.25	50	subword	n/a	n/a

Tabelle 4-2 (Fortsetzung)

Variante	IR-Modus	KS-Gewicht für Aggregation	KS-Gewicht für RR	Gewicht der Unterabfrage	Max. Anzahl der Objekte	Tokenisierung bei KS	IR-Sprachmodell	RR-Sprachmodell
keyword_rerank_000	keyword	n/a	0.0	n/a	50	subword	n/a	feinabgestimmt
keyword_rerank_000_extract_refs_025	keyword	n/a	0.0	0.25	50	subword	n/a	feinabgestimmt
keyword_rerank_000_word_tokenize	keyword	n/a	0.0	n/a	50	word	n/a	feinabgestimmt
keyword_rerank_025	keyword	n/a	0.25	n/a	50	subword	n/a	feinabgestimmt
keyword_word_tokenize	keyword	n/a	n/a	n/a	50	word	n/a	n/a
semantic	semantic	n/a	n/a	n/a	50	n/a	feinabgestiimmt	n/a
semantic_base	semantic	n/a	n/a	n/a	50	n/a	vortrainiert	n/a
semantic_rerank_000	semantic	n/a	n/a	n/a	50	n/a	feinabgestiimmt	feinabgestimmt
weighted_joiner_025	hybrid	0.25	n/a	n/a	50	subword	feinabgestiimmt	n/a
weighted_joiner_050	hybrid	0.5	n/a	n/a	50	subword	feinabgestiimmt	n/a
weighted_joiner_075	hybrid	0.75	n/a	n/a	50	subword	feinabgestiimmt	n/a

4.1.2 Ergebnisse im Basismodus

Rückverfolgbarkeit zwischen Kundenanforderungen und Projektspezifikationen (Testfall 1)

Die Qualität der Erstellung der Rückverfolgbarkeit zwischen den Kundenanforderungen und den Designspezifikationen (Testfall 1) wird im Basismodus anhand der Daten der Projekte P17, P18, P22 und P25 evaluiert. Tabelle 4.3 zeigt die Ergebnisse für die Metriken Recall@5 und MAP@20 für die besten acht Varianten, die schlechtesten acht Varianten und die Varianten, bei denen die vortrainierten Modelle eingesetzt werden.

Tabelle 4-3 Ergebnisse der Evaluierung für den Testfall 1 im Basismodus

Beste Varianten mit feinabgestimmten Modellen										
Variante	**P22**		**P18**		**P25**		**P17**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
hybrid_rerank_000_extract_refs_025	0,827	0,796	0,786	0,758	0,812	0,792	0,721	0,649	0,798	0,763
hybrid_rerank_000_extract_refs_050	0,827	0,799	0,777	0,746	0,793	0,768	0,721	0,655	0,791	0,757
keyword_rerank_000_extract_refs_025	0,793	0,775	0,762	0,727	0,811	0,79	0,712	0,639	0,779	0,748
hybrid_rerank_000_extract_refs_075	0,81	0,777	0,742	0,713	0,771	0,744	0,719	0,654	0,772	0,736
hybrid_rerank_025	0,79	0,743	0,779	0,744	0,791	0,763	0,648	0,581	0,763	0,72
keyword_rerank_025	0,79	0,741	0,768	0,74	0,786	0,761	0,653	0,582	0,761	0,718
hybrid_rerank_000_get_25	0,77	0,728	0,788	0,757	0,786	0,759	0,721	0,54	0,768	0,708
hybrid_rerank_000_get_10	0,755	0,713	0,795	0,756	0,789	0,745	0,747	0,59	0,769	0,707
Schlechteste Varianten mit feinabgestimmten Modellen										
Variante	**P22**		**P18**		**P25**		**P17**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
keyword_word_tokenize	0,447	0,418	0,458	0,401	0,546	0,5	0,112	0,092	0,417	0,381
keyword_rerank_000_word_tokenize	0,537	0,531	0,535	0,522	0,58	0,577	0,127	0,118	0,476	0,469
semantic	0,642	0,448	0,761	0,712	0,725	0,708	0,662	0,574	0,687	0,584
keyword	0,705	0,609	0,745	0,692	0,738	0,7	0,605	0,509	0,702	0,629
hybrid_rerank_075	0,72	0,633	0,751	0,704	0,757	0,725	0,616	0,532	0,717	0,652
semantic_rerank_000	0,689	0,661	0,767	0,74	0,73	0,703	0,672	0,508	0,709	0,657
weighted_joiner_075	0,744	0,647	0,767	0,724	0,769	0,727	0,647	0,553	0,737	0,664
weighted_joiner_025	0,72	0,602	0,795	0,748	0,758	0,732	0,696	0,631	0,738	0,666
Varianten mit vortrainierten Modellen										
Variante	**P22**		**P18**		**P25**		**P17**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
hybrid_rerank_000_base	0,738	0,635	0,699	0,536	0,742	0,62	0,732	0,565	0,732	0,603
semantic_base	0,599	0,458	0,774	0,728	0,684	0,666	0,492	0,353	0,63	0,539

Abb. 4.1 zeigt die Variation der Ergebnisse beim Testfall 1 für die Metriken Recall@5 und MAP@20 während der Aggregation und des Rankings in Abhängigkeit von der Gewichtung des Relevanzwertes aus der Stichwortsuche, der Gewichtung der aus dem Anforderungstext extrahierten Unterabfrage und der maximalen Anzahl der abgerufenen Ergebnisse für jede Abfrage.

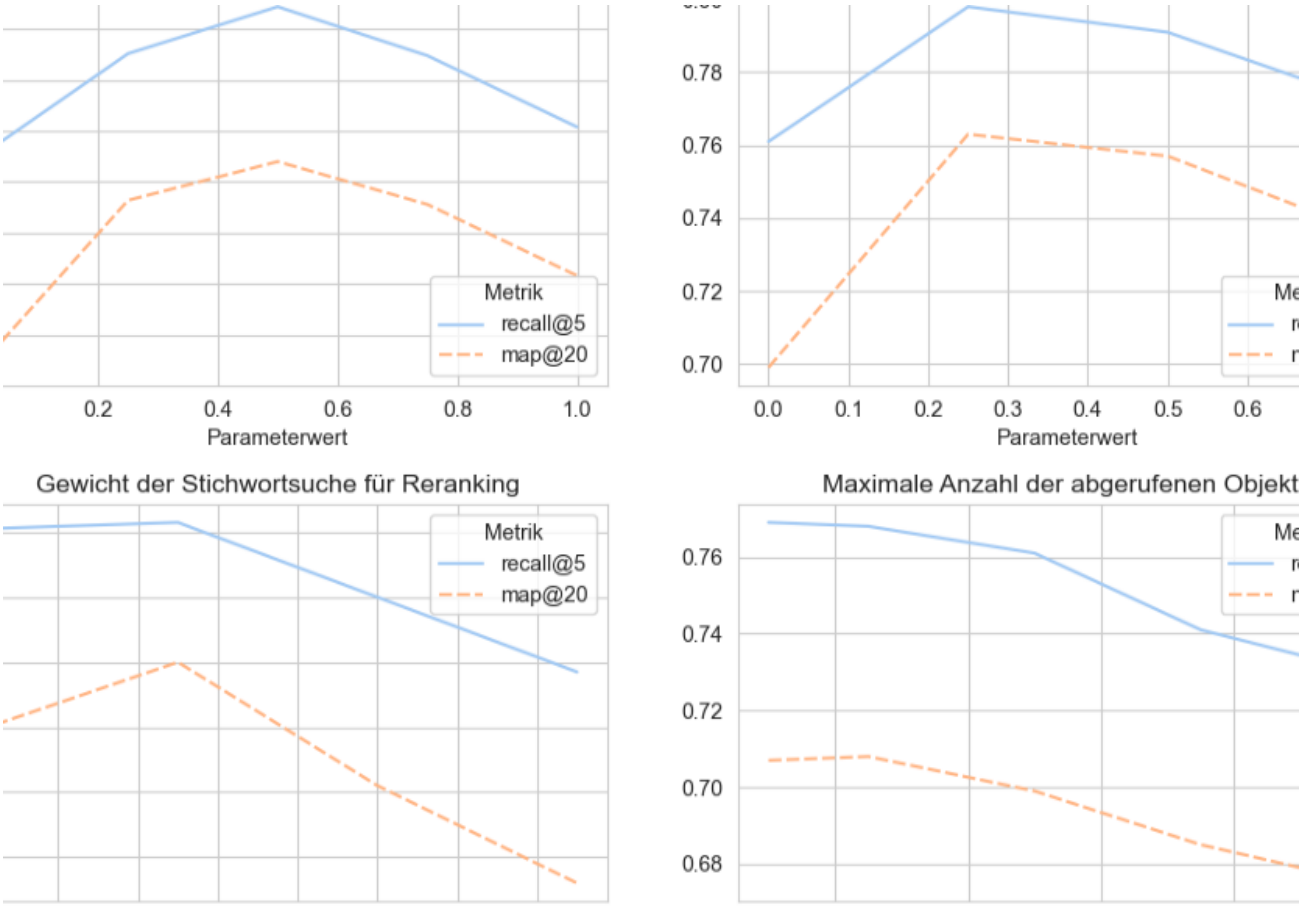

Abbildung 4-1 Untersuchung der Einflüsse von Suchparametern auf die Leistung des Empfehlungssystems im Testfall 1

Rückverfolgbarkeit zwischen Kunden- und Systemanforderungen (Testfall 2)

Die Qualität der Erstellung der Rückverfolgbarkeit zwischen den Kundenanforderungen und den Systemanforderungen (Testfall 2) wird im Basismodus anhand der Daten der Projekte P18, P20 und P22 evaluiert. Tabelle 4.4 zeigt die Ergebnisse für die Metriken Recall@5 und MAP@20 für die besten acht Varianten, die schlechtesten acht Varianten und die Varianten, bei denen die vortrainierten Modelle eingesetzt werden.

Tabelle 4-4 Ergebnisse der Evaluierung für den Testfall 2 im Basismodus

Beste Varianten mit feinabgestimmten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
weighted_joiner_025	0,735	0,657	0,942	0,87	0,609	0,532	0,765	0,689
hybrid_rerank_000 _get_10	0,704	0,623	0,929	0,884	0,623	0,543	0,751	0,682
semantic	0,723	0,656	0,941	0,879	0,549	0,486	0,743	0,679
hybrid_rerank_000 _get_25	0,669	0,613	0,929	0,885	0,621	0,542	0,736	0,677
hybrid_rerank_000	0,66	0,603	0,936	0,887	0,622	0,546	0,734	0,675
hybrid_rer-ank_000_get_50	0,66	0,603	0,936	0,887	0,622	0,546	0,734	0,675
hybrid_rerank_000_ word_tokenize	0,655	0,601	0,932	0,884	0,611	0,541	0,728	0,672
hybrid_rerank_000 _get_75	0,65	0,596	0,935	0,885	0,615	0,538	0,728	0,669
Schlechteste Varianten mit feinabgestimmten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
keyword_word_tokenize	0,365	0,332	0,225	0,181	0,429	0,37	0,339	0,296
keyword_rerank_000_ word_tokenize	0,378	0,358	0,381	0,362	0,481	0,438	0,406	0,38
keyword	0,516	0,449	0,354	0,283	0,514	0,436	0,466	0,395
keyword_extract_refs_02 5	0,515	0,45	0,36	0,288	0,514	0,436	0,467	0,397
hybrid_rerank_075	0,529	0,463	0,397	0,308	0,521	0,443	0,487	0,41
hybrid_rerank_050	0,543	0,486	0,476	0,354	0,524	0,45	0,517	0,436
weighted_joiner_075	0,574	0,486	0,497	0,353	0,548	0,461	0,544	0,439
keyword_rerank_025	0,549	0,507	0,524	0,446	0,527	0,469	0,536	0,478
Varianten mit vortrainierten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
semantic_base	0,614	0,563	0,959	0,915	0,557	0,474	0,704	0,647
hybrid_rerank_000_base	0,622	0,487	0,903	0,797	0,623	0,527	0,708	0,592

Abb. 4.2 zeigt die Variation der Ergebnisse beim Testfall 2 für die Metriken Recall@5 und MAP@20 während der Aggregation und des Rankings in Abhängigkeit von der Gewichtung des Relevanzwertes aus der Stichwortsuche, der Gewichtung der aus dem Anforderungstext extrahierten Unterabfrage und der maximalen Anzahl der abgerufenen Ergebnisse für jede Abfrage.

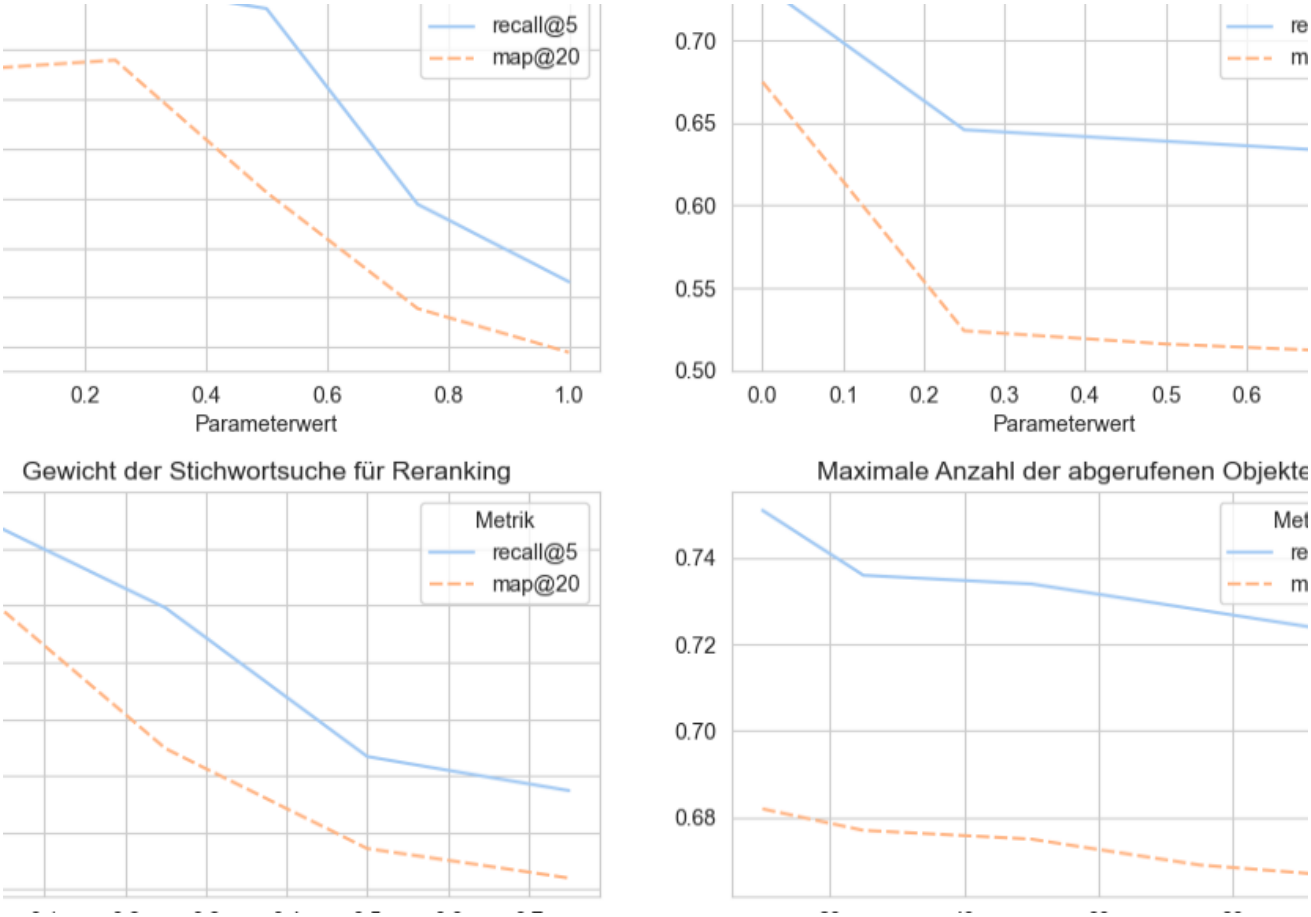

Abbildung 4-2 Untersuchung der Einflüsse von Suchparametern auf die Leistung des Empfehlungssystems im Testfall 2

Rückverfolgbarkeit zwischen System- und Softwareanforderungen (Testfall 3)

Die Qualität der Erstellung der Rückverfolgbarkeit zwischen den System- und Softwareanforderungen (Testfall 3) wird im Basismodus anhand der Daten der Projekte P18, P20 und P22 evaluiert. Tabelle 4.5 zeigt die Ergebnisse für die Metriken Recall@5 und MAP@20 für die besten acht Varianten, die schlechtesten acht Varianten und die Varianten, bei denen die vortrainierten Modelle eingesetzt werden.

Tabelle 4-5 Ergebnisse der Evaluierung für den Testfall 3 im Basismodus

Beste Varianten mit feinabgestimmten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
weighted_joiner_025	0,434	0,352	0,924	0,903	0,93	0,913	0,69	0,64
semantic	0,425	0,356	0,919	0,894	0,918	0,895	0,681	0,635
hybrid_rerank_000_get_10	0,394	0,312	0,908	0,881	0,918	0,902	0,663	0,612
weighted_joiner_050	0,388	0,291	0,924	0,903	0,937	0,911	0,669	0,61
hybrid_rerank_000_get_25	0,379	0,309	0,897	0,878	0,905	0,895	0,65	0,608
hybrid_rerank_000_word_tokenize	0,362	0,301	0,897	0,877	0,905	0,894	0,641	0,604
hybrid_rerank_000	0,366	0,298	0,897	0,877	0,905	0,895	0,643	0,603
hybrid_rerank_000_get_50	0,366	0,298	0,897	0,877	0,905	0,895	0,643	0,603
Schlechteste Varianten mit feinabgestimmten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
keyword_word_tokenize	0,243	0,206	0,912	0,881	0,901	0,839	0,587	0,546
keyword_rerank_000_word_tokenize	0,253	0,222	0,889	0,867	0,897	0,889	0,585	0,562
keyword	0,278	0,232	0,923	0,896	0,917	0,868	0,611	0,569
keyword_extract_refs_025	0,278	0,233	0,923	0,896	0,917	0,868	0,611	0,57
keyword_rerank_000_extract_refs_025	0,284	0,243	0,899	0,873	0,9	0,884	0,603	0,572
keyword_rerank_000	0,285	0,243	0,9	0,877	0,903	0,892	0,605	0,576
hybrid_rerank_000_extract_refs_075	0,292	0,254	0,895	0,871	0,904	0,886	0,607	0,578
hybrid_rerank_075	0,28	0,239	0,923	0,901	0,922	0,89	0,613	0,58
Varianten mit vortrainierten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
semantic_base	0,313	0,261	0,925	0,903	0,93	0,892	0,632	0,591
hybrid_rerank_000_base	0,378	0,274	0,881	0,751	0,773	0,586	0,612	0,481

Abb. 4.3 zeigt die Variation der Ergebnisse beim Testfall 3 für die Metriken Recall@5 und MAP@20 während der Aggregation und des Rankings in Abhängigkeit von der Gewichtung des Relevanzwertes aus der Stichwortsuche, der Gewichtung der aus dem Anforderungstext extrahierten Unterabfrage und der maximalen Anzahl der abgerufenen Ergebnisse für jede Abfrage.

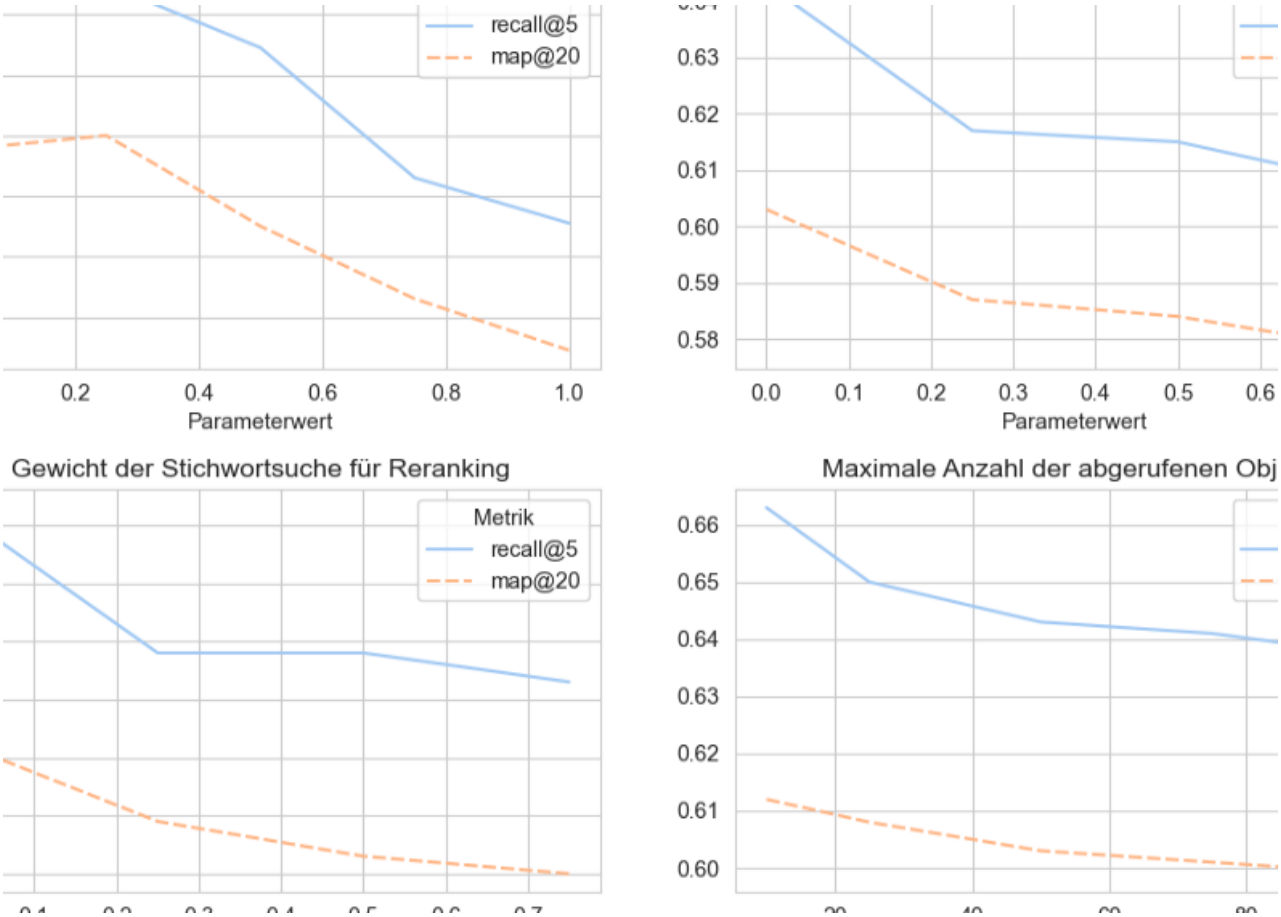

Abbildung 4-3 Untersuchung der Einflüsse von Suchparametern auf die Leistung des Empfehlungssystems im Testfall 3

Rückverfolgbarkeit zwischen System- und mechanischen Anforderungen

Die Qualität der Erstellung der Rückverfolgbarkeit zwischen den System- und mechanischen Anforderungen (Testfall 4) wird im Basismodus anhand der Daten der Projekte P18, P20 und P22 evaluiert. Tabelle 4.6 zeigt die Ergebnisse für die Metriken Recall@5 und MAP@20 für die besten acht Varianten, die schlechtesten acht Varianten und die Varianten, bei denen die vortrainierten Modelle verwendet wurden.

Tabelle 4-6 Ergebnisse der Evaluierung für den Testfall 4 im Basismodus

Beste Varianten mit feinabgestimmten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
weighted_joiner_025	0,674	0,539	0,762	0,72	0,81	0,777	0,734	0,652
weighted_joiner_050	0,546	0,454	0,812	0,75	0,813	0,771	0,693	0,622
semantic	0,652	0,529	0,693	0,631	0,783	0,754	0,696	0,614
weighted_joiner_075	0,479	0,415	0,82	0,77	0,798	0,757	0,662	0,608
keyword_extract_refs_025	0,471	0,408	0,794	0,753	0,798	0,744	0,65	0,596
hybrid_rerank_050	0,467	0,404	0,812	0,746	0,794	0,76	0,653	0,596
hybrid_rerank_075	0,471	0,404	0,804	0,751	0,798	0,756	0,653	0,596
keyword	0,473	0,403	0,794	0,752	0,798	0,752	0,651	0,595
Schlechteste Varianten mit feinabgestimmten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
keyword_rerank_000_word_tokenize	0,361	0,34	0,623	0,582	0,748	0,733	0,534	0,508
keyword_rerank_000_extract_refs_025	0,38	0,349	0,638	0,595	0,746	0,727	0,547	0,515
keyword_rerank_000	0,39	0,352	0,638	0,594	0,76	0,743	0,555	0,52
hybrid_rerank_000_get_100	0,39	0,362	0,632	0,59	0,77	0,753	0,555	0,526
hybrid_rerank_000_get_75	0,387	0,366	0,632	0,594	0,77	0,754	0,554	0,529
hybrid_rerank_000	0,413	0,376	0,633	0,59	0,775	0,755	0,567	0,532
hybrid_rerank_000_get_50	0,413	0,376	0,633	0,59	0,775	0,755	0,567	0,532
hybrid_rerank_000_word_tokenize	0,405	0,374	0,638	0,59	0,775	0,758	0,565	0,532
Varianten mit vortrainierten Modellen								
Variante	**P22**		**P18**		**P20**		**aggregiert**	
	Recall @5	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**	**Recall @5**	**MAP @20**
semantic_base	0,457	0,393	0,847	0,777	0,8	0,738	0,661	0,596
hybrid_rerank_000_base	0,487	0,413	0,791	0,745	0,795	0,733	0,655	0,593

Abb. 4.4 zeigt die Variation der Ergebnisse beim Testfall 4 für die Metriken Recall@5 und MAP@20 während der Aggregation und des Rankings in Abhängigkeit von der Gewichtung des Relevanzwertes aus der Stichwortsuche, der Gewichtung der aus dem Anforderungstext extrahierten Unterabfrage und der maximalen Anzahl der abgerufenen Ergebnisse für jede Abfrage.

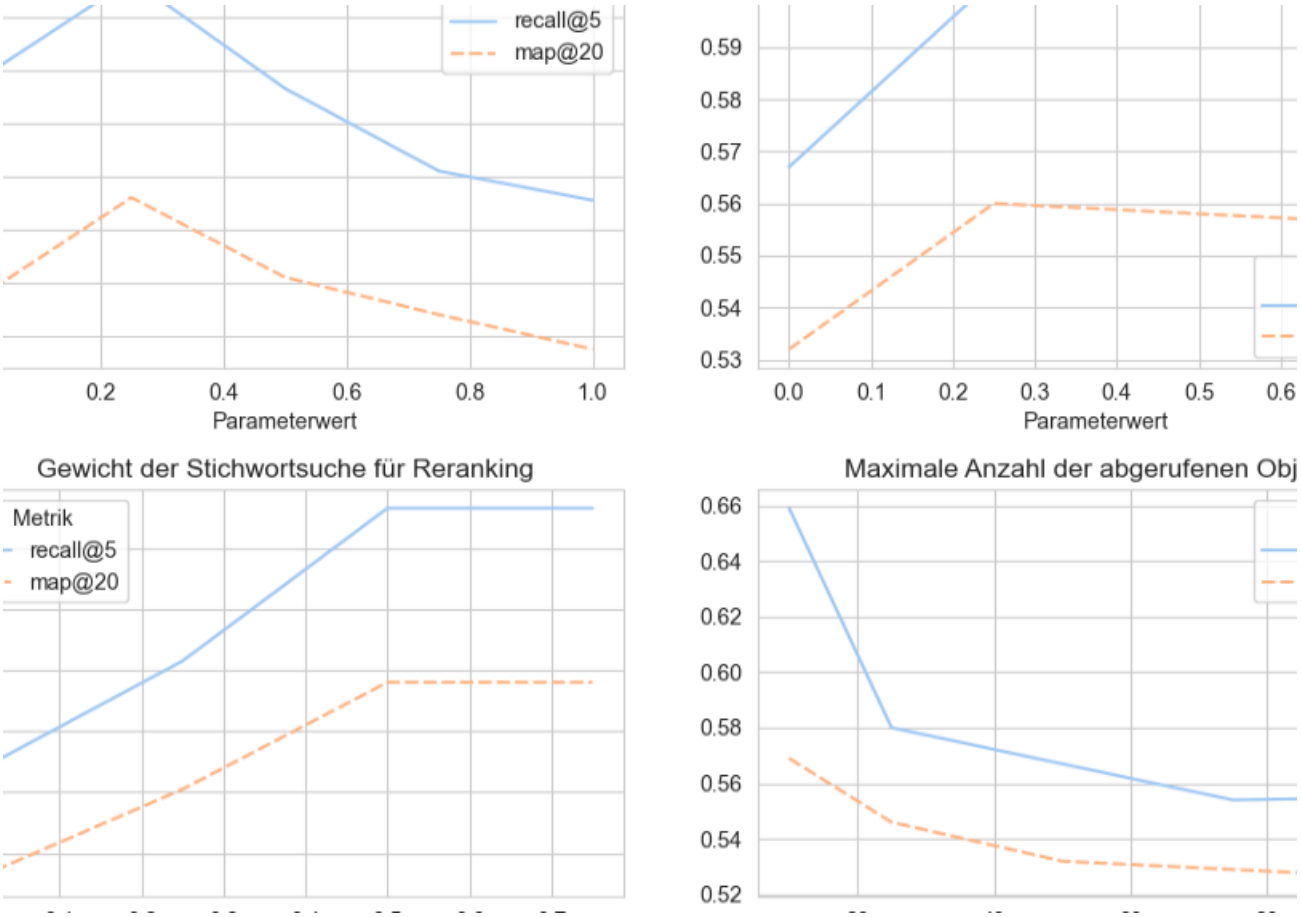

Abbildung 4-4 Untersuchung der Einflüsse von Suchparametern auf die Leistung des Empfehlungssystems im Testfall 4

4.1.3 Diskussion der Ergebnisse im Basismodus

Basierend auf den Ergebnissen des vorherigen Abschnitts lässt sich sagen, dass die Leistung des Empfehlungssystems je nach Testfall und Projekt, dessen Daten im Test verwendet werden, signifikant variieren kann. Das beste Ergebnis wird im ersten Testfall erzielt, wo die aggregierte MAP-Metrik für drei Projekte 0,76 und die Sensitivität nahezu 0,8 beträgt. Das schlechteste Ergebnis wurde im dritten Fall erzielt, wo die Rückverfolgbarkeit zwischen System- und Softwareanforderungen erstellt wird. In diesem Fall beträgt die aggregierte MAP-Metrik 0,64 und die Sensitivität fast 0,7. Dabei ist das Ergebnis für das größte Projekt P22 wesentlich schlechter: MAP - 0,34 und Sensitivität - 0,43.

Die besseren Ergebnisse im ersten Testfall im Vergleich zu den nachfolgenden drei Testfällen sind nicht überraschend, da es sich hier um spezifischen Informationen - den Dokumentennummern im Text - handelt, deren Anwesenheit bestimmt, ob Anforderungen miteinander verbunden werden können. In dieser Situation trägt die Stichwortsuche (KS) und das Extrahieren der Dokumentennummer in der Abfrage zur Verbesserung aller Metriken bei. Insbesondere führt eine Gewichtserhöhung der Unterabfrage auf 25% zu einer Verbesserung der MAP-Metrik um 7%, und eine Gewichtserhöhung der KS auf 50% zu einer Verbesserung um 6%. In den Testfällen 2-4 bestimmt die semantische Suche die Leistung des Systems, während ein extrahierter Unterabfrage nur die Metriken verschlechtert.

KS liefert besonders schlechte Ergebnisse im Testfall 2, wo es viele interlinguale Beziehungen gibt. In solchen Situationen ist nur die semantische Suche in der Lage, ähnliche oder sogar identische Bedeutungen der verbundenen Anforderungen zu erfassen.

Während im ersten Testfall die hybride Suche mit Reranking am erfolgreichsten war, zeigten sich in den darauffolgenden drei Tests enttäuschende Ergebnisse für den Reranker. Hingegen erwies sich die hybride Suche mit einfacher gewichteter Aggregation als die effektivste Methode. Dabei sollte das Gewicht der KS nicht höher als 0,25 sein, da ein weiterer Anstieg das Ergebnis erheblich beeinträchtigt. Die Verschlechterung im zweiten Testfall beträgt 30%, während sie in den Testfällen 3 und 4 etwa 9% beträgt.

Die Erhöhung der Anzahl der vor dem Reranking erhaltenen Objekte verschlechtert Leistung des Empfehlungssystems in allen Fällen außer dem ersten, was hauptsächlich mit der schwächeren Performance des Rerankers im Vergleich zum Modell für semantische Suche zusammenhängt.

Wenn man die feinabgestimmten Modelle mit den Basismodellen vergleicht, zeigt sich praktisch in allen Fällen eine bessere Leistung der feinabgestimmten Modelle. Insbesondere zeigten die feinabgestimmten Modelle eine deutliche Verbesserung in komplizierteren Fällen, z.B. im Testfall 4 für das Projekt P22, wo die Verbesserung des semantischen Suchmodells fast 20% und im Testfall 2 etwa 10% für die Sensitivität beträgt. Dabei sollte jedoch betont werden, dass auf einfacheren Daten Grundmodelle überraschenderweise bessere Ergebnisse liefern können, z.B. für das Projekt P18 im Testfall 4 liefert das Basismodell eine Ergebnisverbesserung für die Sensitivität um 15%, während in anderen Situationen das Grundmodell gewöhnlich den feinabgestimmten Modellen unterlegen ist und nur in seltenen Fällen um nicht mehr als 2% verbessern kann.

4.2 Alternativmodus: Erkennung von Verweisen auf Projektspezifikationen in Kundenanforderungen

4.2.1 Beschreibung des Testfalls

Im alternativen Modus werden in den Anforderungstexten ein oder mehrere Zeichenfolgen identifiziert, die der Nummer eines externen Dokuments (Projektspezifikation) ähneln. Alle gefundenen Verweise auf externe Dokumente werden in einer Menge zusammengefasst und dann so umgewandelt, dass die Projektspezifikationen einen Tabellenindex bilden und jeweils einer oder mehreren Anforderungen zugeordnet werden.

Der alternative Modus ist speziell für den oben beschriebenen Anwendungsfall konzipiert. Die Evaluierung basiert auf Kundenanforderungstexten aus mehreren Projekten, die manuell etikettiert wurden und deren Daten nicht für das Training des NER-Netzwerks verwendet wurden (siehe Abschnitt 3.3.5). Als zusätzliche Validierung für dieselbe Aufgabe wird die OpenAI-API eingesetzt, um das GPT3.5-Modell programmatisch abzufragen. Das GPT3.5-Modell ist für diese Aufgabe nicht angepasst, sondern führt die Klassifizierung auf der Grundlage eines Systemprompts durch, der eine Liste von Verweisen auf externe Dokumente, Normen und Standards für jedes der folgenden Dokumente anfordert. Es werden auch mehrere Beispiele für die korrekte Klassifizierung von Anforderungstexten gegeben, d. h. diese Methode kann als sogenanntes „Few-Shot-Learning“ bezeichnet werden (Parnami & Lee, 2022).

4.2.2 Ergebnisse im Alternativmodus

Die Evaluierung erfolgt anhand der Metriken Genauigkeit, Sensitivität und F1-Maß. Letzteres ist entscheidend für die Bestimmung der Leistung des Systems im alternativen Modus. Da in diesem Modus die Reihenfolge, in der die Ergebnisse erzeugt werden, irrelevant ist, werden fortgeschrittene Metriken wie MAP nicht verwendet. Für die Evaluierung im Alternativmodus werden genauso wie für den Testfall 1 im Basismodus die Daten aus Projekten P22, P18, P25 und P17 eingesetzt. Um aus der Liste der Kundenanforderungen, die zu den angegebenen Projekten gehören, einen zuverlässigen etikettierten Testdatensatz zu erstellen, wurde eine 20%-ige Stichprobe nach dem Zufallsprinzip gezogen. Anschließend wurden die Nummern der Dokumente aus den Projektspezifikationen mit Hilfe von bereits in der Datenbank vorhandenen Verknüpfungen zu Objekten des Typs "Projektspezifikation" als Etiketten an die Kundenanforderungen angehängt. Die manuelle Analyse der resultierenden Listen ergab eine beträchtliche Anzahl von Verweisen auf externe Dokumente, die in der Datenbank nicht berücksichtigt wurden, so dass weitere Etiketten manuell zum Datensatz hinzugefügt wurden.

Die Ergebnisse mit dem feinabgestimmten NER-Modell und mit dem GPT-3.5-Modell werden in der Tabelle 4.7 dargestellt.

Tabelle 4-7 Ergebnisse der Evaluierung im Alternativmodus

	P22			P18			P25			P17			aggregiert		
	Genauigkeit	Recall	F1-Maß	Genauigkeit	Recall	F1-Maß	Genauigkeit	Recall	F1-Maß	Genauigkeit	Recall	F1-Maß	Genauigkeit	Recall	F1-Maß
NER	0,90	0,93	0,92	0,83	0,89	0,86	0,81	0,85	0,83	0,75	0,89	0,82	0,83	0,90	0,86
GPT-3.5	0,54	0,92	0,68	0,55	0,87	0,67	0,56	0,89	0,69	0,49	0,81	0,61	0,53	0,88	0,66

4.2.3 Diskussion der Ergebnisse im Alternativmodus

Im alternativen Modus ermöglicht es das Empfehlungssystem nicht, andere Parameter außer dem verwendeten Modell zu variieren. Es kann entweder ein fein abgestimmtes NER-Modell oder das API von GPT-Modell verwendet werden. Wie in Tabelle 4.7 zu sehen ist, liegt der Wert der F1-Maß bei Anwendung des NER-Modells zwischen 0,82 und 0,93, während das GPT-3.5-Modell ein niedrigeres Ergebnis von 0,61 bis 0,68 aufweist.

Die Verschlechterung der Leistung mit dem GPT-3.5-Modell ist ausschließlich auf den Unterschied in der Genauigkeitsmetrik zurückzuführen, was darauf hindeutet, dass dieses Modell dazu neigt, mehr Objekte im Anforderungstext, die der Dokumentennummer ähnlich sind, als nötig zu extrahieren. Zum Beispiel ist ein Verweis auf eine einzelne Anforderung einem Verweis auf das gesamte Dokument sehr ähnlich, daher fügt GPT ein solches Objekt der Liste hinzu. Durch eine Feinabstimmung des NER-Modells ist es möglich, dem Modell beizubringen, zwischen solchen Fällen zu unterscheiden. Gleichzeitig führt dies nicht zu einer Verschlechterung der Sensitivität, die im GPT-Modell nur beim P25-Projekt geringfügig besser ist.

Wenn das Ergebnis im alternativen Modus mit dem Ergebnis im Basismodus im Testfall 1 verglichen wird, lässt sich feststellen, dass in beiden Fällen das gleiche Problem gelöst wird, allerdings unter unterschiedlichen Voraussetzungen (siehe Abschnitt 4.1). Die MAP-Metrik aus der Evaluierung vom Basismodus kann nicht direkt mit der Genauigkeit verglichen werden, aber auch wenn ihrer Wert (0,78) als die höchste marginale Bewertung betrachtet wird, ist er niedriger als im Alternativmodus (0,83). Das Gleiche gilt für die Sensitivität (0,90), die im alternativen Modus deutlich höher ist als die Sensitivität im Basismodus (0,80). Der alternative Ansatz ermöglicht es also, gleichzeitig mehr relevante Anforderungen in Bezug auf die Projektspezifikationen zu erfassen und liefert weniger überflüssige Ergebnisse.

4.3 Fazit

Die in diesem Abschnitt durchgeführte Evaluierung des Empfehlungssystems hat bestätigt, dass es über die in Abschnitt 3.1 aufgeführten Fähigkeiten verfügt.

1) Das System ist in der Lage, Anforderungen aus zwei verschiedenen Mengen auf unterschiedlichen Spezifikationsstufen zu verknüpfen, wie anhand von vier verschiedenen Testfällen gezeigt wurde, bei denen das Empfehlungssystem im Basismodus ("Mapper") verwendet wurde. Die Evaluierung wurde in jedem der Testfälle an mehreren Projekten durchgeführt, deren Daten nicht in den Prozess des Trainings der Sprachmodelle einbezogen wurden. Auf diese Weise wurde das Risiko eines verzerrten Ergebnisses reduziert und es kann festgestellt werden, dass zumindest im Bereich der Automobilelektronik die Qualität der Leistung des Empfehlungssystems erhalten bleibt. Die Leistung des Empfehlungssystems fällt nach der aggregierten MAP-Metrik in den komplexesten Testfällen 2-4 nicht unter 0,64, und nach der Sensitivität nicht unter 0,69. Es ist anzumerken, dass die Werte der beiden grundlegenden Metriken je nach Projekt und Testfall über 0,9 oder umgekehrt unter 0,5 liegen können. Diese Beobachtung ist darauf zurückzuführen, dass die Metriken auf der Grundlage der im Originaldatensatz vorhandenen Trace-Links berechnet werden. Je nach Projekt und Art der Spezifikation enthält der Datensatz möglicherweise keine Verknüpfungen zwischen wirklich relevanten Anforderungen oder er enthält Verknüpfungen zwischen irrelevanten Anforderungen oder zwischen Anforderungen, aus deren Text sich ihre Relevanz nicht ableiten lässt. Das Vorhandensein falscher Etiketten sowie deren Unvollständigkeit führen dazu, dass das Ergebnis für einzelne Projekte unterschätzt wird.

Als Ergebnis der Auswertung ist festzustellen, dass die beste Variante des Systems in den Testfällen 2-4 eine gewichtete Aggregation der Ergebnisse der semantischen Suche und der Stichwortsuche ist, wobei das Gewicht der letzteren 25% nicht überschreiten sollte. Für den Testfall 1 liefert die Variante mit reinem Reranker das beste Ergebnis.

2) Das System ist in der Lage, Verweise auf externe Dokumente im Text der Anforderungen zu erkennen. Im alternativen Modus findet das Empfehlungssystem Entitäten, die den Dokumentennummern ähnlich sind, indem es ein Sprachmodell wie Named Entity Recognition anwendet. Das mit manuell etikettierten Daten feinabgestimmte Sprachmodell erreicht bei der Evaluierung von vier Projekten einen F1-Score von mindestens 0,82, während dieser Wert für einzelne Projekte über 0,9 liegt. Bei der Lösung der Aufgabe, Projektspezifikationen und Kundenanforderungen zu verknüpfen, zeigt der alternative Ansatz einen Vorteil gegenüber dem Basismodus, da er ein korrekteres Ergebnis (höhere Genauigkeit) liefert und gleichzeitig mehr Verknüpfungen zwischen relevanten Objekten erfasst (höhere Sensitivitätswerte). Darüber hinaus hat ein Vergleich des fein abgestimmten NER-Modells mit der Anwendung des großen

generativen Modells GPT-3.5 gezeigt, dass man durch das Training des eigenen Modells eine bessere Genauigkeit erzielen und somit die Arbeitskosten für die Analyse der erhaltenen Ergebnisse reduzieren kann.

3) Das System bietet die Möglichkeit, sowohl im Test- als auch im Produktionsmodus zu arbeiten. Das System ermöglicht die Verknüpfung von etikettierten Daten zur Evaluierung der Ergebnisse des Systembetriebs nach verschiedenen Offline-Metriken, je nach Lauf-Modus (Basis oder Alternative). Das Empfehlungssystem funktioniert auf der Grundlage einer YAML-Konfiguration, die die Möglichkeit bietet, verschiedene Hyperparameter zu ändern, die die Funktionsweise des Systems bestimmen. Ihre Aufzählung kann im automatischen Modus erfolgen und ermöglicht es, die besten Hyperparameter für verschiedene Anwendungsfälle zu finden.

4) Das System kann sowohl online als auch im Batch-Modus ausgeführt werden. Die Architektur des Empfehlungssystems ermöglicht es, es auf einem Server als API-Endpunkt einzusetzen, über den der Benutzer sowohl mehrere als auch einzelne Suchanfragen an den Korpus stellen kann.

5) Das System ermöglicht das Laden von Daten aus einer Vielzahl von Quellen. Das Empfehlungssystem kann Eingabedaten sowohl vom OpenSearch-System empfangen, das bereits vorberechnete Vektordarstellungen und Objektmetadaten speichert, als auch die Möglichkeit, Eingabedaten aus separaten Dateien in verschiedenen Formaten zu erhalten.

Das Potenzial für eine weitere Verbesserung des Empfehlungssystems liegt vor allem in der Verbesserung der Qualität der Daten für das Training der Sprachmodelle. Obwohl sowohl die Sentence-Transformer- als auch die Cross-Encoder-Modelle eine durchschnittliche Verbesserung gegenüber den Basismodellen aufwiesen, war diese nicht so signifikant, wie man es sich vor Beginn der Arbeit vorgestellt hatte. Wie die Datenanalyse in Abschnitt 3.3.2 gezeigt hat, ist die Qualität des Anforderungstextes, der dem Prozess der Feinabstimmung der Sprachmodelle zugrunde liegt, in vielen Spezifikationen gering. Unvollständigkeit und Fehlerhaftigkeit der Daten verschlechtern sowohl die Trainingsergebnisse als auch die Modellevaluierungen. Das Beispiel eines starken NER-Modells, das aus einem neu etikettierten Datensatz abgeleitet wurde, zeigt, wie wichtig die Qualität der Etiketten ist. Eine zusätzliche Etikettierung der Anforderungen für das Training der semantischen Such- und Rerankermodelle erfordert jedoch die Einbeziehung von Experten für jede der betrachteten Domänen, was in dieser Phase nicht möglich war.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Empfehlungssystem zur Erstellung von Rückverfolgbarkeit in technischen Spezifikationen entwickelt und anschließend anhand von realen Projekten getestet. Das System ermöglicht es, Anforderungen verschiedener Ebenen und Domänen miteinander zu verknüpfen und in den Anforderungstexten Verweise auf externe Dokumente zu erkennen, die ebenfalls Teil des Projekts sind und deren Anforderungen ebenfalls erfüllt werden müssen. Das System wurde unter Verwendung der neuesten Entwicklungen im Bereich des Deep Learning unter Berücksichtigung bestehender Ressourcen und traditionellerer Methoden der IR entwickelt. Es ist vorgesehen, dass das System als internes Tool für ein Unternehmen der Automobilelektronik eingesetzt wird.

Bevor mit der Entwicklung begonnen wurde, hatte der Autor der Arbeit die einschlägige Literatur zu den Themen Anforderungsmanagement und Verarbeitung natürlicher Sprache studiert und sich direkt mit den neuesten bekannten Fortschritten bei der Lösung des Problems der Wiederherstellung von Trace-Links sowie verwandter Probleme beschäftigt. Die gewonnenen wissenschaftlichen Erkenntnisse wurden in Kapitel 2 dargelegt und dienen als Grundlage für die weitere Entwicklung.

Die Entwicklung des Systems wurde in fünf Teile unterteilt. Zunächst wurden die hochrangigen funktionalen Anforderungen an das Empfehlungssystem ermittelt, d. h. die Fähigkeiten, die es besitzen soll. Auf der Grundlage der ermittelten Anforderungen wurde eine Architektur vorgeschlagen, die es dem System ermöglicht, mit den angegebenen Eigenschaften zu funktionieren. Da die Systemarchitektur auf Deep-Learning-Modellen basiert, die an den betrachteten Bereich angepasst werden müssen, bestand der nächste Schritt in der Analyse der Daten, die zum Trainieren des neuronalen Netzwerks verwendet werden sollten. Bei der Datenanalyse zeigten sich mehrere bevorstehende Schwierigkeiten bei der Umsetzung der Aufgabe: Unvollständigkeit und teilweise Fehlerhaftigkeit der Trace-Links im Datensatz; die Texte der Anforderungen, einschließlich der verknüpften, liegen in verschiedenen Sprachen (Deutsch und Englisch) vor; der Text eines Drittels der Anforderungen besteht aus unvollständigen Sätzen. Nach der Analyse und Bereinigung der Daten von Ausreißern wurden drei Deep-Learning-Modelle trainiert - ein Modell für die semantische Suche (Sentence-Transformer), ein Modell für das Ranking der Ergebnisse (Cross-Encoder) und ein Modell für die Erkennung von Dokumentennummern in Anforderungstexten (Named Entity Recognition). Im nächsten Schritt wurde die Serversoftware des Empfehlungssystems implementiert, um die Eingabedaten in eine Form aufzubereiten, die für die Ausführung der Algorithmen des neuronalen Netzes und der Information Retrieval geeignet ist, und dann die Ergebnisse nach einem bestimmten Da-

tenschema auszugeben. Das Empfehlungssystem kombiniert mehrere Komponenten - semantische Suche, Volltextsuche, Reranker, Erkennung der Dokumentennummern - die zusammen oder getrennt funktionieren können und deren Grad der Beteiligung durch mehrere konfigurierbare Hyperparameter bestimmt wird.

In Kapitel vier wurde das Empfehlungssystem einem gründlichen Test unterzogen. Siebenundzwanzig Kombinationen von Hyperparametern wurden definiert und für jeden der vier Testfälle im Basismodus, bei dem es um die Rückverfolgbarkeit zwischen zwei Anforderungsmengen geht, und einen zusätzlichen Fall im alternativen Modus, bei dem Verweise auf externe Dokumente erkannt werden sollen, ausgeführt. Jeder Testfall wurde in mehreren Projekten ausgewertet, um eine Verzerrung der Schlussfolgerung über die Leistungsfähigkeit des Empfehlungssystems zu vermeiden. Aus der Evaluierung wurden die folgenden Schlüsse gezogen:

1) Das System ist in der Lage, die Rückverfolgbarkeit zwischen zwei Anforderungsmengen mit einer durchschnittlichen Genauigkeit von mindestens 0,65 herzustellen und im Durchschnitt 70 % der relevanten Anforderungspaare abzudecken.

2) Im alternativen Modus ist das Empfehlungssystem in der Lage, durchschnittlich 90 % der Verweise auf externe Dokumente in Anforderungstexten mit einer durchschnittlichen Genauigkeit von 0,83 zu erkennen.

3) Das Ergebnis des Empfehlungssystems kann erheblich verbessert werden, wenn die Qualität der Daten, die dem Training der Sprachmodelle zugrunde liegen, erhöht wird, insbesondere die Verfügbarkeit von validen Etiketten.

Die weitere Entwicklung des Empfehlungssystems sollte sich auf die Verbesserung der Leistung der Sprachmodelle für die semantische Suche und insbesondere des Rerankers konzentrieren. Angesichts der relativ schwachen Leistung des Rerankers ist es sinnvoll, andere Ansätze für sein Training zu erforschen, insbesondere um andere Verlustfunktionen zu testen oder eine Feinabstimmung auf der Grundlage eines anderen Modells durchzuführen. Das größte Verbesserungspotenzial liegt jedoch in den Trainingsdaten, die zur Verbesserung ihrer Qualität die Einbeziehung von Experten aus den entsprechenden Bereichen - Software, Elektrotechnik, Maschinenbau und System Engineering - erfordern. Um effektiv mit ihnen zu kommunizieren, kann eine Feedback-Funktion in das System eingeführt werden, die es dem Benutzer ermöglichen sollte, die von der Anwendung ausgegebene Ergebnisse zu bewerten.

Weitere Bereiche der zukünftigen Entwicklung sind die Erstellung einer Benutzeroberfläche und die Integration des Systems in bereits bestehende Unternehmenstools.

6 Literaturverzeichnis

Bast, H., & Buchhold, B. (2016). Semantic Search on Text and Knowledge Bases. *Foundations and Trends in Information Retrieval Vol. 10, No. 2-3*, S. 119–271.

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python.* Sebastopol, CA: O'Reilly Media.

Bischof, B., & Yee, H. (2023). *Building Recommendation Systems in Python and JAX.* Sebastopol, CA: O'Reilly Media.

Bojanowski, Piotr, Grave, Edouard, Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv:1607.04606*. Von https://arxiv.org/abs/1607.04606 abgerufen

Bussmann, H. (2008). *Lexikon der Sprachwissenschaft.* Stuttgart: Kröner Verlag.

COEST Community. (2017). *COEST Datasets*. Abgerufen am 12. 06 2024 von http://sarec.nd.edu/coest/datasets.html

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., & Wenzek, G. (8. April 2020). Unsupervised Cross-lingual Representation Learning at Scale. *arXiv:1911.02116 [cs.CL]*.

Cui, N. (2018). Applying Gradient Descent in Convolutional Neural Networks. *Journal of Physics: Conference Series 1004 012027*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs.CL]*.

Dick, J., Hull, E., & Jackson, K. (2017). *Requirements Engineering.* Switzerland: Springer International Publishing.

Elastic. (2015). *ElasticSearch Guide*. Abgerufen am 15. 06 2024 von https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

ExplosionAI. (2015). *Spacy*. Abgerufen am 08. 06 2024 von https://spacy.io/

Gage, P. (1994). A New Algorithm for Data Compression. *Dr. Dobb's Journal*.

Gillick, D., Presta, A., & Tomar, G. (2018). End-to-End Retrieval in Continuous Space. *arXiv:1811.08008 [cs.IR]*. Von https://arxiv.org/abs/1811.08008 abgerufen

GitHub. (2022). *PGVector Repository*. Abgerufen am 15. 06 2024 von https://github.com/pgvector/pgvector

Godoy, D. (2021). *Deep Learning with PyTorch Step-by-Step.* Victoria, British Columbia, Canada: Leanpub.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* Cambridge, Massachusetts: MIT Press.

Herlocker, J., Konstan, J., Terveen, L., & Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS), 22*(1), S. 5-53.

Hey, T. (2023). *Automatische Wiederherstellung von Nachverfolgbarkeit zwischen Anforderungen und Quelltext.* Karlsruhe Institut für Technologie.

Hirschberg, J., & Manning, C. (2015). Advances in natural language processing. *Science*, S. 261-266.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural Computation 9(8)*, S. 1735–1780.

Hofstätter, S., Althammer, S., Schröder, M., Sertkan, M., & Hanbury, A. (2020). Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. *arXiv:2010.02666 [cs.IR]*. Von https://arxiv.org/abs/2010.02666 abgerufen

Hugging Face. (2021). *Transformers Documentation*. Abgerufen am 20. 07 2024 von https://huggingface.co/docs/transformers/index

Hugging Face. (2023). *Hugging Face NLP Course*. (Hugging Face, Inc.) Abgerufen am 08. 07 2024 von https://huggingface.co/learn/nlp-course

IEEE STD 1220-2005: IEEE Standard for Application and Management of the Systems Engineering Process. (2005).

ISO/IEC/IEEE 29148-2018: ISO/IEC/IEEE International Standard - Systems and Software Engineering - Life Cycle Processes - Requirements Engineering. (2018).

ISO 26262-8: International Standard. Road Vehicles - Functional Safety. (2011).

Järvelin, K., & Kekäläinen, J. (2002). Cumulated Gain-Based Evaluation of IR Techniques. *ACM Transactions on Information Systems, 20*(4), S. 422-446.

Jupyter Team. (2022). *Project Jupyter Documentation*. Abgerufen am 20. 07 2024 von https://docs.jupyter.org/en/latest/

Jurafsky, D., & Martin, J. (2000). *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Upper Saddle River, NJ: Prentice Hall PTR.

Kossmann, M. (2016). *Requirements management.* New York: Routledge.

Lavrač, N., Podpečan, V., & Robnik-Šikonja, M. (2021). *Representation Learning: Propositionalization and Embeddings.* Switzerland: Springer Nature.

Luan, Y., Eisenstein, J., Toutanova, K., & Collins, M. (2020). Sparse, Dense, and Attentional Representations for Text Retrieval. *arXiv:2005.00181 [cs.CL]*.

Lyons, J. (1991). *Natural Language and Universal Grammar.* New York: Cambridge University Press.

Manning, C., Raghavan, P., & Schütze, H. (2008). *An Introduction to Information Retrieval.* Cambridge, England: Cambridge University Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781*. doi:https://doi.org/10.48550/arXiv.1301.3781

Mills, C., & Haiduc, S. (2017). A Machine Learning Approach for Determining the Validity of Traceability Links. *2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion*, S. 122-123.

Mills, C., Escobar-Avila, J., & Haiduc, S. (2018). Automatic Traceability Maintenance via Machine Learning Classification. *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, S. 369-380.

OpenAI. (2023). *OpenAI developer platform*. Abgerufen am 20. 07 2024 von https://platform.openai.com/docs/overview

OpenSearch. (2024). *Keyword search*. Abgerufen am 20. 06 2024 von https://opensearch.org/docs/latest/search-plugins/keyword-search/

Pachiana, G., Grunwald, M., Markwirth, T., & Sohrmann, C. (2021). *Automated traceability of requirements in the design and verification process of safety-critical mixed-signal systems.* Dresden, Germany : Fraunhofer IIS/EAS.

Pakhale, K. (2023). Comprehensive Overview of Named Entity Recognition: Models, Domain-Specific Applications and Challenges. *arXiv:2309.14084 [cs.CL]*. Von https://arxiv.org/abs/2309.14084 abgerufen

Pandas. (2020). *Pandas Documentation*. Abgerufen am 20. 07 2024 von https://pandas.pydata.org/docs/

Parnami, A., & Lee, M. (2022). Learning from Few Examples: A Summary of Approaches to Few-Shot Learning. *arXiv:2203.04291 [cs.LG]*.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. doi:10.3115/v1/D14-1162

Reimers, N., & Gurevich, I. (2020). Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. *arXiv:2004.09813v2 [cs.CL]*.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv:1908.10084 [cs.CL]*.

Robertson, S., & Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval 3 (4)*, S. 333–389.

Rodriguez, D., & Carver, D. (2019). Comparison of Information Retrieval Techniques for Traceability Link Recovery. *2019 IEEE 2nd International Conference on Information and Computer Technologies*, S. 187-193.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs.CL]*.

Spärck Jones, K. (1972). A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *Journal of Documentation. 28 (1)*, S. 11–21.

Stahl, P. M. (2022). *Lingua-py Repository*. Abgerufen am 10. 06 2024 von https://github.com/pemistahl/lingua-py

The PyTorch Foundation. (2018). *PyTorch documentation*. Abgerufen am 20. 07 2024 von https://pytorch.org/docs/stable/index.html

Tunstall, L., von Werra, L., & Wolf, T. (2022). *Natural Language Processing with Transformers.* Sebastopol, CA: O'Reilly.

UKPLab. (2023). *SentenceTransformers Documentation*. Abgerufen am 01. 04 2024 von https://www.sbert.net/

Umar, M. A., & Lano, K. (2024). Advances in automated support for requirements engineering: a systematic literature review. *Requirements Engineering*. doi:https://doi.org/10.1007/s00766-023-00411-0

Vaswani, A., Shazeer, N., Parmar, N., Uszk, J., Jones, L., Gomez, A., . . . Polosukhin, I. (2017). Attention Is All You Need. *arXiv:1706.03762v7 [cs.CL]*.

VDA Working Group, 1. (2023). *Automotive SPICE Process Assessment / Reference Model 4.0.* Verband der Automobilindustrie e.V.

Voorhees, E. (2000). The TREC-8 Question Answering Track Report. *The Eight Text Retrieval Conference (TREC-8)*, S. 77-82.

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2018). Deep Learning based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys*. doi:https://doi.org/10.1145/3285029

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements: A Systematic Mapping Study. *ACM Computing Surveys, 3*(54).

Zhao, Z., Liu, T., Li, S., Li, B., & Du, X. (2017). Ngram2vec: Learning Improved Word Representations from Ngram Co-occurrence Statistics. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, S. 244–253.

Zhou, Z.-H. (2021). *Machine Learning.* Singapore: Springer Nature Singapore Pte Ltd.

Anhang A Trainingsskripte

Anhang A.1 Trainingsskript für MLM

```
from datasets import Dataset
import pandas as pd
from transformers import (
    AutoModelForMaskedLM,
    AutoTokenizer,
    DataCollatorForLanguageModeling,
    TrainingArguments,
    Trainer,
)

training_data_directory = "project_datasets/training_datasets"
combined arts directory = "project datasets/combined artifacts"

reqs_dataframe = pd.read_csv(
    training data directory + "/tsdae quality train.csv"
).astype(str)

reqs_dataframe = reqs_dataframe.drop_duplicates(subset=["ObjectText"])

reqs_dataset = Dataset.from_pandas(reqs_dataframe.set_index("ObjectID"))

model_checkpoint = "FacebookAI/xlm-roberta-base"

tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)

def tokenize_function(examples):
    result = tokenizer(
        examples["ObjectText"], truncation=True, padding="max_length",
max_length=128
    )
    if tokenizer.is_fast:
        result["word_ids"] = [
            result.word_ids(i) for i in range(len(result["input_ids"]))
        ]
    result["labels"] = result["input_ids"].copy()
    return result


tokenized_reqs_dataset = reqs_dataset.map(
    tokenize_function, batched=True, remove_columns=["ObjectID", "Ob-
jectText"]
)
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer, mlm_probability=0.15
```

```
)
train_size = 80_000
test_size = int(0.1 * train_size)

mlm_reqs_dataset = tokenized_reqs_dataset.train_test_split(
    test_size, train_size, seed=42
)

model = AutoModelForMaskedLM.from_pretrained(model_checkpoint)

batch_size = 12
logging_steps = len(mlm_reqs_dataset["train"])
model_name = model_checkpoint.split("/")[-1]

training_args = TrainingArguments(
    output_dir=f"{model_name}-reqs",
    overwrite_output_dir=True,
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    weight_decay=0.01,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    push_to_hub=False,
    logging_steps=logging_steps,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=mlm_reqs_dataset["train"],
    eval_dataset=mlm_reqs_dataset["test"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)

trainer.train()
```

Anhang A.2 Trainingsskript für mehrsprachiges Modell

```
import os
from datetime import datetime

import numpy as np

from sentence_transformers import (SentenceTransformer, models,
                                   evaluation, losses)
from torch.utils.data import DataLoader
from sentence_transformers.datasets import ParallelSentencesDataset

teacher_model_name = "paraphrase-distilroberta-base-v2"
student_model_name = "./xlm-roberta-base-reqs/checkpoint-20000"

max_seq_length = 128
train_batch_size = 4
inference_batch_size = 8
train_max_sentence_length = 250
num_epochs = 5
num_warmup_steps = 10000
num_evaluation_steps = 1000

teacher_model = SentenceTransformer(teacher_model_name)
word_embedding_model = models.Transformer(
    student_model_name, max_seq_length=max_seq_length
)
pooling_model = models.Pooling(
      word_embedding_model.get_word_embedding_dimension()
)
student_model = SentenceTransformer(modules=
            [word_embedding_model, pooling_model]
)

train_data = ParallelSentencesDataset(
    student_model=student_model,
    teacher model=teacher model,
    batch_size=inference_batch_size,
    use_embedding_cache=True,
)

parallel_sentences_path = \
            "project_datasets/training_datasets/parallel_en_de_ted.csv"

train_data.load_data(
    parallel_sentences_path,
    max_sentences=500000,
    max_sentence_length=train_max_sentence_length,
```

```
)
train_dataloader = DataLoader(train_data, shuffle=True,
batch_size=train_batch_size)
train_loss = losses.MSELoss(model=student_model)
test_file_path = "project_datasets/training_datasets/paral-
lel_en_de_ted_test.csv"

src_sentences = []
trg_sentences = []
with open(test_file_path, "rt", encoding="utf8") as fIn:
    for line in fIn:
        splits = line.strip().split("\t")
        if splits[0] != "" and splits[1] != "":
            src_sentences.append(splits[0])
            trg_sentences.append(splits[1])

test_mse = evaluation.MSEEvaluator(
    src_sentences,
    trg_sentences,
    name=os.path.basename(test_file_path),
    teacher_model=teacher_model,
    batch_size=inference_batch_size,
)

filepath = "project_datasets/benchmarks/STS.en-de.txt"
filename = os.path.basename(filepath)
sts_data = {"sentences1": [], "sentences2": [], "scores": []}

with open(filepath, "rt", encoding="utf8") as fIn:
    for line in fIn:
        sent1, sent2, score = line.strip().split("\t")
        score = float(score)
        sts_data["sentences1"].append(sent1)
        sts_data["sentences2"].append(sent2)
        sts_data["scores"].append(score)

test_similarity_evaluator = evaluation.EmbeddingSimilarityEvaluator(
    sts_data["sentences1"],
    sts_data["sentences2"],
    sts_data["scores"],
    batch_size=inference_batch_size,
    name=filename,
    show_progress_bar=False,
)

evaluators = [test_mse, test_similarity_evaluator]
output_path = f'output/multilingual_{datetime.strftime(datetime.now(),
format="%Y%m%d_%H%M")}'
```

```
student_model.fit(
    train_objectives=[(train_dataloader, train_loss)],
    evaluator=evaluation.SequentialEvaluator(
        evaluators, main_score_function=lambda scores: np.mean(scores)
    ),
    epochs=num_epochs,
    warmup_steps=num_warmup_steps,
    evaluation_steps=num_evaluation_steps,
    output_path=output_path,
    save_best_model=True,
    optimizer_params={"lr": 2e-5, "eps": 1e-6},
)
```

Anhang A.3 Trainingsskript für Sentence-Transformer-Modell

```
from datetime import datetime

import pandas as pd
from sentence_transformers import (SentenceTransformer, losses,
                                   InputExample, evaluation)
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split

training_data_directory = 'project_datasets/training_datasets'
combined_arts_directory = 'project_datasets/combined_artifacts'
link_pairs_directory = 'project_datasets/link_pairs'

triplets = pd.read_csv(training_data_directory + '/P00000_triplets.csv')

model = SentenceTransformer('./output/multilingual_20240517_2345',
device='cuda')

triplets records = triplets[[
        'ObjectText_a', 'ObjectText_b_p', 'ObjectText_b_n'
    ]
].to dict(orient='records')

train_triplet_set, test_triplet_set = train_test_split(triplets_records,
                                                       test_size=0.2,
                                                       random_state=99)
train_examples = [InputExample(texts=[record['ObjectText_a'],
                                      record['ObjectText_b_p'],
                                      record['ObjectText_b_n']])
                                      for record in train_triplet_set]

train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=32)
```

```
train_loss = losses.MultipleNegativesRankingLoss(model=model)

anchors = [record['ObjectText_a'] for record in test_triplet_set]
positives = [record['ObjectText_b_p'] for record in test_triplet_set]
negatives = [record['ObjectText_b_n'] for record in test_triplet_set]

evaluator = evaluation.TripletEvaluator(anchors,
                                        positives,
                                        negatives,
                                        batch_size=32,
                                        show_progress_bar=True)
output_path = f'output/triplet_{datetime.strftime(datetime.now(),
format="%Y%m%d_%H%M")}'

model.fit(
    [(train_dataloader, train_loss)],
    output_path=output_path,
    evaluator=evaluator,
    evaluation_steps=round(len(train_dataloader)/5),
    save_best_model=True,
    optimizer_params={"lr": 2e-5, "eps": 1e-6},
    warmup_steps=1000,
    epochs=12,
    checkpoint_path = output_path,
    checkpoint_save_steps = 5000,
    checkpoint_save_total_limit = 10
)
```

Anhang A.4 Trainingsskript für Cross-Encoder-Modell

```
from datetime import datetime
import math

import pandas as pd
from sentence transformers import CrossEncoder, InputExample
from sentence_transformers.cross_encoder.evaluation import CECorrelation-
Evaluator
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split

training_data_directory = 'project_datasets/training_datasets'
combined_arts_directory = 'project_datasets/combined_artifacts'
link_pairs_directory = 'project_datasets/link_pairs'

positive_pairs = pd.read_csv(training_data_directory +
                   '/P00000_positive_pairs_cross_train.csv').astype(str)
negative_pairs = pd.read_csv(training_data_directory +
                    '/P00000_negative_pairs_cross_train.csv').astype(str)
```

```
model = CrossEncoder('cross-encoder/msmarco-MiniLM-L12-en-de-v1',
                     num_labels=1,
                     max_length=256,
                     device='cuda')

all_pairs = pd.concat([positive_pairs, negative_pairs])[['ObjectText_a',
                                                         'ObjectText_b',
                                                         'scaled_score']]

all_pairs['scaled_score'] = all_pairs['scaled_score'].astype(float)

train_pairs_set, test_pairs_set = train_test_split(all_pairs,
                                                   test_size=0.2,
                                                   random_state=99,
                                                   shuffle=True)

train_examples = [InputExample(texts=[record['ObjectText_a'],
                                      record['ObjectText_b']],
                                      label=record['scaled_score'])
                  for record in train_pairs_set.to_dict(orient='records')]

train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=24)
test_examples = [InputExample(texts=[record['ObjectText_a'],
                                      record['ObjectText_b']],
                                      label=record['scaled_score'])
                  for record in test_pairs_set.to_dict(orient='records')]

evaluator = CECorrelationEvaluator.from_input_examples(test_examples)

num_epochs = 6
warmup_steps = math.ceil(len(train_dataloader) * num_epochs * 0.1)
output_path = f'output/cross_encoder_{datetime.strftime(datetime.now(),
                                    format="%Y%m%d_%H%M")}'

model.fit(
    train_dataloader,
    output_path=output_path,
    evaluator=evaluator,
    evaluation_steps=round(len(train_dataloader)/5),
    save_best_model=True,
    optimizer_params={"lr": 2e-5, "eps": 1e-6},
    warmup_steps=warmup_steps,
    epochs=num_epochs
)
```

Anhang A.5 Trainingsskript für Named-Entity-Recognition-Modell

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from datasets import Dataset, DatasetDict
from transformers import (AutoTokenizer,
                          AutoModelForTokenClassification,
                          TrainingArguments, Trainer)
from transformers import DataCollatorForTokenClassification
from seqeval.metrics import (accuracy_score,
                             classification_report,
                             f1_score)

train_ner_dataset = pd.read_excel(
    'output/ner_results/customer_psl_extraction.xlsx'
    )

p1_ner_dataset = pd.read_excel('metrics_data/Labels_P1.xlsx')
p2_ner_dataset = pd.read_excel('metrics_data/Labels_P2.xlsx')
p3_ner_dataset = pd.read_excel('metrics_data/Labels_P3.xlsx')

train_ner_dataset = train_ner_dataset.reindex(
    columns=[col for col in train_ner_dataset.columns
             if not col.startswith('Unnamed:')])

train_ner_dataset = pd.concat([train_ner_dataset,
                               p1_ner_dataset,
                               p2 ner dataset,
                               p3_ner_dataset])

train_ner_dataset['Label'] = train_ner_dataset['Label'].str.strip()
train ner dataset aggregated = train ner dataset.fillna('') \
            .groupby(['ObjectID', 'ObjectText']) \
            .agg({'Label':list}).reset_index()

train_ner_dataset_aggregated['Label'] = \
      train_ner_dataset_aggregated['Label'] \
                  .apply(lambda x: [s for s in x if s])

train_full =
      train_ner_dataset_aggregated[
            train_ner_dataset_aggregated['Label']
                        .apply(lambda x: len(x) > 0)
            ]
train_empty = train_ner_dataset_aggregated[
      train_ner_dataset_aggregated['Label'].apply(lambda x: len(x) == 0)]
```

```
train_empty_ner, _ = train_test_split(
                train_empty, train_size=len(train_full), random_state=99,
shuffle=True)
train_ner_dataset_aggregated = pd.concat([train_full, train_empty_ner])

model_checkpoint = './xlm-roberta-base-reqs/checkpoint-20000'

tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)

def find_occurence(tokenized_sentence, tokenized_entity):
    pos = 0
    blueprint = [-100] + [0 for _ in range(len(tokenized_sentence)-2)] + [-
100]
    for pos in range(len(tokenized_sentence)):
        found_positions = []
        search_pointer = pos
        for element in tokenized_entity:
            if search_pointer == len(tokenized_sentence) \
                        or element != tokenized_sentence[search_pointer]:
                found_positions = []
                break
            found_positions.append(search_pointer)
            search_pointer += 1
        for i, found_pos in enumerate(found_positions):
            if i == 0:
                blueprint[found_pos] = 1
            else:
                blueprint[found_pos] = 2
    return blueprint

def find_all_occurencies(sentence, entities, tokenizer):
    tokenized_sentence = tokenizer.tokenize(sentence,
                                            truncation=True,
                                            add_special_tokens=True)
    agg_blueprint = [-100] + [0 for _ in range(len(tokenized_sentence)-2)]
+ [-100]
    if not entities:
        return agg_blueprint
    blueprints = []
    for entity in entities:
        if isinstance(entity, str):
            blueprints.append(find occurence(tokenized sentence,
                                             tokenizer.tokenize(en-
tity)))
    for blueprint in blueprints:
        assert len(agg_blueprint) == len(blueprint)
        for pos in range(len(agg_blueprint)):
            if agg_blueprint[pos] == 0 and blueprint[pos] != 0:
```

```
                agg_blueprint[pos] = blueprint[pos]
    return agg_blueprint

train_ner_dataset_aggregated['labels'] =
      train_ner_dataset_aggregated.apply(
         lambda row: find_all_occurencies(row['ObjectText'], row['Label'],
                                                    tokenizer),
                                                    axis=1)

train_ner_set, test_ner_set = train_test_split(
       train_ner_dataset_aggregated[['ObjectID', 'ObjectText', 'labels']],
                test_size=0.2, random_state=99, shuffle=True)

doc_numbers_train_test = DatasetDict({
    "train": Dataset.from_pandas(train_ner_set),
    "test": Dataset.from_pandas(test_ner_set)
    })

def tokenize_function(examples):
    return tokenizer(examples['ObjectText'], truncation=True)

tokenized_datasets = doc_numbers_train_test.map(
    tokenize_function, batched=True,
    remove_columns=['ObjectID', 'ObjectText', '__index_level_0__']
)

for item in tokenized_datasets['train']:
    print(len(item['input_ids']), len(item['labels']))
    assert len(item['input_ids']) == len(item['labels'])

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
label_names = ['O', 'B-DOC', 'I-DOC']

def compute_metrics(p):
  predictions, labels = p
  predictions = np.argmax(predictions, axis=2)
  true_predictions = [
      [label_names[p] for (p, l) in zip(prediction, label) if l != -100]
      for prediction, label in zip(predictions, labels)
  ]
  true_labels = [
      [label names[l] for (p, l) in zip(prediction, label) if l != -100]
      for prediction, label in zip(predictions, labels)
  ]
  results = {
      'accuracy': accuracy_score(true_labels, true_predictions),
      'f1': f1_score(true_labels, true_predictions),
      'classification_report': classification_report(
```

```
                                    true_labels, true_predictions)
  }
  return results

id2label = {i: label for i, label in enumerate(label_names)}
label2id = {v: k for k, v in id2label.items()}

model = AutoModelForTokenClassification.from_pretrained(
    model_checkpoint,
    id2label=id2label,
    label2id=label2id,
)
batch_size = 12
logging_steps = len(tokenized_datasets["train"])

args = TrainingArguments(
    "xlm-reqs-doc-numbers-v3",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    num_train_epochs=9,
    weight_decay=0.01,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    logging_steps=logging_steps,
    push_to_hub=False
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer
)
trainer.train()
```