UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

PALUNO
The Ruhr Institute for Software Technology

Institut für Informatik und Wirtschaftsinformatik (ICB)
Lehrstuhl für Software Engineering, insb. mobile Anwendungen
Prof. Dr. Volker Gruhn

Techniken der Computerlinguistik zur Verbesserung von Suchfunktionen in der Software-Entwicklung

Improving Search Functions in Software Engineering using Computer Linguistics

Bachelorarbeit

vorgelegt der Fakultät für Informatik
der Universität Duisburg-Essen (Campus Essen) von

Leon Zimmermann
Laddringsweg 8
45219 Essen
Matrikelnummer: 3080384

Essen, den 9. Oktober 2023

Betreuung:	Wilhelm Koop, Sascha Feldmann
Erstgutachter:	Prof. Dr. Volker Gruhn
Zweitgutachter:	Prof. Dr. Klaus Pohl
Studiengang:	Angewandte Informatik - Systems Engineering (B. Sc.)

Abstract

Software developers often need to gather Information to continue with their work. An example of this might be the implementation of a feature, where the software developer needs to understand the intent of the feature to be implemented. Such information could be found in a knowledge base. To find information in a knowledge base, search engines can be used. This is especially helpful, when the software developer does not know beforehand, where the needed information is to be found in the knowledge base. However, search engines don't always find the information that the software developer needs.

As part of this Bachelor's Thesis, a semantic search engine is developed. The development of a semantic search engine aims to be an improvement on existing search engines, which use TF-IDF or BM25 for the scoring of documents. An example of such search engines is the search engine of the knowledge base Confluence. To determine whether or not the newly developed search engine resembles an *improvement* over the existing search engine of Confluence, this thesis discusses methods and metrics for the evaluation of search engines. The search engine of Confluence and the newly developed search engine are then compared on the basis of the previously discussed methods and metrics. For that, exemplary queries are formulated, as well as the expected documents to be returned by the search engines. The queries are based on Use Cases that have been previously formulated. These Use Cases describe situations, in which it is realistic for a software developer to be using the search engine of a knowledge base.

The study showed, that for the generated dataset the semantic search has a lower precision value compared to a BM25 search. It is argued, that the results can be explained by the fact, that the dataset that was used, belongs to a specific domain and contains many technical terms from that specific domain. Now, a sentence transformer, which is used to index the documents for the semantic search, is trained on a general purpose dataset containing general purpose terms only. That said, the sentence transformer could not have learned the semantic of the technical terms from that specific domain. Finally the Bachelor's Thesis mentions approaches to adapt sentence transformers to a specific domain, increasing the precision of the semantic search.

Zusammenfassung

Softwareentwickler müssen sich bei ihrer Arbeit Informationen zusammensuchen, welche sie für die weitere Arbeit benötigen. So muss ein Softwareentwickler bei der Implementierung eines Features die Intention des Features kennen. Solche Informationen können in Wissensdatenbanken hinterlegt sein. Um dort die gewünschten Informationen zu finden, können Suchfunktionen verwendet werden. Insbesondere, wenn dem Softwareentwickler im Vorfeld nicht klar ist, wo er die gewünschten Informationen in der Wissensdatenbank finden kann. Aber nicht im-

mer liefert diese Suchfunktion die gewünschten Informationen.

In dieser Arbeit wird eine semantische Suchfunktion entwickelt. Diese soll eine Verbesserung zu bestehenden Suchfunktionen bieten, welche TF-IDF oder BM25 für das Scoring von Ergebnissen verwenden, wie beispielsweise die Suchfunktion von Confluence. Um festzustellen, ob die neue Suchfunktion *besser* ist als die bestehende, werden Methoden und Kriterien zur Evaluierung von Suchfunktionen erörtert. Anhand dieser Kriterien werden die neu implementierte Suchfunktion und die bestehende Suchfunktion von Confluence in einer Studie verglichen. In der Studie werden beispielhafte Sucheingaben definiert, sowie die erwarteten Ergebnisse. Die Definition der Sucheingaben erfolgt auf Basis von Anwendungsfällen. Die Anwendungsfälle beschreiben Situationen, in denen es realistisch ist, dass ein Softwareentwickler die Suchfunktion einer Wissensdatenbank verwendet.

Die Studie hat gezeigt, dass für den generierten Datensatz eine semantische Suche eine geringere Precision besitzt, als eine BM25 Suche. Das Ergebnis wird damit begründet, dass es sich bei dem Datensatz um eine spezifische Domäne mit vielen Fachbegriffen handelt. Ein Sentence Transformer, welcher dazu verwendet wird die Daten für die semantische Suche zu indizieren, wurde nur auf allgemeinen Datensätzen mit allgemeinen Begriffen trainiert. Dadurch hat der Sentence Transformer noch nicht die Semantik der Domäne gelernt. Zuletzt nennt die Arbeit Ansätze, um Sentence Transformer an eine spezifische Domäne anzupassen, und die Precision der semantischen Suche damit zu erhöhen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ANN Approximate Nearest Neighbor

HNSW Hierarchical Navigable Small Worlds

NLP Natural Language Processing

VSM Vector Space Model

1. Einleitung

Für die tägliche Arbeit benötigt ein Softwareentwickler Informationen, welche über den Code, an dem er arbeitet, hinausgehen. Um an diese Informationen zu kommen, kann der Softwareentwickler eine Person suchen, welche ihm die gewünschte Information geben kann. Diese Person kann er im Projekt finden, oder über Websites, wie StackOverflow. Darüber hinaus wird in vielen Projekten eine Wissensdatenbank angelegt, welche Informationen enthält, welche spezifisch für das Projekt sind. Eine solche Wissensdatenbank ist Confluence. Sie bietet eine Suchfunktion, welche es dem Softwareentwickler erleichtern soll, die gewünschten Informationen zu finden. Beispiele für Informationen sind die Spezifikationen oder Dokumentationen eines Teiles der Software, mit welcher der Softwareentwickler gerade arbeitet. Oder aber auch Best-Practices, Guides, oder Informationen darüber, wie die Software gestartet oder ausgeliefert wird. Auch Informationen über den Projektplan sind für einen Softwareentwickler von Bedeutung.

Aber nicht immer finden Softwareentwickler die gewünschten Informationen mithilfe der Suchfunktion. Ziel dieser Arbeit soll es sein, mithilfe von Wissen über Suchalgorithmen und Algorithmen aus dem Natural Language Processing zu zeigen, wie sich bestehende Suchfunktionen verbessern lassen. Software, wie chatGPT[1] zeigt, dass Large Language Models eine valide Möglichkeit zur Information Extraction sind. Es ist denkbar, dass Suchfunktionen für Softwareentwickler durch Large Language Models verbessert werden können. Diese Arbeit erörtert, wie die Technologien von Large Language Models verwendet werden können, um eine semantische Suche zu implementieren. Die implementierte semantische Suche wird mit der bestehenden Confluence-Suche verglichen, um herauszufinden, ob die semantische Suche eine Verbesserung zu der bestehenden Confluence-Suche darstellt.

1.1. Vorgehensweise

Die Gründe, warum eine gewünschte Information schwierig zu finden ist, sind vielzählig. Manchmal kennt der Softwareentwickler nicht das genaue Wording, um die gewünschten Informationen zu finden. Manchmal ist das Abstraktionslevel der gefundenen Informationen nicht das, welches sich der Softwareentwickler gewünscht hat. Beispielsweise, wenn eine allgemeine Definition von Domänenobjekten gesucht wird, aber eine Spezifikation eines Anwendungsfalls gefunden wird, in welchem das Domänenobjekt lediglich erwähnt wird. Im Rahmen dieser Arbeit wird eine neue Suchfunktion entwickelt, welche eine Verbesserung zu bestehenden

[1] https://openai.com/blog/chatgpt

Suchfunktionen von Wissensdatenbanken, wie Confluence, darstellen soll. Nachdem die neue Suchfunktion entwickelt wurde, wird diese mit einer bestehenden Suchfunktion verglichen, um zu untersuchen, ob die neue Suchfunktion besser ist als eine bestehende Suchfunktion. Dazu werden objektive Metriken zur Messung von Suchfunktionen dargestellt. Die einzelnen Schritte für die Entwicklung und Evaluierung der neuen Suchfunktion werden im Folgenden dargestellt:

- **Definition von Anwendungsfällen:** Es werden zuerst Anwendungsfälle definiert. Die Anwendungsfälle beschreiben die konkreten Situationen, in welchen ein Softwareentwickler eine Suche nutzen könnte. Das hilft später dabei Suchfunktionen miteinander zu vergleichen. Denn anhand der Anwendungsfälle können realistische Sucheingaben definiert werden. Diese Sucheingaben können an zwei verschiedene Suchfunktionen übergeben werden. Anschließend können die gefundenen Ergebnisse verglichen werden. Die genaue Vorgehensweise für diesen Vergleich wird in Kapitel 5 erklärt.

- **Technologien von Confluence:** Als Nächstes wird die Technologie von Confluence dargestellt. Das hilft dabei später die Konzeption einer neuen Suchfunktion zu entwickeln und zu begründen, warum die neue Technologie besser sein soll. Im weiteren Verlauf der Arbeit wird erläutert, dass für den Vergleich der Suchfunktionen nicht die tatsächliche Confluence-Suchfunktion als Benchmark für die neue Suchfunktion herangezogen wird. Stattdessen wird die neue Suchfunktion so entwickelt, dass sie so konfiguriert werden kann, dass sie eine gute Heuristik für die Confluence-Suchfunktion darstellt. Dieses Kapitel erfüllt ebenfalls den Zweck eine vergleichbare Suchfunktion entwickeln zu können, welche anhand der objektiven Kriterien gemessen werden kann. Auch bei der Auswertung der Ergebnisse der Suchfunktion sind die Inhalte aus diesem Kapitel relevant.

- **Konzeption der neuen Suchfunktion:** Nachdem die Technologien von Confluence erörtert wurden, wird in diesem Kapitel die Theorie für die Implementierung einer Suchfunktion erläutert. Es wird zuerst erklärt, dass die neue Suchfunktion eine semantische Suchfunktion ist. Dann wird erläutert, was eine semantische Suche ist, und warum eine semantische Suche eine Verbesserung gegenüber den Technologien von Confluence darstellen soll. Anschließend wird erklärt, welche Technologien für die Implementierung einer semantischen Suche notwendig sind. Zuletzt wird die Implementierung der semantischen Suche mithilfe von der Vektordatenbank Weaviate[2] erklärt.

- **Vergleich der Suchfunktionen:** In diesem Kapitel werden zuerst Methoden für die Evaluierung von Suchfunktionen herausgearbeitet. Damit wird die Frage beantwortet, wann eine Suchfunktion *gut* ist. Das hier erläuterte Wissen wird für die Durchführung der Studie benötigt. Denn um festzustellen, ob die Implementierung eine Verbesserung darstellt, muss eine Studie durchgeführt werden. Aufgrund des Scopes der Arbeit wird nur eine rudimentäre Studie durchgeführt. Die Studie vergleicht die Precision-Werte der

[2] https://weaviate.io/

Confluence-Suchfunktion und der neuen Suchfunktion mithilfe eines generierten Datensatzes von Sucheingaben und erwarteten Ergebnissen. Grundlage für die Generierung dieses Datensatzes sind die Anwendungsfälle, welche zuvor definiert wurden. Die Ergebnisse werden dargestellt und diskutiert. Es wird gezeigt, dass eine semantische Suche für die generierten Daten und den spezifischen Kontext der Datengrundlage, einen schlechten Precision-Wert besitzt. Dann wird dieses Ergebnis interpretiert und es wird der Schluss gezogen, dass Sentence Transformer besser geeignet sind für Open-Domain Datensätze, im Gegensatz zu Closed-Domain Datensätzen. Zuletzt wird der Studienaufbau diskutiert.

- **Zusammenfassung und Ausblick:** In diesem Kapitel werden die einzelnen Kapitel der Arbeit nochmal zusammenfassend dargestellt. Außerdem wird das Ergebnis der Arbeit zusammenfassend dargestellt. In Kapitel 6.2 werden Ansätze erörtert, welche die Suchergebnisse von semantischen Suchen in spezifischen Domänen verbessern sollen.

1.2. Verwandte Arbeiten

Es gibt einige Arbeiten, welche die gleichen oder sehr ähnliche Probleme adressieren. Die verschiedenen Herangehensweisen werden in diesem Kapitel erläutert. Zuerst wird der Begriff Traceability behandelt, und wie diese genutzt wird, um Suchfunktionen zu verbessern. Die Herangehensweisen zur Herstellung von Traceability umfassen die Verwendung von Word Embeddings oder auch ein Task-Based Ansatz. Es folgt eine Übersicht über Arbeiten, welche sich mit Question Answering beschäftigen. Dann folgt eine Übersicht über Arbeiten, welche sich mit der Evaluierung von Suchfunktionen auseinandersetzen.

Traceability bedeutet, dass sich von einer Stelle im Code, auf die entsprechenden Stellen in anderen Artefakten zurückschließen lässt. Ein Anwendungsfall für eine solche Traceability-Funktionalität ist *Feature Location*, also das Finden der Spezifikation eines Features, wenn nur der Code vorhanden ist. Analog dazu ist *Bug Localization* ein Anwendungsfall zum Auffinden von Stellen im Code, welche mit einem Bug zusammenhängen. Zur Herstellung von Traceability zwischen Code und Dokumentation gibt es verschiedene Ansätze. Haiduc et al. (2013) schlagen ein System zur Verbesserung von Sucheingaben vor. Das System verwendet Query Reformulations, um die Traceability herzustellen. Query Reformulation bedeutet, dass das System den Softwareentwickler bei der Eingabe einer Suchanfrage zur Suche nach den passenden Artefakten unterstützt. Dazu gibt der Softwareentwickler zunächst eine Suchanfrage ein, und markiert diejenigen Ergebnisse, welche am relevantesten für ihn sind. Auf Grundlage der gewählten Ergebnisse und mithilfe eines Machine Learning Algorithmus werden nun Vorschläge für eine verbesserte Suchanfrage gemacht. Dabei gibt es verschiedene Strategien. Wenn der Softwareentwickler zu Beginn eine sehr lange Suchanfrage eingegeben hat, dann kann das System eine Reduktion der Suchanfrage vorschlagen. Hat der Softwareentwickler

dagegen lediglich einen Suchbegriff angegeben, so kann das System eine Erweiterung der Suchbegriffe vorschlagen. Dazu greift das System auf Synonyme des eingegebenen Suchbegriffes zurück.

Ye et al. (2016) beschreiben, wie Word Embeddings dazu verwendet werden können, um Traceability zwischen Code und anderen Softwareentwicklungs-Artefakten herzustellen. Word Embeddings sind eine Datenstruktur, welche einem Wort einen Vektor in einem n-dimensionalen Raum zuweist. Anhand dieses Vektors kann die Ähnlichkeit zwischen Wörtern beschrieben werden. Ähnliche Wörter haben eine geringe Distanz im n-dimensionalen Raum. Unähnliche Wörter haben eine hohe Distanz. Der Algorithmus, welcher die Ähnlichkeit der Wörter bestimmt, macht Gebrauch von der Distributional Hypothesis (Harris 1954). Dieser besagt, dass Wörter, welche im gleichen Kontext verwendet werden, eine ähnliche Semantik besitzen. Hiermit wird also die Ähnlichkeit der Wörter bestimmt. Dieses Verfahren wird nun sowohl auf den Code angewendet als auch auf die Softwareentwicklungs-Artefakte.

Antoniol et al. (2000) verwenden einen ähnlichen Ansatz, wie Ye et al. Auch hier werden Word Embeddings verwendet um Softwareentwicklungs-Artefakte gegen den Code zu matchen. Hier durchlaufen die Artefakte und der Code zwei verschiedene Pipelines. Die Wörter der Artefakte in natürlicher Sprache werden in lowercase umgewandelt. Anschließend werden Stoppwörter entfernt. Zuletzt werden Flexionen entfernt. Aus dem Code werden zunächst Identifier extrahiert. Identifier, welche mehrere Wörter unter Verwendung von CamelCase oder snake_case beinhalten, werden in die einzelnen Wörter aufgeteilt. Anschließend werden die Identifier auf die gleiche Art und Weise normalisiert, wie die Wörter der Softwareentwicklungs-Artefakte. Dann erfolgt sowohl für die Identifier als auch für die Wörter aus den Artefakten die Indizierung, also die Umwandlung in Word Embeddings.

Treude et al. (2015) entwickeln eine Oberfläche, welche die Suche von *Tasks* ermöglicht. Damit soll ebenfalls Traceability zwischen Code und Dokumentation hergestellt werden. Dabei ist unter Task eine Operation im Code zu verstehen. Sie beschreiben einen Task als Verben, welche mit einem direkten Objekt oder einer Präposition in Verbindung stehen. Die Autoren nennen die Phrasen *get iterator* und *get iterator for collection* als Beispiele. Die Software analysiert nun die gesamte Dokumentation und extrahiert Tasks. Die Tasks werden in einen Index geschrieben, sodass der Softwareentwickler nach ihnen suchen kann.

Neben der Traceability zwischen Code und Dokumentation ist das Question Answering ein Ansatz zur Verbesserung von Suchfunktionen. Suchfunktionen sind Document Retrieval Systeme. Sie liefern Dokumente, welche zu der Sucheingabe des Nutzers passen. Document Retrieval Systeme sind eine Unterkategorie von Information Retrieval Systemen. Information Retrieval Systeme liefern auf Anfrage Informationen an den Nutzer. Im Fall einer Suchfunktion werden Dokumente gelie-

fert, welche diese Information beinhalten. Mithilfe der Dokumente ist der Ort, an dem sich die, vom Nutzer gewünschte, Information befindet eingegrenzt. Nichtsdestotrotz muss der Nutzer aus dieser Eingrenzung die gewünschte Information manuell extrahieren. Ganz im Gegensatz zu Question Answering Systemen. Zhang et al. (2023) untersuchen verschiedene Herangehensweisen zur Implementierung von Open-Domain Question Answering Systemen. Open-Domain Question Answering Systeme beantworten allgemeine Fragen eines Nutzers, z.B. basierend auf Informationen von Wikipedia. Closed-Domain Question Answering Systeme beantworten dagegen Fragen im Kontext einer spezifischen Domäne, z.B. basierend auf unternehmensinternen Informationen.

Für die Messung der Performance einer Suchfunktion gibt es verschiedene Metriken. Metriken können systemspezifisch sein, oder nutzerspezifisch. Systemspezifische Metriken messen die Performance anhand objektiver Kriterien, welche an dem System gemessen werden können. Nutzerspezifische Metriken messen die Performance dagegen anhand subjektiver Kriterien. Die Messung erfordert Probanden, welche die Suchfunktion verwenden. Die Probanden legen die Performance der Suchfunktion anhand der zu betrachtenen subjektiven Kriterien fest. Clarke und Willet (1997) messen die Qualität von Suchfunktionen des World Wide Webs anhand des Recalls. Um dies zu ermöglichen wird ein Datensatz generiert, welcher Sucheingaben beinhaltet, sowie alle relevanten Dokumente für eine Sucheingabe. Bar-Ilan (2002) verwendet die gleichen Metriken. Hier wird darüber hinaus auf die Problematik der Messung des Recalls eingegangen. Es wird erläutert, dass zur Messung des Recalls a-priori bestimmt werden muss, welche Dokumente als relevant für eine gegebene Sucheingabe erachtet werden sollten. Gordon und Pathak (1999) behaupten, dass die Bestimmung der Relevanz lediglich dem Nutzer mit dem Bedürfnis nach der Information zu überlassen ist. Voorhees und Harman (2001) behaupten, dass die Bestimmung der Relevanz durch ein Experten-Panel durchgeführt werden sollte. Sirotkin (2012) betrachtet verschiedene Ansätze zur Messung der Performance von Suchfunktionen. Neben den bereits genannten Metriken von Precision und Recall werden andere Metriken zur Messung der Performance einer Suchfunktion genannt, wie Mean Reciprocal Rank und Maximal Marginal Relevance.

2. Definition von Anwendungsfällen

In diesem Kapitel werden Anwendungsfälle ausgewählt, welche eine Grundlage für die spätere Evaluation der Suchfunktionen bilden. Die Anwendungsfälle beschreiben die Situationen, in denen Softwareentwickler die Suchfunktionen von Wissensdatenbanken verwenden könnten. Zur Identifikation von Anwendungsfällen wurde zunächst Literatur herangezogen. Die Literatur ist bereits unter den verwandten Arbeiten aufgeführt. In den verwandten Arbeiten wurden Arbeiten genannt, welche ähnliche Probleme lösen sollen. Diese fokussieren sich vorallem auf die *Feature Location*, *Bug Localization* und die Traceability zwischen Code und anderen Artefakten. Aus diesen Themen ließ sich der Anwendungsfall identifizieren, welcher in Kapitel 2.2 erörtert wird. Außerdem konnten noch weitere Anwendungsfälle ermittelt werden. Dazu wurde die Wissensdatenbank eines Softwareprojektes untersucht. So ist das Onboarding im Projekt ein Anwendungsfäll für die Verwendung der Suchfunktion einer Wissensdatenbank. Ein weiterer Anwendungsfall ist das Auffinden von Informationen über das Projektmanagement.

Die folgenden Kapitel beschreiben die genannten Anwendungsfälle und erläutern, warum sie als Anwendungsfälle ausgesucht wurden. Bei dem Vergleich zwischen Suchfunktionen werden die Anwendungsfälle herangezogen, um Sucheingaben und erwartete Dokumente zu generieren. Der genaue Aufbau der Studie wird in Kapitel 5.2 erklärt.

2.1. Onboarding im Projekt

Wenn ein neuer Softwareentwickler in einem Softwareprojekt startet, dann muss er sich zunächst einmal mit dem Projekt vertraut machen. Das bedeutet, dass er verstehen muss, was das Projekt eigentlich ist. Er muss verstehen, was das eigentliche Problem des Kunden ist. Außerdem muss er verstehen wie die Software dieses Problem löst.

Für das Onboarding im Projekt muss der Softwareentwickler sehr allgemeine Informationen über das Projekt finden können. Er könnte Dinge suchen, wie einen Projektüberblick oder ein Glossar. Neben diesen allgemeinen Informationen muss sich der neue Softwareentwickler mit dem Code vertraut machen. Er muss verstehen, welche Technologien verwendet werden, welche Best-Practices, Code-Styles, Guidelines, Prozesse und Quality Gates eingehalten werden sollten. Und er muss verstehen, wie die Software lokal oder in einer Testumgebung ausgeführt werden kann.

Die Leitung eines Projektes wünscht sich eine möglichst schnelle Einarbeitung von neuen Mitarbeitern. Dazu können bereits etablierte Mitarbeiter herangezogen werden, welche den neuen Mitarbeiter bei der Einarbeitung unterstützen. Der Nachteil besteht darin, dass hierdurch Kapazitäten gebunden werden, welche für die aktive Entwicklung der Software benötigt werden. Daher kann das Zurückgreifen auf eine Wissensdatenbank durch den neuen Mitarbeiter sinnvoll sein. Dazu kann der neue Mitarbeiter das Inhaltsverzeichnis der Wissensdatenbank verwenden, wenn es vorhanden ist. Dieses hilft dem Mitarbeiter dabei Informationen zu bestimmten Themenbereichen zu finden. Es grenzt die Antwort auf eine Frage auf einen bestimmten Bereich ein, so wie es auch eine Suchfunktion tut. Nichtsdestotrotz bietet eine Suchfunktion die Möglichkeit dieses Inhaltsverzeichnis automatisch auf Basis einer Sucheingabe zu durchlaufen. Damit findet der neue Mitarbeiter schneller die Informationen, die er sucht. Darüber hinaus kann die Suchfunktion spezifischere Ergebnisse liefern. So kann die Suchfunktion dem Nutzer bereits die relevantesten Stellen in den relevantesten Dokumenten liefern. Und die Suchfunktion kann dem Nutzer die Informationen aus diesen relevantesten Stellen zusammenfassen. Voraussetzung hierbei ist eine Suchfunktion, welche gut darin ist die relevantesten Dokumente und die relevantesten Stellen aus diesen Dokumenten zu extrahieren. Im weiteren Verlauf wird erörtert, welche Ansätze es gibt, um diese Voraussetzung zu erfüllen, und wie aus den Dokumenten die relevantesten Stellen mithilfe von Passage Retrieval ermittelt werden können. Außerdem wird erläutert, wie Retrieval Augmented Generation die gefundenen Informationen aufbereitet.

2.2. Implementierung nach Spezifikation

Bei der Implementierung von neuen Anforderungen ist es wichtig, dass sich der Softwareentwickler an die Spezifikation hält. Nur so bekommt der Kunde die Software, die er sich gewünscht hat. Dazu sollte der Softwareentwickler alle relevanten Dokumente finden, die zu der Spezifikation dazugehören. Zuerst sollte er die Feature-Spezifikation selbst finden können. Er sollte die Dokumentation der damit einhergehendenden Prozesse finden, und auch die Domänenobjekte, welche für die Implementierung relevant sind. Er sollte Diagramme finden können, welche zu dem Anwendungsfall gehören, und auch die weiteren Dokumente, welche den Kontext der Anforderung erläutern.

Auch nachdem ein Feature nach Spezifikation umgesetzt wurde, ist es weiterhin wichtig die Spezifikation einfach leicht zu können. Der Abgleich mit einer Spezifikation ist notwendig, um Testfälle zu schreiben, und zu prüfen, ob das Verhalten der Anwendung korrekt ist. Es gibt Fehler, welche erkennbar sind, ohne dafür die Spezifikation heranzuziehen. Eine Anwendung, welche einfriert oder abstürzt ist ein Beispiel für einen solchen Fehler. Dennoch können Fehler auch domänen- und kontextspezifisch sein, wodurch die Spezifikation als Referenz notwendig ist. Die Spezifikation zum Abgleich mit dem Verhalten der Software heranzuziehen

ist besonders in Randfällen hilfreich. Angenommen ein Online-Shop bietet einen kostenlosen Versand ab einem Mindestbestellwert an. Sobald der Preis den Wert von 25$ übersteigt, ist der Versand gratis. Der Versand kostet im Normalfall 5$. Nun muss definiert werden, ob der Versand in dem Mindestbestelltwert miteinbezogen wird oder nicht. Wird er miteinbezogen, dann sorgt das dafür, dass ein Produkt, welches 20$ kostet einen kostenlosen Versand hat, weil der Mindestbestellwert zuzüglich des Versandpreis 25$ beträgt. Ob dieses Verhalten gewünscht ist oder nicht, muss in der Spezifikation festgehalten werden. Wenn nun ein Bugticket dieser Art bei einem Softwareentwickler landet, dann muss er, wie auch bei der Implementierung, alle relevanten Dokumente für die weitere Entwicklung heranziehen.

2.3. Informationen über das Projektmanagement finden

Softwareentwickler sollten darüber Bescheid wissen, was in dem Projekt, in dem sie arbeiten, vor sich geht. Sie sollten über organisatorische Dinge Bescheid wissen, wie die Teamaufteilung und die Zuständigkeiten in den einzelnen Teams und in dem gesamten Projekt. Das ist wichtig für eine gute Kommunikation und entsprechenden Wissensaustausch zwischen Kollegen. Außerdem sollten Softwareentwickler darüber Bescheid wissen, wann das nächste Release der Software ansteht. Diesen Termin sollten sie bei der Planung ihrer Arbeit berücksichtigen, damit sie auch rechtzeitig alle relevanten Features implementiert haben und alle kritischen Fehler behoben haben. Neben der Einhaltung von Terminen ist die Planung wichtig für die Motivation der Softwareentwickler. Softwareentwickler sollten die Vision der Software verstehen können und auch die übergreifende Vision des Unternehmens. Sie sollten verstehen, warum die Arbeit, welche sie erledigen, so wichtig ist. Und sie sollten verstehen, für wen sie diese Arbeit tun. Auch diese Faktoren sind wichtig für eine hohe Motivation bei der Arbeit. Daher ist es so wichtig diese Informationen einfach zugänglich zu machen, also einfach auffindbar zu machen.

In Kapitel 5.1 wird aufgezeigt, wie die Anforderungen der Anwendungsfälle quantifizierbar gemacht werden können. Das wird dabei helfen nachzuvollziehen, inwieweit die Anforderungen der Anwendungsfälle erreicht wurden, welche genannt wurden.

3. Technologien von Confluence

Dieses Kapitel hat den Zweck die Funktionsweise der Suchfunktion von Confluence zu erläutern. Dies ist notwendig, um später die Ergebnisse des Vergleichs der Suchfunktion von Confluence mit der neuen Suchfunktion zu verstehen. Um die Suchfunktion von Confluence zu verstehen wird zuerst erklärt, welche Suchmethoden mit den Technologien von Apache Lucene ermöglicht werden und in Confluence verwendet werden können. Dann wird die zugehörige Technologie erklärt, indem zuerst ein Überblick über die wichtigsten Komponenten einer Suchfunktion gegeben wird. Als Nächstes werden die spezifischen Technologien erläutert, welche Apache Lucene verwendet. Das bedeutet, dass zuerst erklärt wird, was ein Volltext-Index ist. Anschließend werden die Scoring-Algorithmen TF-IDF und BM25 erörtert.

3.1. Suchmethoden von Confluence

Dieses Kapitel stellt die Arten von Sucheingaben vor, welche durch Apache Lucene ermöglicht werden. Es ist wichtig diese zu kennen, um das Qualitätskriterium später bei der Implementierung erfüllen zu können. Es gibt die gängigen Suchalgorithmen Keyword Search, Phrase Search, Boolean Search, Field Search.

Eine Keyword Search durchsucht Dokumente nach der Sucheingabe des Nutzers. Die Eingabe wird dabei nicht als Ganzes betrachtet, sondern jedes Wort einzeln. Für jedes Keyword werden Dokumente als Ergebnis angezeigt, wenn dieses in dem Dokument vorhanden ist. Wenn ein Dokument mehrere der Keywords beinhaltet, wird dessen Relevanz höher eingeschätzt als für Dokumente, welche weniger Keywords enthalten. Dokumente mit höherer Relevanz werden weiter oben in der Ergebnisliste angezeigt.

Eine Phrase Search ist die Suche nach Textausschnitten in Dokumenten. Hier werden nicht mehrere Keywords einzeln betrachtet, sondern die gesamte Eingabe in das Suchfeld als eine Einheit. Es reicht also nicht mehr aus, dass ein Dokument eines der Wörter enthält. Es muss die gesamte Sucheingabe als ein String enthalten sein.

Die Boolean Search bietet die Möglichkeit einen boolschen Ausdruck als Sucheingabe zu machen. Ein Beispiel dafür ist die Sucheingabe *Dokumentation AND Angular*. Die Sucheingabe bedeutet, dass die Suchfunktion nur Dokumente als Ergebnis darstellen soll, welche beide Keywords Dokumentation und Angular enthalten. Die Boolean Search kann auch eine Phrase, wie bei der Phrase Search, beinhalten: *"Dokumentation von Software" AND Angular*. In diesem Beispiel wer-

den nur Dokumente als Ergebnis nur angezeigt, wenn sie den gesamten String *Dokumentation von Software* enthalten, sowie das Keyword *Angular*. Bei einer Boolean Search können die boolschen Operatoren *AND*, *OR*, *NOT* beliebig kombiniert werden.

Eine Field Search sucht Dokumente anhand von Attributen. Der Nutzer kann diese Attribute auswählen. Wenn der Nutzer beispielsweise ein Dokument sucht, welches am 01.01.2005 erstellt wurde, dann kann die Eingabe der Suche so aussehen: *erstelldatum: 01.01.2005*. Es können beliebig viele Attribute verwendet werden, um die Suche einzugrenzen. Neben der Verwendung der Attribute für die Suche selbst können die Attribute komplementär zu einer anderen Art von Suche verwendet werden. So kann eine Suchfunktion Buttons bereitstellen, über welche Filter festgelegt werden. Nun kann eine Keyword Search durchgeführt werden, aber die gefundenen Dokumente werden mithilfe der Filter weiter eingeschränkt. Neben diesen gängigen Suchalgorithmen gibt es weitere Suchalgorithmen, wie die strukturierte Suche und die semantische Suche.

3.2. Architektur einer Suchfunktion

Dieses Kapitel gibt einen Überblick über die wichtigstes Komponenten einer Suchfunktion. Eine einfache Implementierung einer Suchfunktion kann aus drei Komponenten bestehen. Aus dem Crawling, dem Index und dem Scoring-Algorithmus. Das Crawling ist zuständig für das Finden von Dokumenten (Castillo 2005). Der Index speichert Informationen der Dokumente, und die Suche ist zuständig für das Verstehen der Nutzeranfrage und die Abfrage der relevantesten Informationen aus dem Index, sowie dessen Verarbeitung und Darstellung. Der Begriff Dokument wird hier verwendet, um die Dateien zu beschreiben, welche durch einen Crawler gesucht und durch den Index verarbeitet werden. Unter Dokumenten können auch eine Website verstanden werden, welche durch einen Webcrawler durchsucht werden. Für die Implementierung der Suche wird ein Scoring-Algorithmus benötigt. Der Scoring-Algorithmus bestimmt, welche Dokumente am besten auf eine Sucheingabe passen.

Für die Implementierung einer Suchfunktion wird zunächst ein Datensatz von Dokumenten benötigt, welche über die Suchfunktion gefunden werden können. Dieser wird mithilfe eines Crawlers aufgebaut. Ein Crawler ist ein Algorithmus, welcher Techniken aus dem Natural Language Processing nutzt, um Informationen aus einem Dokument zu extrahieren (Khder 2021). Implementierungen können reguläre Ausdrücke verwenden, um die Informationen zu extrahieren, oder auch fortgeschrittenere Verfahren, wie Abstract Syntrax Trees. Im Falle von Websites kann der Crawler Hyperlinks zu weiteren Websites extrahieren. Damit kann der Algorithmus sukzessive den Datensatz von Dokumenten befüllen. Die neuen Dokumente werden durch den Index verarbeitet und wiederum auf neue Links analysiert. Dieses Verfahren kann beliebig lange und beliebig rekursiv durchlaufen

werden, um den Index zu erweitern. Idealerweise, ohne Seiten dabei mehrfach zu durchlaufen. Neben dem Crawling können Indizes befüllt werden, indem eine Liste von Dokumenten übergeben werden, welche dem Index hinzugefügt werden sollen.

Die Indizierung von Dokumenten kann sowohl mit einer Volltext-Indizierung oder auch einer Vektor-Indizierung umgesetzt werden. Ein Volltext-Index bestimmt den Score anhand von Ausschnitten aus dem Volltext. Zum einen gibt es Boolean Models im Document Retrieval, welche boolsche Algebra nutzen, um relevante Dokumente zu bestimmen. Das Ergebnis ist eine Boolean Search, wie in Kapitel 3.1 beschrieben. Zum anderen gibt es Probablistische Modelle im Document Retrieval. Ein Vektorindex berechnet Vektoren anhand des Ursprungstextes. Es gibt zwei Modelle des Document Retrievals, welche Vektorindizes benutzen: Sparse Vectors und Deep Vectors. Die Berechnung von Sparse Vectors kann mithilfe von tf-idf erfolgen, wie in Kapitel 3.4 beschrieben. Die Berechnung von Deep Vectors kann wiederum mithilfe von Sentence Transformers erfolgen, wie in Kapitel 4.3 beschrieben.

3.3. Volltext-Indizierung

Nachdem ein Überblick über die Komponenten einer Suchfunktion gegeben wurde, werden nun die Technologien erläutert, welche Apache Lucene bereitstellt. Apache Lucene verwendet einen Volltext-Index. Eine Möglichkeit zur Umsetzung einer Volltext-Indizierung ist der invertierte Index. Zur Generierung des invertierten Indexes müssen die zu indizierten Dokumente zunächst in einer Datenbank abgelegt werden. Ein invertierter Index wird generiert, indem alle Wörter extrahiert werden, welche in den Dokumenten vorkommen. Nun werden diese Wörter in eine Liste geschrieben, und jedem Wort wird zugeordnet, in welchem Dokument sich dieses Wort wiederfinden lässt. Diese Liste wird invertierter Index genannt, weil nicht die Wörter den Dokumenten zugeordnet sind, sondern die Dokumente den Wörtern. Es wird ebenfalls gespeichert, an welchen Stelle des Dokuments das Wort vorkommt, und auch in wie vielen Dokumenten ein Wort vorkommt.

Bei der Indizierung der Wörter besteht nun die Problematik, dass gleiche Wörter in unterschiedlichen Formen existieren können. So stammen *Heizung* und *heizen* beide von dem gleichen Wortstamm *heiz* ab. Um bei der Indizierung Speicherplatz zu sparen, können Wörter auf diesen Wortstamm reduziert werden, damit sie als ein einziges Wort betrachtet werden können. Die Bildung des Wortstamms wird als Stemming bezeichnet. Beim Stemming kann es jedoch zu Overstemming und Understemming kommen. Overstemming bedeutet, dass zwei Wörter, die eigentlich nichts miteinander zu tun haben, also nicht semantisch gleich sind, den gleichen Wortstamm besitzen und als ein Wort betrachtet werden. Ein Beispiel hierfür sind die Wörter *Wand* und *wandere*, wie in *ich wandere*. Beide besitzen den Wortstamm *wand* und werden entsprechend als ein Wort betrachtet. Understemming bedeutet, dass zwei Wörter, die eigentlich etwas miteinander zu tun haben, also

semantisch gleich sind, nicht den gleichen Wortstamm besitzen und dadurch als zwei verschiedene Wörter betrachtet werden. Ein Beispiel hierfür sind die Wörter *absorbieren* und *Absorption*, welche die Wortstämme *absorb* und *absorp* besitzen. Es gibt Techniken zur Vermeidung solcher Probleme, wie der Einsatz vollständiger morphologischer Analysekomponenten. Hierauf soll aber nicht weiter eingegangen werden.

3.4. Sparse Vector Modelle

Da nun die Dokumente indiziert sind, kann ein Scoring-Algorithmus bestimmen, welche Dokumente die passendsten für eine gegebene Sucheingabe sind. Apache Lucene verwendet für das Scoring von Dokumenten die TF-IDF Metrik. TF-IDF untersucht, wie häufig Terme in einem Dokument vorkommen und wie charakteristisch diese Terme für dieses Dokument sind (Manning et al. 2019). Kommt ein gesuchter Term häufig in einem Dokument vor, dann erhöht dies den Score für dieses Dokument. Kommt dieser Term selten in anderen Dokumente vor, dann erhöht dies den Score ebenfalls. Umgekehrt ist der Score für ein Dokument niedriger, je seltener der gesuchte Term in dem Dokument vorkommt oder je häufiger der Term in allen anderen Dokumenten vorkommt. Der tf-Teil steht für *term frequency* und wird berechnet, indem für das jeweilige Dokument bestimmt wird, wie häufig das Wort in dem Dokument vorkommt. Damit die Metrik nicht abhängig von der Länge des Dokumentes ist, wird dieser Wert durch die insgesamte Anzahl der Vorkommnisse des Wortes dividiert. Damit ist diese Metrik relativ zur insgesamten Anzahl der Vorkommnisse des Wortes, und nicht absolut. Der idf-Teil steht für *inverse document frequency*. Er wird berechnet indem die insgesamte Anzahl der Dokumente durch die Anzahl der Dokumente dividiert wird, welche das Wort enthalten. Das Ergebnis des dividierens wird an die Logarithmus-Funktion übergeben, sodass am Ende der idf-Wert berechnet wurde. Die beiden Werte werden miteinander multipliziert. Das Ergebnis ist die TF-IDF-Metrik:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

$$\text{tf}_{t,D} = \frac{\#_{t,D}}{\max_{t' \in D}(\#_{t',D})}$$

$$\text{idf}_t = \log \frac{N}{\text{df}_t}$$

Neben TF-IDF-Metrik wurde die BM25-Metrik entwickelt, welche Apache Lucene ebenfalls bereitstellt. BM25 ist eine probablistische Scoring Metrik. Sie verwendet ebenfalls die inverse Dokumentenfrequenz zur Ermittlung des Scores von Dokumenten. So wie bei der TF-IDF-Metrik stellt tf hier die Termfrequenz dar. Die Variablen k_1, k_3 und b stellen Hyperparameter dar, welche entsprechend der Datenbank und der Sucheingaben angepasst werden können. Die Variable dl ist die Länge des Dokumentes und die Variable avdl die durchschnittliche Länge der

Dokumente. Q ist die Sucheingabe, welche die Terme T enthält. Die Variable qtf wird definiert als die Termfrequenz innerhalb einer Domäne, aus welcher sich die Sucheingabe ableitet (Beaulieu et al. 2000):

$$\sum_{T \in Q} w^{(1)} \frac{(k_1 + 1)\mathrm{tf}}{K + \mathrm{tf}} \frac{(k_3 + 1)\mathrm{qtf}}{k_3 + \mathrm{qtf}}$$

$$K = k_1((1 - b) + b.\mathrm{dl}/\mathrm{avdl})$$

4. Konzeption der neuen Suchfunktion

Dieses Kapitel dient der Konzeption einer neuen Suchfunktion. Die neue Suchfunktion gebraucht eine andere Technologie als die Suchfunktion von Confluence. Während die Confluence-Suche ein Boolean Model und TF-IDF nutzt, ist die neue Suchfunktion eine semantische Suche. Im Folgenden wird zuerst erörtert, was eine semantische Suche ist. Dann wird erklärt, warum die neue Suchfunktion eine bessere Performance erzielen soll als die bestehende Confluence-Suche. Anschließend werden die Technologien erläutert, welche benötigt werden, um die konzipierte semantische Suche zu implementieren. Dies umfasst Sentence Transformer. Sie indizieren Dokumente, indem sie die Sätze in Vektoren transformieren, welche die Semantik der Sätze abbilden. Außerdem umfasst dies Vektorindizes, welche die generierten Vektoren persistent speichern. Zuletzt umfasst dies auch die Bestimmung der relevantesten Dokumente. Dazu werden Hierarchical Navigable Small Worlds (HNSW) benutzt. Nachdem die Technologien erörtert wurden, wird die Umsetzung der Technologien mithilfe von Weaviate erläutert.

4.1. Semantische Suche

Unter einer semantischen Suche wird im Allgemeinen verstanden, dass die Suche nicht nur eine syntaktische Suche einer Zeichenkette ist. Stattdessen verwenden semantische Suchen Techniken, um die Bedeutung der Sucheingabe nachzuvollziehen (Dengel 2012). Eine semantische Suche hat den Zweck, die Ähnlichkeit und die Beziehungen zwischen Wörtern zu verstehen. Sie kennt Homonyme, Synoyme und Antonyme von Wörtern. Anders als bei den bereits betrachteten Volltext-Indizes. So wird durch sie beispielsweise die Ähnlichkeit von den Wörtern *rollout* und *deployment* abgebildet, und dass diese Wörter oft im gleichen Kontext verwendet werden.

Eine Möglichkeit zur Umsetzung einer semantischen Suche ist die Verwendung von Transformers und Vektordatenbanken. Um zu verstehen, welche Wörter kontextuell zusammengehören, werden hier die Wörter in einem n-dimensionalen Raum positioniert. Wörter, die sich sehr ähnlich sind, also im gleichen Kontext verwendet werden, haben in diesem n-dimensionalen Raum eine geringe Distanz zueinander. Wörter, die sich eher unähnlich sind, wie *"rollout" und "API"*, haben eine größere Distanz. Der Vorteil einer semantischen Suche ist, dass der genaue Begriff, welcher gesucht wird nicht bekannt sein muss. Auch ein Synonym des gesuchten Begriffes reicht aus, um das passende Dokument zu finden. Wenn sich der Nutzer also über ein Thema informieren möchte, mit welchem er nicht gut vertraut ist, dann kann die semantische Suche hilfreich sein. Denn der Nutzer kann nun einen Begriff eingeben, der zu dem Thema passt, und den er kennt. Er findet

anschließend Dokumente, welche vielleicht nicht genau diesen Begriff beinhalten, aber welche thematisch dennoch ähnlich sind. Genau dieser Vorteil soll bei der Implementierung später genutzt werden.

Ein Sentence Transformer erhält als Input eine große Menge an Text und mappt die einzelnen Wörter auf einen Vektor einer bestimmten Länge. Die Länge wird durch das Modell des Transformers vorgegebenen. Der Vektor, der am Ende herauskommt, beschreibt die Position des Wortes in dem n-dimensionalen Raum. Der Vektor beschreibt gewissermaßen, wie stark ein Wort in eine abstrakte Kategorie einzuordnen ist. Jeder Wert im Vektor entspricht einer Kategorie. Mithilfe der Vektoren können verschiedene Wörter hinsichtlich ihrer Ähnlichkeit analysiert werden. Ähnliche Wörter habe eine große räumliche Nähe, während zwei Wörter, die in vollkommen unterschiedlichen Kontexten verwende werden eine sehr große Distanz im Raum besitzen. Nehmen wir für ein Beispiel einen dreidimensionalen Raum an. Die x-Achse ist beschriftet mit dem Wort „Tier“, die y-Achse ist beschriftet mit dem Wort „Computer“ und die z-Achse ist beschriftet mit dem Wort „Mensch“. Nun geben wir einem Transformer das Wort „Katze“, und der Transformer berechnet einen dreidimensionalen Vektor, welcher das Wort „Katze“ im Raum positioniert. Weil eine Katze ein Tier ist, ist der X-Wert des Vektors eins. Der Wert eins bedeutet, dass das Wort vollständig zu dieser Kategorie gehört. Da eine Katze überhaupt nichts mit einem Computer zu tun hat, ist der Y-Wert des Vektors 0.

Nun ist eine Katze kein Mensch, aber eine Katze ist ein Haustier von Menschen. Es ist denkbar, dass die Wörter Katze und Mensch oft im gleichen Kontext verwendet werden, sodass der Wert bei 0,3 liegen könnte. Damit der Transformer einen Vektor berechnen kann, braucht er eine Menge Daten. Diese Daten erhält er aus vielen Texten. Werden zwei Wörter oft im gleichen Text genannt oder kommen zwei Wörter in vielen Texten sehr nahe beieinander vor, dann geht der Transformer davon aus, dass die beiden Wörter ähnlich sind, und berechnet ähnliche Vektoren. Zuvor müssen die Texte allerdings bereitgestellt werden. Dazu kann beispielsweise das Internet gecrawlt werden. Die Ergebnisse des Transformers werden in einer Vektordatenbank gespeichert. Eine Vektordatenbank ist eine Datenbank, welche Vector Embeddings, also ein Objekt als Key und dessen Vektor als Value speichert. Bei dem Objekt kann es sich um Wörter handeln, dann wird auch von Word Embeddings gesprochen. Es können aber auch Daten andere Daten, wie Bilder, Videos oder Audio gespeichert werden. Der Zweck von Vektordatenbanken ist es, Daten nicht einfach linear zu speichern, sondern in einem Raum. Die Distanz zwischen zwei Einträgen in diesem Raum beschreibt dessen Ähnlichkeit. Genau diese Informationen nutzen semantische Suchen.

4.2. Wahl der neuen Suchfunktion

Dieses Kapitel betrachtet verschiedene Quellen, um zu zeigen, welches Verbesserungspotenzial die Verwendung von Sentence Transformern im Document Retrieval bietet. Zuerst wird anhand der Arbeit von Choudhary et al. (2020) gezeigt, wie das Scoring im Document Retrieval durch die Verwendung von Bert verbessert wird. Anschließend wird die Arbeit von Karpukhin et al. (2020) herangezogen, um zu verstehen, warum es zu einer Verbesserung im Document Retrieval kommt, wenn Sentence Transformer verwendet werden. Die Argumente von Karpukhin et al. werden dann durch Berger et al. (2000) unterstützt. Zuletzt werden die Argumente für die Verwendung von Sentence Transformern ebenfalls mithilfe von Benchmark-Werten gestützt.

Choudhary et al. (2020) entwickelten ein Document Retrieval System, welches Bert zur Generierung von Embeddings verwendet. Der Scoring-Algorithmus des Systems kombiniert Bert und TF-IDF, um den Score zu berechnen. Laut dem Paper bietet eine Kombination aus TF-IDF und Bert zur Implementierung eines Document Retrieval Systems signifikante Performance Verbesserungen gegenüber einem Document Retrieval System, welches lediglich TF-IDF verwendet. Die Arbeit von Choudhary et al. bieten damit einen ersten Anhaltspunkt darauf, dass Document Retrieval Systeme auf Basis von TF-IDF mithilfe von Sentence Transformern verbessert werden können. Die Performance Verbesserung wird in dem Paper an dem MS Marco Datensatz gemessen.[1] Die nachgewiesene Verbesserung ist Grund zur Annahme, dass auch die Confluence-Suche durch die Verwendung von Sentence Transformern verbessert werden kann. Denn die Confluence-Suche verwendet laut Dokumentation Apache Lucene (Atlassian 2023). Und laut der Dokumentation von Apache Lucene, verwendet dieses einen Scoring-Algorithmus, welcher sowohl auf VSM als auch auf Boolean Models basiert (Foundation 2023). Im Information Retrieval sind Boolean Models jene Scoring-Algorithmen, welche auf boolscher Algebra basieren. Aus der Dokumentation von Apache Lucene und aus dem genannten Paper geht ebensfalls hervor, dass die Vektoren des VSM mit TF-IDF berechnet werden.

Karpukhin et al. (2020) entwickelten einen Dense-Passage Retriever im Kontext des Open-Domain Question Answering. Ein Dense Passage Retriever bestimmt den Textabschnitt eines gegebenen Textes, welcher mit größter Wahrscheinlichkeit eine Antwort auf eine gestellte Frage beinhaltet. Der Input für einen Dense Passage Retriever sind Dokumente, welche zuvor mithilfe eines Scoring-Algorithmus aus einem Index extrahiert wurden. Karpukhin et al. (2020) zeigen, dass das entwickelte Passage-Retrieval System besser darin ist relevante Textabschnitte zu finden als der BM25 Algorithmus. Sie nennen die semantische Verknüpfung von Synonymen und Paraphrasierungen mit unterschiedlichen Tokens als Vorteil gegenüber BM25 und auch TF-IDF. Das Argument lässt sich auf das Thema dieser Arbeit übertragen. Das bedeutet, dass Sentence Transformer gegenüber TF-IDF

[1] `https://microsoft.github.io/msmarco/`

und BM25 den Vorteil haben, dass sie Synonyme und Paraphrasierungen semantisch verknüpfen können. Damit lässt sich die bessere Performance von Sentence Transformern gegenüber TF-IDF und BM25 im Kontext des Document Retrieval begründen.

Berger et al. (2000) bezeichnen die Problematik der Synonyme im Kontext von TF-IDF und BM25 als die Lexical Gap. Sie erklären, dass Scoring-Algorithmen, wie TF-IDF lediglich eine lexikalische Analyse der Dokumente und der Sucheingabe durchführen. Dies ermöglicht jedoch nicht die Abbildung von Synonymen. Wenn der Nutzer beispielsweise nach *NLP* sucht, dann erwartet der Nutzer Dokumente über Natural Language Processing. Wenn nun der Begriff *NLP* als solches aber nicht explizit in den Dokumenten genannt wird, sondern lediglich in der ausgeschriebenen Form *Natural Langauge Processing*, dann können die Dokumente mithilfe des TF-IDF Scoring-Algorithmus nicht gefunden werden.

Die genannten Quellen zeigen, dass semantische Suchen grundsätzlich den Vorteil gegenüber TF-IDF und BM25 besitzen, dass sie Synonyme verstehen können, und damit relevante Dokumente finden können, welche mit TF-IDF und BM25 nicht gefunden werden können. Nun muss untersucht werden, ob dieser Vorteil sich auch in besseren Messwerten im Document Retrieval wiederspiegelt. Dazu werden nun die Benchmark-Werte für die semantische Suche von Weaviate herangezogen. Weaviate ist eine Vektordatenbank, welche Sentence Transformer nutzt, um Dokumente zu indizieren, und HNSW als Scoring-Algorithmus zur Ermittlung des passendsten Dokumentes. Die Performance des Scoring-Algorithmus wurde durch Weaviate anhand von Benchmarks überprüft. Dabei wird der Recall für das erste Element, die ersten zehn Elemente und die ersten 100 Elemente gemessen. Zusätzlich ist angegeben, wie Weaviate konfiguriert ist und welcher Datensatz benutzt wird. Die Benchmark zeigt, dass HNSW für den Datensatz SIFT1M einen Recall-Wert von 90.91% besitzt (Weaviate 2023). Nachdem nun die Vorteile von semantischen Suchen beschrieben wurden, werden als Nächstes die Technologien beschrieben, welche notwendig sind, um eine solche semantische Suche zu implementieren.

4.3. Sentence Transformer

Ein Sentence Transformer transformiert eine Aneinanderreihung von Tokens beliebiger Länge in einen Vektor einer vorgegebenen Länge. Der Aneinanderreihung der Tokens kann dabei ein Satz sein, ein Teil eines Satzes oder auch ein ganzer Paragraph. Die Länge des Vektors wird durch die Architektur des Sentence Transformers vorgegeben. Um eine Aneinanderreihung von Tokens in einen Vektor zu transformieren, braucht der Sentence Transformer für jedes Wort in dessen Wortschatz einen Input. Da ein neuronales Netzwerk keine Wörter verarbeiten kann, sondern nur Zahlen, müssen die Wörter zuerst in Zahlen umgewandelt werden. Dies erfolgt mithilfe von Word Embeddings und wird in Kapitel 4.3.1 erläutert.

Wenn nun die Semantik eines Satzes in einen Vektor transformiert werden soll, dann ist die Reihenfolge der Wörter in dem Satz wichtig zu beachten, um die Semantik des Satzes abzubilden. Der Satz *Software-Entwickler verwenden die Spezifikation, um Features korrekt zu implementieren* besitzt die gleichen Wörter, wie der Satz *Features verwenden die Spezifikation, um Software-Entwickler korrekt zu implementieren*. Dennoch ist die Sematik zwischen den Sätzen nicht die gleiche. Das bedeutet, dass ein Sentence Transformer die Positionen der einzelnen Wörter in dem Vektor kodieren muss, um die Sematik des Satzes korrekt abzubilden. Die Kodierung der Positionen der Wörter in einem Satz wird als Positional Encoding bezeichnet und wird in Kapitel 4.3.2 erklärt. Zuletzt wird Self-Attention auf die Kodierung der Wörter angewendet. Dieser Schritt wird in Kapitel 4.3.3 erklärt. Die beschriebenen Schritte bilden zusammengefasst den Encoder eines Transformers. Er generiert einen Vektor einer fixen Länge für eine beliebig lange Eingabe. Der generierte Vektor wird auch als Context Vector bezeichnet. Bert wurde entwickelt, um eine gute automatische Übersetzung zwischen Sprachen zu ermöglichen. Die Architektur von Bert umfasst neben dem Encoder, welcher einen Context Vector generiert, auch einen Decoder, welcher diesen Context Vector wieder in einen Satz einer anderen Sprache umwandelt. Im Rahmen dieser Arbeit ist nur die Betrachtung des Encoders relevant. Denn dieser generiert bereits die Context Vectors, und diese werden anschließend in einer Vektordatenbank gespeichert.

4.3.1. Word Embeddings

Word Embeddings sind eine Vektorrepräsentation von Wörtern. Zweck von Word Embeddings ist es, semantische Ähnlichkeiten zwischen Wörtern zu verstehen. Eine einfache Implementierung von Word Embeddings könnte sein, jedem Wort eine zufällige Zahl zuzuordnen. Wenn die Zahlen zufällig sind, werden damit allerdings keine semantischen Ähnlichkeiten kodiert. Um herauszufinden, welche Wörter eine semantische Ähnlichkeit besitzen wird word2vec benutzt, um Word Embeddings zu generieren.

Word2vec ist ein zwei-Layer neuronales Netzwerk, welches die Semantik von Wörtern lernt. Für jedes Wort, für welches eine Vektorrepräsentation bestimmt werden soll, benötigt das neuronale Netzwerk einen Input. Das erste Layer des neuronalen Netzwerkes besteht aus Aktivierungsfunktionen. Jede Node des Layers summiert die Inputs. Die Weights der Aktivierungsfunktionen sind am Ende die Vektorrepräsentationen der Wörter. Die Anzahl der Nodes im Aktivierungslayer bestimmt die Anzahl der Dimensionen im Vektor. Das nächste Layer summiert wiederum die Werte aus dem Aktivierungslayer und wendet eine SoftMax Funktion an. Der Output des neuronalen Netzwerkes besteht, so wie der Input, aus so vielen Nodes, wie es Wörter gibt, welche mit dem Netzwerk repräsentiert werden sollen. Der Output gibt an, mit welcher Wahrscheinlichkeit jedes der Wörter das Folgewort des gegebenen Inputs ist.

Das neuronale Netzwerk wird mithilfe von Backpropagation trainiert. Dazu wird Training Data benötigt. Dieses enthält Sätze, auf welche das neuronale Netzwerk trainiert werden soll. Aufgabe des neuronalen Netzwerkes ist es, auf Basis eines gegebenen Wortes das nächste Wort im Satz zu bestimmen. Für ein einfaches Beispiel könnte das Trainingsdatenset aus zwei Einträgen *NLP ist interessant* und *NLP ist komplizit* bestehen. Da das Trainingsdatenset insgesamt aus vier verschiedenen Wörtern besteht, hat das neuronale Netzwerk genau vier Inputs. Wenn nun das Folgewort für das Wort *NLP* trainiert werden soll, dann bekommt das neuronale Netzwerk als Input die Werte *1, 0, 0, 0*. Die Eins bedeutet, dass das Wort NLP aktiviert ist. Die Nullen bedeuten, dass die anderen Wörter nicht aktiviert sind. Nun muss das neuronale Netzwerk für diesen Input den Output-Wert berechnen. Für die Backpropagation wird die Cross-Entropy Loss Function verwendet. Das Ergebnis ist, dass das trainierte neuronale Netzwerk für das Wört *NLP* lernt, dass das nächste Wort mit der Wahrscheinlichkeit eins das Wort *ist* ist. Die gelernten Weights bilden einen Vektor, welcher in einer Vektordatenbank gespeichert werden kann. Das Ergebnis ist eine Vektordatenbank, welche Informationen darüber enthält, welche Wörter semantisch ähnlich sind, und welche nicht.

4.3.2. Positional Encoding

Positional Encoding wird eingesetzt, um die Position eines Wortes in einem Satz zu kodieren, wenn ein Satz in einen Vektor transformiert wird. Das Positional Encoding wird angewendet, nachdem die Wörter mithilfe von Word Embeddings in Vektoren transformiert wurden. Nun werden die generierten Vektoren mit einem Vektor addiert, welcher die Position des Wortes repräsentiert. Der Vektor, welcher auf die Word Embeddings addiert wird, wird mithilfe von Sinus- und Kosinusfunktionen berechnet. Sinus- und Kosinusfunktion sind surjektiv. Das bedeutet, dass es keine eindeutige Abbildung von einem Inputwert auf einen Output-Wert gibt. Da nun die Position der Wörter in einem Satz eindeutig kodiert werden sollen, wird für jede Dimension in dem Vektor eine andere Sinus- bzw. Kosinusfunktion verwendet. Und mit steigender Dimension wird die Frequenz der Sinus- bzw. Kosinusfunktion verringert. Der Unterschied in der Frequenz hat zur Folge, dass jeder Vektor zur Bestimmung der Position eines Wortes eindeutig ist.

4.3.3. Self-Attention

Der Satz *Software-Entwickler verwenden die Spezifikation, damit sie Features korrekt implementieren* besitzt beispielsweise eine Cross-Referenz. Das Wort *sie* bezieht sich in dem Beispielsatz auf Software-Entwickler, aber es könnte sich auch auf die Spezifikation beziehen. Im Allgemeinen können Menschen solche Cross-Referenzen intuitiv verstehen. Mit der bisher erörterten Architektur von Sentence Transformers, können diese allerdings keine Cross-Referenzen verstehen. Self-Attention ist eine Technologie, welche es Transformern erlaubt die Cross-Referenz zwischen Wörtern zu verstehen.

Um dies zu ermöglichen wird die Ähnlichkeit zwischen Wörtern in einem Satz berechnet. Für jedes Wort wird also die Ähnlichkeit zu allen anderen Wörtern berechnet, sowie die Ähnlichkeit zu sich selbst. Um die Ähnlichkeiten zu berechnen, werden auf Basis der Positional Encodings neue Self-Attention Vektoren generiert. Auf Basis der Positional Encodings werden für jedes Wort in einem Satz eine Query, ein Key und ein Value berechnet. Die Begriffe Query, Key und Value entstammen hierbei wiederum aus dem Information Retrieval. Im Information Retrieval ist die Query die gemachte Sucheingabe, der Key eine Kategorie von Dokumenten und der Value ist ein tatsächliches Dokument. In diesem Kontext wird das Punktprodukt aus der Query und dem Key berechnet und es wird die SoftMax Funktion auf das Ergebnis angewendet. Das Ergebnis repräsentiert für das aktuelle Wort, für welches der Self-Attention Wert berechnet werden soll, wie stark eine Suche nach diesem Wort, mit einer bestimmten Kategorie von Wörtern (mit einem Konzept) übereinstimmt. Dann wird wiederum das Punktprodukt auf dieses Ergebnis und den Value jedes Wortes angewendet. Diese Ergebnisse bedeuten wiederum, wie repräsentativ die einzelnen Wörter für das gegebene Konzept sind. Die Ergebnisse dieser Operationen sind die Self-Attention Werte der Wörter in einem Satz.

4.3.4. seq2seq

Ein seq2seq-Modell soll eine Sequenz auf eine andere Sequenz anhand von erlernten Regeln mappen. Dazu wird die Input-Sequenz zunächst in eine modell-interne Sequenz encodiert, der Context Vector, und anschließend in die Output-Sequenz dekodiert. Wenn eine Wortsequenz in eine andere Wortsequenz umgewandelt werden soll, dann müssen die Input-Wörter zunächst durch ein Word Embedding Layer verarbeitet werden, welches die Wörter in Word Embeddings konvertiert. Auf das Word Embedding Layer folgen eine beliebige Anzahl von LSTM-Layers. Die Anzahl der Nodes der LSTM-Layer entspricht der Anzahl der Nodes im Word Embedding Layer. Zusätzlich kann jede Node in jedem LSTM-Layer mehrere LSTM Netzwerke besitzen. Die Weights und Biases der LSTM Netzwerke bilden den Output und damit den Context Vector. Die bisher beschriebene Architektur wird als Encoder bezeichnet. Die LSTM Netzwerke des Encoders werden im Decoder gespiegelt. Die Outputs des oberen LSTM Layers des Decoders werden in ein Fully Connected Layer gegeben und durchlaufen eine Softmax Funktion. Der Output des Decoders sind die Wahrscheinlichkeiten, mit welchen die möglichen Wörter der Ziel-Sequenz dem Input entsprechen.

Der Context Vector kann in einer Vektordatenbank gespeichert werden, so wie Word Embeddings in einer Vektordatenbank gespeichert werden können. Das hat zur Folge, dass die Ähnlichkeit von Sätzen genauso wie die Ähnlichkeit von Wörtern ermittelt werden kann.

4.4. Vektor-Indizierung

Neben einer Volltext-Indizierung können Dokumente in Form von Vektoren indiziert werden. Dazu werden Dokumente zunächst, wie auch bei der Volltext-Indizierung gecrawlt. Anschließend durchlaufen die Inhalte der Dokumente ein Preprocessing. Dieses kann je nach Implementierung variieren. Kapitel 4.6.2 beschreibt, wie in der hier aufgeführten Implementierung das Preprocessing durchgeführt wird. Das Preprocessing hat den Zweck die Daten an das Schema der Datenbank anzupassen und die Qualität der Daten zu erhöhen. Außerdem sorgt es für eine kürzere Indizierungszeit.

Nach dem Preprocessing werden durch einen Transformer für die Inhalte der Dokumente Vektoren berechnet. Ein Transformer wird mithilfe von Trainingsdaten darauf trainiert, Vektoren für Wörter zu generieren. Das trainierte Modell wird nach dem Preprocessing durchlaufen. Zuletzt werden die Daten in einer Vektordatenbank gespeichert. Eine Vektordatenbank speichert Daten, wie eine dokumentenbasierte Datenbank. Dort werden nun sowohl die rohen Daten als auch die Vektoren gespeichert, welche von dem Transformer berechnet wurden. Die Vektoren haben den Vorteil, dass die Daten in der Datenbank nicht linear gespeichert sind. Sie sind in einem n-dimensionalen Raum gespeichert, mit dessen Hilfe die semantische Nähe zwischen Dokumenten ausgedrückt werden kann.

4.5. Hierarchical Navigable Small Worlds (HNSW)

HNSW ist ein Nearest Neighbor Suchalgorithmus. Der Algorithmus findet die ähnlichsten Nachbar-Vektoren zu einem gegebenen Input-Vektor. Um den Algorithmus umzusetzen werden Skip Lists und Navigable Small Worlds verwendet. Skip Lists sind klassische Linked Lists, welche aus mehreren Schichten besteht. Die unterste Schicht entspricht der originalen Linked List und beinhaltet alle Daten. Jedes höhere Layer beinhaltet nur ein Subset der Nodes des darunterliegenden Layers. Wenn nun eine Suche in der Skip List durchgeführt wird, dann wird das oberste Layer zuerst durchsucht. Das oberste Layer wird sequenziell durchlaufen. Es wird bei dem ersten Element des obersten Layers begonnen. Solange das aktuelle Element kleiner ist als das gesuchte Element, wird das nächste Element untersucht. Wenn das Element n dem gesuchten Element entspricht, dann wurde das gesuchte Element gefunden und wird zurückgegeben. Wenn das nächste Element n+1 größer ist als das gesuchte Element, dann wird das aktuelle Element n in dem aktuellen Layer gefunden. Da dieses Element noch nicht dem gesuchten Element entspricht, wird nun im tieferen Layer weitergesucht, in welchem mehr Elemente vorhanden sind. Denn das gesuchte Element liegt zwischen dem gefundenen Element n und dem untersuchten Element n+1. Der gleiche Algorithmus wird rekursiv für jedes Layer durchgeführt, bis das unterste Layer erreicht wurde. Wurde am Ende des untersten Layers kein Element gefunden, dann konnte das Element in der Skip List nicht gefunden werden (Pugh 1990).

Navigable Small Worlds basieren auf dem Small World Phänomen aus der Sozialpsychologie, welches von Stanley Milgram formuliert wurde. Dieses besagt, dass Menschen untereinander so stark vernetzt sind, dass jeder Mensch über nur wenige andere Menschen mit jedem anderen Menschen vernetzt sind (Milgram 1967). Das Konzept lässt sich auf Graphen in der Informatik übertragen. Das bedeutet, dass in einem stark vernetzten Graphen jeder Datenpunkt mit jedem anderen Datenpunkt über nur wenige Kanten erreichbar ist. Greedy Routing Algorithmen können von dem Konzept Gebrauch machen, um den Datenpunkt zu finden, dessen Distanz zu einem bestimmten Datenpunkt am geringsten ist (Malkov und Yashunin 2020). Die Konzepte der Skip List und der Navigable Small World können nun kombiniert werden. Dazu werden anstelle von LinkedLists Graphen in mehreren Layers gespeichert. Wie bei Skip Lists, beinhaltet der Graph des untersten Layers alle Datenpunkte. Jedes Layer darüber beinhaltet nur ein Subset des Layers darunter. Anstatt die Liste des obersten Layers zu durchlaufen, wird der Graph des obersten Layers durchlaufen. Es wird der Punkt gefunden, welcher am nähesten am gesuchten Punkt ist. Der gesuchte Punkt ist dabei das Embedding der Sucheingabe. Dann wird so lange das nächstuntere Layer betrachtet, bis der Punkt gefunden wurde, welcher der Sucheingabe am nähesten ist.

Um nun die Distanz zwischen zwei Punkten zu bestimmen wird eine Distanzmetrik benötigt. Eine mögliche Distanzmetrik für Vektorräume ist die Cosine Similarity. Eine andere ist L2-squared.

4.6. Implementierung der neuen Suchfunktion

Nachdem erörtert wurde, was eine semantische Suche ist und wie die zugrunde liegende Technologie funktioniert, wird in diesem Kapitel die Implementierung der neuen Suchfunktion dargestellt. Die Implementierung setzt die Vektordatenbank Weaviate ein, welche eine semantische Suche ermöglicht. Es wird als nächstes erklärt, wie die Vektordatenbank aufgesetzt wird. Anschließend wird erläutert, wie die Daten in Weaviate eingespielt werden.

4.6.1. Aufsetzen der Vektordatenbank Weaviate

Für das Aufsetzen von Weaviate werden zwei Komponenten benötigt. Zum einen die Vektordatenbank selbst. Zum anderen ein Transformer, welcher Text entgegennimmt, und diese in Vektoren umwandelt, sodass diese in der Datenbank gespeichert werden können. Um die beiden Komponenten aufzusetzen wird Docker benutzt. Das entsprechende docker-compose.yml File ist im Anhang zu finden. Hier werden die beiden Komponenten definiert. Unter *t2v-transformers* wird der Transformer konfiguriert. Es wird das Image eines bereits vortrainierten Transformers verwendet. Unter *weaviate* wird die Datenbank konfiguriert. Hier wird mithilfe von den Environment-Variablen $DEFAULT\_VECTORIZER\_MODULE$, $ENABLE\_MODULES$ und $TRANSFORMERS\_INFERENCE\_API$ konfiguriert, welcher Transformer verwendet werden soll. So wird konfiguriert, dass

der Transformer der anderen Komponente benutzt werden soll.

Neben den beiden Docker-Containern für die semantische Suche wird ebenfalls eine Spring Boot Anwendung in einem Docker Container eingesetzt. Die Spring Boot Anwendung verwendet Kotlin und Gradle. Mithilfe der Dependency *io.weaviate:client:4.0.1* wird die Client API von Weaviate eingesetzt. So wird nun eine Verbindung zu der Vektordatenbank aufgebaut. Diese beinhaltet zu diesem Zeitpunkt noch keine Daten. Bevor die Daten in die Datenbank eingespielt werden, muss das Schema der Daten angegeben werden. Der entsprechende Code ist ebenfalls im Anhang zu finden. Es wird eine Klasse *Document* definiert. Diese Klasse beinhaltet die Properties *documentUrl*, *h1*, *h2* und *p*. Es werden in dieser Klasse also die URL des Dokuments, sowie die Inhalte aller h1-, h2 und p-Tags gespeichert. Außerdem wird als Vectorizer *text2vec-transformers* konfiguriert.

4.6.2. Einspielen der Daten in Weaviate

Um nun die Daten aus Confluence in Weaviate einzuspielen, werden zuerst die Daten aus Confluence exportiert. Beim Export von Confluence-Seiten werden HTML-Dateien generiert. Es werden keine CSS-, JavaScript- oder Bilddateien generiert. Die HTML-Dateien werden vorverarbeitet, um dem zuvor definierten Schema zu entsprechen. Es werden zuerst mithilfe eines Abstract-Syntax-Trees alle Inhalte von h1-, h2- und p-Tags herausgefiltert. Anschließend werden Punctuations und Stopwords entfernt und die Inhalte druchlaufen einen Tokenizer und einen Stemmer. Punctuations sind Zeichen, wie die folgenden: *.,;:*. Stopwords sind Wörter, welche für einen Leser notwendig sind, aber für die Verarbeitung durch einen Algorithmus als unwichtig erachtet werden (Sarica und Luo 2021). Beispiele für Stopwords sind *aber, denn, der*. Ein Tokenizer trennt einen Text in einzelne Wörter auf. Aus einem Text, wie *das deployment erfolgt durch ein bash-skript* wird also ein Array, welches folgendermaßen aussieht:

$$[\text{"}das\text{"}, \text{"}deployment\text{"}, \text{"}erfolgt\text{"}, \text{"}durch\text{"}, \text{"}ein\text{"}, \text{"}bash\text{"}, \text{"}skript\text{"}].$$

Ein Stemmer bestimmt den Wortstamm für die einzelnen Wörter auf Basis von Grammatikregeln. Die Software nutzt den Porter-Stemmer-Algorithmus. Aus dem Wort *deployment* wird dadurch beispielsweise *deploy*. Duplikate von Wörtern werden anschließend verworfen. Alle Inhalte, welche in h1-Tags stehen, werden konkateniert und einem Datenfeld der Vektordatenbank gespeichert. Das Gleiche gilt für die übrigen Tags. Nachdem die Inhalte das Preprocessing durchlaufen haben, werden sie in die Datenbank eingespielt.

In Kapitel 4.4 wurde bereits von Performancegründen gesprochen, aus denen das Preprocessing durchgeführt wird. Da nun die einzelnen Schritte des Preprocessings erklärt wurden, kann auch erklärt werden, warum das Preprocessing die Performance beim Indizieren erhöht. Zuallererst werden viele Wörter gänzlich ver-

worfen, weil sie Stopwords sind. Durch den Stemmer werden anschließend ähnliche Wörter zusammengruppiert. Die Wörter *Heizung* und *heizen* stammen beide vom gleichen Wortstamm *heiz*. Das bedeutet, dass aus zwei verschiedenen Wörtern, welche beide indiziert werden müssen, ein einziges Wort gemacht wird. Denn am Ende werden Duplikate, wie bereits erwähnt, verworfen. Die Anzahl der Wörter, welche indiziert werden müssen, wird dadurch reduziert, und damit auch die Last auf dem Transformer, welche die Vektoren für die Wörter berechnen muss.

5. Vergleich der Suchfunktionen

Zum Vergleich der beiden Suchfunktionen wird im Folgenden zuerst auf Evaluationsmethoden und -Kriterien eingegangen. Diese sind die Grundlage für den Vergleich von zwei Suchfunktionen. Anschließend wird der Aufbau der Studie dargestellt, sowie der zugrundeliegende Datensatz für die Durchführung der Studie. Nachdem der Aufbau der Studie erklärt wurde, werden die Daten der Studie ausgewertet und dargestellt. Das Ergebnis wird diskutiert. Zuletzt wird auf die Validität der Daten eingegangen und es wird der Versuchsaufbau diskutiert.

5.1. Evaluationsmethoden und -Kriterien

Um zu verifizieren, dass die Implementierung der neuen Suchfunktion ihre gewünschte Wirkung erzielt, müssen zuerst Methoden und Kriterien zur Messung herangezogen werden. Dazu soll ein Experiment durchgeführt werden. Bei einem Experiment wird zunächst eine Hypothese aufgestellt. Diese wird anschließend durch das Experiment untersucht. Ein solches Experiment wird für die Evaluation der Suchfunktionen erstellt. Diese stützt sich auf den Anwendungsfällen, welche in Kapitel 2 dargestellt wurden. Dazu werden für die Anwendungsfälle Sucheingaben erstellt. Für jede Sucheingabe wird definiert, welche Dokumente als Ergebnis erwartet werden. Die Hypothese: Mit der implementierten Suchfunktion werden die erwarteten Dokumente *besser* gefunden als mit der bisherigen Suchfunktion. Diese Hypothese muss nun quantifizierbar und messbar gemacht werden. Die subjektive Wahrnehmung reicht nicht für eine wissenschaftliche Arbeit aus.

Die Formulierung der Hypothese ist für sich genommen zu unspezifisch. Es muss geklärt werden, wann eine Information *besser* zu finden ist. Im Folgenden werden Precision, Recall und F-Maß herangezogen, um Suchfunktionen anhand dieser Eigenschaften vergleichbar zu machen. Mithilfe dieser Messwerte kann eine objektive Entscheidung darüber getroffen werden, welche Suchfunktion *besser* funktioniert.

5.1.1. Precision, Recall und F-Maß

Zur Evaluation der Suchfunktionen können die statistischen Messwerte Precision und Recall verwendet werden. Die Messwerte beschreiben, inwieweit eine Hypothese zutrifft. Wenn also die Hypothese ist, dass ein bestimmtes Dokument gefunden wird, dann ist der Precision-Wert das Verhältnis zwischen allen gefundenen Dokumenten und den gefundenen Dokumenten, die tatsächlich relevant sind. Der Wert lässt sich im Kontext der Suche nach Dokumenten wie folgt definieren (Sirotkin 2012):

$$Precision = P = \frac{gefundenerelevanteDokumente}{gesamteAnzahlgefundenerDokumente}$$

Der Recall-Wert gibt wiederum an, wie viele von den tatsächlich relevanten Dokumenten auch gefunden wurden. Er lässt sich in diesem Kontext wie folgt definieren (Sirotkin 2012):

$$Recall = R = \frac{gefundenerelevanteDokumente}{gesamteAnzahlrelevanterDokumente}$$

Es ist schwierig beide Werte zu optimieren, da der Precision-Wert versucht die Anzahl der gefundenen Dokumente einzugrenzen und der Recall-Wert versucht die Anzahl der gefundenen Dokumente zu erweitern. Das F1-Maß fasst beide Werte zu einem neuen Wert zusammen (Sirotkin 2012):

$$F_1 = 2\frac{PR}{P+R}$$

Neben der Verwendung des F1-Maß müssen sich Gedanken darüber gemachen werden, welcher der beiden Messwerte wichtiger ist. In diesem Fall ist es sinnvoll eher die Precision zu optimieren. Denn, wenn ein Dokument gesucht wird, aber überhaupt nicht gefunden werden kann, dann erfüllt die Suchfunktion nicht ihren Zweck. Wenn die Suchfunktion irrelevante Dokumente darstellt, kann sie trotzdem ihren Zweck erfüllen, solange die relevantesten Dokumente zuerst in der Liste der Ergebnisse dargestellt wird. Dieser Faktor gilt auch bei der Implementierung zu berücksichtigen.

5.2. Aufbau der Studie

Im Rahmen der Bachelorarbeit wird eine Studie durchgeführt, welche die neue Suchfunktion mit der bestehenden Confluence-Suche vergleicht. Da der Zeitrahmen begrenzt ist, in welchem die neue Suchfunktion entwickelt wird, wird nicht die tatsächliche Suchfunktion von Confluence als Benchmark für die neue Suchfunktion herangezogen. Das ist notwendig um eine gute Vergleichbarkeit zwischen den Suchfunktionen zu gewährleisten. Die Produktreife von Confluence könnte die Performance dessen Suchfunktion positiv beeinflussen. Da die neue Suchfunktion in nur kurzer Zeit entwickelt wurde, kann der Unterschied Produktreife ein Störfaktor bei dem Vergleich der Suchfunktionen sein. Um dies zu vermeiden wird die Confluence-Suche repräsentiert durch eine BM25 Suche, welche ebenfalls mit Weaviate implementiert wurde. Es wurde bereits erwähnt, dass die Suche von Confluence auf Apache Lucene basiert, und dass dieses ein VSM auf Basis von TF-IDF verwendet. Sparck et al. (2000) beschreiben, dass eine große Ähnlichkeit bei der Performance zwischen TF-IDF und BM25 besteht. Auf diese Weise wird die Vergleichbarkeit zwischen den beiden Suchfunktionen also verbessert. Die BM25-Suche in Weaviate ist eine von drei Konfigurationen der neuen Suchfunktionen. Sie ist die Benchmark für die neue Suchfunktion und repräsentiert die Confluence-Suche.

Eine weitere Konfiguration verwendet eine Mischung aus einer BM25 Suche, und einer semantischen Suche. Die beiden Suchalgorithmen werden zu gleichen Teilen verwendet. Zuletzt verwendet eine andere Konfiguration lediglich die semantische Suche auf Grundlage von Sentence Similarity. Die Eigenschaft, welche untersucht werden soll ist, ob eine semantische Suche für den gegebenen Datensatz, und im Kontext einer Wissensdatenbank in der Softwareentwicklung, ebenfalls bessere Suchergebnisse liefert, als eine Suche auf Basis von BM25. Durch die Verwendung einer einheitlichen Implementierung wird sichergestellt, dass lediglich die Unterschiede der Suchalgorithmen untersucht werden. Es wird damit verhindert, dass die Confluence-Suche besser abschneidet, weil sie ausgereifter ist. Es wird ebenfalls sichergestellt, dass die Datensätze der Suchfunktionen identisch sind.

Für die Studie wurde ein Datensatz generiert, welcher Dokumente beinhaltet, welche durch die Suchfunktion gefunden werden sollen. Der Datensatz wurde aus einem realen Softwareentwicklungs-Projekt generiert, indem ein Teil der tatsächlichen Confluence-Seiten exportiert wurde. Die Software soll im Gesundheitswesen eingesetzt werden. Das bedeutet, dass die Wissensdatenbank Fachbegriffe aus dem Gesundheitswesen beinhaltet. Darüber hinaus beinhaltet die Wissensdatenbank Definitionen, welche sich spezifisch auf das Softwareprojekt beziehen. Daher handelt es sich bei dem Datensatz um eine Closed-Domain, also ein Datensatz, welcher spezifisch aus einer bestimmten Domäne generiert wurde. Das steht im Gegensatz zu einem Open-Domain Datensatz, welcher Daten beinhaltet, welche allgemein gültig sind, und nicht nur in einer bestimmte Domäne. Für jedes Dokument sind mehrere Sucheingaben definiert, mit dessen Eingabe das Dokument gefunden werden soll. Darüber hinaus ist für jedes Dokument festgehalten, zu welchem Anwendungsfall sich dieses zuordnen lässt. Tabelle 5.1 zeigt die Dokumente, welche gefunden werden sollen, sowie die Sucheingaben, mit welcher sie gefunden werden sollen. Die Einträge sind jeweils Anwendungsfällen zugeordnet, sowie einem Bereich in der Wissensdatenbank. Die Spalte für die Sucheingaben enthält mehrere Sucheingaben, welche durch ein Komma getrennt sind.

Anwendungsfall	Bereich	Interessante Seiten	Sucheingaben
Informationen über PM	Projekt Branchensoftware	Erstellung einer Lieferung	Lieferung erstellen,Software ausliefern, Software deployen, an Kunden liefern
Onboarding	Projekt Branchensoftware	On- und Offboarding - Willkommen im Projekt	Onboarding,Offboarding,Einleitung Projekt,Getting Started
Onboarding	Projekt Branchensoftware	EP - DoR & DoD	Definition of Done,Definition of Ready, DoD, DoR, Akzeptanzkriterien
Onboarding	Entwicklung	Getting Started	Getting Started,Einleitung Entwicklung
Onboarding	Entwicklung	The Pragmatic Programmer	The Pragmatic Programmer, Pragmatischer Programmierer, Konventionen
Onboarding	Entwicklung	Guidelines	Guidelines, Richtlinien, Konventionen
Bug Localization	Entwicklung	FE - Debugging	Debugging, Frontend Debugging, FE Debugging, Debuggen im Frontend, Bugs im Frontend finden
Onboarding	Entwicklung	Dev-Stage in Docker-Compose	Dev-Stage lokal, Dev-Stage Docker-Compose, Lokal ausrollen, Lokal deployen, Docker Deploy
Onboarding	Entwicklung	Java-Testing - Tipps, Stolperfallen und Best-Practices	Java Testing, Backend Testing, Unit-Tests, Integration-Tests, Testing Best Practices
Implementierung nach Spec	Architektur nextGen	Bearbeiter Prinzip	Bearbeiterprinzip,Prinzip der Bearbeitung, Bearbeiter Prinzip, Auftrag bearbeiten, Recht zur Bearbeitung
Onboarding	Architektur nextGen	Clientinstallation und -update	Client installieren, Clientinstallation, Software installieren, Branchensoftware installieren
Implementierung nach Spec	Spezifikation nextGen - QP	V-AW-107 Prüfaufträge aus Excel-Datei importieren	V-AW-107, Prüfaufträge importieren, Excelimport
Implementierung nach Spec	Spezifikation nextGen - QP	V-I-R-1 Übermitteln von Prüfaufträgen	V-I-R-1, Übermitteln von Prüfaufträgen, <ausgeblendet>, Übermitthlung von Prüfaufträgen
Implementierung nach Spec	Spezifikation nextGen - QP	V-D-10 Prüfauftrag	V-D-10,Prüfauftrag

Tabelle 5.1.: **Studienaufbau, mit den interessanten Seiten, welche mit den Sucheingaben gefunden werden sollen. Mehrere Sucheingaben für eine Seite sind durch Kommata getrennt. Die Seiten sind einem Anwendungsfall und einem Bereich in der Wissensdatenbank zugeordnet**

Tabelle 5.2 zeigt einen Ausschnitt aus den Daten, welche durch die Studiedurchführung generiert wurde. Der Name des Projektes, aus welchem die Daten

entnommen wurden, wurde in beiden Tabellen unkenntlich gemacht.

Anwendungsfall	Bereich	Sucheingabe	Erwartetes Dokument
Onboarding	Entwicklung	Getting Started	Getting Started

Gefundene Dokumente	Hit	Hit in Five	Hit in Three	Hit in One
getting started (legacy)..., onboarding usecase x..., onboarding re..., ui-debian-build-base..., ep...	true	true	true	true

Tabelle 5.2.: **Ausschnitt aus den Ergebnissen der Studie, mit den Dokumenten, welche für eine Sucheingabe gefunden wurden, sowie der Hit-in-One bis Hit-in-Five Wert**

Die Suchfunktion gibt für jeden Algorithmus fünf Dokumente als Antwort auf eine Sucheingabe zurück. Diese fünf Dokumente werden nach dessen Score sortiert. Das bedeutet, dass das erste Dokument der Liste jenes ist, welches von dem Algorithmus als das passendste erachtet wird. Die fünf Dokumente werden darauf untersucht, ob sich das gewünschte Dokument unter den Dokumenten befindet. Das Ergebnis ist ein Precision-Score für den Suchalgorithmus. Um eine detailliertere Analyse zu ermöglichen wird nicht nur die Precision in Bezug auf die ersten fünf Dokumente gemessen. Es wird die Precision für das erste Dokument (Hit in One), die ersten drei Dokumente (Hit in Three) und alle fünf Dokumente (Hit in Five / Hit) gemessen. Anschließend werden die Precision-Scores der Algorithmen miteinander verglichen. Dazu wird die Summe der Hits für alle Sucheingaben gebildet. Die Summe wird durch die gesamte Anzahl der Sucheingaben dividiert.

Die Studie wird vollkommen automatisch durchgeführt. Dadurch können Ergebnisse der Studie nicht durch Teilnehmer verfälscht werden. Es bedeutet auch, dass die Studie eine hohe Reliabilität hat und mit jeder Durchführung das gleiche Ergebnis liefert. Sie ist also reproduzierbar. Der Nachteil dieser Herangehensweise ist, dass die subjektive Wahrnehmung des Nutzers, in Bezug auf die Precision der Suchfunktion, nicht beachtet werden kann. So ist es denkbar, dass eine Sucheingabe nicht das gewünschte Dokument beinhaltet, aber andere Dokumente, welche ein Nutzer als sinnvoll erachten würde. Die Ergebnisse der Studie sind damit abhängig von der Vorauswahl der Dokumente, welche gefunden werden sollen, und der Sucheingaben, welche a priori als sinnvoll bestimmt wurden. Es ist möglich, dass eine Suchfunktion für eine Sucheingabe durchaus ein sinnvolles Ergebnis liefert, aber nicht das Ergebnis, welches durch den Aufbau der Studie erwartet wird.

5.3. Auswertung der Ergebnisse

Die Studie wurde mehrfach durchgeführt. Die Precision-Scores werden auf zwei Nachkommastellen gerundet. Die Precision-Scores werden in der Tabelle 5.3 zusammengefasst.

Es ist auffällig, dass der Precision-Score der semantischen Suche wesentlich schlechter ist als der Precision-Score der BM25 Suche. Dies könnte sich wie folgt

Precision	Hit in Five	Hit in Three	Hit in One
BM25	0,56	0,5	0,39
Hybrid	0,56	0,52	0,35
Semantic	0,15	0,13	0,07

Tabelle 5.3.: **Precision der Suchfunktionen**

erklären lassen. Bei der Implementierung der Suche wurden alle H1-Header, H2-Header und Paragraphen eines Dokuments zusammengefasst in jeweils einen String für das entsprechende HTML-Tag. Der Algorithmus der Suchfunktion vergleicht nun den String der Sucheingabe mit den Strings in den Dokumenten. Also mit dem String des Titels, dem String der H1-Header, dem String der H2-Header und dem String der Paragraphen. Weil die einzelnen Inhalte der HTML-Tags zusammengefasst werden, können diese Strings größer werden als die Sucheingabe, welche in der Regel zwei bis drei Wörter umfasst. Dadurch, dass die Sätze einen großen Größenunterschied haben, fällt entsprechend auch der Score der Dokumente niedrig aus. Da nun alle Scores niedrig sind, kann die Suchfunktion nicht so leicht die passendsten Dokumente finden. Dadurch werden zum Teil weniger relevante Dokumente gefunden und der Precision-Score fällt besonders niedrig aus.

Auch das Postprocessing könnte eine Auswirkung auf den Precision-Score der semantischen Suche haben. Für Bag of Words Modelle, wie BM25, ist es typisch ein Stemming vor der Indizierung durchzuführen. Zum einen hat dies positive Auswirkungen auf den Precision-Score. Zum sorgt dies dafür, dass verschiedene Tokens als das gleiche Wort betrachtet werden, auch wenn sie nicht in der gleichen morphologischen Form stehen. Die Details zu dieser Problematik wurden bereits in Kapitel 3.3 erläutert. Für die semantische Suche könnte ein Stemming allerdings negative Auswirkungen haben. Denn bei dem Training eines Sentence Transformers werden ganze Sätze betrachtet und nicht nur die Stammformen der einzelnen Wörter. Das bedeutet, dass der Sentence Transformer die Wörter nicht mehr korrekt semantisch zuordnen kann. Dies wirkt sich wiederum negativ auf den Precision-Score der semantischen Suche aus.

Die Hypothese ist also, dass der Precision-Score der semantischen Suche nur aus dem Grund gering ist, weil neben dem Title-Tag auch die weiteren Tags indiziert werden. Um diese Hypothese zu überprüfen wurde die Studie ein weiteres Mal durchgeführt. Dieses Mal wurden lediglich die Title-Tags indiziert. Das Title-Tag ist für jedes Dokument nur einmal vorhanden. Außerdem sind die Titelsätze der Dokumente wesentlich kürzer als ganze Paragraphen. Mit dieser Konfiguration sollte der Größenunterschied des Sucheingabe-Strings und der Titel-Strings kein entscheidender Faktor mehr sein. Das Ergebnis dieser Konfiguration ist in Tabelle 5.4 dargestellt.

In dieser Konfiguration hat die semantische Suche eine bessere Precision. Die Hit-in-Five Precision liegt bei 0,28 für die Konfiguration, welche nur den Titel

Precision	Hit in Five	Hit in Three	Hit in One
BM25	0,54	0,5	0,39
Hybrid	0,54	0,48	0,35
Semantic	0,28	0,22	0,11

Tabelle 5.4.: **Precision der Suchfunktionen (2. Durchführung der Studie)**

der Dokumente indiziert, gegenüber 0,15 bei einer vollständigen Indizierung. Die Verbesserung der Precision kann mit drei Faktoren begründet werden. Zum einen die beiden Hypothesen, welche mit dieser Beobachtung untersucht werden sollten. Also zum einen der Größenunterschied von Sucheingabe-String und den Strings, welche indiziert werden. Zum anderen die Tatsache, dass sich das Postprocessing der Dokumente negativ auf die Fähigkeit des Sentence Transformers auswirkt, die Semantik der Wörter zu verstehen. Außerdem besteht die Möglichkeit, dass die Inhalte der Dokumente leicht von den Titeln der Dokumente abweichen können. Das würde bedeuten, dass für den gegebenen Versuchsaufbau der Titel besser geeignet ist, um die gewünschten Dokumente zu finden, als das gesamte Dokument. Die Precision für die BM25-Suche ist bei vollständig indizierten Dokumenten besser als bei alleiniger Beachtung des Titels. Lediglich die semantische Suche hat sich bei der zweiten Konfiguration verbessert. Das bedeutet, dass sich der gezeigte Effekt der Verbesserung der Suchfunktion tatsächlich mit der Hypothese begründen lässt, dass die Länge der Strings ein zu berücksichtigender Faktor ist. Da die semantische Suche weiterhin schlechter ist als die BM25-Suche, scheint es aber weitere Faktoren zu geben, welche zu berücksichtigen sind.

Eine mögliche Erklärung für die weiterhin schlechten Ergebnisse der semantischen Suche ist die Tatsache, dass es sich bei dem verwendeten Datensatz um eine Closed-Domain handelt. Das bedeutet, dass der Textcorpus Fachbegriffe beinhaltet, welche der Transformer mit hoher Wahrscheinlichkeit nicht kennt. Dementsprechend schwerfällt es dem Transformer folglich, passende Vektoren zu generieren, welche die Semantik von den domänenspezifischen Fachbegriffen abbilden. In Kapitel 6.2 wird auf Ansätze in der Literatur eingegangen, welche dieses Problem versuchen zu lösen.

5.4. Diskussion des Studienaufbaus

Wie bereits beschrieben gibt der Recall an, wie viele der relevanten Dokumente gefunden wurden. Die Precision gibt lediglich an, wie viele der Dokumente, welche gefunden wurden, relevant sind. Eine optimale Suchfunktion würde per Definition alle Dokumente finden, welche relevant sind, und keine irrelevanten Dokumente. Umgekehrt ist eine schlechte Suchfunktion eine Suchfunktion, welche keine relevanten Dokumente findet, sondern nur irrelevante. Dabei spielt es für den Nutzer aber keine Rolle, ob irrelevante Dokumente gefunden wurden. Die Tatsache, dass die relevanten Dokumente nicht gefunden wurden, machen die Suchfunktion für den Nutzer nicht benutzbar. Die Studie untersucht die Precision der Suchalgorith-

men. Auf Grundlage der obigen Argumentation zeigt sich, dass der Recall-Score wichtiger ist als der Precision-Score. Denn ist der Recall gering, dann ist die Suchfunktion nicht benutzbar. Eine Suchfunktion mit einem hohen Recall, aber eine niedrigen Precision könnte dagegen viele relevanten Dokumente finden, aber auch viele irrelevante Dokumente. Solange die relevanten Dokumente in der Liste weiter oben dargestellt werden, ist diese Zusammensetzung aus Precision und Recall gut.

Die Studie hat die Precision für das erste, die ersten drei und alle fünf Dokumente gemessen. Die Studie hat allerdings keinen Recall gemessen. Grund dafür ist die Tatsache, dass um den Recall messen zu können, alle relevanten Dokumente für eine Sucheingabe bekannt sein müssen. Um für eine Studie a priori Sucheingaben zu bestimmen, und alle Dokumente, welche für diese Sucheingabe relevant sind, müsste der gesamte Datensatz bekannt sein. Da der Datensatz mehrere hundert Dokumente beinhaltet ist dies nicht möglich. Und auch wenn der gesamte Datensatz bekannt wäre, dann müsste trotzdem eine Entscheidung darüber getroffen werden, welche Dokumente relevant sind und welche nicht. Das wäre wiederum eine subjektive Entscheidung, sodass die Validität des Ergebnisses anzweifelbar wäre.

6. Zusammenfassung und Ausblick

In dem letzten Kapitel wurde der Vergleich zwischen der Confluence-Suche und der neuen Suchfunktion durchgeführt. Die Ergebnisse sind ebenfalls diskutiert worden. Dieses Kapitel die fasst die Arbeit zusammen und stellt einen Ausblick dar.

6.1. Zusammenfassung

Zweck der Arbeit war die Konzeption einer neuen Suchfunktion, welche nach festgelegten Kriterien besser ist als die Confluence-Suchfunktionen. Dazu wurden in Kapitel 2 Anwendungsfälle definiert, anhand dessen die Nutzung der Suchfunktionen konkretisiert wurde. Dies hat den Rahmen geschaffen, um später einen Vergleich der beiden Suchfunktionen durchführen zu können. Nach der Definition der Anwendungsfälle wurde die Technologie der Suchfunktion von Confluence in Kapitel 3 erläutert. Damit wurde ein besseres Verständnis für die Suchfunktion von Confluence geschaffen. Dieses bessere Verständnis war die Grundlage für die Konzeption einer neuen Suchfunktion in Kapitel 4. Denn das Verständnis war notwendig, um Schwachstellen in der Confluence-Suche zu identifizieren und Verbesserungspotenziale zu verstehen. In Kapitel 4 wurde zuerst die Technologie der neuen Suchfunktion erörtert. Während Confluence einen Volltext-Index verwendet, verwendet die neue Suchfunktion Sentence Transformer für die Indizierung. Dazu wird eine Vektordatenbank benötigt, sowie der Scoring-Algorithmus HNSW. Auf Grundlage der Verbesserungspotenziale der Confluence-Suche und der dargestellten alternativen Technologien wurde die neue Suchfunktion abgeleitet. Dazu wurde argumentiert, warum sich die dargestellten Technologien als eine Alternative zu der bestehenden Confluence-Suche eignen sollen. Es wurde anhand verschiedener Quellen belegt, dass eine semantische Suche auf Basis von Sentence Transformers gute Ergebnisse für Open-Domain Datensätze erzielen. Dann wurde die Implementierung der neuen Suchfunktion in Kapitel 4.6 erklärt.

Nachdem die neue Suchfunktion implementiert wurde, wurden die Precision-Werte für einen angefertigten Datensatz zwischen den beiden Suchfunktionen verglichen. Das nötige Wissen für diesen Vergleich wurde in Kapitel 5.1 erläutert. Es wurde ebenfalls auf alternative Möglichkeiten zur Messung der Performance von Suchfunktionen eingegangen. Der angefertigte Datensatz leitete sich dabei aus den dargestellten Anwendungsfällen aus Kapitel 2 ab. Der Datensatz entstammte einer Wissensdatenbank eines Softwareprojektes. Die Software soll im Gesundheitswesen eingesetzt werden und umfasste entsprechend Fachbegriffe aus dem Gesundheitswesen. Darüber hinaus beinhaltet die Wissensdatenbank Definitionen, welche sich spezifisch auf das Softwareprojekt beziehen.

Die Durchführung der Studie wurde in Kapitel 5 erklärt. Die Studie kam zu dem Ergebnis, dass die neue Suchfunktion eine wesentlich schlechtere Performance besitzt als die Confluence-Suche. Begründet wurde dieses Ergebnis mit der Tatsache, dass es sich bei dem Datensatz, welcher in der Studie verwendet wurde, um eine Closed-Domain handelt. Da der Sentence Transformer nicht auf die Domäne angepasst wurde, kennt der Sentence Transformer die domänenspezifischen Fachbegriffe nicht und kann daher keine passenden Vektoren generieren, welche dessen Semantik wiederspiegeln. Im Ausblick wird erläutert, welche Ansätze es in der Literatur gibt, um dieses Problem zu lösen.

6.2. Ausblick

Die Studie hat ergeben, dass die semantische Suche, so wie sie implementiert war, nicht besser ist als eine Suche auf Basis von BM25. Der Kontext, in welchem die Suchen verwendet werden, ist eine Closed-Domain und beinhaltet domänenspezifische Fachbegriffe. Probablistische Modelle unterscheiden sich in der Performance nicht zwischen Closed-Domain und Open-Domain. Grund dafür ist, dass probablistische Modelle die Dokumente ohne ein vorheriges Training indizieren, anders als dies bei Sentence Transformern der Fall ist. So könnte der Schluss gezogen werden, dass sich probablistische Modelle besser für Closed-Domain Datensätze eignen. Dennoch gibt es in der Literatur Ansätze, um die Probleme von Sentence Transformern mit Closed-Domain Datensätzen zu beheben. In der Literatur wird dabei von Domain Adaption oder von Fine-Tuning gesprochen. Gururangan et al. (2020) zeigen beispielsweise, dass eine weitere Phase des pre-trainings zur Adaption an eine Domäne Performance Verbesserungen mit sich bringt. Dazu führen sie das pre-training fort, auf einem unlabeled Datensatz von domänenspezifischen Texten.

Wang et al. (2022) entwickelten das sogenannte Generative Pseudo Labeling, um Sentence Transformer an eine Domäne anzupassen. Zuerst extrahieren sie Paragraphen aus der Domäne, an welche der Sentence Transformer angepasst werden soll. Anschließend generieren sie automatisch drei Sucheingaben für jeden Paragraphen. Diese Sucheingaben werden an einen bestehenden Dense Retriever gesendet. Ein Dense Retriever ist dabei eine Vektordatenbank mit einem entsprechenden Sentence Transformer zum Indizieren der Daten und einem Scoring-Algorithmus, wie HNSW. Nun werden die Ähnlichkeiten der Sucheingaben mit den Paragraphen mithilfe eines Cross-Encoders bestimmt. Ein Cross-Encoder ist ein Sentence Transformer, welcher zwei Sätze entgegennimmt und die Ähnlichkeit der beiden Sätze bestimmt. Er funktioniert anders als der Bi-Encoder, welcher für die Implementierung der neuen Suchfunktion verwendet wurde. Denn dieser generiert für jeden Satz zuerst Embeddings, welche anschließend mithilfe des Scoring-Algorithmus verglichen werden. Nun werden die berechneten Ähnlichkeiten zusammen mit den Sucheingaben und Paragraphen als pseudo-gelabelte

Trainingsdaten verwendet. Der Sentence Transformer, welcher an eine Domäne angepasst werden soll, wird mithilfe der MarginMSE-Loss Funktion auf die generierten Trainingsdaten trainiert.

Die Quellen zeigen, dass es Möglichkeiten gibt, Sentence Transformer an eine Domäne anzupassen. Die Anpassung eines Sentence Transformer an eine Domäne setzt dabei einen Datensatz aus dieser Domäne voraus, auf welchem der Sentence Transformer trainiert werden kann. Ist ein solcher Datensatz nicht vorhanden, dann sind klassische Document-Retrieval Systeme auf Basis von TF-IDF oder BM25 vermutlich die bessere Wahl. Wenn ein solcher Datensatz aber vorhanden ist, dann kann es sich lohnen einen Sentence Transformer an die gewünschte Domäne zu adaptieren und eine semantische Suche zu implementieren.

7. Literaturverzeichnis

Antoniol, Canfora, Casazza und De Lucia (2000). „Information retrieval models for recovering traceability links between code and Documentation“. In: *Proceedings International Conference on Software Maintenance ICSM-94*.

Atlassian (2023). *Ranking of Search Results*. URL: `https://confluence.atlassian.com/doc/ranking-of-search-results-1188406620.html` (besucht am 26. 09. 2023).

Bar-Ilan, Judit (2002). „Methods for measuring search engine performance over time“. In: *Journal of the American Society for Information Science and Technology* 53.4, S. 308–319.

Beaulieu, Track, Karen Sparck und Peter Willett (Jan. 2000). „Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track“. In.

Berger, Adam, Rich Caruana, David Cohn, Dayne Freitag und Vibhu Mittal (2000). „Bridging the lexical chasm“. In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*.

Castillo, Carlos (2005). „Effective web crawling“. In: *ACM SIGIR Forum* 39.1, S. 55–56.

Choudhary, Sneha, Haritha Guttikonda, Dibyendu Roy Chowdhury und Gerard P. Learmonth (2020). „Document retrieval using deep learning“. In: *2020 Systems and Information Engineering Design Symposium (SIEDS)*.

Clarke, Sarah J. und Peter Willett (1997). „Estimating the recall performance of web search engines“. In: *Aslib Proceedings* 49.7, S. 184–189.

Dengel, Andreas (2012). *Semantische Technologien Grundlagen - Konzepte - Anwendungen*. Spektrum, Akad. Verl.

Foundation, The Apache Software (2023). *Apache Lucene - Scoring*. URL: `https://lucene.apache.org/core/2_9_4/scoring.html` (besucht am 26. 09. 2023).

Gordon, Michael und Praveen Pathak (1999). „Finding information on the World Wide Web: The Retrieval Effectiveness of search engines“. In: *Information Processing & Management* 35.2, S. 141–180.

Gururangan, Suchin, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey und Noah A. Smith (2020). „Don't stop pretraining: Adapt language models to domains and tasks“. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Haiduc, Sonia, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia und Tim Menzies (2013). „Automatic query reformulations for text retrieval in software engineering“. In: *2013 35th International Conference on Software Engineering (ICSE)*.

Harris, Zellig S. (1954). „Distributional structure“. In: *WORD* 10.2–3, S. 146–162.

Karpukhin, Vladimir, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen und Wen-tau Yih (2020). „Dense passage retrieval for open-domain question answering“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Khder, Moaiad (2021). „Web scraping or web crawling: State of Art, Techniques, approaches and application“. In: *International Journal of Advances in Soft Computing and its Applications* 13.3, S. 145–168.

Malkov, Yu A. und D. A. Yashunin (2020). „Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4, S. 824–836.

Manning, Christopher D., Prabhakar Raghavan und Hinrich Schütze (2019). „Term frequency and weighting“. In: *Introduction to information retrieval*. Cambridge University Press, S. 117–120.

Milgram, Stanley (1967). „The small-world problem“. In: *PsycEXTRA Dataset*.

Pugh, William (1990). *A Skip List Cookbook*. Techn. Ber. USA.

Sarica, Serhad und Jianxi Luo (2021). „Stopwords in technical language processing“. In: *PLOS ONE* 16.8.

Sirotkin, Pavel (2012). *On Search Engine Evaluation Metrics*.

Sparck Jones, K., S. Walker und S.E. Robertson (2000). „A probabilistic model of information retrieval: Development and comparative experiments“. In: *Information Processing & Management* 36.6, S. 779–808.

Treude, Christoph, Mathieu Sicard, Marc Klocke und Martin Robillard (2015). „TaskNav: Task-based navigation of software documentation“. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*.

Voorhees, E M und D K Harman (2001). In: *The Ninth text retrieval conference (TREC-9)*.

Wang, Kexin, Nandan Thakur, Nils Reimers und Iryna Gurevych (2022). „GPL: Generative Pseudo Labeling for Unsupervised Domain Adaptation of Dense Retrieval“. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Weaviate (2023). *Weaviate vector database RSS*. URL: `https://weaviate.io/developers/weaviate/benchmarks/ann` (besucht am 26. 09. 2023).

Ye, Xin, Hui Shen, Xiao Ma, Razvan Bunescu und Chang Liu (2016). „From word embeddings to document similarities for improved information retrieval in software engineering“. In: *Proceedings of the 38th International Conference on Software Engineering*.

Zhang, Qin, Shangsi Chen, Dongkuan Xu, Qingqing Cao, Xiaojun Chen, Trevor Cohn und Meng Fang (2023). „A survey for Efficient Open Domain Question Answering“. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Anhang

A. Anhang

A.1. docker-compose.yml File für Weaviate

```
version: '3.4'
services:
  weaviate:
    image: semitechnologies/weaviate:1.18.3
    command:
      - --host
      - 0.0.0.0
      - --port
      - '2000'
      - --scheme
      - http
    ports:
      - "2000:2000"
    restart: on-failure:0
    environment:
      PROMETHEUS_MONITORING_ENABLED: 'true'
      QUERY_DEFAULTS_LIMIT: 20
      AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: 'true'
      PERSISTENCE_DATA_PATH: "/var/lib/weaviate"
      DEFAULT_VECTORIZER_MODULE: text2vec-transformers
      ENABLE_MODULES: text2vec-transformers,qna-transformers
      TRANSFORMERS_INFERENCE_API: http://t2v-transformers:8080
      QNA_INFERENCE_API: "http://qna-transformers:8080"
      CLUSTER_HOSTNAME: 'node1'
    volumes:
      - /var/weaviate:/var/lib/weaviate
  t2v-transformers:
    image: semitechnologies/transformers-inference:sentence-
      transformers-msmarco-distilroberta-base-v2
    environment:
      ENABLE_CUDA: 0
  qna-transformers:
    image: electra-qna
    environment:
      ENABLE_CUDA: 0
```

A.2. Initialisieren des Schemas in Weaviate

```
WeaviateClass.builder()
  .className(DOCUMENT_CLASS)
  .properties(
    dataServiceHelper.buildProperties(
      mapOf(
        DOCUMENT_URL to WEAVIATE_TEXT_DATATYPE,
        TITLE_TAG to WEAVIATE_TEXT_DATATYPE,
        H1_TAG to WEAVIATE_TEXT_DATATYPE,
        H2_TAG to WEAVIATE_TEXT_DATATYPE,
        PARAGRAPH_TAG to WEAVIATE_TEXT_DATATYPE
      )
    )
  ).vectorIndexConfig(
    VectorIndexConfig.builder()
      .distance("l2-squared")
      .ef(100)
      .efConstruction(128)
      .build()
  )
  .invertedIndexConfig(
    InvertedIndexConfig.builder()
      .bm25(
        BM25Config.builder()
          .b(.5f)
          .k1(.5f)
          .build()
      )
      .build()
  )
  .vectorizer(VECTORIZER)
  .build()
```

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Essen, den 9. Oktober 2023