Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Sichere dezentrale Berechnung eines Dokument-Ranking-Ansatzes

Studienarbeit

eingereicht von: Philipp Waack
geboren am: 11.08.1991
geboren in: Hamburg

Gutachter/innen: Prof. Dr. Björn Scheuermann
M. Sc. Phillipp Schoppmann

eingereicht am: .....

Inhaltsverzeichnis

1 Einleitung

Der Schutz personenbezogener Daten spielt aufgrund fortschreitender Digitalisierung eine wichtige Rolle im Diskurs um jegliche elektronische Unterstützung und der Entwicklung von IT-Lösungen. Dabei ist die Durchsetzung des Datenschutzes als Grundrecht für das Individuum wesentlich, um dieses vor wirtschaftlichen Interessen von Unternehmen, die mit personenbezogenen Daten Profite erzielen, zu schützen. Die nach dem BDSG besonders sensiblen personenbezogenen Daten sollen unter anderem zudem vor Diskriminierungen und Benachteiligungen schützen. Eine dieser Daten angehörende Kategorie sind Angaben zur Gesundheit einer Person [1]. Der Gesetzesschutz deckt jedoch nicht immer beim Entstehen von Daten durch Nutzung eines IT-Systems. So gibt das Abrufen von Informationen zu einer Krankheit von der die Suchende Person betroffen ist über eine Suchmaschine den Betreibern implizit gesundheitliche Daten preis. Diese Informationen können beispielsweise für das Schalten von personenbezogener Werbung missbraucht werden.

Andererseits kann die Verarbeitung von personenbezogenen Daten für eine größere Wissensbasis einen sozialen Nutzen haben. Beispielsweise können größere Mengen von Daten zur Gesundheit im Krankenhaus zur Forschung genutzt werden und eine fundiertere Behandlung von Patienten ermöglichen [2]. Wesentlicher Bestandteil einer solchen Anwendung wären Operationen auf einem verteilten Datenbestand, bspw. der Austausch von Daten zur Behandlung von Patienten zwischen mehreren Krankenhäusern, um eine zur Entscheidung eines Falles.

Ob bei der Suche über eine Suchmaschine als profitorientierte Dienstleistung oder bei der Suche von Informationen innerhalb einer Branche, unstrukturierte, textbasierte, großen Datenmengen sind wesentliche Informationsquellen. Hierbei ist neben der Einhaltung von Datenschutzgesetzen auch das möglichst effektive Abrufen von relevanten Daten unabdingbar. Mit Effektivität ist hier die möglichst akkurate Sortierung der abgerufenen Daten nach Relevanz für das subjektive Anliegen der suchenden Person gemeint.

Das allgemeine Setting auf das sich diese Arbeit bezieht, besteht somit zum einen aus einer anfragenden Partei und zum anderen aus einer Menge von Parteien, welche die gewünschten textbasierten Informationen der Partei auf verteilte Weise bereitstellen. Dabei ist es auch möglich, dass die anfragende Partei selbst einen Teil der verteilten Dokumentenmenge liefert. Ziel ist es einerseits eine Relevanz-basierte Suche einer Partei auf dem Korpus auszuführen. Des Weiteren sollen die Daten der einzelnen Parteien, also die Suchanfrage der anfragenden Partei sowie die Herkunft der einzelnen Textquellen des verteilten Korpus geschützt werden. Ebenso soll das Ergebnis der Suche nur der anfragenden Partei offenbart werden. So können weder die Quellen den einzelnen Parteien eindeutig zugeordnet werden, noch sind die Anfrage-Daten der suchenden Partei bekannt.

In der hier beschriebenen Arbeit sollen erste Schritte zu einem Ansatz umgesetzt werden, der Relevanz-basierte Suchen auf einer verteilten Datenmenge textbasierter Informationen sicher ausführt. Hierbei ist mit sicher die Datenschutz-gewährleistende Ausführung der Suche für alle beteiligte Parteien gemeint. Für einen angemessenen

Umfang dieser Arbeit wurde sich auf ein Szenario beschränkt, in dem eine anfragende Partei (Client) und eine Partei mit dem Korpus existiert, auf dem die Suche ausgeführt werden soll.

Die nachfolgend beschriebenen Methoden sollen die Erfüllung der dargelegten Ziele gewährleisten.

2 Methoden

2.1 Relevanz-basierte Suche

Während im Kontext einer öffentlichen und kommerziellen Suchmaschine aufgrund hoher Anzahl an Nutzern, Klicks der Nutzer und Referenzen zwischen Dokumenten (beispielsweise Websites) ein effektives Maß für die Dokument-Relevanz sein kann, sind diese Quantitäten für eine Anwendung in einem kleinen, zum Beispiel branchenspezifischen Kontext unzureichend für die Ableitung der Relevanz. Im Beispiel eines Krankenhauses sind möglicherweise nur Ärzte berechtigt, Suchen durchzuführen. Die daraus resultierende Aktivität einer geringen Nutzeranzahl wäre nicht repräsentativ für eine Relevanzmessung auf Basis von Klicks. Im folgenden wird ein Ansatz beschrieben, der semantische Informationen nutzt, um über die Relevanz von Dokumenten in einem Korpus, gegeben eine Suchanfrage, zu entscheiden. Diese Methode soll die Suche in einem großen Dokumentenraum für einen Nutzer vereinfachen, sodass dieser in möglichst wenig Zeit zu den gewünschten Informationen gelangt.

2.1.1 Word Embeddings

Um Berechnungen für ein Relevanz-Ranking auf den Inhalten der Dokumente eines Korpus' selbst durchzuführen, ist es hilfreich die Dokumente in eine neue Repräsentation zu überführen, mit der mathematische Berechnungen durchgeführt werden können. Dazu muss der Wort-Raum (auch Vokabular) in einen Vektorraum transformiert werden. Nach Abbildung des Korpus' und der Suchanfrage in diesen Vektorraum kann über ein Abstandsmaß (beispielsweise der Kosinus-Abstand) die Nähe der Suchanfrage zu den Dokumenten im Korpus berechnet werden.

Eine mögliche Transformation wäre die Bag-Of-Words-Repräsentation (BOW), bei der jeder Term im Korpus eine Dimension darstellt. Die bekanntesten BOW-Repräsentationen nutzen hier entweder eine binäre Darstellung der Wortvorkommen oder die tatsächlichen Häufigkeiten im jeweiligen Dokument. Diese Vektor-Repräsentation von Dokumenten kann zusätzlich gewichtet werden (zum Beispiel mittels der Term-Frequency-Inverse-Document-Frequency (TFIDF)). Dies führt dazu, dass Wörter die in vielen Dokumenten eines Korpus' vorkommen, weniger stark ein einzelnes Dokument repräsentieren, während Wörter, die nur in wenigen Dokumenten vorkommen, ein Dokument stärker repräsentieren.

Die BOW-Methode hat jedoch zwei wesentliche Nachteile. Zum einen benutzt sie das Wortvorkommen als Repräsentation, was die Reihenfolge der Worte vollständig vernachlässigt. Des Weiteren muss jedes Dokument durch das gesamte Vokabular des Korpus'

repräsentiert werden. Das führt zu einem enorm hoch-dimensionalem Vektorraum. Die hier genutzte Methode Continous-Bag-Of-Words (CBOW) zur Repräsentation der Dokumente bildet diese in deutlich kleineren Vektoren ab, die komprimierte Informationen über die Dokumente enthalten. Diese Abbildung wird auch als Word Embeddings bezeichnet. Zudem kodiert die Methode semantische Informationen mit Hilfe des Umfelds eines Wortes im Dokument. Dies folgt der Annahme, dass ähnliche Worte in einem ähnlichen Wort-Umfeld stehen.

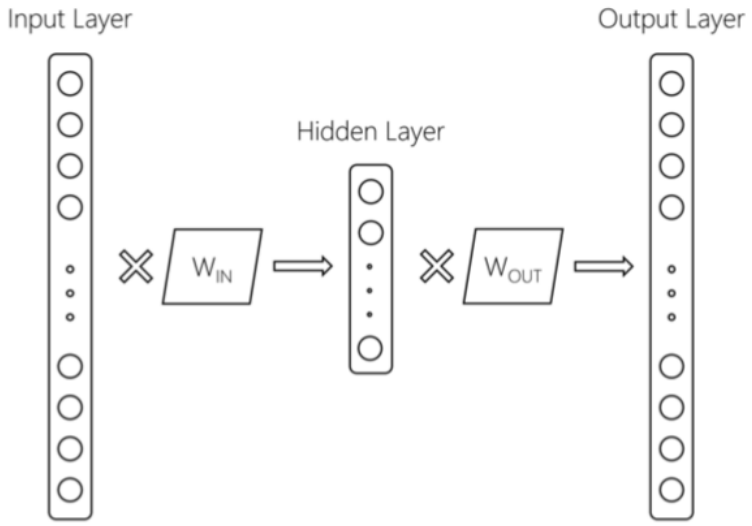


Abbildung 1: Architektur des Neuronalen Netzes CBOW zum Training der Word Embeddings. Hier ist eine Hidden Unit dargestellt. Im Falle des genutzten Models existiert diese Form für 200 Hidden Units. Die Input-Units pro Kontext-Wort werden durch One-Hot-Encoded Vektoren dargestellt.

Die CBOW-Methode wurde im Zuge der Entwicklung des word2vec-Werkzeugs eingeführt. Das Training eines CBOW-Models wird mittels eines probabilistic feedforward shallow neural networks [3] durchgeführt. Dabei wird eine Klassifikationsaufgabe genutzt, um zwei Gewichte-Matrizen zu trainieren. Jedes Wort wird mittels eines festgelegten Wort-Kontexts klassifiziert indem die Input-Neuronen, den Wortkontext darstellen und nach einer einfachen Projektion im Hidden Layer über Linearkombinationen, eine Ausgabe des wahrscheinlichsten Wortes im Output-Layer berechnet wird. Das neuronale Netz hat zwei Gewichtsmatrizen, die "gelernt"werden. Eine Matrix zwischen Input- und Hidden-Layer und eine weitere zwischen Hidden- und Output-Layer. In Abbildung 1 wird die Architektur des neuronalen Netzes dargestellt, wobei die Gewichte durch W_IN und W_OUT angedeutet werden. Die Gewichte werden im Training an den Trainingskorpus angepasst und können anschließend als Word Embeddings genutzt werden. Die Units im Hidden Layer begrenzen hier die Dimensionalität der Word Embeddings, da die Anzahl der Gewichte pro Wort abhängig von der Anzahl der Units ist [4].

Durch die auf den Wortkontext basierenden Klassifikationsaufgabe werden semantische Information eines Wortes in den Word Embeddings kodiert. So liegen Worte ähnlicher Bedeutung näher im Vektorraum beieinander, während sie in einer BOW-Repräsentation unabhängig von ihrer semantischen Ähnlichkeit behandelt werden [4].

2.1.2 Dual Embedding Space Model

Die eigentliche Aufgabe des Relevanz-Rankings für eine Suchanfrage soll mittels des Dual Embedding Space Models (DESM) durchgeführt werden. Dieses mathematische Model nutzt beide Word Embedding Räume eines trainierten CBOW-Models. Die Gewichte der Input-Units werden im folgenden als In-Embeddings und die der Hidden-Units als Out-Embeddings bezeichnet. Dabei werden die Out-Embeddings zur Repräsentation der Dokumente D als D_{Out} und die In-Embeddings zur Repräsentation der Suchanfrage Q als D_{In} verwendet. Als Distanzmaß dient der Kosinus-Abstand zwischen dem Dokument-Zentroiden und jedem Word Embedding der Suchanfrage.

Während der Kosinus-Abstand definiert ist als

$$sim(v_i, v_j) = cos(v_i, v_j) = \frac{v_i^T \cdot v_j}{\|v_i\| \cdot \|v_j\|},$$

wird die Verwendung der Word Embeddings als DESM wie folgt abgebildet:

$$DESM_{In-Out}(Q, D) = \frac{1}{|Q|} \cdot \sum_{q_i \in Q} \frac{q_{In,i}^T \cdot \overline{D_{Out}}}{\|q_{In,i}\| \cdot \|\overline{D_{Out}}\|},$$

wobei ein Dokument-Zentroid definiert ist als

$$\overline{D} = \frac{1}{|D|} \cdot \sum_{d_j \in D} \frac{d_j}{\|d_j\|}.$$

und die Norm eines Vektors als

$$\|v\| = \sqrt{\sum_{i=1}^{n} |v_i|^2}$$

Intuitiv formuliert, wird für jedes Anfrage-Word-Embedding der Kosinus-Abstand zum aktuellen Dokument-Zentroid berechnet, über alle Anfrage-Embeddings aufsummiert und durch die Anzahl der Anfrage-Worte normalisiert.

Hierbei sind Ähnlichkeitswerte höher für Wörter die gemeinsam im Trainings-Korpus bzw. in einem ähnlichen Umfeld auftreten, was eine thematische Ähnlichkeit abbildet. Somit ist beispielsweise das Wort *yale* nicht sehr ähnlich zu *harvard*, wie im Falle von einer In-In-Embedding Repräsentation, sondern *yale* ist ähnlich zu *faculty* [5].

2.2 Sichere verteilte Berechnung

2.2.1 Multi-Party Computation

Damit eine effektive Suche mittels Ranking auf einem verteilten Korpus möglich ist, werden Methoden benötigt, die eine Berechnung ermöglicht, bei der die Eingabe-Werte mehrerer Parteien verwendet werden, ohne das den Parteien die Eingaben der jeweils anderen Parteien offenbart werden. Ebenfalls soll keine Partei involviert sein, der die restlichen Parteien vertrauen müssen. Methoden zur dezentralen Berechnung, ohne eine Partei der die restlichen Parteien vertrauen müssen, liefert der Bereich der Multi-Party Computation (MPC). MPC-Methoden ermöglichen die Berechnung einer Funktion zwischen 2 oder mehreren Parteien, während den Parteien bei der Berechnung nur ihre eigenen Eingaben bekannt sind und je nach Konfiguration das Resultat der Berechnung oder Implikationen die sich aus dem Resultat ergeben. Im Falle des Millionär-Problems, bei dem die Person mit dem größten Vermögen identifiziert werden soll, ohne das explizite Vermögen der Personen zu verraten, würde klar werden, dass die identifizierte Person ein größeres Vermögen hat als der Rest der Personen, mehr jedoch nicht [6].

Es gibt zwei grundlegende Herangehensweisen bei der Multi-Party Computation [6]

1. Yao Circuits [7]: Hier wird eine Funktion durch einen binären Circuit repräsentiert. Eine Partei verschlüsselt die Gates dieses Circuits, auch Garbled Circuit genannt. Eine zweite Partei evaluiert den Circuit abhängig von seinen Eingabe-Werten und erhält auf diese Weise das Resultat.

2. Secret Sharing [8]: In diesem Ansatz wird eine Funktion durch einen arithmetischen Circuit repräsentiert.

Abhängig von der Herangehensweise kann die MPC-Methode gegen verschiedene Angreifermodelle sicher sein [6].

1. Honest-But-Curious Angreifer: In diesem Model wird davon ausgegangen, dass die Parteien das Protokoll zwar verfolgen, aber an einer Offenbarung der privaten Werte der anderen Parteien interessiert sind.

2. Malicious Angreifer: Hier wird von einem Angreifer ausgegangen, der nicht nach dem Protokoll handelt und aktiv daran beteiligt ist die Berechnung zu korrekte Berechnung zu verfälschen.

In dieser Arbeit wird, wie in der Einleitung beschrieben, von einem Zwei-Parteien Fall ausgegangen und das Verfahren der Yao Circuits eingesetzt, weshalb hier nur auf Yao's Protokoll und dessen Umsetzung genauer eingegangen wird. Die Umsetzung geht zudem von einem Honest-but-Curious Angreifermodell aus.

2.2.2 Yao's Protokoll

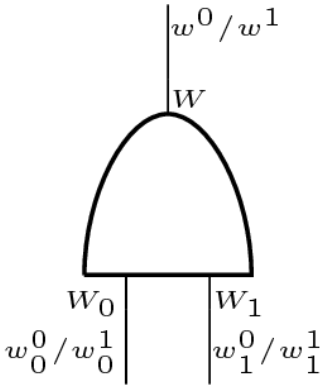
Damit zwei Parteien P_1 und P_2 eine Funktion $f(x, y)$ sicher berechnen können, ohne ihre Eingaben x und y der jeweils anderen Partei zu offenbaren, müssen nach Yao's

Protokoll folgenden Schritte berücksichtigt werden:

Boolschen Circuit konstruieren
Jede Funktion mit festen Eingabewerten kann mittels eines boolschen Circuits C berechnet werden.

Ciruit unkenntlich machen (Garbled Circuit)
P_1 transformiert den Circuit in einen Garbled Circuit, indem sie für jede Leitung (Wire) W_i, zufällig zwei geheime Werte w_i^j wählt, wobei $j \in \{0, 1\}$. Da j aus w_i^j nicht abgeleitet werden kann, muss P_1 sich i und j merken. Dies muss für alle Input- und Output-Leitungen durchgeführt werden.

Unkenntliche Wahrheitstabelle erstellen
Zudem muss P_1 eine unkenntliche Wahrheitstabelle (Garbled Truth Table) T_i für jedes Gate G_i in C anlegen, sodass mittels Garbled Values als Input, der Garbled Output von G_i rekonstruiert werden kann, ohne weitere Informationen zu offenbaren. Dazu verschlüsselt P_1 jede mögliche Ausgabe der boolschen Eingaben mit Hilfe einer symmetrischen Verschlüsselung. Hier dienen die korrespondierenden Garbled Values der möglichen Eingaben als Schlüssel für die Verschlüsselung der Ausgabe.



W_0	W_1	W
0	0	0
0	1	0
1	0	0
1	1	1

Abbildung 2: Darstellung eines Garbled Boolean Circuits und der entsprechenden Garbled Truth Table

Transfer des Eingabewerte von P_2
P_2 erhält den verschlüsselten Garbled Circuit von P_1. Nun muss die Partei P_2 die Input-Bits an P_1 übergeben, damit P_1 der Partei die korrespondierenden Garbled Values mitteilen kann. Mittels Oblivious Transfer [9] kann dies geschehen, ohne das P_1 die wahren Eingabewerte von P_2 offenbart werden. So kennt P_2 nicht den Inhalt des anderen Garbled Values und P_1 kennt nicht die Eingabe von P_2.

P_2 berechnet den Output des Circuits
Nun muss P_2 die Output Garbled Values mit den korrekten Eingabe Garbled Values für

jedes Gate entschlüsseln. Da P_2 am Anfang des Circuits nur seine Eingabe-Bits kennt und alle anderen Werte keine Bits, sondern Garbled Values sind und zudem die Typen der Gates, welche die Eingabe-Werte auf eine Ausgabe abbilden, nicht kennt, werden ihm bis zum Output nur Garbled Values offenbart. Der Output wird nicht verschlüsselt, sodass dieser für P_2 sichtbar wird. P_2 erhält somit das Ergebnis der Funktion $f(x, y)$ und kann es P_1 mitteilen, ohne die Eingabe von P_1 zu kennen oder die eigene zu offenbaren [10].

So kann P_2 beispielsweise ein Dokument-Ranking mit einer Ranking-Score-Funktion $f(x, y)$ und gegebenen Word Embeddings berechnen, wobei die Dokument-Zentroide x darstellen und die Anfrage-Word-Embeddings y. P_1 weiß weder wie die Suchanfrage lautete, noch was das Resultat der Anfrage war, sollte P_2 das Ergebnis der Evaluation des konstruierten Garbled Circuit nicht weitergeben.

Um die hier beschriebene verteilte Berechnung eines Dokument-Rankings sicher mit Yao's Garbled Circuits umzusetzen, wird die auf C basierende Sprache Obliv-C genutzt. Diese hat eine Version von Yao's Protokoll implementiert und bietet eine Umsetzung für eine verteilte Berechnung mit zwei Parteien an [11]. Auf Obliv-C wird im Laufe des nächsten Abschnitts im Detail eingegangen.

Im folgenden wird die Umsetzung des beschriebenen Ziels mit Hilfe der hier erläuterten Methoden besprochen.

3 Umsetzung

3.1 Zielsetzung

Die Umsetzung des DESM-Rankings in einem verteilten Kontext mittels MPC-Methoden ist in 3 Teilziele untergliedert. Als erstes wurde das DESM-Ranking aus dem Paper in C implementiert. Hier wurde die Aufgabe der verteilten Ausführung vernachlässigt, um eine Implementierung mit Fokus auf das Ranking zu ermöglichen. Ziel war die Reproduktion der Ergebnisse aus dem Paper [5] mit allen nötigen Schritten, von der Vorverarbeitung der Dokumente bis zur Berechnung des Rankings mit Hilfe der Word Embeddings aus dem CBOW-Modell.

Daraufhin wurden die Schritte Implementierung, welche in einem 2-Parteien-Fall verteilt ausgeführt werden müssen in den C-Wrapper Obliv-C ausgelagert, welcher eine einfache Implementierung des Yao-Protokolls zur sicheren verteilten Berechnung ermöglicht.

Die Implementierung wurde zum Schluss einerseits mit Hilfe eines neuen Korpus' auf die Effektivität der Relevanz geprüft und außerdem wurde die Performance der C-Implementierung mit der Performance der Obliv-C-Implementierung verglichen.

3.2 Implementierung des DESM-Rankings

Die Implementierung des Rankings sollte in C umgesetzt werden, um die anschließende Umsetzung in Obliv-C zu erleichtern, da es sich bei Obliv-C um einen Wrapper des C-Compilers gcc handelt. Um die Korrektheit der Implementierung zu prüfen, sollte ein Experiment aus dem entsprechenden Paper reproduziert werden. In einem Vergleich der Resultate, sollte das Ergebnis der Implementierung auf Plausibilität geprüft werden. Das Experiment beinhaltet fünf Dokumente. Zwei der Dokumente handeln von Städten mit bedeutsamen Universitäten (Cambridge und Oxford). Ein weiteres Dokument handelt von Giraffen. Idee des Experiments ist es in dem Dokument über Cambridge das Wort Cambridge mit dem Wort Giraffe auszutauschen und dasselbe mit dem Dokument über Giraffen zu tun. Das Ergebnis sollte sein, dass eine BOW-Repräsentation der Dokumente zu einem falschen Ranking aufgrund der vertauschten Worte kommt, während die CBOW-Repräsentation aufgrund der Semantik des Textes weiterhin in einem sinnvollen Ranking resultiert, die Vertauschung von Worten also einen minimalen Einfluss auf das Ergebnis hat.

Ausgangssituation
Bei der Umsetzung war das im Paper verwendete CBOW-Model gegeben, dass auf Datensatz *Bing's large scale search logs* (von August 19, 2014 bis August 25, 2014) trainiert wurde [12]. Außerdem waren 5 Dokumente gegeben, die im zu reproduzierenden Experiment genutzt wurden. Im Anhang sind die Texte zu Cambridge und die vertauschte Variante dargestellt beispielhaft dargestellt.

Somit mussten die Texte vorverarbeitet und tokenisiert werden, die Token durch ihre Word Embeddings aus dem CBOW-Modell abgebildet werden und anschließend die Berechnung des Rankings durch das DESM umgesetzt werden.

Vorverarbeitung
Die Schritte zur Vorbereitung für die Berechnung im DESM wurden mit Python umgesetzt. Die Skript-Sprache eignet sich besonders für die Vorverarbeitung des Korpus' gut, da viele Bibliotheken zur Vorverarbeitung zur Verfügung stehen. Die Vorverarbeitung ist nicht nur für die Abbildung der Worte auf ihre Word Embeddings notwendig. Die Reduktion von Wörtern oder Zeichen in einem Dokument, die keine oder nur sehr wenig Informationen in sich tragen, kommt einem Entfernen von Rauschen gleich und kann somit die Effektivität des Rankings verbessern. Zudem führen weniger Worte in einem Dokument bzw. einer Anfrage zu einer schnelleren Verarbeitung und Berechnung des Rankings. Im Anhang sind die einzelnen Schritte zur Vorverarbeitung und die dazu genutzten Bibliotheken aufgelistet werden.

DESM: Lokale Berechnung der Dokument-Zentroide
Nachdem die Dokumente in eine Menge von vorverarbeiteten Token transformiert wurden, mussten diese durch ihre Word Embeddings abgebildet werden.

Mit Hilfe der Repräsentation der Dokumente durch die Menge der jeweiligen Word Embeddings, musste entsprechend der Dokument-Zentroid berechnet werden. Auch

dies wurde in Python implementiert, sodass nur die nötigen Eingabedaten für die Berechnung des DESM an die C Implementierung übergeben werden müssen.

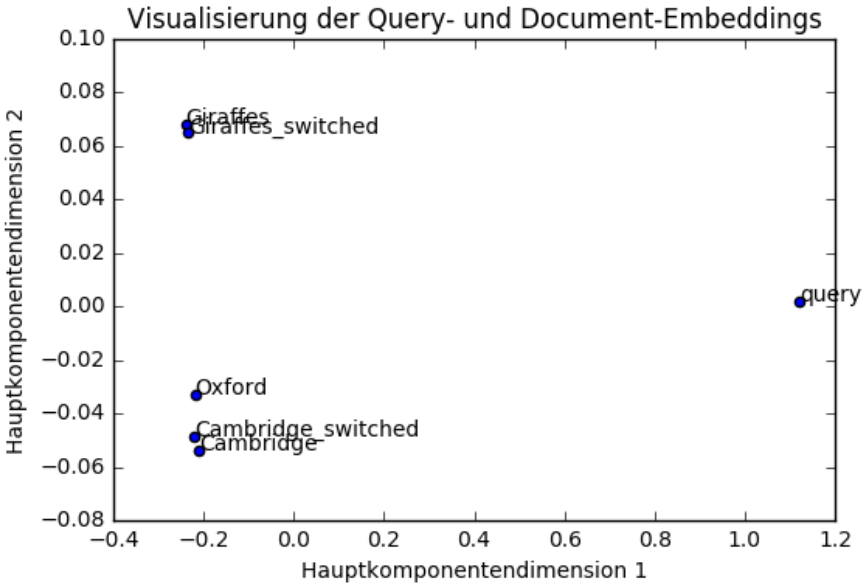


Abbildung 3: Visualisierung der Dokument- und Anfrage-Embeddings in einem Vektorraum, mittels Principle Component Analysis (PCA) auf zwei Dimensionen reduziert.

CFFI: Schnittstelle zwischen Python und C
Als Schnittstelle zwischen Python und C dient die Bibliothek Common Foreign Function Interface (CFFI). Diese ermöglicht die Einbindung von C-Code als Shared Library oder direkt als Code in der CFFI-API, der direkt von der Bibliothek kompiliert werden kann. Für die C-Implementierung wurde letztere Lösung genutzt, da nur C-eigene Bibliotheken bei der Implementierung verwendet wurden und diese Methode in der Dokumentation empfohlen wird [13].

DESM: Implementierung
Die Berechnung des DESM-Rankings erfolgt im C-Programm, wobei über die Menge der Dokumente iteriert wird und pro Dokument der DESM-Score berechnet wird. Dabei wird wiederum die Summierung der normierten Kosinus-Abstände zwischen Dokument-Zentroid und jeweiligem Word Embedding der Anfrage ebenfalls mittels einer Iteration umgesetzt. Die resultierenden Scores werden sortiert nach Größe ausgegeben.

Reproduktion der Test-Rankings
Der Vergleich der resultierenden Scores mit den Scores im Paper zeigt, dass das Ranking zwar äquivalent ist, die Scores selbst jedoch nicht exakt denen des Papers entsprechen. Da im Paper die Vorverarbeitungsschritte nicht erklärt wurden, ist es möglich, dass

Worte ausgelassen oder verwendet wurden, die in der hier implementierten Version berücksichtigt oder vernachlässigt wurden. Dies würde natürlich zu anderen Werten führen, das eigentliche Ranking jedoch nicht maßgeblich beeinflussen, solange es sich nicht um signifikant relevante Word Embeddings für das Dokument handelt. Die Entfernung von Stopworten oder Zahlen ebenso wie die Normalisierung der Dokumente auf Kleinbuchstaben wären mögliche Gründe.

Titel	**Paper**	**C-Implementierung**
Cambridge	-0.062	-0.042
Oxford	-0.070	-0.048
Giraffes	-0.102	-0.081
Giraffes getauscht	-0.094	-0.074
Cambridge getauscht	-0.076	-0.054

Tabelle 1: Vergleich der DESM-Scores aus einem Experiment des Papers. "Giraffes getauschtßowie "Cambridge getauscht"betiteln die Dokumente, in denen das jeweilige Wort durch das andere (Cambridge oder Giraffes) ausgetauscht wurde.

Beim Vergleich der Scores in Tabelle 1 ist zu erkennen, dass die Reihenfolge des Rankings die selbe ist. Auch die Differenzen zwischen den einzelnen Score-Werten ist sehr ähnlich zu einander. Aufgrund der in Relation zu den Ergebnissen des Papers plausiblen Resultaten, wird daher angenommen, dass die Berechnung denen des Papers entsprechen.

3.3 Implementierung eines verteilten, sicheren Rankings

Nachdem die verschiedenen Schritte von der Vorverarbeitung bis zur Ausgabe des Rankings wie im letzten Abschnitt beschrieben in Python und C umgesetzt wurden, sollte die Umsetzung nun auf eine verteilte Berechnung erweitert werden. Dabei soll, wie in der Einleitung erläutert, eine Partei den Korpus bereitstellen und eine zweite Partei die Suchanfrage übergeben. Ziel ist es, dass nur der zweiten Partei das Ranking der Dokumente offenbart wird und ihre Suchanfrage für die erste Partei nicht erkennbar ist. Für die Umsetzung mittels Python, CFFI und Obliv-C müssen derzeit jedoch folgende Einschränkungen definiert werden:

1. Die Menge der Dokumente muss beiden Parteien bekannt sein.
2. Die Anzahl der Worte in der Anfrage muss beiden Parteien für die Iteration während der Berechnung bekannt gemacht werden.

Diese Einschränkungen bestehen auch aufgrund der Ausführung von Iterationen in Obliv-C, die beispielsweise durch die Menge der Anfrage-Word-Embeddings begrenzt sind. Dazu könnte die im Paper [11] erwähnte Festlegung einer maximalen Anzahl an

Iterationen und der tatsächlichen Begrenzung mittels *obliv if*-Kontrollstruktur umgesetzt werden. Dies hätte jedoch wiederum negative Folgen für die Laufzeit-Performance der Umsetzung, weshalb auf diese Herangehensweise verzichtet wurde.

Die Schritte zur Vorbereitung der Berechnung des Rankings entsprechen hier der Beschreibung aus dem letzten Abschnitt. Da diese lokal ausgeführt werden, ist es nicht nötig erneut darauf einzugehen. Um das Obliv-C Programm über CFFI in Python einzubinden, wurde das Programm als Shared Library ausgegeben.

Lokale Berechnung der euklidischen Norm
Die Berechnung des DESM-Rankings kann teils lokal berechnet werden. Die notwendige Berechnung der Norm des jeweiligen Dokuments bzw. der Anfrage-Word-Embeddings benötigt keine Informationen der anderen Partei, weshalb dieser Teil im C-Code berechnet wird.

Sichere, verteilte Berechnung des Kosinus-Abstands und Score
Da die eigentliche Berechnung des DESM-Scores die Dokument-Zentroide sowie die Anfrage Word Embeddings benötigt, muss die restliche Berechnung in Obliv-C umgesetzt werden. Obliv-C ist eine Sprache zur Implementierung von Multi-Party Computation Protokollen und unterstützt erweiterbare sichere Programmierung auch für Entwickler, die keine Experten im Bereich der Kryptographie sind. Mittels syntaktischer Elemente, werden Prozesse sicherer bzw. verschlüsselter Berechnungen im Programm sichtbar gemacht. Obliv-C ist eine strikte Erweiterung der Sprache C und unterstützt somit alle C-eigenen Datentypen und Kontrollstrukturen. Hinzu kommt unter anderem der obliv-Qualifier, womit ein Datentyp oder eine Variable in einer if-Kontrollstruktur für die andere Partei *versteckt* bzw. verschlüsselt wird.

Ein Obliv-C Programm wird grundlegend so ausgeführt, dass beide Parteien dasselbe Programm ausführen. Daten die zu einer verteilten Berechnung gehören, werden durch den obliv-Qualifier erweitert und müssen über eine Funktion kryptographisch gesichert werden. Detailliertere Informationen zu Obliv-C können dem Paper [11] entnommen werden.

Da im Besonderen die Anfrage geschützt werden muss, in weiteren Umsetzungen jedoch auch unklar sein soll, von wem welche Dokumente bereitgestellt werden, müssen die Dokument-Zentroide sowie die Anfrage Embeddings in obliv-qualified Variablen konvertiert werden. Der weitere Verlauf der Umsetzung gleicht der Erläuterung im letzten Abschnitt mit dem Unterschied der Verwendung von obliv-Variablen.

Ausgabe des Ergebnisses
Das Ranking wird im Obliv-C-Teil ausschließlich der zweiten Partei, welche die Anfrage gestellt hat, offenbart. Auf das Ranking kann wiederum vom C-Code zugegriffen werden. Wenn der C-Code nun von der zweiten Partei ausgeführt wird, sortiert das Programm zuletzt die ausgegebenen Rankings und gibt die Indices in, nach Score sortierter Reihenfolge, zurück. In einer realen Anwendung könnten hier entweder die vollständigen Texte zurückgegeben werden oder aber die Anfragepartei benötigt Zugang zu allen Dokumenten, um diese über den Index zu erhalten.

3.4 Evaluation von Effektivität und Effizienz

Die Implementierung wurde zum einen auf einem Korpus zur Validierung von Rankings überprüft. Außerdem wurde die Performance der C-Implementation und der Obliv-C-Implementation anhand der Laufzeit gemessen und soll hier verglichen werden.

Evaluation der Effektivität
Die Effektivität der DESM-Ranking-Implementation wurde mit Hilfe des SIGIR 2013: News Vertical Search Dataset evaluiert. Der Korpus besteht aus Nachrichten-bezogenen Anfrage-Texten und Dokumenten aus verschiedenen Quellen aus dem Zeitraum vom 10. bis 16. Januar 2013 [14]. Jede Datei des Datensatzes enthält eine Tabelle von Anfragen, die zu einem Thema gehören und entsprechenden gerankten Dokumenten. Während der ursprüngliche Datensatz aus den Quellen Twitter, News, Blogs, Reddit, Wikipedia und Bing bestand, wurde der hier genutzte Korpus auf Ergebnissen von News und Reddit beschränkt, da nur diese Quellen Dokumente enthielten, die aus mehreren Sätzen bestanden. Hieraus wurden wiederum 1000 Artikel extrahiert, wobei 10 Artikel zu einer Anfrage gehören. Somit wurden 100 Anfragen zufällig aus dem gesamten Datensatz gewählt zusammen mit jeweils 10 Artikeln. Nach einer Vorverarbeitung des Korpus', in der nicht-valide Dokumente entfernt wurden, bleibt ein Korpus von insgesamt 830 Dokumenten.

Aus zeitlichen Gründen wurde hier ein einfacher Ansatz zur Evaluation gewählt bei dem angenommen wurde, dass Dokumenten, die bei der Suchanfrage auftreten als relevant und alle anderen als irrelevant gelten. Dies führt jedoch dazu, dass semantisch ähnliche Artikel, die nicht bei der Suchanfrage auftraten, nicht als relevant berücksichtigt wurden. Im folgenden ist eine Tabelle zu sehen, welche zufällig gewählten Suchanfragen zeigt und die dazugehörige relevante Anzahl an Dokumenten sowie der Median der Ranking-Positionen dieser Dokumente nach Anwendung des DESM-Rankings. Aufgrund der vereinfachten Evaluation, sind die Ergebnisse nur als Orientierung für die Effektivität des Ansatzes gedacht.

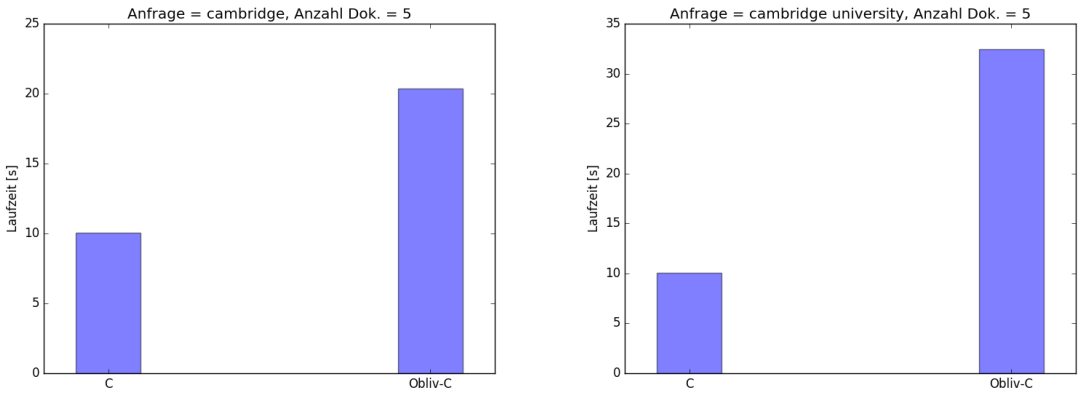
Anfrage	**Anzahl Relevant**	**Ranking Median**
whats the point	10	27.5
aaron is dead	10	53.5
justin timberlake	10	18.5
devil may cry	20	27.5

Tabelle 2: Median der Ranking-Positionen der relevanten Dokumente mit einem Korpus von 830 Dokumenten.

Anhand der beispielhaften Ergebnisse in Tabelle 2 kann man sehen, dass die Ranking-Ergebnisse im Median höchstens bei etwas über 50 liegen. In Anbetracht der Ausgangssituation mit insgesamt 830 Dokumenten und der Relativität solcher Relevanzeinschätzungen weicht das Ergebnis im Median nicht maßgeblich von der relevanten Dokumentmenge ab. Hier muss jedoch nochmals darauf hingewiesen werden, dass

Dokumente mit hohem Ranking durchaus als relevante Ergebnisse eingestuft werden könnten, jedoch nicht unter dem entsprechenden Suchbegriff im Datensatz aufgeführt sind. Für eine detailliertere Darstellung, wird auf den Anhang verwiesen.

Evaluation der Laufzeit
Des Weiteren wurde mit verschiedenen Dokumentenmengen und Anfrageworten die Laufzeit des C- und Obliv-C-Programms gemessen. Die folgenden Plots enthalten die Laufzeit pro Programmausführung. Zudem befinden sich im Anhang Tabellen mit den jeweiligen genauen Messwerten. Dabei ist die geringere Laufzeit in der Spalte der Obliv-C Implementierung dadurch zu erklären, das hier die Ausführung der Server-Partei dargestellt ist, während die C-Implementierung die Schritte, im besonderen die Vorverarbeitung, beider Parteien ausführt.



(a) Laufzeit mit einem Anfragewort (b) Laufzeit mit zwei Anfrageworten

Abbildung 4: Vergleich zwischen verschiedener Anzahl von Anfrageworten

Der Vergleich der Laufzeiten zwischen der C und der Obliv-C Implementierung, welche im Anhang in der Tabelle 3 im Detail dargestellt ist, zeigt, dass im Gegensatz zur C Implementierung die Laufzeit bei der Berechnung des Rankings signifikant zunimmt, während die Verteilung der Daten bzw. die Umwandlung in obliv-qualified Daten, nur wenig Laufzeit beansprucht. So benötigt das Ranking der C Implementierung deutlich weniger als eine Sekunde. Das Obliv-C Programm hingegen benötigt über 14 Sekunden. In Abbildung 4 ist die Gesamtlaufzeit abgebildet.

Die Hinzunahme eines Anfragewortes in Abbildung 4, Plot (b), verhält sich ungefähr linear zur Laufzeit der Ranking-Berechnung. Dies wird zudem in Abbildung 5 illustriert. Hier wird ausschließlich die Laufzeit des Rankings der Obliv-C Implementierung mit ein, zwei und drei Anfrageworten abgebildet.

Für eine Darstellung der Laufzeit mit logarithmisch ansteigender Zahl an Dokumenten dient die Abbildung 6.

Die Abbildung 6 zeigt, dass die Laufzeit bei Erhöhung der Dokumentanzahl von 2^1 bis 2^9 nahezu linear ansteigt. Grund für dieses Verhalten ist vermutlich der kryptographische Prozess sowie die verteilte Berechnung durch zusätzlichen Netzwerk-Verkehr

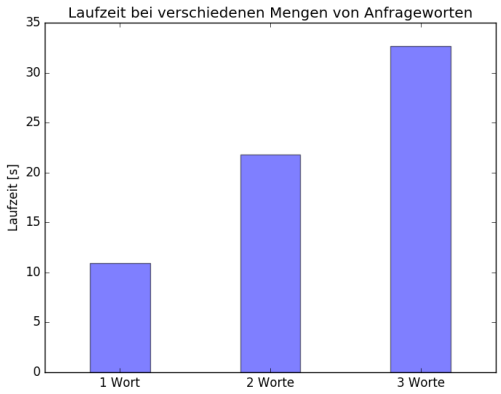


Abbildung 5: Ranking-Laufzeit der Obliv-C Implementierung mit unterschiedlicher Anzahl an Anfrageworten.

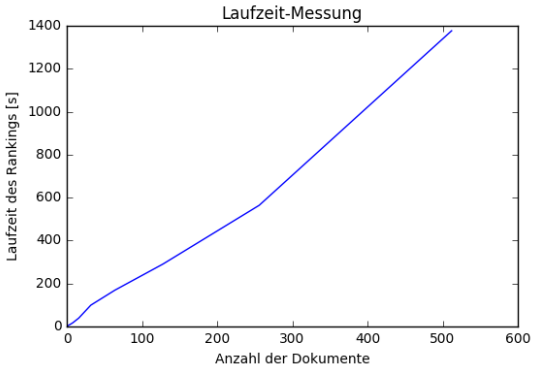


Abbildung 6: Ranking-Laufzeit der Obliv-C Implementierung mit logarithmisch ansteigender Anzahl an Dokumenten.

pro Dokument. Bei über 30 Minuten Laufzeit des Rankings, bei über 800 Dokumenten, ist diese Implementierung aus Sicht der Performance praktisch unzureichend.

Noch detailliertere Informationen können dem Anhang oder dem Log der Ausführung entnommen werden. Dort sind auch die Laufzeiten der 2. Partei angegeben. Hier soll nur erwähnt werden, dass für die Vorverarbeitung eines Wortes 3.4563 Sekunden notwendig sind. Die Abbildung des Word Embeddings auf ein Wort beträgt 0.023139 Sekunden. Das Laden und Konvertieren eines Vektors in eine obliv-Variable benötigt 0.045731 Sekunden.

4 Ausblick

Die Umsetzung und Evaluierung des verteilten DESM-Rankings zeigt, dass besonders auf kleinen Korpora ein Ranking auf sichere und verteilte Weise möglich und in vertretbarer Zeit berechenbar ist, auf einem größeren Korpus, besonders aber auch bei der Erhöhung von Anfrageworten, ist die Laufzeit der derzeitige Implementierung jedoch unzureichend. Die Laufzeit könnte dabei durch eine andere Lösung für die Abbildung der Token auf die Word Embeddings reduziert werden. Möglicherweise könnte eine vollständige Implementation, des Rankings sowie der Vorverarbeitung, in C die Gesamtlaufzeit beschleunigen. Eine weitere Möglichkeit zur Verbesserung der Performance, wäre die Nutzung mehrerer Threads. Neben Bibliotheken für Python, welche dies umsetzen, gibt es auch in Obliv-C bereits Möglichkeiten Parallelisierung durchzuführen [11]. Da die Berechnung der Rankings der Dokumente nicht untereinander abhängig sind, bietet sich, neben der Vorverarbeitung, auch hier eine Parallelisierung an.

Außerdem könnte die Implementation weiter generalisiert werden. Derzeit ist die Dimensionalität der Word Embeddings auf 200 festgelegt, da es nicht möglich war dynamische 2-dimensionale Arrays über CFFI von Python an C zu übergeben. Des Weiteren könnten die derzeitigen Einschränkungen bzgl. der beidseitigen Kenntnis der Anzahl der Worte einer Anfrage sowie der Dokumentenmenge umgangen werden, sodass diese Informationen nicht jeder Partei bekannt sein müssen.

Wünschenswerte Erweiterungen dieser Implementation wäre einerseits die Verteilung des Korpus auf beide Parteien, sodass die Anfragepartei zusätzlich auf eigenen Dokumenten sucht, die der anderen Partei nicht bekannt sein soll.

Außerdem wäre die Verbesserung des DESM-Rankings durch eine Kombination mit der BM25-Methode, die auf einer BOW-Methode basiert, sinnvoll, um die Effektivität des Rankings besonders auf größeren Korpora zu verbessern [5].

Ein nächster Schritt, der über diese Implementierung hinaus reicht, wäre die Erweiterung des Rankings auf mehr als zwei Parteien. Dies würde einem Anwendungsfall, wie beispielsweise der Suche auf Diagnoseberichten in einem Zusammenschluss von Krankenhäusern, näher kommen, um die Machbarkeit eines solchen Anwendungsfalls mit Methoden der Multi-Party Computation evaluieren zu können.

Literatur

[1] Mathis(Vorsitzender) Ruff. Datenschutz im internetzeitalter, Aufgerufen am: 18.05.2018.

[2] Thomas Tolxdorff and Frank Puppe. Data warehouse, klinisches, Juni 08, 2016, Aufgerufen am: 18.05.2018.

[3] Yoshua Bengio, Regean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. 2003.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013.

[5] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. Improving Document Ranking with Dual Word Embedding. 2016.

[6] Nigel Smart. *Cryptography Made Simple*. 2016.

[7] A.C. Yao. How to generate and exchange secrets. 1986.

[8] Adi Shamir. How to Share a Secret. 1979.

[9] Michael O. Rabin. How to Exchange Secrets with Oblivious Transfer. 1981.

[10] Yolan Romailler. Yao's garbled circuits and how to construct those, Juni 09, 2017, Aufgerufen am: 18.05.2018.

[11] Samee Zahur and David Evans. Obliv-C: A Language for Extensible Data-Oblivious Computation. 2015.

[12] Bhaskar Mitra, Nick Craswell, and Rich Caruana. Dual Embedding Space Model, January 21, 2016, aufgerufen am: 04.03.2018.

[13] Armin Rigo and Maciej Fijalkowski. Cffi documentation, 2012-2018, Aufgerufen am: 04.05.2018.

[14] IadhOunis. Sigir 2013: News vertical search dataset, Juli 28, 2013, Aufgerufen am: 15.05.2018.

5 Anhang

Beispieltexte aus dem reproduzierten Experiment

Text über Cambridge
The city of Cambridge is a university city and the county town of Cambridgeshire, England. It lies in East Anglia, on the River Cam, about 50 miles (80 km) north of London. According to the United Kingdom Census 2011, its population was 123,867 (including 24,488 students). This makes Cambridge the second largest city in Cambridgeshire after Peterborough, and the 54th largest in the United Kingdom. There is archaeological evidence of settlement in the area during the Bronze Age and Roman times; under Viking rule Cambridge became an important trading centre. The first town charters were granted in the 12th century, although city status was not conferred until 1951.

Text über Cambridge, wobei "Cambridge"mit Giraffe ausgetauscht wurde
The city of Giraffe is a university city and the county town of Cambridgeshire, England. It lies in East Anglia, on the River Cam, about 50 miles (80 km) north of London. According to the United Kingdom Census 2011, its population was 123,867 (including 24,488 students). This makes Giraffe the second largest city in Cambridgeshire after Peterborough, and the 54th largest in the United Kingdom. There is archaeological evidence of settlement in the area during the Bronze Age and Roman times; under Viking rule Giraffe became an important trading centre. The first town charters were granted in the 12th century, although city status was not conferred until 1951.

Vorverarbeitungsschritte im Experiment

- Konvertierung zu Kleinbuchstaben, um den Korpus maßgeblich zu verkleinern.
- Tokenisierung der Sätze, um Vorverarbeitungsschritte auf Satzebene durchzuführen.
- Eventuelle Encoding-Fehler durch entsprechendes Zeichen ersetzen, da diese das Vokabular vergrößern, ohne eine wesentliche Information zu beinhalten.
- Token die mit http beginnen entfernen (naive Erkennung von URLs), da jede URL das Vokabular unnötig vergrößert und die Wahrscheinlichkeit, dass das CBOW-Model eine URL nicht kennt groß ist.
- Satzzeichen entfernen, da diese in der Regel keine bedeutsame Informationen beinhalten und das Vokabular der Dokumentmenge unnötig vergrößeren.
- Tokenisierung der Worte, um die Token später auf die Word Embeddings abbilden zu können.
- Alle Token der Länge 1 entfernen (mit Ausnahme von i für das englische Pronomen), da diese im allgemeinen Falle keine Dokument-spezifisch Information liefern.
- Zahlen entfernen, da diese in den meisten Fällen keine aussagekräftige Information über ein Dokument liefern.

Performance Test-Ergebnisse (Ausschnitt)

Ausführungsschritt	**Laufzeit in C**	**Laufzeit in Obliv-C**
Vorverarbeitung	0.064650	0.066417
Abbildung auf Word Embeddings	3.378596	3.648695
Zentroide berechnen	0.161661	0.170682
Gesamte Vorbereitung	10.077279	5.725848
Normen berechnen	0.000005	0.000423
In obliv umwandeln	na	0.074633
Ranking berechnen	0.000004	10.906001
Gesamtes Yao-Protokoll	na	11.132056
Gesamtes Ranking	0.000298	14.637919
Gesamtzeit	10.077695	20.364021

Tabelle 3: Laufzeit-Vergleich zwischen C und Obliv-C. Anfrage = cambridge, Anzahl Dokumente= 5

Ausführungsschritt	**Laufzeit in C**	**Laufzeit in Obliv-C**
Vorverarbeitung	0.065603	0.065777
Abbildung auf Word Embeddings	3.391233	3.513459
Zentroide berechnen	0.162693	0.171808
Gesamte Vorbereitung	10.089242	5.598006
Normen berechnen	0.000006	0.000735
In obliv umwandeln	na	0.121308
Ranking berechnen	0.000006	21.817545
Gesamtes Yao-Protokoll	na	22.103311
Gesamtes Ranking	0.000271	26.849374
Gesamtzeit	10.089631	32.447652

Tabelle 4: Laufzeit-Vergleich zwischen C und Obliv-C. Anfrage = cambridge university, Anzahl Dokumente= 5

Ausführungsschritt	**Laufzeit in C**	**Laufzeit in Obliv-C**
Vorverarbeitung	0.064574	0.065942
Abbildung auf Word Embeddings	3.377046	3.598886
Zentroide berechnen	0.161456	0.187473
Gesamte Vorbereitung	10.081188	5.692507
Normen berechnen	0.000006	0.001031
In obliv umwandeln	na	0.167084
Ranking berechnen	0.000008	32.658706
Gesamtes Yao-Protokoll	na	32.981617
Gesamtes Ranking	0.000284	36.969102
Gesamtzeit	10.081592	42.661763

Tabelle 5: Laufzeit-Vergleich zwischen C und Obliv-C. Anfrage = cambridge university giraffes, Anzahl Dokumente= 5

Ausführungsschritt	**Laufzeit in C**	**Laufzeit in Obliv-C**
Vorverarbeitung	5.990394	6.252674
Abbildung auf Word Embeddings	27.430468	28.654633
Zentroide berechnen	25.712314	26.225077
Gesamte Vorbereitung	66.602793	62.819893
Normen berechnen	0.000332	0.027224
In obliv umwandeln	na	4.374763
Ranking berechnen	0.000349	1815.724618
Gesamtes Yao-Protokoll	na	1820.302117
Gesamtes Ranking	0.008706	1845.986122
Gesamtzeit	66.611886	1908.806134

Tabelle 6: Laufzeit-Vergleich zwischen C und Obliv-C. Anfrage = cambridge, Anzahl Dokumente > 800

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 18. Juni 2018 .....