

Konzeption und Implementierung eines Wissensgraphen auf  
Basis von Daten aus semantischen Analysen

**Masterthesis**

zur Erlangung des akademischen Grades  
**Master of Science (M.Sc.)**

im Studiengang Angewandte Informatik mit Schwerpunkt Umwelt-  
und Wirtschaftsinformatik am Umwelt-Campus Birkenfeld

vorgelegt von  
Jana Albert

Erstprüfer: Prof. Dr. Ing. Guido Dartmann  
Zweitprüferin: Marlies Morgen (M.Sc)

Wintersemester 2022/2023  
Abgabedatum: 06.02.2023

---

## **Eidesstattliche Erklärung**

Ich erkläre hiermit, dass ich die vorliegende Masterthesis mit dem Titel „Konzeption und Implementierung eines Wissensgraphen auf Basis von Daten aus semantischen Analysen“ selbst verfasst habe und dass ich dazu keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe. Alle Textstellen, die wörtlich oder sinngemäß aus anderen Werken übernommen wurden, sind als solche gekennzeichnet. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle vorgelegen.

---

Ort, Datum

---

Unterschrift der Kandidatin/des Kandidaten

---

## **Kurzfassung**

---

## Tabellenverzeichnis

3.1	Definition der Begriffe Wissensbasis, Wissensgraph und Ontologie ..	9
3.2	Definition der Begriffe Morphologie, Pragmatik, Syntax und Semantik im Zusammenhang mit NLU .....	18
3.3	Überblick bekannter Ansätze zur Extraktion von keywords aus Textdokumenten[Del21]. .....	22
5.1	Exemplarischer Aufbau der metadata.list.xlsx Datei, welche die Metadaten der Datensammlung speichert.....	32
B.1	Auflistung aller verwendeten Bibliotheken, Packages und Modulen mit Versionen und Beschreibungen [pip] Installation über pip: <a href="https://pypi.org">https://pypi.org</a> .....	51
B.2	Auflistung aller verwendeten in-build python Module .....	52

---

## Abbildungsverzeichnis

2.1	Existierende Lösungen für graphische Datenbanken mit Zuordnung der zugehörigen Organisationen [Yea] . . . . .	4
2.2	Mögliche Differenzierung der Anwendungsgebiete von Wissensgraphen [Zou20] . . . . .	5
2.3	Einflüsse verschiedener Komponenten auf GQL [Gre19] . . . . .	6
3.1	Vergleich eines Wissensgraphen (oben) mit einer Ontologie (unten) [Sch20] . . . . .	9
3.2	Gegenüberstellung des syntaktischen und semantischen Webs am Beispiel einer Suchanfrage [BHL <sup>+</sup> 14] . . . . .	11
3.3	Beispielhafte Darstellung de Google Knowledge Graphs bei einer Suchanfrage "Hauptstadt Australien" [Kop19] . . . . .	12
3.4	RDF Beschreibung des Beispiels Nike, Dahliastraat 24, 2160 Wommelgem. Rot: Subjekt, Blau: Prädikat, Grün: Objekt [KNLG15].	13
3.5	Graph zur Beispiel RDF Beschreibung in Abbildung 3.4. Rot: Subjekt, Blau: Prädikat, Grün: Objekt. Die Webseite der Organisation kann wird hier als Subjekt und Objekt gelesen. . . . .	14
3.6	Beispiel SPARQL Funktion in vereinfachter Darstellung [Ontb]. . . . .	15
3.7	Neo4j Graph [Neoa] . . . . .	16
3.8	Allgemeiner Prozess vom natürlichen Text zum Wissensgraph [AMOOV20] . . . . .	18
4.1	Programmablaufplan vom Einlesen der Metadaten der original PDF Dateien . . . . .	26
4.2	Konzipierter Graph, der durch das Auslesen der Metadaten der Textdokumente entsteht. . . . .	28
4.3	Konzipierter Graph, der durch das Erstellen von keywords und keyphrases aus den Textdokumenten entsteht. Hier vereinfachte Darstellung des Knotens Thema (umfasst Thema Konferenz und Thema paper). . . . .	29
5.1	Hinzufügen einer Test PDF Datei energy_efficiency_Test zu Excel Datei metadata_list . . . . .	33

5.2	Programmablaufplan des Skripts PreProcess_NLP . . . . .	34
5.3	Generierter Graph nur mittels Networkx. Mit Knoten Beschriftung (oben). Ohne Knoten Beschriftung (unten). . . . .	35
5.4	Programmablaufplan des Skripts KG . . . . .	36
5.5	Erstellter Graph durch RDFLib und Networkx. . . . .	37
7.1	Zusammengefasster Ablaufplan mit allen Unterprogrammen . . . . .	42

---

## Listings

---

# Inhaltsverzeichnis

<b>1 Einleitung und Problemstellung .....</b>	1
1.1 Motivation .....	3
1.2 Problemstellung und Zielsetzung .....	3
<b>2 Stand der Technik .....</b>	4
<b>3 Theoretische Grundlagen .....</b>	8
3.1 Ontologien, Wissensgraphen und Wissensbasen .....	8
3.1.1 Anwendungsgebiete und das Semantische Web .....	10
3.1.2 Property Graphs und Cypher .....	15
3.2 Natural Language Processing .....	17
3.2.1 Pre- processing .....	18
3.2.2 Datenextraktion .....	21
<b>4 Konzeption .....</b>	24
4.1 Anforderungen und Formulierung der verschiedenen Problemstellungen .....	24
4.2 Einlesen und Aufbereitung der Rohdaten .....	25
4.3 Entitäten und Beziehungen .....	27
4.4 Aufbau und Abfrage des Graphen .....	29
<b>5 Implementierung und Anwendung .....</b>	31
5.1 Rahmen und Sammlung der Daten .....	31
5.1.1 Erweiterung und Aufbereitung der Datenbasis .....	32
5.2 Entstehung und Visualisierung des Graphen .....	33
5.2.1 Abfragen des Graphen .....	37
<b>6 Ergebnisse .....</b>	39
<b>7 Zusammenfassung und Ausblick .....</b>	41
<b>Literaturverzeichnis .....</b>	44
<b>Glossar .....</b>	48

<b>Anhang</b> .....	49
B.1 Stopword Liste .....	49
B.2 Nutzbare Transformer und Modelle für KeyBert .....	49
B.3 Pakete, Module, Frameworks .....	50

# 1

---

## Einleitung und Problemstellung

Im digitalen Zeitalter, in dem Maschinen und künstlicher Intelligenz eine immer wichtiger Rolle zukommt und die Art und Weise wie Daten gehalten, sowie auch repräsentiert werden, gibt es viele Ansätze mit denen sich eingehend beschäftigt werden kann. Ein bis heute beliebtes Datenmodell sind relationale Datenbanken. Hier werden Daten in tabellarischer Form gehalten. Dabei können diese Tabellen auf andere Tabellen durch unique keys (eindeutiger Schlüssel) aufeinander verweisen. Somit können in einer Tabelle Kundeninformationen sein, die wiederum auf eine andere Tabelle mit Produktinformationen, die Kunden gekauft haben, verweisen [Joy21]. Diese bieten jedoch nicht die Möglichkeit Beziehungen zwischen den Daten abzubilden, die ebenfalls als Daten gehalten werden. Auch können komplexe Anfragen nicht schnell beantwortet werden. Abhilfe schaffen hier Graph Datenbanken. Hier werden die Daten in graphischer Form, also in der Form eines Netzwerks gehalten, in dem Entitäten durch Beziehungen miteinander verbunden sind [Joy21]. Eine besondere Art einer Graph Datenbank sind Wissensgraphen, die auch zum Themengebiet der Wissensrepräsentation zählen. Diese Arbeit wird sich mit dem Themengebiet der Wissensrepräsentation und Wissensgraphen auseinandersetzen, welches auch zum Gebiet des semantischen Webs zählt.

Der Begriff Wissensrepräsentation beschreibt die Problematik, wie aufgebautes Wissen weitergegeben, genutzt und erweitert werden bzw. auch wie es erhalten bleiben soll. Dies soll mittels Maschinen und Menschen ermöglicht werden [DZ07]. Der Begriff des semantischen Webs ist meist im Zusammenhang mit Suchmaschinen bekannt. Dabei geht es darum, dass einer Suchanfrage auch eine semantische Bedeutung zugeordnet wird, wodurch bessere Suchergebnisse erzielt werden können. Google nutzt diese Technik schon länger und arbeitet fortlaufend an deren Verbesserung. Doch was ist nötig, um natürlicher Sprache durch Maschinen auch eine semantische Bedeutung zukommen zu lassen?

Es existieren bereits eine Vielzahl an Algorithmen, welche natürliche Sprache verarbeiten können, Natural Language Processing (NLP) genannt. Durch diese Algorithmen können wichtige Informationen aus Fließtexten ausgelesen und im Anschluss weiter verarbeitet werden. Es ist auch möglich, diese Informationen in einem Graphen zusammenzutragen und durch Beziehungen miteinander in Ver-

bindung zu setzen. Somit entstehen Netzwerke, die von Maschinen verwendet und ausgelesen werden können. Auf diese Weise können diverse Informationen in einer Form gespeichert werden, die es ermöglicht einen Gesamtzusammenhang zwischen diesen zu schaffen. In Kombination mit künstlicher Intelligenz können sogar neue Zusammenhänge erkannt werden, die der Mensch eventuell nicht erkennen kann. Die Erstellung von Prognosen auf Basis einer Ansammlung an Daten und ein schnelles Finden von spezifischen Informationen wird ermöglicht.

In dieser Arbeit wird eine Datenbasis geschaffen, die als Grundlage für solch weiterführende Funktionalitäten genutzt werden kann und als System fungiert, welches abgefragt und erweitert werden kann. Dieses System ist durch Entitäten und Beziehungen gekennzeichnet.

Im Kapitel Theoretische Grundlagen wird eine Basis an grundlegendem Wissen geschaffen, um die folgenden Techniken, Methoden und Implementierungen verständlich zu gestalten. Neben der theoretischen Funktionalität von Wissensgraphen wird auf die Entstehung und den Aufbau des semantischen Webs eingegangen, sowie auf konkrete Anwendungsfälle aus der realen Welt. Auch werden die Grundlagen von NLP und einige relevanten Sprachwissenschaftliche Aspekte vorgestellt. Darüber hinaus wird auf einige bestimmte Techniken eingegangen, die in der späteren Implementierung ihre Anwendung finden.

Nachdem diese Basis geschaffen ist, wird die Konzeption der zu erstellenden Wissensbasis aus natürlicher Sprache ein Themengebiet der Arbeit einnehmen. An dieser Stelle werden Anforderungen und Problemstellungen gebündelt und formuliert. Auch die Vorgehensweise der Sammlung von Rohdaten und deren Aufbereitung zum späteren Einspeisen in die Wissensrepräsentation wird definiert. Es werden Kernschritte zur Umsetzung formuliert, Programmablaufpläne sowie der aufzubauende Graph mit Entitäten und Beziehungen konzipiert. Zuletzt wird das Gesamtergebnis vorgestellt.

Nachfolgend wird auf die Implementierung und Anwendung eingegangen. Hier wird auf die konkrete Datengrundlage eingegangen und beschrieben. Auch die verwendete Umgebung und organisatorische Aspekte des Programms werden dargestellt. Alle Funktionen, deren Ergebnisse und die Einordnung in den Gesamtrahmen der Implementierung werden vorgestellt und eingehend erläutert. Zwischenergebnisse werden gezeigt und bestimmte Methoden, die bereits in der Theorie vorgestellt wurden, werden erneut aufgegriffen und in Verbindung des jeweiligen Anwendungsfalls erläutert. Auch Anwendungsbeispiele und die Nutzung des Programms wird an einigen Beispielen gezeigt und erklärt. Im Anschluss daran werden die Daten in einem Graphen zusammengetragen und visualisiert.

Nach der Umsetzung werden in Kapitel 6 die erzielten Ergebnisse verglichen und bewertet. Dabei wird der implementierte Ansatz mit anderen Ansätzen verglichen, sowie Vor-und Nachteile aufgezeigt.

Am Ende wird in Kapitel 7 noch einmal die gesamte Arbeit und ihrer Ergebnisse reflektiert, zusammengefasst und bewertet. Umgesetztes wird kritisch betrachtet

und weitere Möglichkeiten zur Nutzung oder Erweiterung in Form eines Ausblicks diskutiert.

## 1.1 Motivation

Das Vorgehen Daten so miteinander zu verknüpfen, dass ihnen eine semantische Bedeutung zugeordnet wird, kann in den verschiedensten Bereichen angewendet werden. Ein Bereich stellt hier die Extraktion von gesuchten Informationen. Es existieren bereits viele Datenbanken mit wissenschaftlichen papern, doch die Suche nach bestimmten Themen, Autoren oder Konferenzen ist meist mühsam. Schon bei der Recherche nach einer geeigneten Datenbasis wurde klar, dass im wissenschaftlichen Raum das Finden von passenden Informationen sich zeitaufwendig gestaltet. Aus diesem Grund wird im Rahmen dieser Arbeit eine Datenbasis aus wissenschaftlichen papern geschaffen, die es ermöglicht gewünschte Informationen an den Nutzer herauszugeben. Die Suche kann beispielsweise über Autoren, Veröffentlichungsdaten oder Themen erfolgen, die den Nutzer interessieren. Somit soll es bei einer weit ausgebauten Wissensbasis möglich sein, schneller wissenschaftliche Arbeiten zu bestimmten Themen oder spezielle Autoren zu finden. In dieser Arbeit werden solche paper genutzt, die zum Einen frei zugänglich sind und zum Anderen sich mit dem Themengebiet der Informatik oder auch Mathematik beschäftigen. Auch werden Konferenzen auf denen die paper veröffentlicht wurden miteinbezogen, um die Palette an Nutzungsmöglichkeiten zu erweitern.

## 1.2 Problemstellung und Zielsetzung

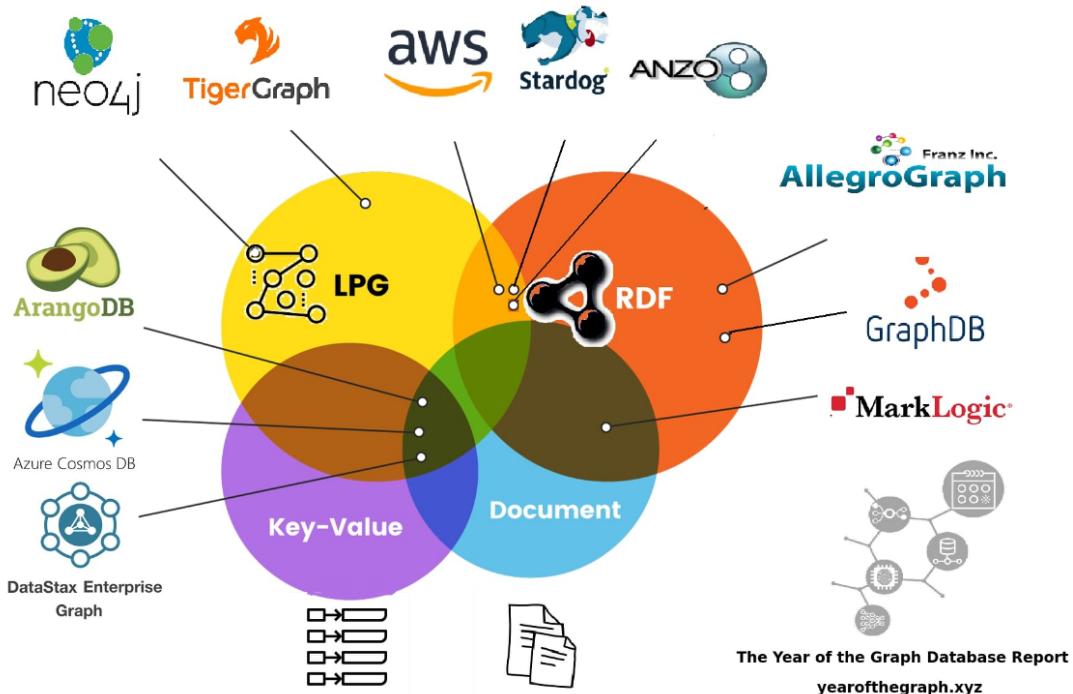
Im Rahmen dieser Arbeit soll eine Wissensbasis aufgebaut werden, die aus wissenschaftlichen papern und zugehörigen Konferenzen besteht. Dabei werden, nach Schaffung einer theoretischen Wissensbasis zu dieser Thematik, Methoden und Techniken untersucht, die sich dazu eignen natürliche Fließtexte und Textdokumente so aufzubereiten, dass bestimmte Kerninformationen wie Erscheinungsjahr, Autor, Titel und weitere Aspekte herausgelesen und in einen Wissensgraphen eingespeist werden können. Die Informationen sollen in diesem Graphen sinnvoll miteinander Verknüpft werden, um sie abfragbar und semantisch in einen Kontext bringen zu können. Zur Implementierung wird die Programmiersprache Python und zugehörige Bibliotheken sowie Pakete und Module genutzt, die im Verlauf der Arbeit vorgestellt werden. Das Ziel ist es eine Sammlung an Informationen zu erhalten, die von einem Nutzer abfragbar ist und es ermöglicht, durch einfache Suchanfragen relevante Ergebnisse zu bekommen. Somit wird die Suche nach passenden papern, Konferenzen, bestimmten Autoren oder auch die Recherche zu speziellen Themen vereinfacht und beschleunigt.

## 2

---

### Stand der Technik

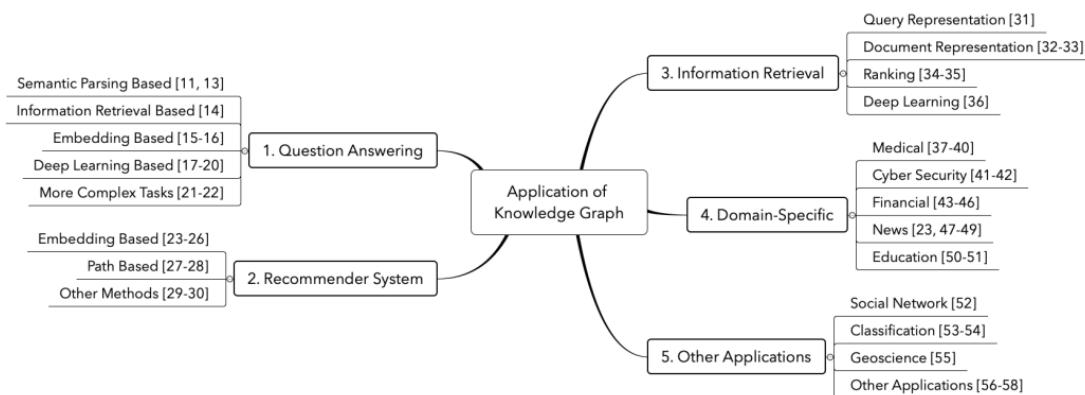
Die Nutzung von graphischen Datenbanken gewinnt immer mehr an Bedeutung und somit steigt auch die Vielfalt ihrer Ausprägungen. Viele Unternehmen stellen Graph Datenbanken bereit. Eine Übersicht über die verschiedenen Datenbanken und deren Art ist in Abbildung 2.1 zu sehen.



**Abbildung 2.1.** Existierende Lösungen für graphische Datenbanken mit Zuordnung der zugehörigen Organisationen [Yea]

Dabei stellt RDF (Resource Description Framework) und LPG (Labeled Property Graph) die für diese Arbeit interessanten Ansätze dar und werden im Folgenden näher betrachtet.

Im Umfeld der Wissensrepräsentation gibt es verschiedene Ansätze und noch mehr Anwendungsgebiete. Eine mögliche Klassifikation der Anwendungsgebiete kann in Abbildung 2.2 eingesehen werden. Ein großes Anwendungsgebiet stellt das semanti-



**Abbildung 2.2.** Mögliche Differenzierung der Anwendungsgebiete von Wissensgraphen [Zou20]

sche Web dar, bei welchem Google wohl eine wichtige Rolle spielt. Der Knowledge Graph von Google ist eine Technik, die Wissen nutzt um das Durchsuchen des Internets zu verbessern [Zou20]. 2013 wurde der Hummingbird Algorithmus<sup>1</sup> eingeführt und ermöglichte die Realisierung des Wissensgraphen von Google [Kop19]. Der Knowledge Graph von Google nutzt strukturierte, unstrukturierte<sup>2</sup>, sowie semi-strukturierte<sup>3</sup> Daten. Strukturierte Daten werden mittels Resource Description Framework (RDF) erfasst, welches in Anwendungsgebiete und das Semantische Web näher betrachtet wird. Die Daten entspringen oft Wissensbasen wie DBpedia<sup>4</sup>, YAGO<sup>5</sup>, Wikidata oder Freebase<sup>6</sup>[Zou20]. Aber nicht nur das semantische Web, sondern auch andere Anwendungsfälle wie das Durchsuchen von bereits aufgebauten Wissensgraphen spielt eine bedeutende Rolle. Wissensgraphen, die RDF basiert aufgebaut sind, werden üblicherweise mit der Sprache SPARQL abgefragt[PS08]. Doch können Graphen nicht nur mittels RDF erstellt werden, sondern auch als sogenannter Property Graph. Diese Art eines Wissensgraphen kann mittels mehreren Abfragesprachen analysiert werden. Die meist genutzten sind:

- Gremlin<sup>7</sup>

<sup>1</sup> <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8355941>

<sup>2</sup> <https://www.sem-deutschland.de/blog/informationen-entitaeten-websites/>

<sup>3</sup> <https://www.sem-deutschland.de/blog/google-informationen-wikipedia/>

<sup>4</sup> Bizer, Christian, et al. "Dbpedia-a crystallization point for the web of data." Journal of web semantics 7.3 (2009): 154-165.

<sup>5</sup> Suchanek, Fabian M., Gjergji Kasneci, and Gerhard Weikum. "Yago: A large ontology from wikipedia and wordnet." Journal of Web Semantics 6.3 (2008): 203-217.

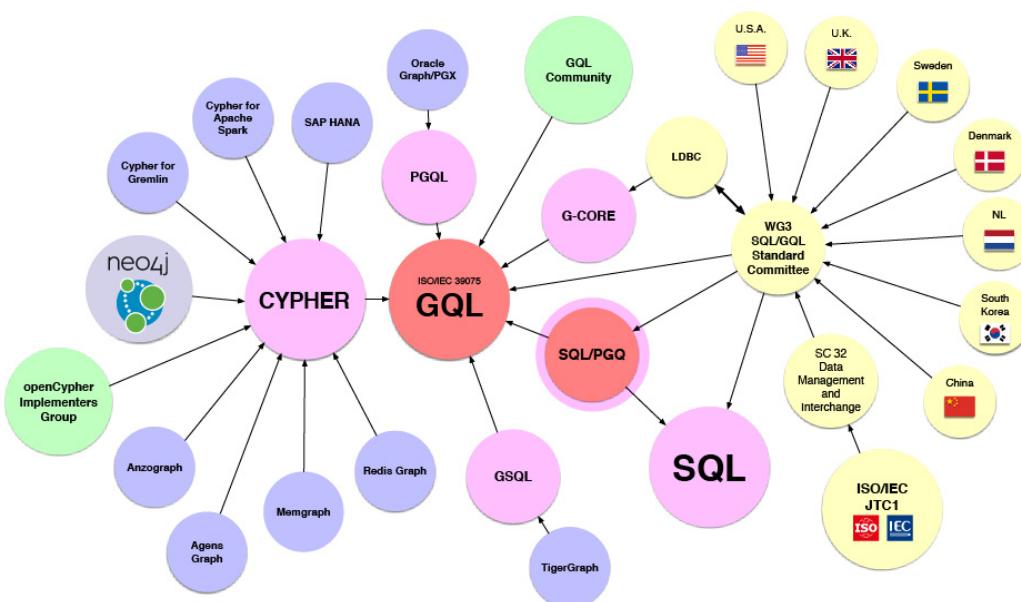
<sup>6</sup> Bollacker, Kurt, Robert Cook, and Patrick Tufts. "Freebase: A shared database of structured general human knowledge." AAAI. Vol. 7. 2007.

<sup>7</sup> <https://medium.com/geekculture/basics-about-graph-database-and-gremlin-5a26c3b51377>

- Cypher<sup>8</sup> von Neo4j<sup>9</sup>
- PGQL (Property Grapg Query Language von Oracle<sup>10</sup>
- GSQl (Graph SQL) von TigerGraph<sup>11</sup>

[Way21]

Dabei ist eine vorgeschlagene Standard Lösung für Property Graphen entstanden: GQL(Graph Query Language)<sup>12</sup>. Diese Sprache soll Cypher, GSQl, PGQL und weitere Ansätze vereinheitlichen [Gre19]. In Abbildung 2.3 ist eine Übersicht über die Einflüsse von GSQl zu sehen.



**Abbildung 2.3.** Einflüsse verschiedener Komponenten auf GQL [Gre19]

Wissensgraphen werden oft in Kombination mit der Verarbeitung von natürlicher Sprache genutzt. Diese Vorgehensweise wird natural processing language, kurz NLP<sup>13</sup>, genannt. Auch hier können verschiedene Ansätze verfolgt werden. Zum Beispiel zur Extraktion von Informationen aus Fließtexten, die dann zur Befüllung eines Wissensgraphen dienen können. Nach einer Vorverarbeitung der Texte durch beispielsweise der Entfernung von bestimmten Wörtern, die Aufteilung einzelner

<sup>8</sup> <https://neo4j.com/developer/cypher/>

<sup>9</sup> <https://neo4j.com/docs/getting-started/current/>

<sup>10</sup> <https://docs.oracle.com/en/database/oracle/property-graph/20.4/spgdg/property-graph-query-language-pgql.html>

<sup>11</sup> <https://www.tigergraph.com/gsql/>

<sup>12</sup> <https://gql.today>

<sup>13</sup> Khurana, D., Koli, A., Khatter, K. et al. Natural language processing: state of the art, current trends and challenges. *Multimed Tools Appl* (2022). <https://doi.org/10.1007/s11042-022-13428-4>

Wörter in Tokens und deren linguistische Klassifikation oder auch Veränderung, kann der entstehende Text dazu genutzt werden gewünschte Informationen zu Extrahieren oder auf eine bestimmte Art darzustellen. Mögliche Anwendungsgebiete sind die Erstellung von keywords, die die Texte inhaltlich zusammenfassen, die Klassifikation von Texten in bestimmte Kategorien oder das Finden bestimmter Informationen innerhalb eines Textes [KKKS22].

# 3

---

## Theoretische Grundlagen

In diesem Kapitel soll grundlegendes Wissen über Wissensgraphen, das semantische Web, die Verarbeitung von natürlicher Sprache und zugehöriger Algorithmen geschaffen werden. Zur Verdeutlichung der Theorie werden die Anwendungsfälle von Wissensgraphen anhand von Beispielen, wie den *Knowledge Graph* von Google und *Wkidata*, vorgestellt. Darüber hinaus wird die formale Sprache Ressource Description Framework (RDF), Linked Data und zugehörige Komponenten wie Protocol And RDF Query Language (SPARQL) von W3C vorgestellt. Zuletzt werden Plattformen zur Erstellung und Visualisierung von Wissensgraphen vorgestellt, die im Rahmen meiner Recherche zum Einsatz gekommen sind.

### 3.1 Ontologien, Wissensgraphen und Wissensbasen

Der Begriff Ontologie hat seinen Ursprung in der Philosophie. Dort bedeutet er *die Lehre vom Sein*. Aus Sicht der Informatik gibt es zwar keine einheitliche Definition, im Rahmen dieser Arbeit aber wird der Begriff als *formale Definition von Begriffen und deren Beziehungen als Grundlage für ein gemeinsames Verständnis* [BHL<sup>+</sup>14] definiert. In einem nächsten Schritt bei der Definition von Wissensgraphen sollten die Begrifflichkeiten geklärt werden, da verschiedene Termini im Zusammenhang mit dieser Thematik genutzt werden. Stichwörter wie Ontologie, Wissensbasis und Wissensgraph werden im Zusammenhang mit dem Wissensmanagement genutzt.  
**Doch wie unterscheiden sich diese Begriffe?**

In der Literatur gibt es keine einheitliche Abgrenzung. Für diese Arbeit gelten die Definitionen aus Tabelle 3.1. Aus dieser ist zu entnehmen, dass sich ein Wissensgraph aus einer Wissensdatenbank zusammensetzt und eine Ontologie ein Schema eines Wissensgraphen darstellt.

Der Unterschied eines Wissensgraphen und einer Ontologie lässt sich an einem Beispiel erklären: Wie in Abbildung 3.1 dargestellt, enthält eine Ontologie ein Schema zu einer bestimmten Domäne, während der Wissensgraph spezifische Instanzen aus dieser Domäne enthält.

Wissensbasis	Wissensgraph	Ontologie
Wissensdatenbank aus semi-strukturierten Daten in Form von Entitäten, deren Beziehungen zueinander und Attributen [RMdR <sup>+</sup> 20]	Wissensgraphen sind Wissensbasen, die in Form eines Graphen dargestellt werden [RMdR <sup>+</sup> 20]	generalisierte, domainbezogene Form eines Wissensgraphen [Sch20]

Tabelle 3.1. Definition der Begriffe Wissensbasis, Wissensgraph und Ontologie

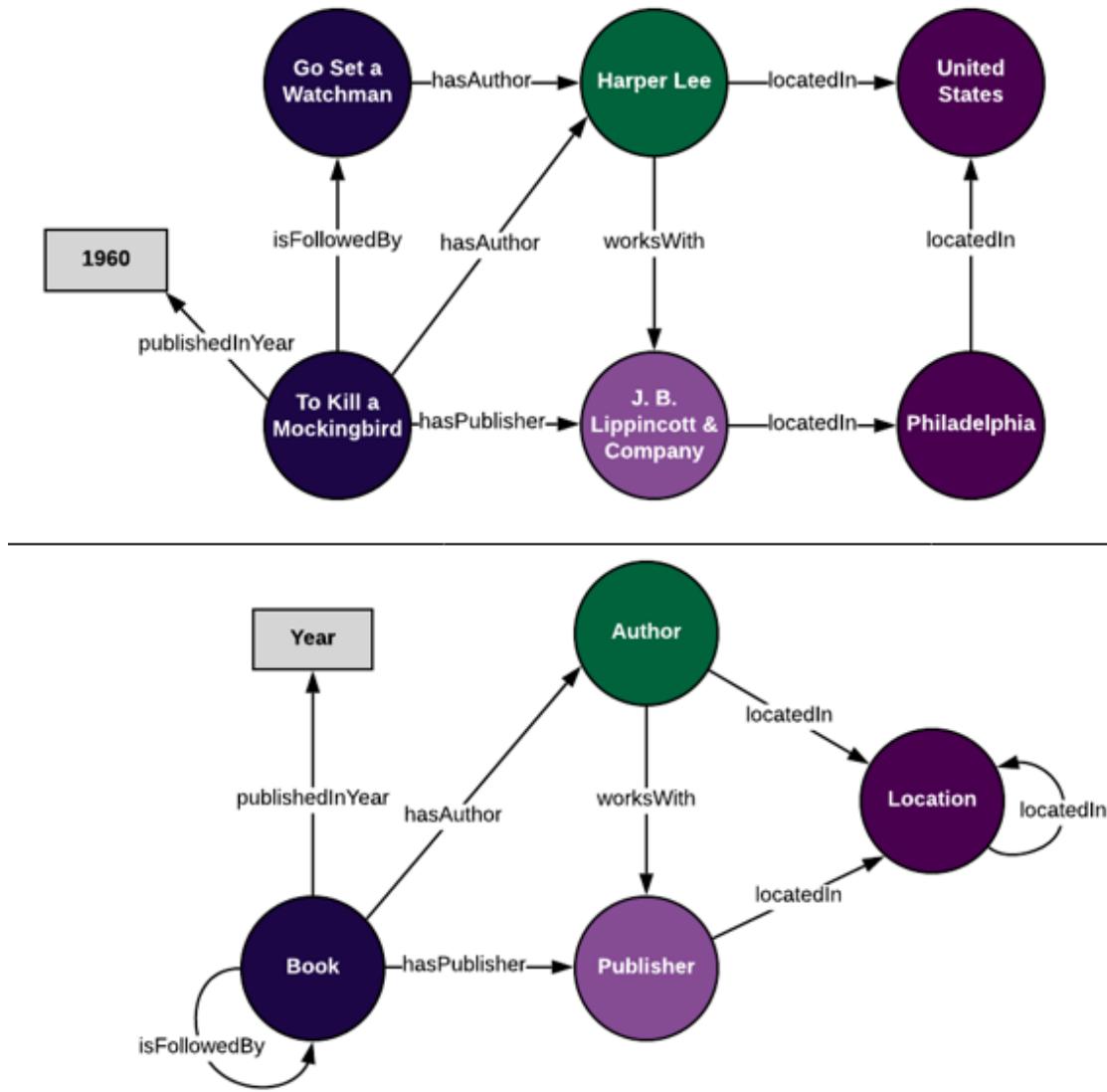


Abbildung 3.1. Vergleich eines Wissensgraphen (oben) mit einer Ontologie (unten) [Sch20]

### 3.1.1 Anwendungsgebiete und das Semantische Web

Bei dem Wissensgraphen werden bestimmte Ausprägungen zu den Entitäten in der Ontologie verwendet. Statt der Entität *Year*, wird die Instanz *1960* eingetragen. Weitere Begriffsdefinitionen innerhalb eines Wissensgraphen sind:

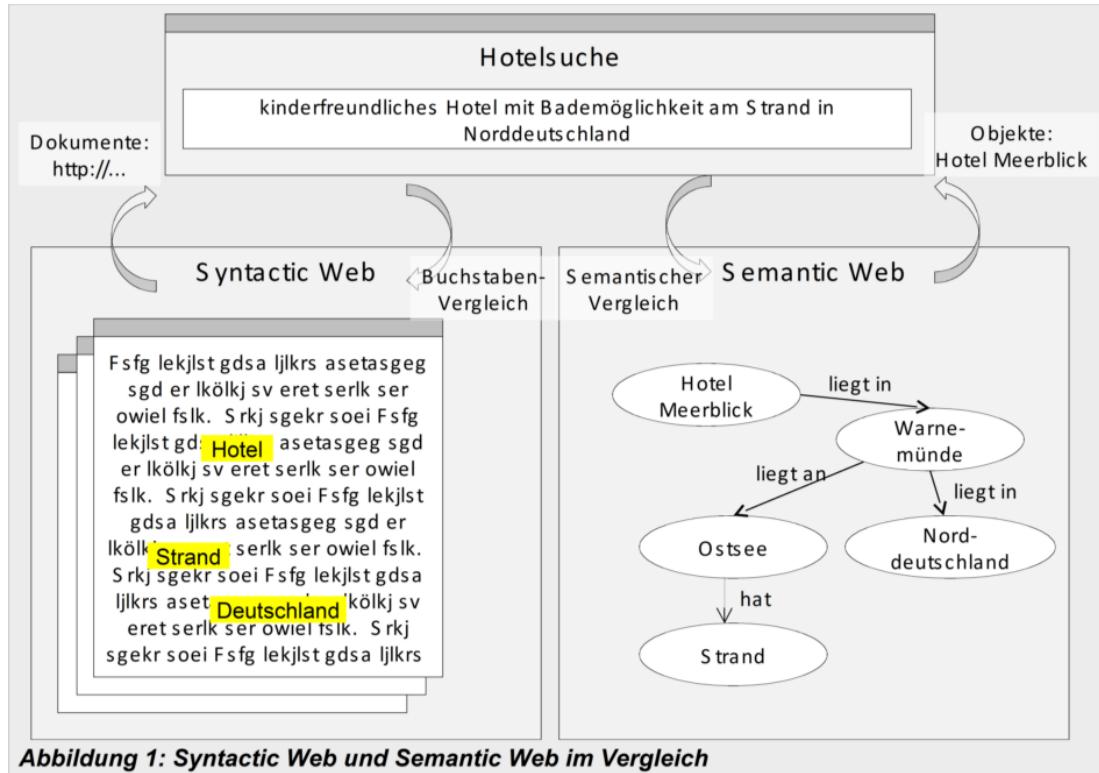
- Knoten: Stellen Entitäten, also Objekte aus unserer realen Welt dar
- Kanten: Darstellung der Beziehung zwischen Entitäten
- Attribute: Eigenschaften von Entitäten und/oder Beziehungen

Zunächst sollte geklärt werden wo Wissensgraphen eingesetzt werden. In diesem Zusammenhang ist es wichtig, den Begriff *semantisches Web* oder der auch im deutschen genutzte englische Begriff *Semantic Web* zu definieren. Denn der Hintergrund bei der Nutzung von Wissensgraphen oder Ontologien besteht darin, Daten eine Bedeutung, also Semantik zuordnen zu können, die von Mensch und Maschine ausgelesen werden kann. Die Funktionsweise des Semantic Web, welches von Tim Berners-Lee, dem Begründer des World Wide Web, ins Leben gerufen wurde [BLFD99], lässt sich an einem Beispiel erklären: Ein Kunde besucht ein Reisebüro um nach einem kinderfreundlichen Hotel mit Bademöglichkeit am Strand in Norddeutschland zu fragen. Der Mitarbeiter wird dem Kunden ein familienfreundliches Hotel anbieten, welches Zugang zu einem Nordsee- oder Ostseestrand hat. Diese Antwort kann er geben, da er weiß, dass Nord- und Ostsee mögliche Strandregionen in Norddeutschland sind. Vergleichen wir diese Mensch-zu-Mensch Anfrage mit einer Mensch-zu-Maschinen Anfrage, beispielsweise eine Suchanfrage bei Google, könnte ein mögliches Ergebnis ein kinderfreundliches Hotel in einem anderen Land mit Nähe zum Strand sein, da bei diesem Ergebnis eine aus Deutschland stammende Kundenbewertung verfasst wurde. Dies geschieht, weil die Google Suche Übereinstimmungen von einzelnen Wörtern überprüft und der Mitarbeiter im Reisebüro den Satz als Ganzes versteht [BHL<sup>+14</sup>]. Zumindest war dies der Fall, bevor Google seinen eigenen Wissensgraphen namens *Knowledge Graph* nutzte. Dieser macht es seit 2013 möglich, den Sinn hinter Suchanfragen zu verstehen. Die Unterscheidung des syntaktischen (Übereinstimmungen einzelner Wörter) und des semantischen Webs nach Busse et al.[BHL<sup>+14</sup>] ist in Abbildung 3.2 dargestellt. Hier ist zu erkennen, dass die Wörter aus der Suchanfrage durch das syntaktische Web einfach abgeglichen werden. Anders ist es bei der semantischen Komponente: Durch die Erkennung von grammatischen Regeln innerhalb der Anfrage und der Verknüpfung dieser Elemente durch Beziehungen ist es möglich, den Sinn hinter der Anfrage zu verstehen und bessere Ergebnisse zu liefern.

Nach Busse et al.[BHL<sup>+14</sup>] lässt sich der Vorgang in einem semantischen Netz in vier Schritte aufteilen:

1. Linguistische Analyse der Anfrage, beispielsweise durch NLP Algorithmen, welche die Bausteine der Anfrage grammatischen Regeln zuordnen können
2. Synonymersetzung, wie zum Beispiel das Ersetzen von *familienfreundlichen* mit *kinderfreundlich*

3. Schlussfolgern durch die Beziehungen innerhalb des Graphen. Beispielsweise Norddeutschland ist in Deutschland und grenzt an das Meer
4. Abgleich welche Ergebnisse passen zu der Anfrage

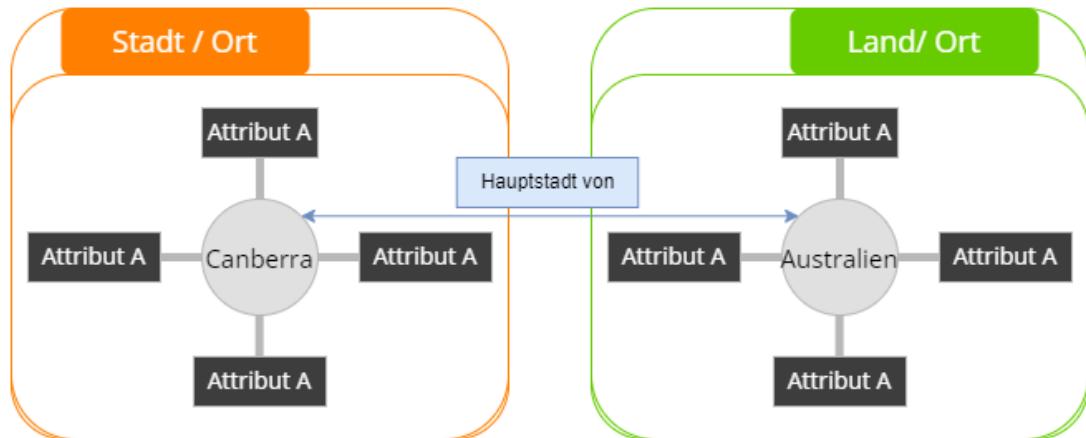


**Abbildung 3.2.** Gegenüberstellung des syntaktischen und semantischen Webs am Beispiel einer Suchanfrage [BHL<sup>+</sup>14]

Die Grundlage des Knowledge Graphs spiegelt sich in der vorherigen Definition wieder. Es gibt einen Entitäten Katalog, der eine Menge an Entitäten bzw. Objekten der realen Welt in ein Knowledge Repository (Knowledge Vault) speichert und mit Beschreibungen ergänzt. Diese Daten dienen als Grundlage für die Wissensbasis und den Knowledge Graph, in dem Attribute und Beziehungen ergänzt werden. Ein weiteres Beispiel verdeutlicht die Funktionsweise: Wird die Hauptstadt von Australien gesucht, bekommt der Nutzer das Ergebnis "Canberra". In Abbildung 3.3 ist der Graph zu diesem Anwendungsfall dargestellt [Kop19].

Über die Beziehung *Hauptstadt von* kann Google die Anfrage beantworten, ohne dass das Wort *Canberra* explizit in der Anfrage genutzt wird. Dies ist möglich durch das Herstellen von Zusammenhängen mit grammatischen Regeln. Die einzelnen Wörter einer Suchanfrage werden zum Beispiel als Nomen, Adjektive, Verben oder Adverbien kategorisiert. Außerdem werden Synonyme gebildet und Schlussfolgerungen gezogen [BHL<sup>+</sup>14]. Im Beispiel der Google Suchanfrage ist *Canberra* das Subjekt, *Australien* das Objekt und *Hauptstadt* oder *ist die Hauptstadt* ist das

Prädikat. Diese Zuordnungen sind durch das Natural Language Processing (NLP) möglich. Durch dieses Vorgehen werden Nomen bzw. Objekte und Subjekte meist als Entitäten und Beziehungen als Prädikat oder Verben interpretiert [Kop19]. Doch nicht nur Google trägt bei der Entwicklung von Wissensgraphen oder des



**Abbildung 3.3.** Beispielhafte Darstellung de Google Knowledge Graphs bei einer Suchanfrage "Hauptstadt Australien" [Kop19]

Semantic Web bei. Auch ein Projekt der bekannten Wikimedia-Stiftung, zu der auch Wikipedia gehört, entwickelt ihre eigene Wissensbasis. Wikidata<sup>1</sup> verfolgt verschiedene Ziele:

- Bereitstellung einer frei zugänglichen und kostenlosen Wissensbasis
- Kann von jedem Menschen und von Maschinen verändert oder erweitert werden
- Realisierung in möglichst vielen Sprachen
- Zugang zu einem zentralen Speicherort für strukturierte Daten zur Unterstützung anderer Wikimedia Projekte

Ein Wissensgraph besteht jedoch nicht nur aus Knoten, Kanten und Beziehungen, sondern enthält durch die dahinter liegende Wissensbasis auch Aussagen. Diese Aussagen werden formal, also maschinenlesbar, definiert und geben zum Beispiel Instanzen wieder. Ein Beispiel für eine formale Definition einer Aussage wäre: `:HotelMeerblick rdf:type :Hotel` heißt so viel wie: Hotel Meerblick ist eine Instanz, also ein Beispiel für die Klasse Hotel [BHL<sup>+</sup>14]. Das in Beispiel Abbildung 3.1.1 beschriebene Format ist eine standardisierte und formale Vorgehensweise, um Objekten zu beschreiben und wurde vom World-Wide Web Consortium (W3C)<sup>2</sup> entwickelt und nennt sich *RDF* [dic]. Der RDF Standard basiert auf Extensible Markup Language (XML)<sup>3</sup> und setzt sich aus einem Triple (dreiteilig)

<sup>1</sup> [https://www.wikidata.org/wiki/Wikidata:Main\\_Page](https://www.wikidata.org/wiki/Wikidata:Main_Page)

<sup>2</sup> <https://www.w3.org>

<sup>3</sup> <https://www.w3.org/TR/REC-xml/>

zusammen, die aus einer Internationalized Resource Identifier (IRI)<sup>4</sup>, aus Literalen<sup>5</sup> oder Blank Nodes<sup>6</sup> bestehen [RDF14]:

1. Subjekt: Die zu beschreibende Ressource [AI03].
2. Prädikat: Die Beschreibung der Ressource durch Attribute und Beziehungen [AI03].
3. Objekt: Der Wert der beschriebenen Eigenschaft der Ressource [AI03].

In Abbildung 3.4 ist eine Beispiel RDF Beschreibung von der Organisation **Nike**, **Dahliastraat 24, 2160 Wommelgem** gegeben. Weitere Syntax Möglichkeiten zum Aufbau von RDF Tripeln werden in dieser Arbeit nicht beleuchtet.

```
<rdf:RDF
  xmlns:rov="http://www.w3.org/TR/vocab-regorg/"
  xmlns:org="http://www.w3.org/TR/vocab-org/"
  xmlns:locn="http://www.w3.org/ns/locn#" >

<rov:RegisteredOrganization rdf:about="http://example.com/org/2172798119">
  <rov:legalName> "Nike" </rov:legalName>
  <org:hasRegisteredSite rdf:resource="http://example.com/site/1234"/>
</rov:RegisteredOrganization>

<locn:Address rdf:about="http://example.com/site/1234">
  <locn:fullAddress> Dahliastraat 24, 2160 Wommelgem </locn:fullAddress>
</locn:Address>

</rdf:RDF>
```

**Abbildung 3.4.** RDF Beschreibung des Beispiels **Nike, Dahliastraat 24, 2160 Wommelgem**. Rot: Subjekt, Blau: Prädikat, Grün: Objekt [KNLG15].

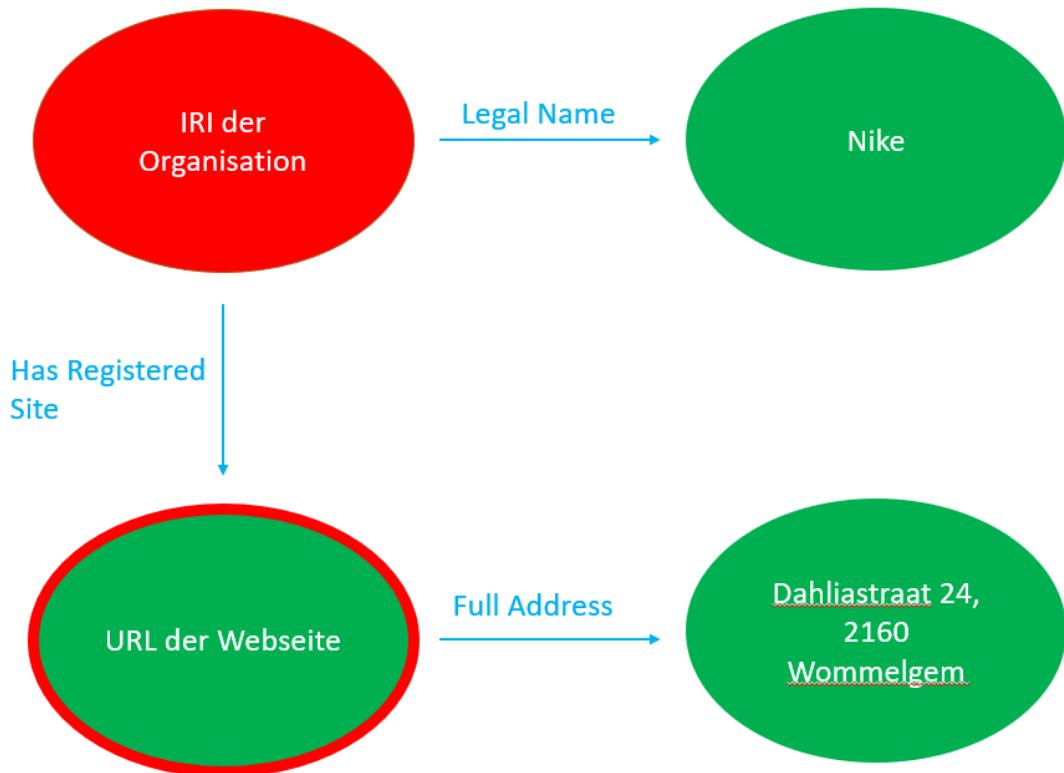
Die rot hinterlegten Felder beschreiben dabei die Subjekte, die Blauen die Prädikate und die Grünen die Objekte. In Abbildung 3.5 ist das Beispiel als Graph interpretiert. Auf die Syntax wird an dieser Stelle nicht näher eingegangen, da dieses Vorgehen der semantischen Beschreibung von Ressourcen in dieser Arbeit nicht verwendet wird. Die Syntax und eine Sammlung an Vokabular kann jedoch in der [W3C Dokumentation](#) nachgelesen werden.

Ein weiteres Kernelement des Semantic Web ist **Linked Data**, bzw. bei freien Inhalten **Linked Open Data (LOD)**. Dies sind Daten, welche über die im RDF Standard verwendete URI identifizierbar sind und über weitere URIs auf andere Daten verweisen - also miteinander verlinkt sind. Diese Verlinkungen sind auch von Maschinen lesbar und bilden ein weltweites Netz, welches Linked Open Data Cloud

<sup>4</sup> internationalisierte Form der URI mit erweiterten erlaubten Zeichen [RDF14]

<sup>5</sup> Definiert mögliche Werte und den Datentyp [RDF14]

<sup>6</sup> Repräsentiert einen leeren Knoten wenn die IRI oder das Literal nicht bekannt sind [RDF14]



**Abbildung 3.5.** Graph zur Beispiel RDF Beschreibung in Abbildung 3.4. Rot: Subjekt, Blau: Prädikat, Grün: Objekt. Die Webseite der Organisation kann hier als Subjekt und Objekt gelesen.

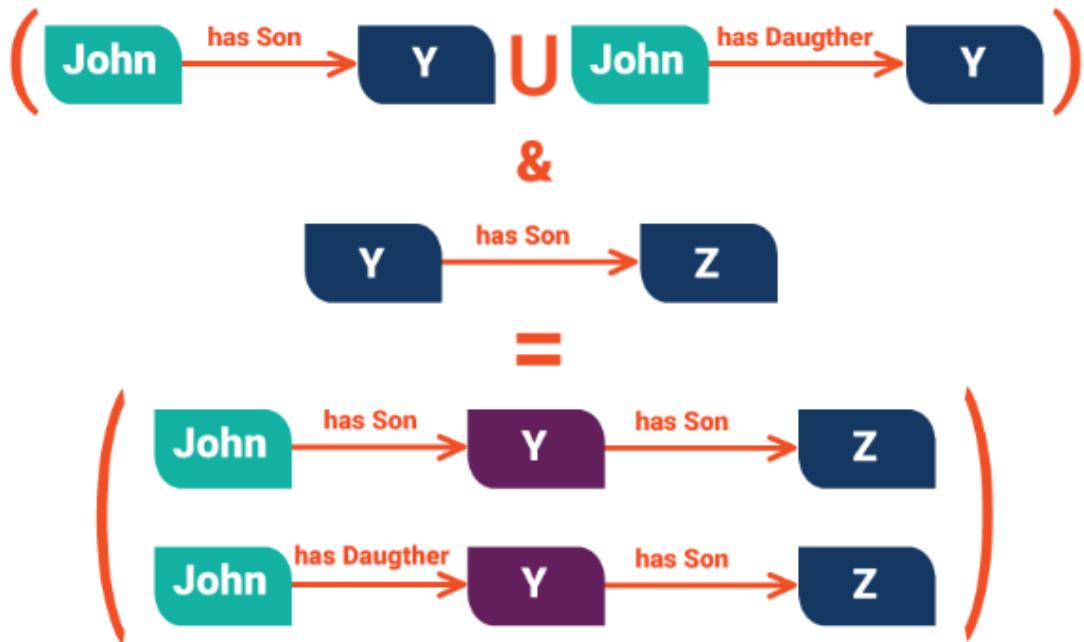
oder Giant Global Graph genannt wird. Beispiele für Linked Data sind bereits im Kapitel Stand der Technik genannt worden. Dazu zählt DBPedia, wodurch unter Anderem Wikipedia Daten in RDF Form verfügbar gemacht werden [Onta] [RDF].

Die Standardmethodik diese RDF Triple abzufragen nennt sich **SPARQL**<sup>7</sup>. Diese Sprache ähnelt SQL sehr in seiner Funktionalität. Es ist möglich, ähnlich wie die bekannten Form (SELECT FROM WHERE) von SQL, nach bestimmten Informationen zu suchen, oder auch die Struktur der zugrundeliegenden Datenbasis zu modifizieren. Besonders ist, dass bei dieser Methode Triple abgefragt werden. Triple in der Form Subjekt, Prädikat und Objekt [Ontb]. Darüber hinaus existieren verschiedene Arten von Abfragen:

- ASK
- SELECT
- CONSTRUCT
- DESCRIBE

<sup>7</sup> <https://www.w3.org/TR/rdf-sparql-query/>

Wie SPARQL funktioniert, wird mit Hilfe der Abbildung 3.6 deutlich.



**Abbildung 3.6.** Beispiel SPARQL Funktion in vereinfachter Darstellung [Ontb].

Angenommen, eine Graphdatenbasis enthält das Subjekt John und zugehörige Prädikate has Son (hat Sohn) und has daughter (hat Tochter). Mittels einer SPARQL Abfrage sollen alle Enkel von John gefunden werden, ohne zu wissen, wer die Kinder von John sind. In einem ersten Schritt werden alle Söhne und Töchter von John gesucht und die Ergebnisse vereinigt (union). Dann werden zu allen gefundenen Kindern von John (Y in Abbildung 3.6) wiederum alle Söhne gesucht (Gefundene Enkel sind Z in Abbildung 3.6) [Ontb].

Wie bereits erwähnt existiert noch eine modifizierende Funktionalität, namens UPDATE, die das Einfügen, Löschen und Manipulieren von Daten innerhalb von RDF Graphen ermöglicht.

### 3.1.2 Property Graphs und Cypher

Neben der bekannten Verwendung von RDF, gibt es auch die Möglichkeit Wissensgraphen in Form von Property Graphen oder Labeled Property Graphen zu erstellen. Bei diesem Stil der Implementierung werden Informationen ebenfalls in Form von Knoten, Kanten dargestellt. Hinzu kommen Eigenschaften (properties). Knoten stellen hier ebenfalls Entitäten dar, können aber mit Label versehen werden, die die Knoten oder Entitäten näher beschreiben. Dies könnte beispielsweise

<sup>8</sup> <https://www.w3.org/TR/rdf-sparql-query/>

*Person* sein, wenn der Knoten den Namen einer Person darstellt. Die Eigenschaft wird als key-value Paar dargestellt. Im selbigen Beispiel wäre dies zum Label *Person* ein *name* und die zugehörige Ausprägung, zum Beispiel *Max*. Die Relationen zwischen Knoten geben eine gerichtete und benannte Kante zwischen zwei Knoten an. Diese können ebenfalls Eigenschaften besitzen [Pro]. Eine Möglichkeit diese Graphen zu erstellen, beziehungsweise zu speichern ist Neo4j. Diese open-source Graph Datenbank stellt auch eine Abfragesprache *Cypher* bereit. Darüber hinaus kann die Nutzung von Neo4j in andere Programmiersprachen wie zum Beispiel Java, JavaScript und Python einfach übertragen werden [Neob]. Cypher ist eine SQL ähnliche Abfragesprache, die für Graphen optimiert ist. Die Sprache wird anhand des folgenden Beispiels und der Abbildung 3.1.2 erläutert.

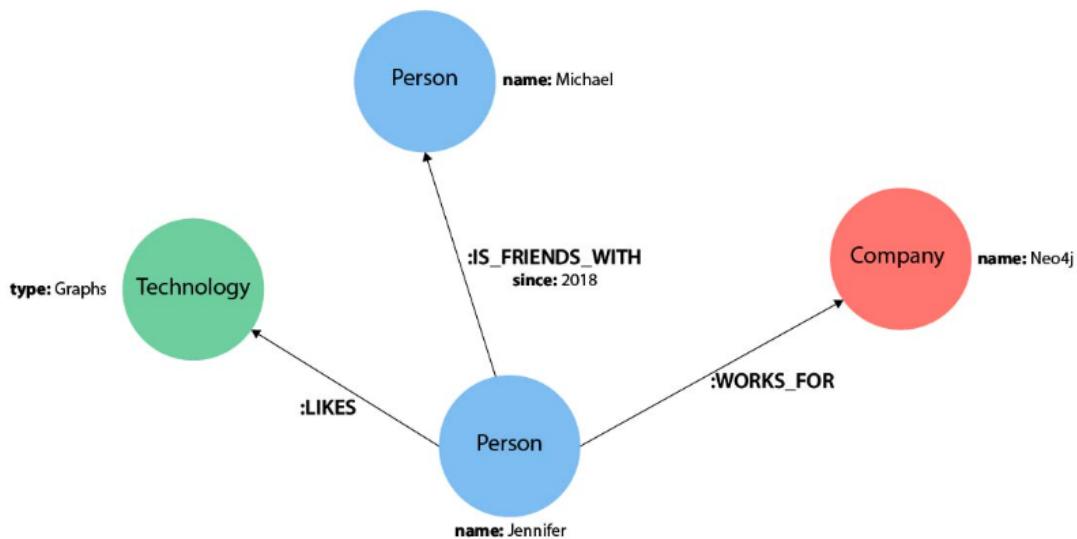


Abbildung 3.7. Neo4j Graph [Neoa]

Die Knoten in dem Beispiel sind *Jennifer*, *Michael*, *Neo4j* und *Graphs*. Die Abbildung von Knoten in Cypher geschieht durch Klammerung des jeweiligen Knotens: `(Knoten)`. Darüber hinaus kann ein Knoten eine Label aufweisen, die diesen klassifiziert. Beispielsweise *Person*. Ein Label kann noch einer Variable zugeordnet werden, wie `p` für *Person*. Darüber hinaus müssen auch Relationen in Cypher abgebildet werden. Diese werden durch `-->` oder `<--` (gerichtet) oder `--` (ungerichtet) repräsentiert. Typen von Beziehungen werden durch `:Relationentyp` dargestellt. Diese Typen sind ähnlich zu Label bei Knoten und zeigen an, wie Knoten miteinander in Beziehung stehen. Ebenso können diese Typen mit einer Variable versehen werden [Neoa].

Nach dieser Beschreibung wie Wissen, bzw. Semantik ins Netz integriert oder auch von Maschinen lesbar gemacht wurde, wird im Folgenden auf eine Erstellung von

Wissensgraphen in einem speziellen Anwendungsfall eingegangen. Es ist ebenso möglich, auch mittels der Programmiersprache Python Wissensgraphen aufzubauen. Im Rahmen dieser Arbeit wird ein Graph implementiert, der essentielle Informationen aus wissenschaftlichen Publikationen miteinander verknüpft. Das Format des Graphen bleibt, wie in Abbildung 3.5 dargestellt, in RDF, doch die Herkunft der Daten und deren Art der Verknüpfung ändert sich.

## 3.2 Natural Language Processing

Um einen Wissensgraphen auf der Grundlage von natürlicher Sprache aufzubauen, werden NLP Algorithmen benötigt um die benötigten Daten erst einmal zu extrahieren und maschinenlesbar zu machen. Denn einen Graphen manuell aufzubauen, ist mit stetig wachsenden Datenmengen nicht tragbar. NLP Algorithmen ermöglichen es Maschinen, natürliche Sprache, beispielsweise Fließtext aus einem PDF Dokument, zu lesen und zu klassifizieren. Dieser Vorgang wird auch als Text Mining bezeichnet. Es existieren bereits verschiedene Techniken um solche Vorhaben in die Tat umzusetzen. In einem ersten Schritt wird Basiswissen zu diesem Gebiet der Arbeit vermittelt. Die Definition von NLP nach Liddy (2001) lautet: *Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications* [Lid01]. Auf deutsch heißt dies etwa: Natural Language Processing ist ein theoretischer Ansatz von Computertechniken zur Analyse und Repräsentation von Texten in natürlicher Sprache auf einer oder mehreren Ebenen der linguistischen Analyse mit dem Ziel, eine menschenähnliche Verarbeitung dieser Texte für gewissen Aufgaben zu erreichen.

Nach dieser sehr allgemeinen Definition sollte der Begriff NLP, bzw. auch allgemeine Kenntnisse zur Linguistik erläutert werden. NLP kann in zwei Komponenten aufgeteilt werden. Zum Einen *Natural Language Understanding (NLU)* und zum Anderen *Natural Language Generation (NLG)*<sup>9</sup>. Bei NLG handelt es sich um die Fähigkeit von Maschinen, natürliche, also für den Menschen verständliche, Texte zu verfassen [KKKS22] und wird in dieser Arbeit aufgrund mangelnder Relevanz nicht näher betrachtet. Die NLU Komponente hingegen befasst sich mit dem Verstehen von natürlicher Sprache, also auch damit Semantik aus einem Text herauszulesen oder Texte zu analysieren und beispielsweise in Entitäten umzuwandeln [KKKS22]. Die in dieser Arbeit relevanten Teilgebiete von NLU, bzw. allgemein von der Linguistik, sind Morphologie<sup>10</sup>, Pragmatik<sup>11</sup> und damit auch die Syntax bzw. Semantik von Sätzen und Texten. Ein Überblick über die Teilgebiete sind in Tabelle 3.2 abgebildet. Bei der Morphologie ist es wichtig den Begriff Morphem einzuführen. Hierbei handelt es sich um die Kleinste Einheit eines Wortes. Ein Beispiel wäre: *Ich habe Bauchweh*. In diesem Satz befinden sich fünf Morpheme:

<sup>9</sup> <http://www.lacsc.org/papers/PaperA6.pdf>

<sup>10</sup> <https://linguistik.online/2021/09/30/was-ist-morphologie/>

<sup>11</sup> <https://linguistik.online/2021/10/03/was-ist-pragmatik/>

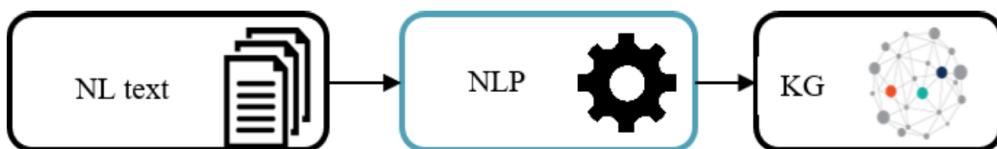
Morphologie	Aufbau und Struktur von Wörtern
Pragmatik	Situationsbedingte Bedeutung eines Satzes
Syntax	Grammatikalische Struktur von Sätzen
Semantik	Die Bedeutung eines Satzes

**Tabelle 3.2.** Definition der Begriffe Morphologie, Pragmatik, Syntax und Semantik im Zusammenhang mit NLU

Ich/ hab/e Bauch/weh. Durch dieses Verfahren können Wörter eine gänzlich andere Bedeutung haben. Das Nomen *Sitz* wird durch das Morphem *en* zu dem Verb *sitzen*, welches bei der Verarbeitung von natürlicher Sprache einen erheblichen Unterschied macht [KKKS22].

Die Syntax eines Satzes beeinflusst auch die Bedeutung. Der Satz *Max besiegt Anna im Tennis* und *Anna besiegt Max im Tennis* hat sich nur in seiner Reihenfolge geändert, dennoch ist die Semantik eine andere. Um den Sinn eines Satzes mittels Maschinen auszulesen, werden logische Strukturen, wie vorher erläuterte linguistische Konzepte, genutzt. So kann beispielsweise die Thematik eines Satzes anhand verwandter Wörter herausgelesen werden, ohne dass die explizite Bezeichnung des Themas wörtlich innerhalb des Satzes genannt wird. Ebenso ist es möglich die Bedeutung von Wörtern anhand ihrer lexikalischen Bedeutung oder durch andere Wörter innerhalb eines Satzes, also durch den Kontext, zu identifizieren [KKKS22].

Im Zusammenhang mit der Datenextraktion aus wissenschaftlichen Papieren wird sich in dieser Arbeit mit NLP bezüglich der Informationsextraktion beschäftigt. Das Themenfeld der Informationsextraktion (IE) aus Fließtexten hat die Aufgabe unstrukturierten Text so aufzubereiten, dass dieser später für die Repräsentation von Wissen genutzt werden kann [Sin18]. Dabei werden, wie in Abbildung 3.2 dargestellt, verschiedene Prozesse durchlaufen, die im folgenden erläutert werden.



**Abbildung 3.8.** Allgemeiner Prozess vom natürlichen Text zum Wissensgraph [AMOOV20]

### 3.2.1 Pre-processing

In einem ersten Schritt sollte der unstrukturierte vorliegende Text aufbereitet werden, sodass er von Maschinen gelesen werden kann. Die Vorverarbeitung von Texten ist ein wichtiger Schritt, um diesen für Maschinen lesbar zu machen und gehört

zu NLP. In dieser Arbeit wird die Verarbeitung der englischen Sprache fokussiert. Mögliche Schritte zur Aufbereitung sind:

- Tokenization
- Stemming und Lemmatizing
- Stopwords
- Part of speech tags
- Chunking
- Parsing

**Tokenization** oder Tokenisierung ist ein Vorgang um Texte aus Dokumenten in einzelne, kleinere Abschnitte (Tokens) aufzuteilen. Die Aufteilung kann auf verschiedene Weisen erfolgen, dabei ist die Aufteilung in einzelne Sätze (Sentence Segmentation genannt), Wörter oder sogar auf Buchstabenebene möglich. Auch sind individuelle Aufteilungen durch z.B. reguläre Ausdrücke realisierbar. Einzelne Wörter können auch, wie in Tabelle 3.2 erklärt, auf morphologischer Ebene aufgeteilt werden. Auf die englische Sprache beschränkt, sind Wörter meist durch Leerzeichen und Sätze durch Satzzeichen getrennt, anhand dessen Algorithmen erkennen können wann ein Wort oder ein Satz endet. Doch hier können mehrere Probleme auftreten: Satzzeichen, wie etwa Punkte können auch für abgekürzte Wörter stehen (Beispiel: Bzw.). Dies gilt auch für weitere Satzzeichen [Pal00]. Das Unterteilen eines Textes in Tokens ist essenziell für die Verarbeitung von natürlicher Sprache durch Maschinen [Pai22]. Es existieren verschiedene Ansätze Texte in Tokens zu unterteilen. Alle folgenden Arbeitschritte des Pre-Processings bauen auf der Segmentierung des Textes in Tokens auf.

Das **Stemming** bezeichnet die Bildung des Wortstammes. Die Wörter gepflanzt, pflanzen, umpflanzen werden alle zu ihrem Wortstamm *pflanz* zurückgeführt, indem Suffixe und Präfixe entsprechend umgeformt werden. Somit wird erreicht, dass die zu Beginn unterscheidbaren Wörter ununterscheidbar werden und in weiteren Analysen gleich behandelt werden können [Kip18]. Ein bekannter Stemmer aus dem Python NLTK Package ist der **Porter Stemmer**. Der Porter-Stemmer-Algorithmus<sup>12</sup> interpretiert Zeichenfolgen als eine Vokal-Konsonant-Sequenz. Jedes Wort, kann auf in der Form

$$[K](VK)^m[v] \quad (3.1)$$

darstellen. Das heißt, dass jedes Wort mit einem, keinem oder mehreren Konsonanten beginnt, gefolgt von (einem) Vokal-Konsonant-Paar(en) und mit einem, keinem oder mehreren Konsonanten endet. Die Anzahl von Vokal-Konsonant-Paaren ist das Maß eines jeden Wortes und wird mit m abgekürzt. Das Ersetzen oder Entfernen von Wortendungen wird durch Regeln realisiert. Diese Regeln befinden sich in verschiedenen Gruppen und pro Gruppe darf nur eine Regel angewandt werden.

<sup>12</sup> Porter, Martin F. "An algorithm for suffix stripping." Program (1980).

Eine Regel wird angewandt, wenn eine Bedingung erfüllt wird. Als Beispiel wird nun das englische Wort *Replacement* genutzt. eine Regel lautet:

$$(m > 1) EMENT - > \quad (3.2)$$

Dies bedeutet, dass wenn m größer 1 ist und das Wort mit EMENT endet, soll dies mit nichts ersetzt werden, also wegfallen. Replace besitzt den Wert m=2 und endet auf EMENT. Somit wird das Wort *REPLAC* gebildet [Mal17]. Andere Vorgehensweisen zum Stemming sind der Lancaster Stemming Algorithmus<sup>13</sup>, der Snowball Stemmer (auch Porter2 Stemmer genannt, da dieser eine Verbesserung des Porter Stemmers darstellt) oder Stemmer, die auf regulären Ausdrücken basieren.

Sehr ähnlich ist das **Lemmatizing**, bei dem aber nicht der Wortstamm, sondern die Grundform eines Wortes gebildet wird [San20]. Die Grundform oder das Lemma eines Wortes ist die Form, die im Wörterbuch eingetragen ist [lem] und somit auch eine Bedeutung hat. Im obigen Beispiel wäre dies das Verb *pflanzen*. Eine Möglichkeit zur Kreation der Lemma von Wörtern ist das Nachschlagen der Grundformen in einer Datenbank. So funktioniert beispielsweise der WordNet Lemmatizer aus dem bekannten NLTK Package [Jab18].

Das Entfernen von **Stopwords** ist auch ein Schritt des Pre-Processings. Durch dieses Verfahren können Wörter, die wenig zur Analyse und zur Semantik eines Textes beitragen, eliminiert werden. Dies sind beispielsweise Artikel (der, die, das) oder Bindewörter (oder, also, und) [San20]. Die Wörter werden vorher Definiert und dann aus den Texten entfernt.

Ein weiterer wichtiger Bestandteil der Vorverarbeitung von Texten ist das Part Of Speech Tagging (POS-Tagging). Bei diesem Schritt werden Wörtern ihre grammatische Bedeutung zugeordnet, also die Klassifikation in Subjekt, Prädikat und Objekt und weitere Wortarten und Satzglieder. Diese Aufteilung wurde bereits in Unterabschnitt 3.1.1 erläutert. Es existieren verschiedene Ansätze zur Durchführung von POS-Tagging. Beispielsweise werden Hidden-Markov-Modelle<sup>14</sup> verwendet, das Eric Brills<sup>15</sup> Verfahren oder Entscheidungsbäume<sup>16</sup> genutzt. Auch stochastische Verfahren finden ihre Anwendung<sup>17</sup> [Tut]. Durch die Einteilung in Wortarten mittels POS Tagging, können anschließend gefilterte Informationen, die auf Regular Expressions basieren können, durch **Chunking** herausgefiltert werden. Dabei können zum Beispiel nur Verben aus einem Teil des Textes extrahiert

<sup>13</sup> Paice, C. D. (1990, November). Another stemmer. In ACM Sigir Forum (Vol. 24, No. 3, pp. 56-61). New York, NY, USA: ACM.

<sup>14</sup> Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. Computer speech & language, 6(3), 225-242.

<sup>15</sup> Eric Brill: A simple rule-based part-of-speech tagger. In Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP-92). S. 152-155, 1992.

<sup>16</sup> Helmut Schmid: Probabilistic part-of-speech tagging using decision trees. In Proceedings of the International Conference on New Methods in Language Processing 1994.

<sup>17</sup> Nugues, P. M. (2006). Part-of-Speech Tagging Using Stochastic Techniques. An Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German, 163-184.

werden. Aber auch inhaltlich können Textbausteine extrahiert werden. Es ist zum Beispiel möglich, Früchte nach bestimmten Eigenschaften zu gruppieren [V21].

Das **Parsing** beschäftigt sich mit der Analyse der grammatischen Struktur von Sätzen. Dabei basiert die Analyse entweder auf der Abhängigkeit von Wörtern in einem Satz (Dependency Parser) oder auf der Verkleinerung von Sätzen in kleinere Bestandteile (Constituency Parsing). Der Dependency Parser beschreibt die Beziehung zwischen zwei Wörtern in einem Satz. Dabei fungiert ein Wort als *head* und eines als *dependent*. Die Abhängigkeit der beiden Wörter wird durch einen Tag beschrieben, der für eine bestimmte Art von Abhängigkeit steht. Ein Beispiel dazu wäre der Satz *regnerisches Wetter*. Hier ist *Wetter* der head und *regnerisch* die dependency. Die Kategorisierung erfolgt dadurch, dass das Wort *regnerisch* die Bedeutung von *Wetter* beeinflusst und verändern kann. Somit entsteht eine Abhängigkeit von *Wetter* vom Wort *regnerisch*. Daraus resultiert, dass das Wort *regnerisch* eine Abhängigkeit (dependency) ist und das Wort *Wetter* der head ist. Der Tag ist in diesem Fall *amod*, was für einen Modifikator als Adjektiv (Der Modifikator als beschreibendes Wort) steht. Hier kann beispielsweise der Stanford Parser<sup>18</sup> genutzt werden. Das Constituency Parsing<sup>19</sup> teilt Sätze in mehrere kleinere Phrasen auf und vergibt auch dort Tags zur Kategorisierung. Das können beispielsweise Teile für Verben oder Teile für Nomen sein [Sha20].

Probleme und Lösungen im Zusammenhang mit der Vorverarbeitung der in dieser Arbeit vorliegenden Texte werden in Kapitel 5 näher beleuchtet und anhand der Problemstellung angewandt.

### 3.2.2 Datenextraktion

Nachdem Text für Maschinen durch die Vorverarbeitung lesbar gemacht wurde, ist der nächste Prozessschritt die Extraktion von gewünschten oder wichtigen Informationen. Im Bereich der IE ist dieser Aufgabenteil im Zusammenhang mit NLP ein essenzieller Vorgang. Ein Ansatz ist die Extraktion von keywords oder keyphrases: Dabei werden solche Wörter oder n-Gramme aus den Texten extrahiert, die den Inhalt des Textes am besten widerspiegeln [Gro20b].

In Bezug zu der vorliegenden Problemstellung sollen solche keywords aus Dokumenten ausgelesen werden, die später die behandelten Themen in den wissenschaftlichen Papieren repräsentieren. Zur Generierung von keywords existieren bereits eine Menge an Ansätzen und Algorithmen, die in Python implementiert sind. Eine Auswahl ist in Tabelle 3.3 aufgezeigt.

In dieser Arbeit wird das Modell KeyBert verwendet, weswegen auf näheres Eingehen der anderen Ansätze verzichtet wird. KeyBert entstammt aus Bidirectional Encoder Representations from Transformers, kurz: BERT<sup>20</sup>. BERT wurde von

<sup>18</sup> <https://nlp.stanford.edu/software/lex-parser.html>

<sup>19</sup> <https://web.stanford.edu/~jurafsky/slp3/13.pdf>

<sup>20</sup> <https://arxiv.org/pdf/1706.03762.pdf>

Rake	Yake	Multi-Rake	PKE	KeyBert
unsupervised learning	unsupervised learning und statistische Methoden	toolkit für Ende-zu-Ende keyphrase Extraktions pipeline	Rake für jede Sprache	bi-directional transformer model

**Tabelle 3.3.** Überblick bekannter Ansätze zur Extraktion von keywords aus Textdokumenten [Del21].

Google veröffentlicht und beschreibt einen Ansatz, dem im Bereich NLP eine wichtige Rolle zukommt [Hor18]. Dabei werden Sätze in Vektoren transformiert. Diese Vektoren erfassen auch die semantische Bedeutung der Sätze [Gro20b]. Die dann entstehenden numerischen Werte können dann mittels mathematischer Methoden ausgewertet und verglichen werden.

KeyBert selbst wurde von Maarten Grootendorst [Gro20a] entwickelt. KeyBert arbeitet, im Gegensatz zu den Anderen in Tabelle 3.3 aufgezeigten Ansätzen nicht mit statistischen Methoden, sondern eher auf semantischer Ebene [Gro20b]. In einem ersten Schritt werden so genannte Candidate keywords (Kandidaten für relevante keywords) kreiert. Dabei können verschiedene Ansätze genutzt werden. Durch das Festlegen eines *vectorizers* können zum Beispiel n-Gramme für die Länge der keyphrases festgelegt werden, oder Wörter die bei der Generierung von keywords- und phrases nicht berücksichtigt oder in jedem Fall berücksichtigt werden sollen. Auch die Art der Tokenisierung, falls es sich um eine Sprache handelt die einen anderen Satzbau wie englisch oder deutsch aufweist, wie etwa chinesisch, kann angepasst werden. Verschiedene vectorizer bieten auch weitere Möglichkeiten, wie zum Beispiel grammatisch korrekte keyphrases oder das individuelle Anpassen von POS, also beispielsweise das Berücksichtigen der Abfolge von Wortarten [Gro].

Danach wird ein vortrainiertes Bert Modell gewählt, welches die keyword Kandidaten in numerische Daten, genauer in Vektoren, umwandelt [Gro20b]. Dieser Vorgang wird auch *embedding* genannt, also das Einbetten von Texten oder Wörter in einen Vektorraum. Dabei sind verschiedene Transformer mit unterschiedlichen zugehörigen Modellen nutzbar und können im Anhang eingesehen werden. Auch auf das vorhandene Modell wird im späteren Kapitel eingegangen.

Eine weitere Funktionalität ist die Kosinus-Ähnlichkeit<sup>21</sup> (cosine similarity). Durch dieses Maß wird die semantische Distanz zwischen den gebildeten Vektoren, also den kreierten n-Grammen, und des gesamten Textes errechnet. Somit erhält man solche n-Gramme, die eine hohe Ähnlichkeit mit dem vorliegenden Textdokument aufweisen und dieses am besten repräsentieren [Mis21].

Weitere Algorithmen zur Anpassung der Diversität der resultierenden keywords sind *max sum similarity* und *maximal marginal relevance*. Ersteres beschreibt die

<sup>21</sup> <https://www.geeksforgeeks.org/cosine-similarity/>

den maximalen Summenabstand zwischen zwei Datenpaaren. Es wird also das Datenpaar gesucht, bei dem die Distanz am größten ist. Um eine hohe Differenz zwischen den gefundenen Kandidaten, aber eine niedrige zum gesamten Textdokument zu erzielen, kann dieser Parameter angepasst werden [Gro20b]. Der zweite Algorithmus versucht eine minimale Redundanz und eine maximale Diversität der keyword Resultate zu erzeugen. Dazu werden zuerst solche keywords genommen, die das vorliegende Textdokument am besten repräsentieren. Danach werden solche Kandidaten aufgenommen, die zwar zum Dokument, aber keine Ähnlichkeit mit den bereits gesammelten keywords aufweisen [Gro20b]. Die Wahl der Parameter werden im späteren Verlauf der Arbeit vorgestellt.

# **4**

---

## **Konzeption**

Bevor die Implementierung eines Wissensgraphen stattfinden kann, muss eine ausgiebige und geeignete Konzeption eine Rahmenvorgabe schaffen. Fragen zur Herangehensweise und der genutzten Methoden werden in diesem Kapitel aufgezeigt. Darüber hinaus wird erklärt warum diese genutzt werden bzw. zu welchem Ziel die jeweiligen Schritte führen sollen. Dazu gehört die Sammlung der Rohdaten die es im Laufe der Arbeit zu verarbeiten gilt, die Konzeption der Verknüpfung von verschiedenen Daten und Entitäten, sowie auch die Erläuterung einzelner Teilschritte die implementiert werden müssen, um zu einem ganzheitlichen Wissensgraphen zu gelangen. Dazu zählen erwünschte Funktionen, Designentscheidungen, genutzte Plattformen und Anforderungen die das Programm erfüllen soll.

### **4.1 Anforderungen und Formulierung der verschiedenen Problemstellungen**

Der erste Schritt bei der Erstellung von Wissensgraphen ist die Frage nach einer geeigneten Datenbasis. Bei der gegebenen Problemstellung werden wissenschaftliche paper benötigt, um diese miteinander verknüpfen zu können. Hierbei ist es wichtig darauf zu achten, dass die paper frei zugänglich sind (open Access), um Datenrechtliche Richtlinien nicht zu verletzen. Die Sammlung der Daten befindet sich im Anhang B.

Neben den wissenschaftlichen und frei zugänglichen Veröffentlichungen wurden auch zugehörige Konferenzen betrachtet. So konnten beispielsweise alle paper einer Konferenz genutzt werden, da es ganze Konferenzen gibt, die frei zugänglich sind. Darüber hinaus können Beschreibungen über die Konferenzen selbst gesammelt werden (oftmals als aims & scope betitelt). Um den Wissensgraphen am Ende nicht zu groß zu gestalten, wird sich in dieser Arbeit die Thematik der paper im Rahmen der Informatik und Mathematik bewegen. Zu Beginn der Planung der Vorgehensweise wird sich auf zehn paper beschränkt, die zu verschiedenen Konferenzen gehören. Sobald die grundlegenden Funktionsweisen umgesetzt wurde, erweitert sich die Anzahl an Daten. Doch bevor es um die Implementierung von Funktionalitäten geht, sollte im Vorhinein geplant werden, was das Programm im

grundlegenden Anforderungen erfüllen muss und mit welchen Daten gearbeitet werden soll. Da das grundlegende Ziel darin besteht bestimmte Informationen aus wissenschaftlichen Papieren auszulesen, muss das Programm in der Lage sein, diese einzulesen und zu verarbeiten bzw. zu verknüpfen. Im Anschluss müssen die Daten graphisch visualisiert werden können und auch durch graphentheoretische Algorithmen abfragbar sein. Aus diesen grundlegenden Funktionen ergeben sich verschiedene Teilschritte, die im Programm implementiert werden sollten. Daraus ergeben sich Fragestellungen zur Konzeption:

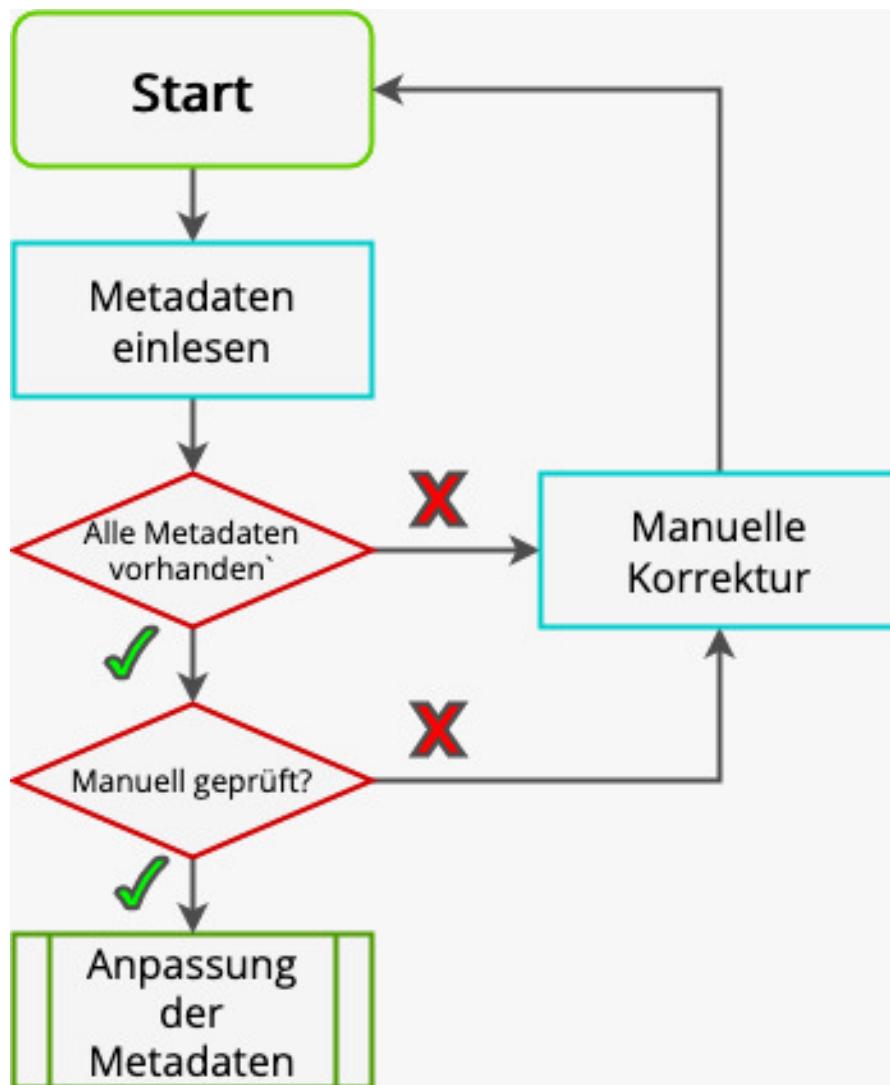
- Wie werden wissenschaftliche Veröffentlichungen eingelesen?
- Welche Informationen sollen verarbeitet werden?
- Wie werden Informationen miteinander verknüpft?
- Wie sollen die verknüpften Informationen dargestellt werden?
- Wie werden meine Informationen abgefragt?

Anhand dieser Auflistung an Fragen erfolgt die Konzeption des Programms.

## 4.2 Einlesen und Aufbereitung der Rohdaten

Nach näherer Betrachtung der wissenschaftlichen Paper wird klar, dass viele Kerninformationen nicht mittels NLP Algorithmen ausgelesen werden sollten. Der Aufbau der einzelnen PDF-Dokumente ist zu unterschiedlich, als dass hier eine ganzheitliche Regel zum Finden der Autoren, Titel und anderen Kerninformationen möglich wäre. Aus diesem Grund werden solche Informationen aus den Metadaten der einzelnen PDF's ausgelesen. Nach einer Prüfung der Vollständigkeit der Metadaten stellt sich heraus, dass hier eine Vorverarbeitung stattfinden muss, um Metadaten entweder zu vervollständigen, zu korrigieren oder gar initial zu befüllen. Das Programm sollte also in der Lage sein nicht nur den Fließtext der Dateien einzulesen, sondern auch deren Metadateninformationen. In Kapitel 5 werden die dazu genutzten Bibliotheken und Packages näher betrachtet und erläutert. Zuerst wird das Einlesen der Metadaten und deren Korrektheit konzipiert. Das manuelle Korrigieren der Informationen sollte so gering wie möglich gehalten werden, um Fehleranfälligkeit zu minimieren. Somit fällt die Wahl auf einen Programmablauf, wie er in Abbildung 4.1 abgebildet ist.

Beim Start des Programms werden die Metadaten so eingelesen, wie sie in der originalen Datei vorhanden sind. Danach wird geprüft, ob alle Informationen vorhanden sind. Dazu wird eine Hilfsdatei genutzt, die ein manuelles Anpassen sowie ein Ein- und Auslesen ermöglicht. Bei der manuellen Befüllung soll es zusätzlich die Möglichkeit geben, die Dateien als *geprüft* zu markieren. Somit soll der Fall abgedeckt werden, dass auch wenn alle Informationen gegeben sind, diese immer noch falsch sein könnten und somit trotzdem angepasst werden müssten. Wenn alle Metadaten korrekt befüllt sind, sollten diese auch in den eigentlichen PDF-Dateien programmatisch angepasst werden. Das Programm bricht ab, wenn die Prüfspalte



**Abbildung 4.1.** Programmablaufplan vom Einlesen der Metadaten der original PDF Dateien

nicht befüllt ist und soll den Nutzer bitten, die Metadaten manuell zu befüllen beziehungsweise zu korrigieren. Diese Funktion ist der letzte Schritt des gezeigten Programmablaufs, weswegen hier der *Stop* Baustein des Programmablaufs entfällt.

Folgende Metadaten sollen im späteren Verlauf hinterlegt, ausgelesen und verarbeitet werden:

- Titel der paper
- Autor(en) der paper
- Veröffentlichungsdatum der paper
- zugehörige Konferenz der paper

Die Korrektur der Metadaten in den originalen PDF Dateien soll in einer weiteren Funktion umgesetzt werden, nachdem die Hilfsdatei erstellt und die Metadaten korrekt hinterlegt wurde. Die Datei soll die einzelnen Daten einlesen und in den jeweiligen PDF Dateien als Metadaten hinterlegen. Selbiges soll für die Konferenzbeschreibungen gelten, bei denen der Konferenztitel ebenfalls in den Metadaten hinterlegt werden soll.

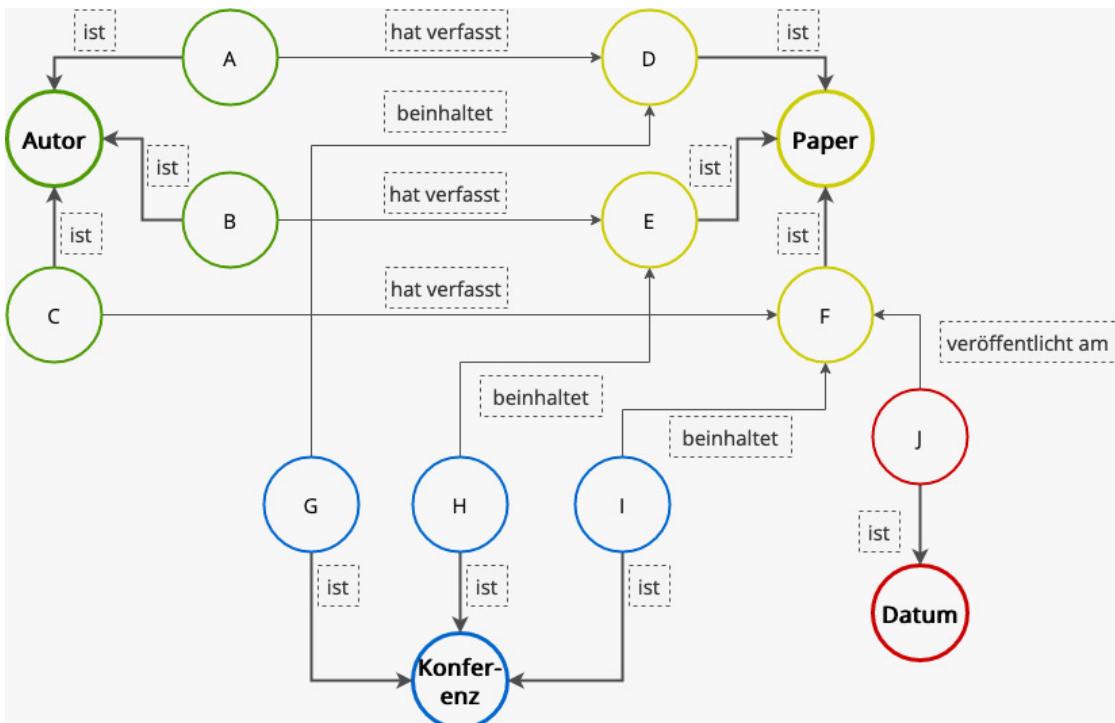
Neben den Metadaten sollen auch Fließtextinformationen mittels NLP ausgelesen werden. Diese Informationen sollen die Basis für die behandelten Thematiken in papern oder Konferenzen bilden. Grundlegend soll das Programm die Funktionalität bieten, Texte aus den PDF Dateien zu extrahieren. Hierbei soll es möglich sein, nur das abstract oder zumindest die Seite, auf der sich das abstract befindet, einzulesen, falls dieses existiert. Ansonsten wird der komplette Text extrahiert. Bei Konferenzen soll immer der komplette Text extrahiert werden. In einem nächsten Schritt sollen die eingelesenen Texte für die Weiterverarbeitung aufbereitet werden. Dazu sollen sogenannte stop words, also Wörter, die in der englischen Sprache zwar oft Verwendung finden, auf den semantischen Kontext aber keinerlei Auswirkungen haben, entfernt werden. Darüber hinaus sollen auch Zahlen bei der Weiterverarbeitung nicht berücksichtigt und herausgefiltert werden.

Nachdem der Fließtext umgestaltet ist, sollen die Kerninformationen in Form von keywords herausgelesen werden. Diese keywords werden dann in den Wissensgraphen eingebunden. Die Bildung der keywords soll mittels eines geeigneten Algorithmus geschehen, wobei neben den keywords auch paper- bzw. Konferenztitel miteingelesen werden, um im späteren Verlauf die Zuordnung der Informationen zu den jeweiligen papern und Konferenzen erhalten zu können.

### 4.3 Entitäten und Beziehungen

Neben dem Einlesen und der Aufbereitung der Daten ist es wichtig zu wissen, welche Informationen dem Nutzer bereitgestellt werden sollen und wie diese miteinander in Verbindung stehen. Dazu ist ein Festlegen von Entitäten und Beziehungen unabdingbar. Da die Daten durch unterschiedliche Vorgehensweisen der Vorverarbeitung aufbereitet werden, können die gewünschten Entitäten und Beziehungen in zwei Teile untergliedert werden. Der eine Teil stammt aus dem Auslesen der Metadaten und der Andere aus den Schritten des NLP. Der Graph der entstehen soll, setzt sich sozusagen aus zwei Teilgraphen zusammen. Der Graph der Metadaten ist in Abbildung 4.2 zu sehen und der aus den Prozessschritten des NLP entstehende Graph ist in Abbildung 4.3 abgebildet. In beiden Abbildungen sind die Entitäten und deren Beziehungen untereinander zu erkennen. Dabei werden manchen Entitäten wie Autoren, papern oder Konferenzen jeweils eine weitere Entität zugeordnet, die einen Namen eines Autors als Autor klassifiziert, bzw. einen papertitel als paper kategorisiert. Dies soll zu einer besseren und verständlicheren Lesbarkeit des Graphen führen. Im Folgenden wird eine Liste der Entitäten aufgeführt, welche als Label von Instanzen dienen.

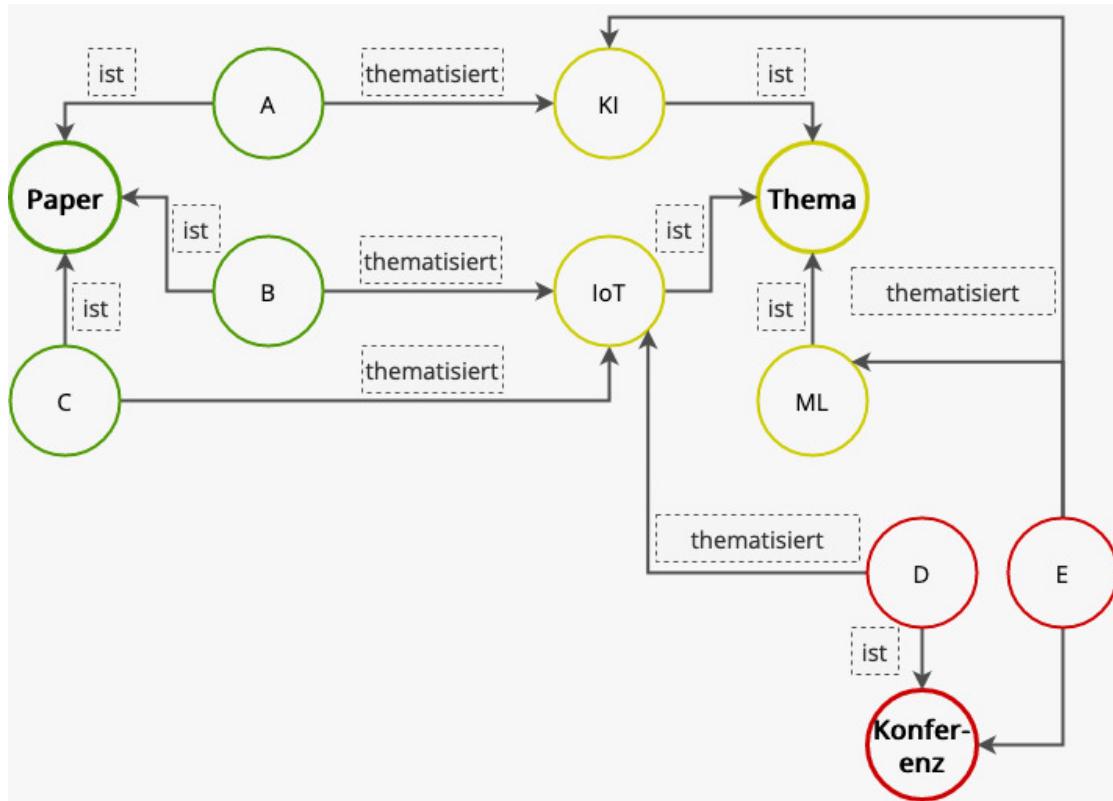
- Autor
- paper
- Konferenz
- Thema Paper
- Thema Konferenz



**Abbildung 4.2.** Konzipierter Graph, der durch das Auslesen der Metadaten der Textdokumente entsteht.

Neben den Hauptentitäten gibt es einige Instanzen die aus den papern und Konferenzen ausgelesen werden sollen. Dabei sollen Namen der Autoren, die Titel der paper und Konferenztitel die einzelnen Instanzen der jeweiligen Überknoten bilden. Auch das Veröffentlichungsdatum von Papertiteln wird über die Metadaten ausgelesen. Dabei werden Veröffentlichungsdaten nicht als eigener Knoten, sondern als Attribut der jeweiligen papertitel Knoten integriert. Die Thematik von papern und Konferenzen soll über NLP generiert werden und bildet die im vorherigen Abschnitt erläuterten keywords.

Nachdem die gesuchten Entitäten definiert sind, ist die Verbindung zwischen diesen Knoten zu konzipieren. Zuerst sind alle Überknoten mit ihren Instanzen über eine Kante *ist* verbunden. Das heißt, jeder Name eines Autors ist mit der Entität *Autor*, Namen von Konferenzen mit *Konferenz* und Titel der paper mit *paper* verbunden. Auch werden keywords von paper Instanzen mit *Thema paper* über



**Abbildung 4.3.** Konzipierter Graph, der durch das Erstellen von keywords und keyphrases aus den Textdokumenten entsteht. Hier vereinfachte Darstellung des Knotens Thema (umfasst Thema Konferenz und Thema paper).

handelt von und Konferenztitel mit *Thema Konferenz* über *thematisiert* verbunden und jedes Thema zu dem jeweiligen Überknoten *Thema paper* oder *Thema Konferenz*. Darüber hinaus sollen Beziehungen zwischen Namen von Autoren und der Titel von papern bestehen, die als *hat verfasst* deklariert sind. Somit werden Autoren ihren verfassten papern zugeordnet. Selbiges soll für eine Verbindung zwischen Konferenztiteln und paper Titeln erfolgen, wobei diese Beziehung mit *ist Teil von* betitelt werden soll. So soll die Zugehörigkeit von papern zu einer Konferenz dargestellt werden. Diese Beziehungen entstehen zusammen mit dem Auslesen der Metadaten und der Integration von keywords und ergeben einen Graphen.

## 4.4 Aufbau und Abfrage des Graphen

Nachdem passende Rohdaten, deren Aufbereitung und die Entitäten und deren Beziehungen konzipiert wurden, kann über die Visualisierung des entstandenen Geflechts nachgedacht werden. So soll jede Entität einen eigenen Knoten im Graphen darstellen und die im Vorhinein konzipierten Verbindungen als Kanten aufweisen. So soll beispielsweise eine Instanz eines wissenschaftlichen papers (ein konkreter Titel) mit der Überkategorie *paper*, einer Instanz Autor(ein konkreter Name), einer Instanz Konferenz (ein konkreter Konferenztitel) und mit passenden key-

words, die aus demselben paper generiert wurden, verbunden sein. Zusätzlich soll das Veröffentlichungsdatum als an die Knoten von papertiteln gehängt werden. Der entstehende Graph soll zusätzlich einigen visuellen und funktionellen Kriterien entsprechen. Zu den visuellen Kriterien zählen die variable Größe von Knotenpunkten. Dabei sollen Knoten, die viele Ein- oder Ausgänge besitzen, größer als Andere dargestellt werden. Jede Gruppe an Entitäten soll ein eigenes Farbschema erhalten. Zusätzlich sollen funktionale Aspekte wie das Abfragen des Graphen möglich sein. Der Nutzer soll die Möglichkeit haben über einfache, programmatiche Abfragen Ergebnisse zu erhalten. Dies kann beispielsweise die Frage nach einem oder mehreren papern sein, welches sich mit IoT (Internet of Things) beschäftigt. Der Nutzer soll die Möglichkeit haben ein bestimmtes Thema einzugeben und die Kategorie der Ausgabe zu spezifizieren. Somit sollen beispielsweise alle Titel (wenn die Spezifikation Titel ist) ausgegeben werden, die sich mit der Thematik deep learning (gesuchtes Thema) beschäftigen. Die Abfragen sollen mittels *SPARQL* realisiert werden, wodurch der Nutzer die Abfragen selbst formulieren kann.

# 5

---

## Implementierung und Anwendung

In diesem Teil der Arbeit wird auf die konkrete Umsetzung der Konzeption zum Aufbau einer Wissensbasis über Konferenzen und wissenschaftlichen papern eingegangen. Dabei werden zuerst die verwendete Sammlung an PDF Dateien vorgestellt, dann der Ablauf der verschiedenen Programmteile sowie verwendete Pakete und weitere Rahmenanwendungen. Die Umsetzung stützt sich auf die im Kapitel 4 konzipierte Vorgehensweise. Erläuterungen der Anwendung des Programms sind die mögliche Erweiterung des Graphen durch weitere paper und Konferenzen, die visuelle Darstellung des resultierenden Wissensgraphen, ein Auszug von Nachbar-knoten um die Verbindungen im Einzelnen darstellen zu können und beispielhafte Suchanfragen mit zugehörigen Ergebnislisten.

Verwendete Pakete, Module und Bibliotheken sind im Abschnitt B.3 enthalten.

### 5.1 Rahmen und Sammlung der Daten

Wie schon im Verlauf der Arbeit erwähnt, arbeitet das Programm mit wissenschaftlichen papern um aus deren Inhalt eine Wissensbasis aufzubauen. Diese paper wurden in Form von PDF Dateien zusammengestellt, da die meisten Publikationen in diesem Format auch erhältlich sind. Zu den Konferenzen wurde jeweils die Konferenzbeschreibung ebenfalls in Form von PDF Dateien extrahiert und zur Verarbeitung genutzt, um ein einheitliches Datenformat zu erhalten. Die Dateien befinden sich in einer lokalen Ordnerstruktur, die von dem Python Skript abgefragt wird. Die Python Skripte wurden mittels der Benutzeroberfläche jupyter lab<sup>1</sup> verfasst und ausgeführt. Der Grund für die ausgewählte Umgebung war die Übersichtlichkeit durch die Erstellung von Code Blöcken sowie deren gegliederte Ausgaben.

Die Datensammlung beschränkt sich auf open Access Dateien, welche von verschiedenen Portalen heruntergeladen wurden. Das Herunterladen und aufbereiten dieser Daten wird teilweise manuell und programmatisch gelöst. In einem ersten Schritt wurden die paper und die zugehörigen Konferenzen heruntergeladen und

---

<sup>1</sup> [https://jupyterlab.readthedocs.io/en/stable/getting\\_started/overview.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html)

als PDF gespeichert. Manche Konferenzbeschreibungen sind nicht in einem Dateiformat vorhanden und mussten direkt von der Webseite kopiert, in ein Word Dokument eingefügt und als PDF Datei exportiert werden.

Bevor der erster programmatische Schritt erfolgen kann, wurde eine Excel Datei (`metadata_list.xlsx`) erstellt, in der die Metadaten der paper hinterlegt werden sollen. Diese Datei ist in Tabelle 5.1 exemplarisch dargestellt.

PDF	Name der PDF Datei
Titles	Titel der paper
Authors	Verfasser der paper
Date	Veröffentlichungsdatum der paper
Conference	Name der zugehörigen Konferenz
Check	Prüffeld

**Tabelle 5.1.** Exemplarischer Aufbau der `metadata_list.xlsx` Datei, welche die Metadaten der Datensammlung speichert.

Die Datei dient zum Einen dazu, bereits vorhandene Metadaten aus den PDF Dateien auszulesen und zum Anderen für die weitere Verarbeitung.

### 5.1.1 Erweiterung und Aufbereitung der Datenbasis

Im ersten programmatischen Schritt sollen die Metadateninformationen der paper und Konferenzen gesammelt, gespeichert und weiterverarbeitet werden können. Dazu wird die bereits erwähnte Excel Liste erstellt, die die Metadaten von PDF Dateien ausliest. Diese PDF Dateien stellen wissenschaftliche paper dar. Sobald eine neue Datei hinzukommt erzeugt das Programm folgenden Fehler:

Exception: Anpassen der Metadaten in Datei notwendig.  
 Datei wird geöffnet.  
 Anschließend Programm neustarten.

Gleichzeitig wird die Excel Datei geöffnet. Die neue Datei wird (in diesem Fall energy efficiency-Test), wie in Abbildung 5.1 gezeigt, als letzte Zeile angehängt. Da die Datei bereits Metadaten enthält, sind die Spalten, bis auf `Check` gefüllt. Diese Spalte ist immer leer wenn eine neue Datei zu der Excel Liste hinzugefügt wird. Sobald diese Spalte von einem Nutzer gefüllt wird, beispielsweise mit x, kann die Funktion erneut aufgerufen werden und das Programm läuft weiter. Somit wird eine menschliche Korrektur garantiert und die Qualität der späteren Ausgabewerte im Graphen erhöht.

Folgende Metadaten werden ausgelesen:

- Titel
- Author

PDF	Titles	Authors	Date	Conference	Check
Ćiprijanović_2022_Mach._LDeepAdversaries: e;Aleksandra Ćipriji	DeepAdversaries: e;Aleksandra Ćipriji	21 July 2	Machine Learning: Science and Technology	x	

**Abbildung 5.1.** Hinzufügen einer Test PDF Datei `energy_efficiency_Test.pdf` zu Excel Datei `metadata.list`

- CreationDate
- ConferenceName

Dabei stellt ConferenceName eine benutzerdefinierte Information dar, die hinzugefügt wird. Durch diese Information werden im späteren Verlauf Konferenzen wissenschaftliche paper zugeordnet.

Um den Wissensgraphen aufzubauen werden jedoch nicht nur Metadaten genutzt. In einem nächsten Schritt werden in einem weiteren Python Skript die eigentlichen Texte der PDF Dateien ausgelesen. Der Ablauf des Programms kann in Abbildung 5.2 eingesehen werden. Es werden die Texte von wissenschaftlichen papern, sowie auch von Konferenz Beschreibungen eingelesen und verarbeitet. Als Ergebnis liefert das Skript eine Liste an keywords mit zugehörigen Relevanzen zu jedem paper und jeder Konferenz. Diese keywords dienen später ebenfalls zur Befüllung von Knoten für den Wissensgraphen.

An dieser Stelle werden dictionaries genutzt, wodurch die Texte und Titel von Konferenzen und wissenschaftlichen papern sowie deren keywords- und phrases bereits miteinander verknüpft sind und somit ein Verlieren der Zuordnungen vermieden wird. Hier wird ebenfalls die Relevanz mit berücksichtigt. Diese gibt an, wie gut die keywords den zugehörigen text widerspiegeln.

## 5.2 Entstehung und Visualisierung des Graphen

Neben der Erzeugung von Phrasen und Schlüsselwörtern, muss auch der Graph mit Knoten und Kanten gefüllt, sowie auch die Darstellung definiert werden. Der am Ende entstandene Graph ist in Abbildung 5.3 zu sehen. Dieser wurde jedoch ohne das Einbinden von RDF Elementen und nur mittels Networkx erzeugt.

Die untere Darstellung enthält keine Label, also Beschriftungen, um das Netzwerk übersichtlicher zu gestalten. Die größeren Knoten bilden bis auf eine Aufnahme jeweils die Überknoten. Es gibt Konferenz Instanzen, die größer dargestellt sind als der Überknoten Konferenz an sich. Der Grund dafür sind die höhere Anzahl an paper, die zu einer Konferenz gehören können, als Konferenz Instanzen zu Konferenz.

Der Ablauf bis zur Erstellung beziehungsweise das Abfragen des Graphen ist im Programmablauf in Abbildung 5.4 zu sehen.

Die Farbverteilung ist wie folgt:

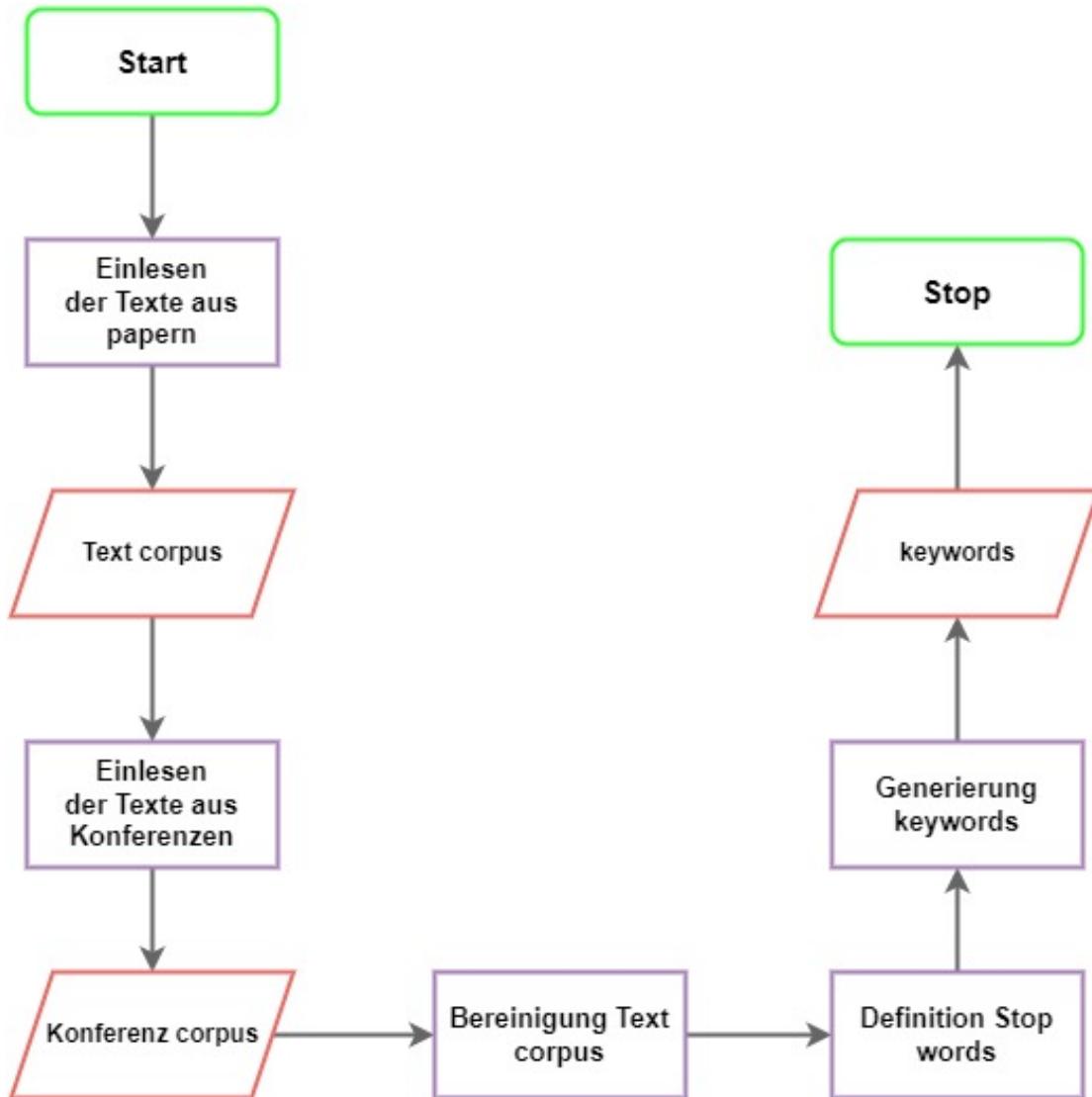
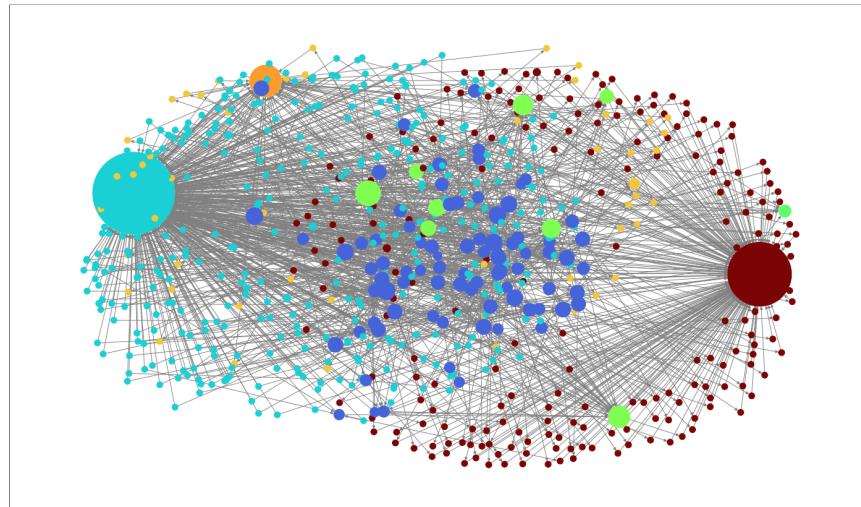
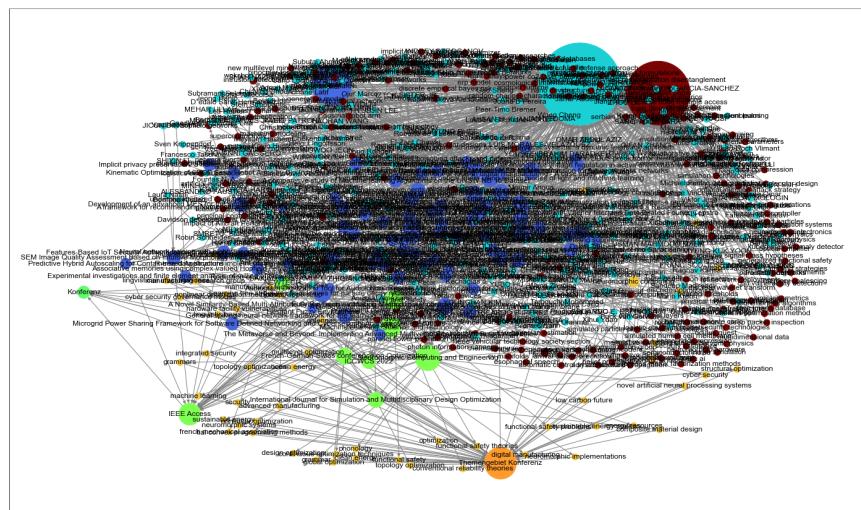


Abbildung 5.2. Programmablaufplan des Skripts PreProcess\_NLP

- Hellblau: Autor
- Dunkelblau: paper
- Dunkelrot: Themengebiet paper
- Orange: Themengebiet Konferenz
- Hellgrün: Konferenz

Die kleineren Knoten mit ähnlichen Farben sind jeweils Instanzen eines Überknotens.  
Die grauen Pfeile stellen die Verbindungen zwischen den Knoten dar.

Die Erstellung des Graphen erfolgte zuerst nur mittels *Networkx*. Diese Methode bietet ein breiteres Spektrum an visuellen Anpassungen, wie beispielsweise das Hinzufügen einer colormap (Knotenabhängige Farbverteilung). Dies ist durch die



**Abbildung 5.3.** Generierter Graph nur mittels Networkx. Mit Knoten Beschriftung (oben). Ohne Knoten Beschriftung (unten).

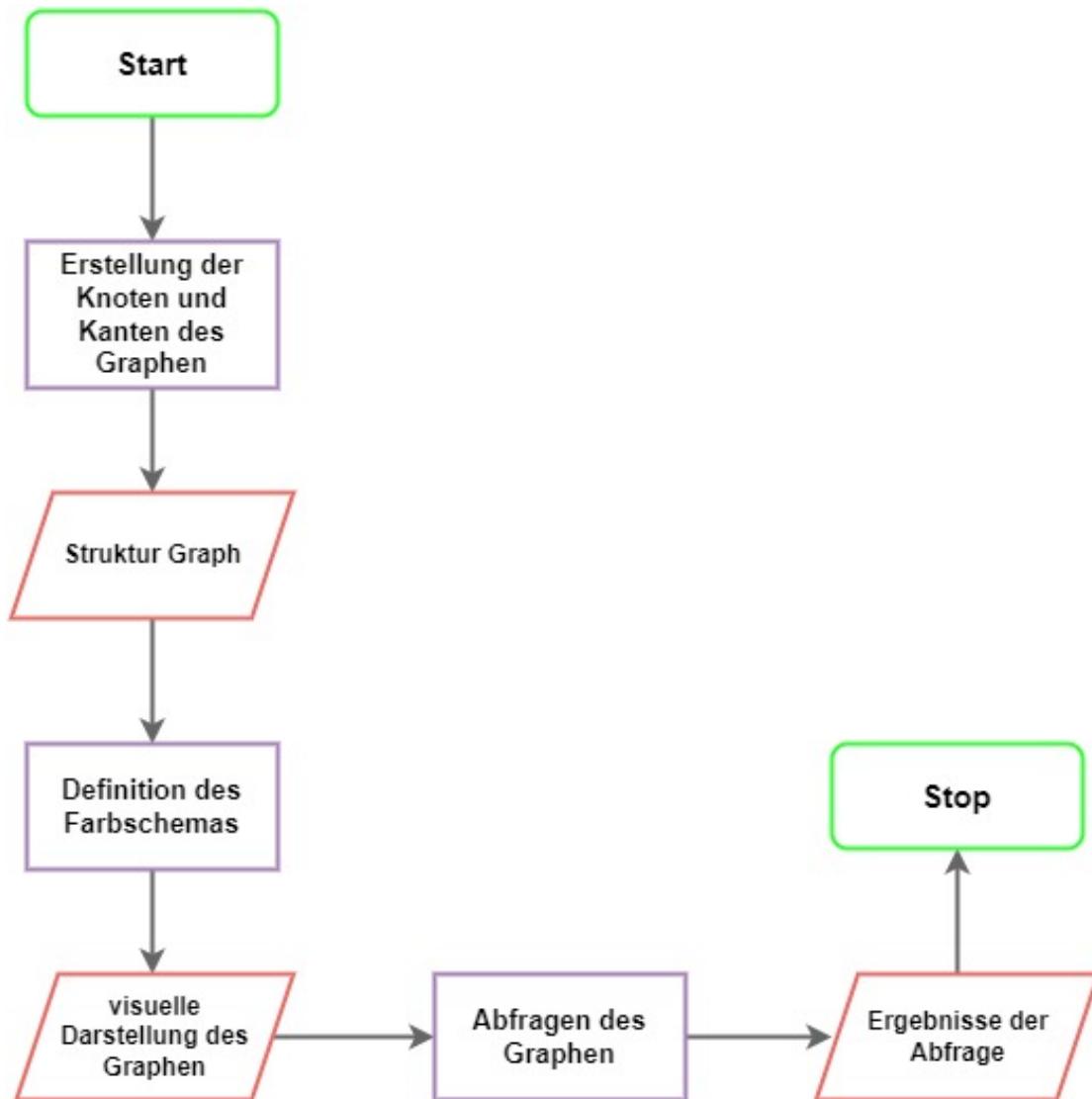
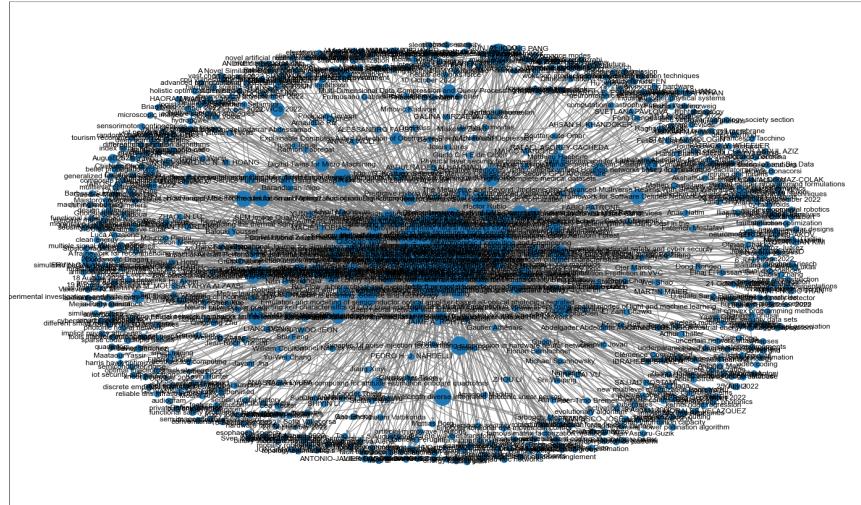


Abbildung 5.4. Programmablaufplan des Skripts KG

Ergänzung von Attributen bei Knoten möglich. Im späteren Verlauf wurde der Graph mittels RDF Struktur ergänzt. In Abbildung 5.5 ist der durch **RDFLib** erstellte Graph zu sehen.

Durch diese Veränderung bzw. Ergänzung ist ein Hinzufügen von Knotenabhängigen Farben nicht mehr erfolgt. Die Implementierung mittels der **RDFLib** von Python ermöglicht die Verknüpfung von Knoten in der Form Subjekt, Prädikat und Objekt. Somit sind Abfragen mittels *SPARQL* möglich. Die Prädikate, also die Verbindungen zwischen den Entitäten, erhalten durch die Einbindung des RDF Vocabulars andere Namen, es gelten jedoch schematisch alle Verbindungen, wie sie in Abschnitt 4.4 konzipiert wurden. Im Folgenden werden die Namen der Relationen in der Form Subjekt, Prädikat, Objekt, aufgelistet:

- Autor, Name, Instanz eines Autors



**Abbildung 5.5.** Erstellter Graph durch RDFLib und Networkx.

- paper, Titel, Instanz eines papers
- ThemengebietPaper, ist, Instanz paper keywords
- Instanz paper, subject, Instanz paper keywords
- ThemengebietKonferenz, ist, Instanz Konferenz keyword
- Instanz Konferenz, subject, Instanz Konferenz keyword
- Instanz paper, publisher, Instanz einer Konferenz
- Konferenz, Titel, Instanz einer Konferenz
- Instanz paper, date, Instanz eines Datums
- Instanz Autor, creator, Instanz eines papers

Subject steht hierbei für das Thema, creator für den Verfasser und publisher für die Herausgebende Konferenz. Bei der Erstellung der keyword Verbindungen kommt nun der Relevanz eine Bedeutung zu. Eine Schranke von 0.4 bei papern und 0.44 bei Konferenzen. Somit sind keywords, die das jeweilige Dokument schlecht wiedergeben ausgenommen und die Menge wird reduziert.

### 5.2.1 Abfragen des Graphen

Um nach Knoten in dem Graphen zu suchen wird SPARQL genutzt. Der Nutzer sollte hier bereits Erfahrung mit derartigen Abfragen haben. Im Folgenden werden verschiedene Beispiele gezeigt, wodurch gleichzeitig die Syntax solcher Abfragen

und die jeweiligen Ergebnisse gezeigt werden. Generell kann durch folgenden Code überprüft werden, ob sich ein Knoten oder ein Teil eines Triples in dem Graphen befindet:

```
searchy = 'Audiogram Digitization Tool for Audiological Reports'

if (Literal(searchy), None, None) in G_KG:
    print("This graph contains triples about", searchy)
```

Dabei können auch Prädikate und Subjekte gleichzeitig gesucht werden oder ausschließlich Objekte. Jede Kombination ist hier möglich.

# 6

---

## Ergebnisse

Zu Beginn dieser Arbeit in Stand der Technik, wurden bereits zwei Ansätze zur Erstellung von Wissensgraphen vorgestellt. In dieser Arbeit wurde der Ansatz von einem RDF Format verfolgt, was auch die Standard Vorgehensweise zum jetzigen Zeitpunkt darstellt. Aber auch das Verfahren der Erstellung von Property Graphen, im Besonderen mittels Neo4j, hat seine Berechtigung. Im Folgenden werden diese beide Ansätze miteinander verglichen und jeweilige Vor- und Nachteile aufgezeigt.

Wie bereits in vorherigen Kapiteln erläutert, ist jedes Triple in einem RDF formalisierten Graphen in der Form Subjekt, Prädikat, Objekt. Dabei ist jeder Knoten oder jede Entität durch eine eindeutige URI gekennzeichnet. Darüber hinaus existiert die Abfragesprache SPARQL um solche Graphen abzufragen. Der Vorteil bei dieser Art eines Graphen besteht in der Standardisierung des gesamten Konzepts. Die Nutzung von anderen RDF basierten Daten oder des Vokabulars ist über das WWW möglich und verlinkt somit eine immer größer werdende Datenbasis, die für jeden zugänglich ist. Auch die Abfrage solcher Graphen durch die standardisierte Sprache SPARQL bleibt immer gleich. Durch diese Standards ist auch die Verknüpfung verschiedener Wissensgraphen leicht umsetzbar. Gleichzeitig führt die Standardisierung zu Nachteilen, wie mangelnde Flexibilität beim Aufbau einer Wissensbasis. So können immer nur Triple in der festgelegten Struktur aufgebaut werden und keine Attribute oder sonstige weiterführende Informationen einfach an Knoten angehängt werden. Auch können nicht mehr als 2 Entitäten gleichzeitig miteinander verknüpft werden [Vet22].

Property Graphen besitzen neben ihren Knoten und Kanten auch Attribute. Diese Attribute geben nähere Informationen über Entitäten oder Beziehungen. Die Form der Informationen werden als Quell- und Ziel Entitäten bezeichnet. Hierbei existiert keine Standardisierung der Erstellung des Graphen oder bei der Nutzung einer Abfragesprache. Viele größere Unternehmen, darunter auch Neo4j, verknüpfen ihre eigenen Graphen, da es übergreifend nicht möglich ist und nutzen ihre eigene Abfragesprache [Vet22]. Bei Neo4j handelt es sich dabei um Cypher. Dabei hat Neo4j eine eigene Benutzeroberfläche um solche Graphen zu erstellen und ist somit wesentlich Nutzerfreundlicher und gerade für Neueinsteiger einfacher zu bedie-

nen. Auch Cypher hat eine gute Dokumentation mit vielen Beispielen. Durch die Möglichkeit Kanten und Knoten Eigenschaften in Form von Attributen hinzufügen zu können, werden Entitäten teilweise detaillierter beschrieben, ohne neue Knoten im Graphen erstellen zu müssen [Vet22]. Doch die fehlende Standardisierung bringt zwar Flexibilität, jedoch keine Interoperabilität. Das Teilen von Informationen über verschiedene Datenbanken und verschiedenen Organisationen hinweg ist nur schwer möglich [Vet22]. Somit entstehen erneut Redundanzen an Datenhaltungen (bei verschiedenen Organisationen, die solche Graphen implementieren), die nicht miteinander geteilt werden können.

## Zusammenfassung und Ausblick

Am Ende dieser Arbeit steht eine Zusammenfassung dessen, welche Ergebnisse, insbesonders im Hinblick auf die zu Beginn definierte Vorstellung des Gesamtresultates, entstanden sind und wie diese für mögliche spätere Arbeiten als Basis dienen können. Damit werden auch bestimmte Programmteile hervorgehoben. Zusätzlich wird der Programmablauf zusammengefasst und übersichtlich dargestellt. Außerdem wird die Arbeit kritisch reflektiert und Verbesserungen für verschiedene Teile der Implementierung in Form eines Ausblicks vorgeschlagen.

In Abbildung 7.1 ist eine minimalistische Darstellung des gesamten Programmablaufs zu sehen.

Zu Beginn werden Funktionen aus PreProcessMetadata aufgerufen, wie auch in Abbildung 4.1 bereits dargestellt. Nachdem die `metadata_list` befüllt und geprüft wurde, werden dies Metadaten in die PDF Dateien der Konferenzen und paper selbst eingetragen und der nächste Programmabschnitt beginnt. Nun werden in `PreProces_NLP`, in Abbildung 5.2 schon näher betrachtet, die Texte aus den PDF Dateien ausgelesen und bereinigt. Folgend werden die keywords- und phrases für jedes einzelne Textdokument generiert. In Folge dessen wird der letzte externe Programmabschnitt `KG` ausgeführt. Dieser wurde ebenfalls bereits in Abbildung 5.4 präziser gezeigt. An dieser Stelle wird der Graph anhand der `metadata_list` Datei und den generierten keywords gebaut. Zusätzlich wird dieser visuell aufbereitet und dargestellt. Im Anschluss kann der Graph nach bestimmten Themen durchsucht werden. Die Ausgabe kann ebenfalls auf Autoren, paper oder Konferenzen eingeschränkt werden.

Rückblickend war zu Beginn dieser Arbeit nicht klar, dass ein großer Teil des Programms für das Ein- und Auslesen von Metadaten zuständig sein wird. Nach näherer Betrachtung der Aufgabenstellung ist jedoch schnell hervorgegangen, dass es wenig sinnvoll wäre jede einzelne Entität mittels intelligenten Ansätzen wie beispielsweise `KeyBert` auszulesen. In Anbetracht des zeitlichen Rahmens weisen wissenschaftliche Publikationen und zugehörige Konferenzen eine zu diverse Struktur auf, als dass diese auf eine einheitliche Art und Weise und mit annehmbaren Ergebnissen verarbeitet werden können. Aus diesem Grund entschied ich mich für die manuelle Korrektur der Metadaten in einer externen Datei. Somit werden

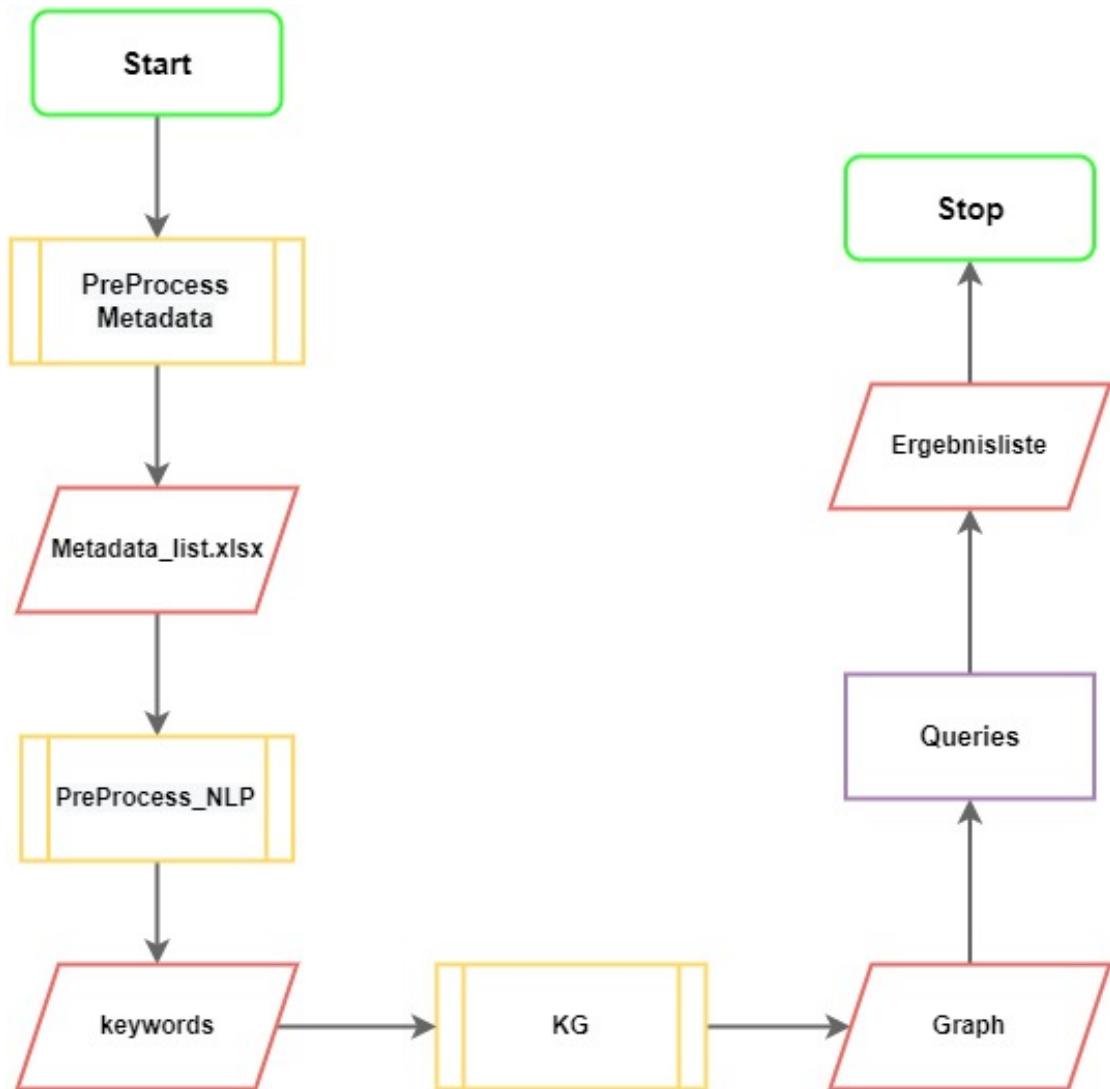


Abbildung 7.1. Zusammengefasster Ablaufplan mit allen Unterprogrammen

fehlerhafte Titel, Autoren sowie Konferenz Zugehörigkeiten gesenkt und ein valider Graph konnte entstehen. An den keywords ist zu erkennen, dass das Modell KeyBert nicht immer gute Ergebnisse liefert. Selbst bei einer definierten Schranke der Relevanz sind hier Phrasen dabei, die nicht viel zur Thematik der Konferenz oder des papers beitragen. Bei der Implementierung der Abfragen sind nur beispielhaft einige Möglichkeiten vorgestellt worden. Im Allgemeinen könnte jede Abfrage, die sich innerhalb der Funktionalität von SPARQL bewegt, möglich sein. Die Problemstellung und Zielsetzung sind zwar teilweise anders als im Vorhinein geplant, jedoch vollumfänglich erfüllt und bearbeitet worden. Beispielsweise gibt es einen visuell ansprechenderen Graphen der mittels Networkx erstellt worden, jedoch nicht durch SPARQL abfragbar ist. Der Aufbau einer kleinen Wissensbasis mit einer Auswahl an Konferenzen mit zugehörigen papern und Autoren ist gelungen. Ebenso wurden diese Daten sinnvoll miteinander verknüpft, sowie auch

durchsuchbar und abfragbar gemacht. Es können bestimmte paper, Konferenzen oder Autoren durch eine Suche nach einer Thematik gefunden werden. Zusätzlich wird ein Erscheinungsjahr von wissenschaftlichen Publikationen mit in den Graphen eingepflegt.

Kritisch betrachtet und ebenso wichtig für eventuelle Folgearbeiten zu dieser Thematik ist, dass das genutzte Modell zur Generierung von keywords eventuell bessere Ergebnisse liefern kann, wenn beispielsweise andere Parameter genutzt werden oder ein anderes Modell genutzt werden würde. Auch das trainieren auf ausschließlich wissenschaftlichen Publikationen zu verschiedenen Themengebieten wäre denkbar. Ebenso könnten verschiedene Modelle gegeneinander gehalten werden um ein vielleicht besser passenden Ansatz als **KeyBert** zu finden. Die Anzahl an solchen Modellen ist groß. Darüber hinaus könnten Entitäten, die bisher durch Metadaten generiert werden, ähnlich wie die keywords- und phrasen entstehen. Hier könnten Modelle speziell zum Auslesen von Autoren und Titeln möglich sein. Auch die Optimierung des Programms, hinsichtlich des Volumens aber auch der Effizienz ist ein Aspekt, der immer eine Rolle solcher Umsetzungen spielt.

Am Ende ist zu sagen, dass der Ausbau und die Verbesserung der Basis, die in dieser Arbeit geschaffen wurde, durchaus zu einer vereinfachten Suche nach wissenschaftlichen Publikationen oder bestimmten Autoren führen kann und es sich wahrscheinlich lohnt mehr auf dem Gebiet von Wissensgraphen und generell zu Semantik in der technischen Welt zu forschen und diese Möglichkeiten in ein breites Spektrum von Anwendungsgebieten einzubetten.

---

## Literaturverzeichnis

- AI03. ALBRECHT, KIRSTEN und ROLAND ILLIG: *RDF und RDF Schema*, Nov 2003.
- AMOOV20. AL-MOSLMI, TAREQ, MARC GALLOFRÉ OCAÑA, ANDREAS L OPPDAHL und CSABA VERES: *Named entity extraction for knowledge graphs: A literature overview*. IEEE Access, 8:32862–32881, 2020.
- BHL<sup>+</sup>14. BUSSE, JOHANNES, BERNHARD HUMM, CHRISTOPH LUBBERT, FRANK MOELTER, ANATOL REIBOLD, MATTHIAS REWALD, VEROPIKA SCHLÜTER, BERNHARD SEILER, ERWIN TEGTMEIER und THOMAS ZEH: *Was bedeutet eigentlich Ontologie?* Informatik-Spektrum, 37, 08 2014.
- BLFD99. BERNERS-LEE, TIM, MARK FISCHETTI und MICHAEL L. DERTOUZOS: *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1st Auflage, 1999.
- Del21. DELGADO, CARLOS: *Top 5: Best Python Libraries to Extract Key-words From Text Automatically*, 2021.
- dic. *RDF / RDFS*. <https://www.dictajet.de/rdf-rdfs/>.
- DZ07. DITTMANN, L.U. und P.D.S. ZELEWSKI: *OntoFMEA*. Gabler Edition Wissenschaft : Information - Organisation - Produktion. Deutscher Universitätsverlag, 2007.
- Gre19. GREEN, ALASTAIR: *GQL Is Now a Global Standards Project alongside SQL*, Sep 2019.
- Gro. GROOTENDORST, MAARTEN: *CountVectorizer Tips & Tricks*.
- Gro20a. GROOTENDORST, MAARTEN: *KeyBERT: Minimal keyword extraction with BERT*., 2020.
- Gro20b. GROOTENDORST, MAARTEN: *Keyword Extraction with BERT*, October 2020.

- Hor18. HOREV, RANI: *BERT Explained: State of the art language model for NLP*, November 2018.
- Jab18. JABEEN, HAFSA: *Stemming and Lemmatization in Python*. <https://www.datacamp.com/tutorial/stemming-lemmatization-python>, Oct 2018.
- Joy21. JOYCE, KARA E.: *Graph database vs. relational database: Key differences*. <https://bit.ly/3WumI4a>, Jan 2021.
- Kip18. KIPP, CHRISTOPHER: *Einstieg in Natural Language Processing – Teil 2: Preprocessing von Rohtext mit Python*. <https://bit.ly/3p6TlW0>, Oct 2018.
- KKKS22. KHURANA, DIKSHA, ADITYA KOLI, KIRAN KHATTER und SUKHDEV SINGH: *Natural language processing: State of the art, current trends and challenges*. Multimedia Tools and Applications, Seiten 1–32, 2022.
- KNLG15. KEYZER, MICHAEL DE, CHRISTOPHE COLAS NIKOLAOS LOUTAS und STIJN GOEDERTIER: *Einführung in RDF und SPARQL*, Aug 2015.
- Kop19. KOPP, OLAF: *Google Knowledge Graph einfach erklärt: Definition und FAQ*. <https://www.sem-deutschland.de/seo-glossar/knowledge-graph/>, Sep 2019.
- lem. <https://www.retresco.de/ressourcen/lexikon/lexikoneintrag/lemma>.
- Lid01. LIDDY, ELIZABETH D: *Natural language processing*. SCHOOL OF INFORMATION STUDIES - FACULTY SCHOLARSHIP, 2001.
- Mal17. MALLAWAARACHCHI, VIJINI: *PORTER stemming algorithm – BASIC INTRO*. <https://vijinimallawaarachchi.com/2017/05/09/porter-stemming-algorithm/>, May 2017.
- mat. <https://docs.python.org/3/library/math.html>.
- Mis21. MISHRA, PRAKHAR: *KeyBERT: Keyword Extraction using BERT*, December 2021.
- Neoa. *Getting Started with Cypher*.
- Neob. *Drivers & Language Guides*.
- Onta. *What Are Linked Data and Linked Open Data?* <https://bit.ly/3wnvGoZ>.
- Ontb. *What is SPARQL?* <https://bit.ly/3j1ctWk>.
- os. <https://docs.python.org/3/library/os.html>.

- Pai22. PAI, ARAVINDPAI: *What is Tokenization in NLP? Here's All You Need To Know.* <https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/>, Jun 2022.
- Pal00. PALMER, DAVID D: *Tokenisation and sentence segmentation.* Handbook of natural language processing, Seiten 11–35, 2000.
- pip. <https://pypi.org>.
- Pro. *What is a Graph Database?*
- PS08. PRUD'HOMMEAUX, ERIC und ANDY SEABORNE: *SPARQL Query Language for RDF.* <https://www.w3.org/TR/rdf-sparql-query/>, Jan 2008.
- RDF. *LINKED DATA.* <https://www.w3.org/standards/semanticweb/data>.
- RDF14. *RDF 1.1 Concepts and Abstract Syntax.* <https://www.w3.org/TR/rdf11-concepts/>, Feb 2014.
- re. <https://docs.python.org/3/library/re.html>.
- RMdR<sup>+</sup>20. REINANDA, RIDHO, EDGAR MEIJ, MAARTEN DE RIJKE et al.: *Knowledge graphs: An information retrieval perspective.* Foundations and Trends® in Information Retrieval, xx(xx):1–153, 2020.
- San20. SANAGAPATI, PAVAN: *Knowledge Graph and NLP Tutorial-(BERT,spaCy,NLTK).* <https://bit.ly/3CvAkVI>, 2020.
- Sch20. SCHRADER, BESS: *What's the Difference Between an Ontology and a Knowledge Graph?* Technischer Bericht, Enterprise Knowledge, Jan 2020.
- Sha20. SHARMA, ABHISHEK: *How Part-of-Speech Tag, Dependency and Constituency Parsing Aid In Understanding Text Data?* <https://bit.ly/3Acy4jy>, Jul 2020.
- Sin18. SINGH, SONIT: *Natural language processing for information extraction.* arXiv preprint arXiv:1807.02383, 2018.
- Tut. *Part of Speech (PoS) Tagging.* <https://bit.ly/3W8cVkv>.
- V21. V, NITHYASHREE: *What is Chunking in Natural Language processing?* <https://bit.ly/3wmkaKE>, Oct 2021.
- Vet22. VETTRIVEL, VISHNU: *Knowledge Graphs: RDF or Property Graphs, Which One Should You Pick?* <https://bit.ly/3FBkJUQ>, Aug 2022.
- Way21. WAYNER, PETER: *What are graph database query languages?,* Sep 2021.
- Yea. <https://yearofthegraph.xyz/graph-database-report/>.

- Zou20. ZOU, XIAOHAN: *A Survey on Application of Knowledge Graph*. Journal of Physics: Conference Series, 1487(1):012016, mar 2020.

# A

---

## Glossar

NLP	Natural Language Processing
NLTK	Natural Language Toolkit
W3C	World-Wide Web Consortium
RDF	Resource Description Framework
OWL	Web Ontology Language
XML	Extensible Markup Language
URI	Uniform Resource Identifier
IRI	Internationalized Resource Identifier
NLU	Natural Language Understanding
NLG	Natural Language Generation
LOD	Linked Open Data
SPARQL	SPARQL Protocol And RDF Query Language
IE	Informationsextraktion
POS-Tagging	Part of Speech Tagging
MUC	Message Understanding Conference

# B

---

## Anhang

### B.1 Stopword Liste

'his', 'does', 'was', 'mighthn', "mustn't", 'having', 'then', 'needn',  
'other', 'from', "hasn't", 'your', 'and', 'for', 'fig', 'it', 'thought',  
'et', 'whom', 't', 'my', 'is', 'most', 'been', "weren't", 'am', 'll',  
ßhan't", 'herself', 'between', 'yourselves', 'our', 'in', 'all', 'yet',  
'while', 'he', 'they', "hadn't", ßhe's", 'very', 'you', its', 'the',  
'her', 's', 'too', 'de', 'were', 'because', 'com', 'himself', 'she',  
'own', 'yourself', 'yours', 'should', 'into', 'wasn', 'term', 'agreement',  
'd', 'over', "you're", it's", 're', 've', 'eu', 'open', 'shouldn', "you'll",  
'such', 'al', 'what', 'to', 'above', 'now', 'but', 'some', "wouldn't",  
'will', 'than', 'o', 'shan', 'won', 'has', 'did', 'under', "didn't",  
'open access', 'table', 'down', 'i', 'are', 'mustn', 'of', 'with', 'didn',  
"doesn't", 'their', 'ours', 'here', 'only', 'ourselves', "mighthn't",  
'hers', 'once', 'm', 'more', 'themselves', 'ieee access', isn't", 'nor',  
'aren', "won't", 'how', 'don', 'me', 'about', 'do', 'being', "wasn't",  
'up', 'again', 'access', 'citiations', 'sees', 'had', 'we', 'www', "haven't",  
'before', "you've", 'same', 'any', 'y', 'who', 'there', 'against', 'few',  
'hadn', 'below', 'a', 'people', 'haven', 'citations', 'which', 'this',  
'out', 'no', 'doesn', 'ieee', 'just', "don't", 'http', 'authors', 'itself',  
'why', 'that', 'doing', 'further', 'each', 'hasn', 'can', "couldn't",  
'through', 'initially', 'these', 'off', 'where', 'them', 'th', 'theirs',  
'org', 'not', ßshould've", 'those', "needn't", ßshouldn't", 'isn', 'until',  
'an', 'or', 'couldn', "you'd", 'when', 'so', 'after', 'if', 'aaai', 'myself',  
'on', 'wouldn', 'both', 'by', 'weren', 'conference', 'have', 'be', 'as',  
'at', "that'll", 'during', ären't", 'him', 'ma', 'ain'

### B.2 Nutzbare Transformer und Modelle für KeyBert

- Sentence Transformer
- Hugging Face

- Flair
- Spacy
- Universal Sentence Encoder (USE)
- Gensim

### B.3 Pakete, Module, Frameworks

Modul	Version	Zusammenfassung	Doku
PyPDF2	2.11.1	Python Bibliothek zum aufteilen, zusammenfügen, kürzen und transformieren von PDF Dateien	<a href="https://pypdf2.readthedocs.io/en/latest/">https://pypdf2.readthedocs.io/en/latest/</a>
PyMuPDF (fitz)	1.20.2	PDF toolkit	<a href="https://github.com/pymupdf/PyMuPDF">https://github.com/pymupdf/PyMuPDF</a>
pathlib	1.0.1	Objekt orientierte Datei-System Pfade	<a href="https://pathlib.readthedocs.org/">https://pathlib.readthedocs.org/</a>
pandas	1.5.1	Datenstrukturen für Datenanalyse Zeitreihen und Statistiken	<a href="https://pandas.pydata.org">https://pandas.pydata.org</a>
numpy	1.23.4	Integration von Arrays	<a href="https://www.numpy.org">https://www.numpy.org</a>
spacy	3.4.2	Natural Language Processing (NLP) in Python	<a href="https://spacy.io">https://spacy.io</a>
nltk	3.7	Natural Language Toolkit	<a href="https://www.nltk.org/">https://www.nltk.org/</a>
KeyBert	0.7.0	keyword Extraktion mittels state-of-the-art Transformern	<a href="https://github.com/MaartenGr/keyBERT">https://github.com/MaartenGr/keyBERT</a>
keyphrase vectorizers	0.0.10	Vectorizer Sammlung zur Extraktion von wichtigen Sätzen mittels part-of-speech patterns von einer Sammlung an Textdokumenten und deren Konvertierung zu einer Matrix	<a href="https://github.com/TimSchopf/KeyphraseVectorizers">https://github.com/TimSchopf/KeyphraseVectorizers</a>
networkx	2.8.8	Python package zur Erstellung und Manipulation von Graphen und Netzwerken	<a href="https://networkx.org/">https://networkx.org/</a>
Sentence Transformer	2.2.2	Multilinguale Texteinbettungen	<a href="https://www.sbert.net">https://www.sbert.net</a>
matplotlib	3.6.2	Python plotting package	<a href="https://matplotlib.org">https://matplotlib.org</a>

**Tabelle B.1.** Auflistung aller verwendeten Bibliotheken, Packages und Modulen mit Versionen und Beschreibungen [pip] Installation über pip: <https://pypi.org>

Modul	Version	Zusammenfassung
os	3.11.0	stellt allgemeine Betriebssystemfunktionalität zur Verfügung[os]
re	3.11.0	stellt Operationen mittels reguläre Ausdrücke zur Verfügung[re]
math	3.11.0	stellt mathematische Funktionen zur Verfügung[mat]

**Tabelle B.2.** Auflistung aller verwendeten in-build python Module