Universität Rostock Traditio et Innovatio

SkenData

Masterarbeit

Extraktion und Integration heterogener unstrukturierter und semistrukturierter Daten für die Tarifierung von gewerblichen Gebäudeversicherungen

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

---

vorgelegt von:	Sina Ghashghaie
Matrikelnummer:	219200010
geboren am:	10.05.1989 in Teheran
Erstgutachter:	Prof. Meike Klettke
Zweitgutachter:	Dr. Holger Meyer
Betreuer an Universität:	M.Sc. Hannes Grunert
Betreuer im Unternehmen:	Dipl.-Ing. Mathias Perlet
Abgabedatum:	18. Oktober 2021

https://doi.org/10.18453/rosdok_id00003362

Abstract

In dieser Masterarbeit geht es um die dynamische Information-Extraktion (IE) und Integration heterogener unstrukturierter und semistrukturierte Daten für die Tarifierung der gewerblichen Gebäude-Versicherungen bei der *SkenData GmbH*. Integriert werden die Daten in einer strukturierten JavaScript Object Notation (JSON)-Datei. Der Leser erhält damit auch einen Überblick über verwandte Arbeiten und die Herausforderungen bei der Extraktion der Informationen aus semistrukturierten Daten bzw. Datenquellen und damit speziell aus deutschsprachigen gewerblichen Texten.

Schlagwörter: Information Retrieval, Informationsintegration, Informationsextraktion, Web Crawling, Natural Language Processing

Inhaltsverzeichnis

Quellcodeausschnitt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1 Einleitung

Die SkenData GmbH[1] wurde im Jahr 2014 von Sven Jantzen und Jon Meis in Rostock gegründet, und ist als IT Dienstleister mit dem Fokus der digitalen Gebäudewertermittlung für die Versicherungswirtschaft tätig. Sie ist ein Insurtech (Versicherung und Technologie) und hat sich vom Startup zum Marktführer entwickelt.
Die *SkenData* ermöglicht die Gebäudewertermittlung einfach, schnell und digital für Versicherer. Viele Faktoren wie *Gebiet, Alter, Lage* usw. können für die Ermittlung des Gebäudewertes berücksichtigt werden, die alle aus verschiedene Datenquellen kommen, und in ein System zusammengeführt werden müssen. Darüber hinaus werden alle diese Dateien und Informationen über die “Wert14”-App[2] von SkenData automatisiert, damit man ganz nutzerfreundlich durch 3D-Gebäudemodellierung den Gebäude-Report anschauen bzw. erhalten kann. Im Bereich Gebäudewertermittlung kommt man somit nicht an der SkenData und damit der Wert14-App vorbei.

1.1 Motivation

Die digitale Gebäudewertermittlung von SkenData arbeitet stufenweise mit einer Automatik, schnellen sowie detaillierten Benutzereingaben. Insbesondere durch die Automatik soll das Konfidenzniveau der Werte und die Vorauswahl der Benutzereingaben verbessert werden.
Für die Automatik soll zukünftig neben der Adresse als Initialisierung der Gebäudewertermittlung auch ein Unternehmensname als Startpunkt möglich sein. Durch die Gewinnung von strukturierten Attributen aus heterogenen und unstrukturierten Datenquellen soll der Prozess beschleunigt und die Benutzereingaben reduziert werden. Im Rahmen dieser Masterarbeit soll ein Konzept zur Extraktion von Informationen aus unstrukturierten und semistrukturierten Datenmodellen in ein strukturiertes Datenmodell entwickelt werden. Der Fokus liegt dabei auf dem Integrationsprozess, über den heterogene Daten in eine einheitliche Struktur überführt werden sollen.
Zusammenfassend gibt der Nutzer den Namen des Unternehmens, wie *SkenData GmbH*, ein, und als Ausgabe erscheint:

[1] `https://www.skendata.de/` aufgerufen am 01.08.2021
[2] `https://www.wert14.de/` aufgerufen am 01.08.2021

```
    "Company": {
              "name": "SkenData GmbH",
              "branch":"Softwareentwickler in Rostock",
              "boss_name": "Jon Meis, Sven Jantzen",
              "address": "Muehlendamm 8B, 18055 Rostock",
              "tell_number" : "+49 (0) 381-2524 9294",
              "fax_number" : "",
              "email_address": "info@skendata.de",
              "num_of_employee": "",
              "ust_ID_Nr": "DE295446443",
              "website":"https://www.skendata.de/",
              }
```

Quellcodeausschnitt 1.1: Ergebnis vom Programm in einem strukturierten JSON-Datei

1.2 Problemstellung

Das Erreichen der unten genannten Faktoren dürfen ständig problematisch werden, deswegen hat es uns motiviert, dieses System zu entwickeln.

- Weniger Kosten, da meistens die Verwendung der Application Programming Interface (API) kostenpflichtig ist.
- Aktualisierten Informationen zu finden.
- Zugriff auf allen beliebigen Informationen zu haben.
- Gegebenenfalls schnelle Extraktion anderer Informationen ermöglicht werden.
- Beliebige Weiterentwicklung der eigenen Software zu ermöglichen.
- Weniger Zeit zum Erreichen der Daten zu ermöglichen.

Parallel zu dieser schriftlichen Arbeit wird ein Web-IR-Werkzeug in Zusammenarbeit mit einem IE-System implementiert. Das besteht aus ein paar Crawlern, die drei Informationen bzw. Attribute wie den komplette Namen, Branche, und Links der Webseiten von Unternehmen bzw. Geschäften aus zwei Datenquellen bzw. Suchmaschinen, *"Google"*[3] sowie *"Gelbe-Seiten"*[4], indexieren. Dann wird im zweiten

[3] `https://www.google.com/` aufgerufen am 20.08.2021
[4] `https://www.gelbeseiten.de/` aufgerufen am 20.08.2021

Schritt ein Crawler umgesetzt, damit die Texte von Homepage-, Über-uns- sowie der Impressums-Seite aus der Geschäfts-Website gesammelt werden. Im nächsten Schritt kommt ein Text-Mining-Algorithmus zum Einsatz, um die relevanten Informationen aus den gesammelten, unstrukturierten Texten zu extrahieren. Danach werden die Informationen letztendlich in einer strukturierten JSON-Datei, wie oben gezeigt, integriert.
Die unten genannten Punkte können bei dieser Arbeit als die allgemeinen Herausforderungen des Projektes betrachtet werden:

- Verschiedene Suchmaschinen-Architekturen bzw. Strukturen
- Unterschiedliche Strukturen der beliebig implementierten Webseiten
- IE aus deutschsprachigen, unstrukturierten Dokumenten bzw. Texten

1.3 Gliederung der Arbeit

Im Kapitel 2 ‚Stand der Technik' wird über IR bzw. Web IR-, IE- Systemen sowie auch Metriken zur Evaluation des Systems geschrieben. Im Abschnitt 2.1 ‚Web Information Retrieval' wird über die verwandten Arbeiten in Bezug auf Web IR bzw. Web-Crawler geschrieben. Abschnitt 2.2 ‚Information Extraktion' beschreibt, welche verwandte Arbeiten bei der Extraktion der Informationen aus unstrukturierten Daten erledigt wurden. Im Kontext der Integration der Informationen wird in Abschnitt 2.3 ‚Informationsintegration' beschrieben.
Im Kapitel 3 ‚Konzept' wird beschrieben, welches Konzept für die parallele Umsetzung des Projektes geplant ist. in den Unterpunkten dieses Abschnitts wird geklärt und durch Diagramme gezeigt, wie die Integration der festgelegten Informationen durch das Projekt erfolgt.
Kapitel 4 ‚Implementierung' zeigt und beschreibt ausführlich, wie die Implementierung durchgeführt wurde, und wie die Einzelheiten der Implementierung funktionieren. In diesem Abschnitt wird auch erläutert, wie die Evaluation der Arbeit erfolgte.
Kapitel 5 ‚Fazit und Ausblick' beschreibt, in welchem Umfang die Problemstellung gelöst wurde und welche möglichen Ansätze für Untersuchungen in künftigen Arbeiten hieraus resultieren.

2 Stand der Technik

In diesem Kapitel wird auf die grundlegenden sowie verwendeten Verfahren in den Bereichen des IR (2.1) und IE (2.2) eingegangen. Anschließend werden mehrere nahezu identische Forschungsarbeiten im Gebiet Informationsintegration (2.3) betrachtet.

2.1 Information Retrieval

Wie es im Buch "*Modern IR*" [5] erklärt wurde, gehören die Sammlung sowie Repräsentation der Dokumente bzw. Webseiten zu IR. IR liegt im Fachbereich Informatik und Computerlinguistik, damit die bestehenden Inhalte von Informationen in Form von Textphrasen sowie Bilder gesucht werden können. Laut Stefano Ceri et al. [6] in ihrem Buch mit dem Namen "*Web IR*" werden die relevanten Dokumente anhand des lexikalischen Patterns von Nutzer-Abfragen gefunden und indexiert. Das wurde hier als einen Aspekt von IR erwähnt, dass die relevanten Ergebnisse bezüglich des Informationsbedarfs und nicht auf die Anfrage bewertet werden [5, 6]. Baeza-Yates et al. [5] meinen, dass nach der Nutzer-Abfrage zum ersten Schritt ein IR System die bestehenden Informationen aus einer Kollektion oder durch Crawling im Internet bzw. Web sammelt. Nach der Speicherung der Dokumente in einem Repository, werden die zum Retrieval und Ranking indexiert. Zur Indexierung wird erwähnt, dass eine Invertierte Liste Infrage kommen kann und damit ein gutes Retrieval schnell funktioniert [5]. Invertierte Listen wurden so genannt, weil die Liste der Dokumente ist, die den Term bzw. die Worte enthalten [7]. Wie es im Abbildung 2.1 zu sehen ist, wird Retrieval und Ranking nach der Indexierung durchgeführt [5]. Ranking wird als eine Funktion auf Menge der Ergebnisse genannt, die eine nach Relevanz geordnete Liste zurückgibt [8].

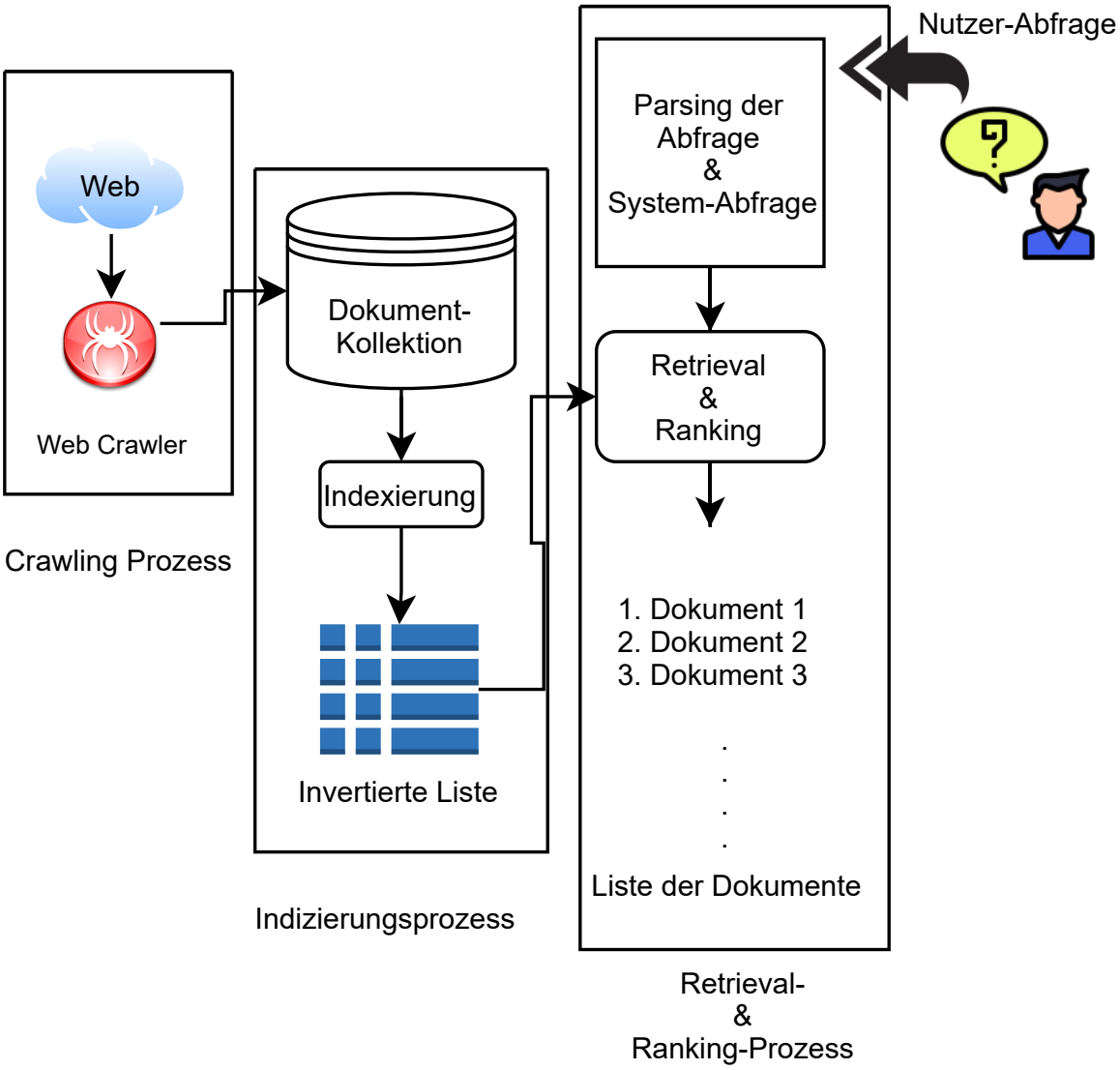

Abbildung 2.1: IR-System-Architektur nach [5]

Für die Bewertung von IR-Systemen wurden *"Precision"* und *"Recall"* zur *"Relevanz-"* und *"Nachweis-Quote"* neben anderen Bewertungskriterien erwähnt [9]. Darüber hinaus spielt die Bewertung der IR Systemen anhand ihrer Effektivität eine große Rolle. Das bedeutet, dass neben den gefundenen auch die nicht gefundenen Dokumente wertvoll sind, und das Ziel ist die Optimierung der Suche. Um genauer zu sein, wenn eine Suchmaschine ein Ergebnis liefert, sollen gefundene relevante Dokumente nicht wenig sein [9, 10]. Im Folgenden wurde auch erwähnt, dass die Recall und Precision sowie F1-Score in folgende Formel gerechnet werden [9, 10]. F1-Score wurde als eine Mischung vom Precision- und Recall-Formel, wie

unten nach [9, 10], geklärt.

A = gefundene, relevante Dokumente
B = nicht gefundene, relevante Dokumente
C = gefundene, nicht relevante Dokumente

$$Recall = \frac{\text{A}}{\text{A} + \mathbf{B}}$$

$$Precision = \frac{\text{A}}{\text{A} + \mathbf{C}}$$

$$F_1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

Als ein Beispiel Anhand der Definition in Quellen für Precision und Recall kann man diese so erklären, dass beispielsweise 200 Daten in einem System stehen. Wenn die Abfrage eingegeben ist, werden 70 Items als Ergebnisse gefunden, dass nur 50 Items von diesen gefundenen Items die relevanten Ergebnisse sind [10]. Auch stehen noch 30 Items, die nicht gefunden sind, obwohl die relevante Items waren. Dann kann man so rechnen, dass 62% der Daten als Recall, und 71% als Precision festgelegt sind [10].

2.1.1 Web-Suche und Hyperlinks

Wie es im Buch Modern IR beschrieben wurde, ist die Web-Suche einer der wichtigsten bzw. bekanntesten Anwendungen von einem IR System und die Komponenten von einer Suchmaschine beinhaltet diese IR Innovationen [5]. In diesen stehen viele Webseiten, welche durch “*Hyperlinks*” gefunden, bewertet und repräsentiert werden [5]. Hyperlinks sind die kurzen und oft blau unterstrichenen Verlinkungen, welche durch die Verknüpfung im Internet mit anderen Links, die Funktionen auf einer Seite durchführen. Dadurch wird auf einem an anderer Stelle auf dem Netzwerkserver im Intranet oder Internet verwiesen [5, 6]. Infolge der Hyperlink Definition wurde in “*Seo-Nerd*” [11] über “*Absolute*” und “*Relative*” Pfade wie unten gezeigt, sodass jede Formen von Pfaden seine Nachteile und Vorteile hat [11].
Laut Alexander Shen [12] ist der effizienteste Algorithmus, der für das Web-Crawling verwendet wird, der “*Depth First Search*” [12]. Das ist ein Algorithmus zum Verwenden, um die Suche zu durchlaufen, indem er auf der Basisseite beginnt

und jede gefundene Hyperlink-Seite tiefer durchläuft. Es wird dann zurückverfolgt und zu den Hyperlink-Seiten verschoben [12].

Tabelle 2.1: Beispiel für Relative und Absolute Pfade
nach [11, 13]

	Pfade
Relative	Impressum
Absolute	Impressum
Relative	Über uns
Absolute	Über uns

Der angezeigte "*relativ*"-link durch 2.1 in einem Tag vom Hypertext Markup Language (HTML) Element verweist auf Impressums-Seite auf Website [11, 13].

2.1.2 Suchmaschinen

Nach Erklärung im Modern IR stehen vielzahl der Daten wie Bilder, Audio, Video und Text im Web [5]. Suchmaschinen wurden als ein universelles Werkzeug bzw. IR System bei der Nutzung des World Wide Web (WWW) genannt [14]. Als eine breite Informationsquelle ermöglichen sie einfachen Zugang für Web-Benutzer, die Informationen für jeden Zweck suchen, wie z. B. Beruf und Privat. Allerdings zeigen Suchmaschinen eine riesige Menge an Informationen, indem eine kritische Bewertung von Informationen in Bezug auf Relevanz und Vertrauenswürdigkeit von Informationen ist entscheidend, da eine falsche von Informationen zu Fehlentscheidungen und schwerwiegenden Konsequenzen führen kann. Web-Benutzer definieren eine Such-Anfrage, formulieren eine Abfrage und geben diese in eine ausgewählte Suchmaschine, wie z. B. Google, Bing, Yahoo oder anderen Arten von Suchmaschinen ein [14, 5].

Laut "*Statcounter*" [15] ist Google mit über 80% die meist genutzte Suchmaschine in Deutschland, sowie der Welt genannt, und wird als Recherche-Funktion im Internet genutzt [15].

2.1.3 Arten sowie Funktionen der Suchmaschinen

Anhand Modern IR gibt es viele verschiedene Arten neben oben genannten Suchmaschinen, die mit verschiedene Ergebnis-Ansichten durch verschiedene Methoden angezeigt werden [5]. Eine Suchmaschine wurde als ein Computersystem genannt, welche die Inhalte aus dem Web durch Crawling erfasst und über eine Benutzer-Schnittstelle durchsuchbar macht, sodass die Ergebnisse in einer nach oben genannten systemseitig angenommenen Relevanz geordneten Darstellung durchgeführt werden [5, 16]. Im folgende werden zwei Typen von Suchmaschinen genannt, anhand [5, 16, 17]:

- ***"Indexbasierte Suchmaschinen"***
  Hierbei wurde beschrieben, dass durch die Suchmaschine mit Hilfe von Crawlern eine große Anzahl von WWW-Dokumenten automatisch eingelesen werden, algorithmisch analysiert und anschließend ein Suchindex erstellt, der bei nachfolgenden Suchanfragen kontaktiert wird. Die berühmtesten Beispiele für Index-basierte Suchmaschinen sind Google, Bing und Yahoo.

- ***"Katalog-basierte Suchmaschinen"***
  Es wurde so erklärt, dass ein Katalog Such-Ergebnisse enthält, die vorher von Personen zusammengestellt, sortiert und bei Bedarf auch manuell gewichtet wurden. In der Regel basiert ein Katalog auf einer alphabetischen oder nach thematischen Kriterien geordneten Liste. Beispiele hierfür sind offene Verzeichnisse und Kindersuchseiten (wie gelbeseiten.de [18]).

"Gelbe-Seiten" ermöglicht durch Zusammenarbeit mit seinen Kooperationspartnern umfangreiche Online-Services für die Suche nach Unternehmen und Adressen. Außerdem zeigt verschiedene Informationen über getroffenes Unternehmen wie Branchen, Bewertungen usw [18].

Knowledge Graph und Suche-API

Wie es im Artikel "*What Is a Knowledge Graph?*" [19] beschrieben wurde, wurde in den letzten Jahren "*Knowledge-Graph*" [20], auch "*Knowledge Panels*" [19] genannt, im Bereich Künstliche Intelligenz (KI) entwickelt. Infolge wurde es beschrieben, dass Google die erste Suchmaschine war, die sogenannte "*Google Knowledge Graph*" [19] in seinem System entwickelt hat [19].
Es wurde von Uyar et al. beschrieben, dass bei andere Desktop-Suchmaschine wie zum Beispiel Bing Knowledge-Graph mit dem sogenannten Name "*Bing Satori*" [21] eingesetzt wurde, die in Bezug auf unserem Kontext Google-Knowledge-Graph wegen der Popularität von Google(sehe Abbildung 2.2) Suchmaschine für den Ansatz entschieden wurde.

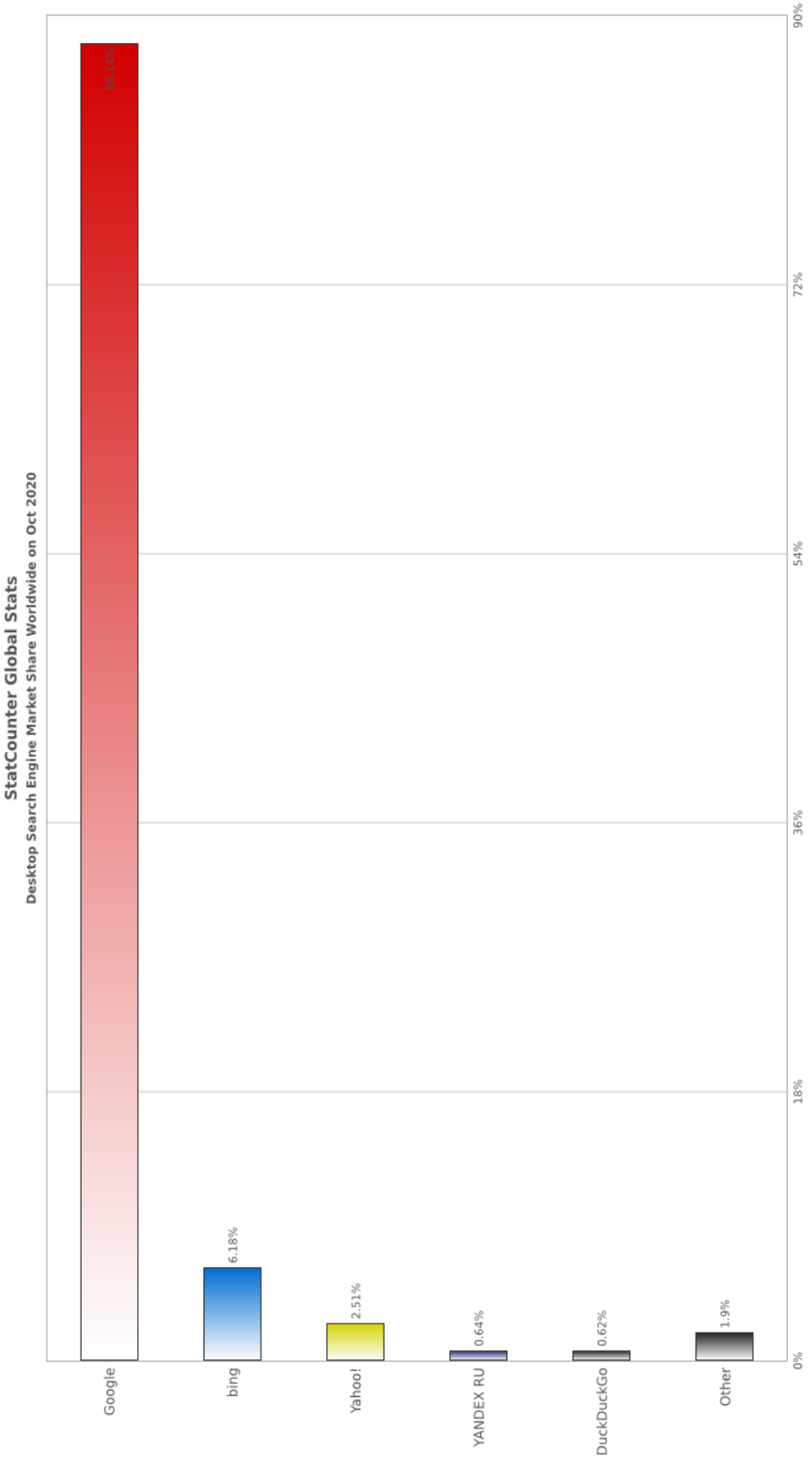

Abbildung 2.2: Weltweite Anzahl der Suchmaschinen-Anfragen nach[1][15]

[1] Dank an "statcounter" für die Herunterladen-Funktion des Graphen, , hochgeladen am 10.08.2021

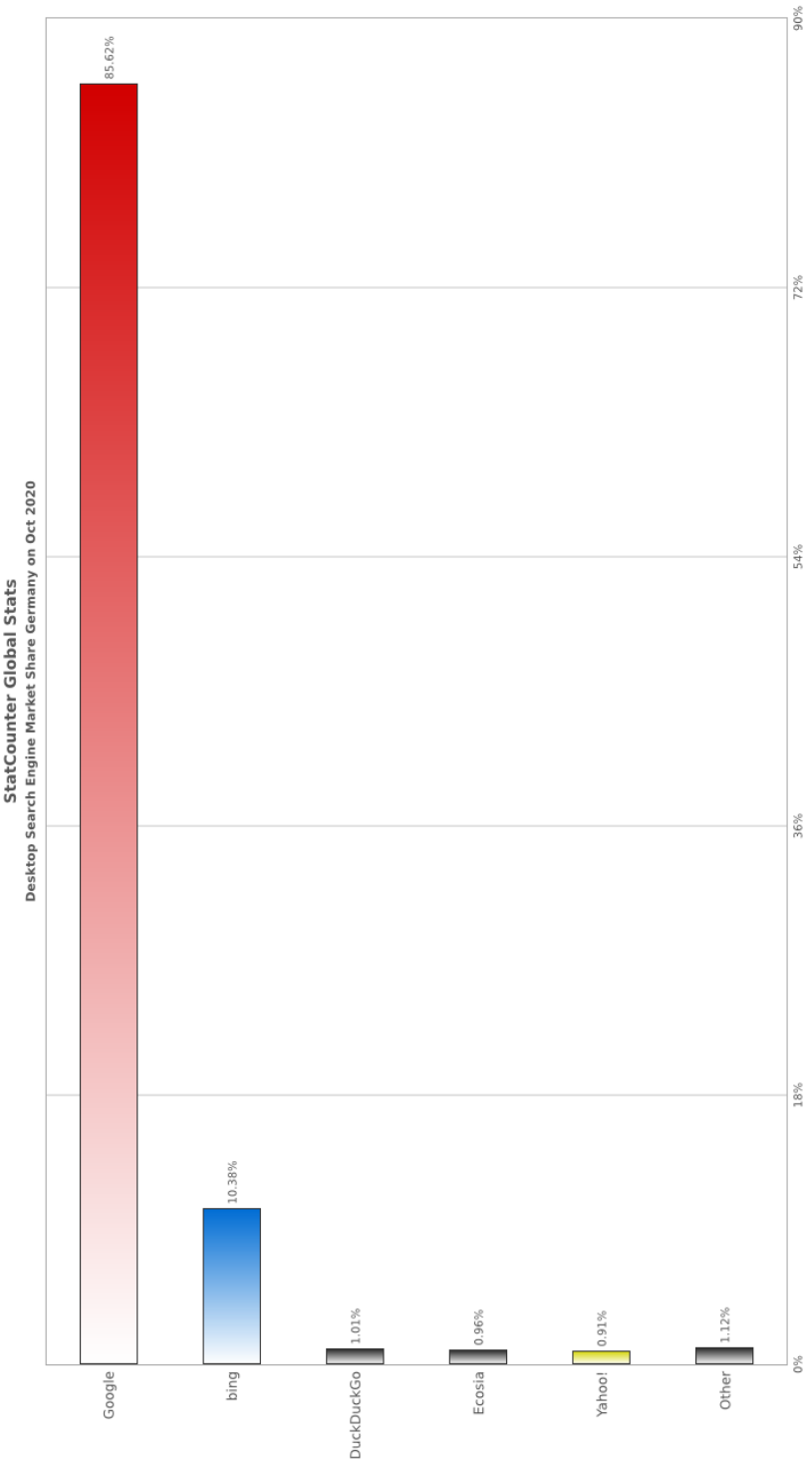

Abbildung 2.3: In Deutschland Anzahl der Suchmaschinen-Anfragen nach[2][15]

[2] Dank an "Statcounter" für die Herunterladen-Funktion des Graphen, hochgeladen am 10.08.2021

Google bietet aktuell einige “*Google Knowledge Graph Suche API’s*” an, womit man durch die Verbindung mit dem API in der Lage ist, die Entitiäten-Ergebnis von Knowledge Graph über zum Beispiel Lokal-Business, Organisation, Ort usw. wie unten in der Abbildung 2.4 kostenlos und bei einigen Entitiäten gegen Geld in einem strukturiertem JSON Datei zu erhalten (In unserem Kontext ist Bezahlung für einige Entitiäten unabdingbar wie Webseite-Uniform Resource Locator (URL) der Geschäfte) [22].

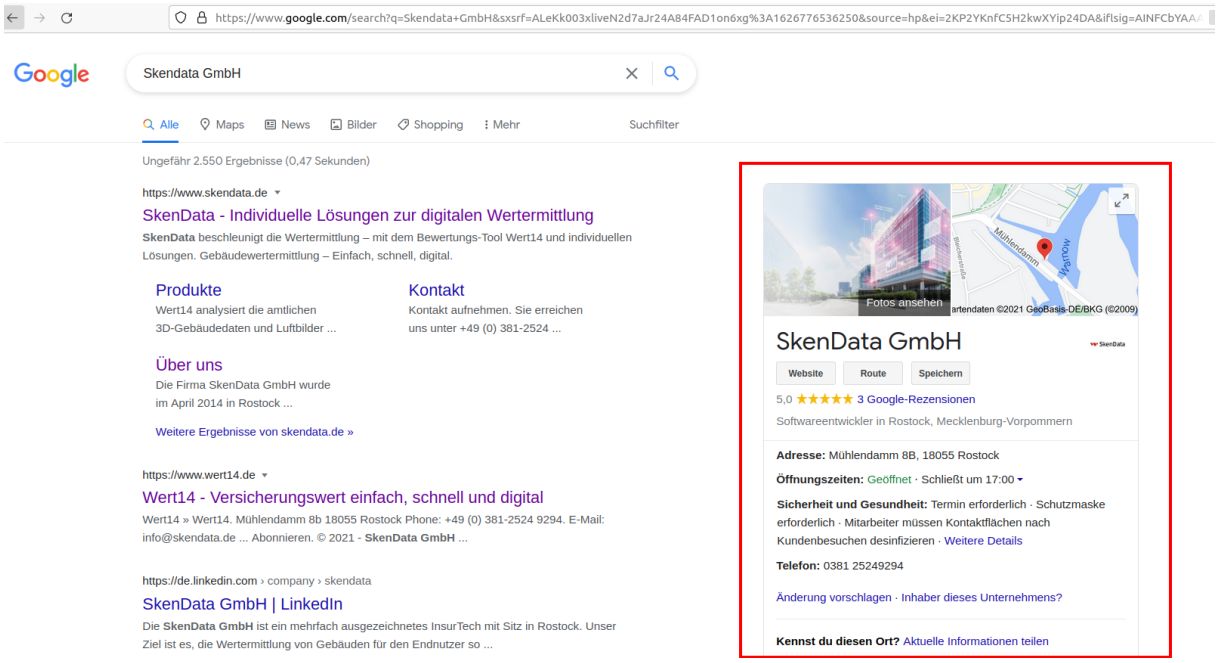

Abbildung 2.4: SERP beim Google Knowledge Graph[3]
Rot umrandet ist das Ergebnis durch die Eingabe *Skendata GmbH*

Wenn man nach lokalen Unternehmen suchen würde, werden Name, Telefonnummer, Kategorie, Stadt, Bundesland, Postleitzahl usw. auch durch Verbindung mit Bing API angezeigt, wobei hier auch für einige Entitäten bezahlt werden soll [23].

SERP bei Lokal-Suche

Wie es auf Unterstützung-Seite von Google beschrieben wurde, zeigt Google normalerweise die drei ersten Lokal-Empfehlungen, die Nah an der Nutzerabfrage sind, sodass diese Ergebnisse dann angezeigt werden, wenn die Nutzerabfrage nicht genau zu Dokument im System passt. Dafür wurde auch auf Support-Seite erzählt,

[3]Mit Respekt auf Google für Bildschirmfoto, aufgerufen am 28.09.2021

dass diese Entwicklung durch Kombination aus drei Algorithmen wie “*Relevance*”, “*Distance*” sowie “*Prominence*” ausgeführt wird [24]. Das SERP von Local-suche zeigt sich wie in Abbildung 2.5.

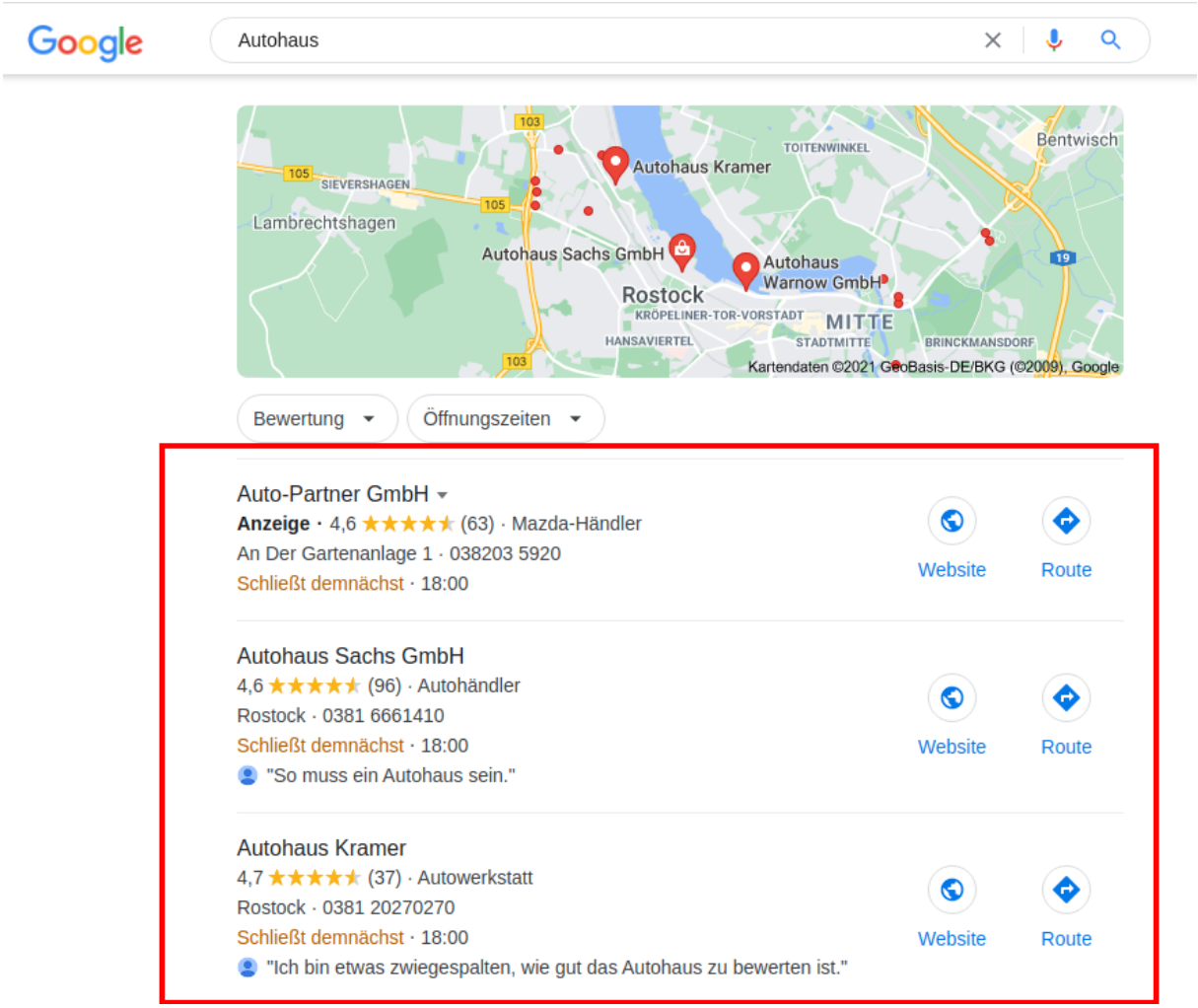

Abbildung 2.5: SERP bei Google-Lokal-Suche[4]
Rot umrandet ist das Ergebnis durch die Eingabe *Autohaus*

Mit anderen Worten, wenn man in unserem Kontext nach einem Unternehmen sucht, erhält man als Ergebnis bei der eingegebenen Abfrage drei Formen in SERP, anhand [5].

- “*satisfied*” Bei diesem Ergebnis sieht man ein Knowledge Panel für das Unternehmen bzw. Geschäft

[4]Mit Respekt auf Google für Bildschirmfoto, aufgerufen am 28.09.2021

- "*partially satisfied*" Dadurch erhält der Nutzer normalerweise erste drei Objekte bei der Lokal-Suche
- "*not satisfied*" Da in diesem Fall die von Nutzer eingegebene Abfrage nicht richtig formuliert wurde, erhält er keine oben genannte Ergebnisse als Relevantes Ergebnis

2.1.4 Web Crawling

Wie es durch Abbildung 2.1 anhand Modern IR Buch gezeigt wurde, wurde Crawling als einen Teil der Sammlung, Indexierung sowie Repräsentation der Informationen genannt [5]. In unserem Kontext spielt der Crawler, und auch manchmal "*Spider*" sowie "*Robot*" genannt [25], als das IR System gerechnet werden kann, da hier einige der bezüglichen Informationen wie kompletter Name des Unternehmens, Branche sowie Website-URL aus Suchmaschinen indexiert und gesammelt werden. Außerdem kommt bei uns der Crawler zum Einsatz, wie beschrieben in "*Web Crawling*"-Buch [25], zum Herunterladen der Webseiten, um andere benötige Attribute aus der Webseiten-Texte zu extrahieren, wie in 3 geklärt wird, und auch im Modern IR Buch beschrieben wurde [5].
Laut "*Ricardo Baeza et al.*" [5] der Kern von einem Web-Crawler zur Indexierung der Inhalte einer Web-Seite genannt wurde [5]. Aus dem Begriff Web-Crawling kann man anhand Modern IR Buch die Verschiedene Type bzw. Arten von Webseiten erwähnen. Im Buch wurde beschrieben, dass bei einem Crawler mindestens zwei Dimensionen von Webseiten als "*Privat/ Publik*" sowie "*Statik/ Dynamik*" gibt. Als Private Webseiten wurde eine Seite genannt, die man zum Einloggen Benutzer-Name und Passwort benötigt, wie Soziale Netzwerke, die in diesem Fall nicht einfach indexiert und gecrawlt werden. Aber bei öffentlichen Websites ist es beschrieben, dass diese indexiert werden können. Auf der anderen Seite stehen Statik Webseiten, die zum Crawlen auf einem Request warten, aber bei den dynamischen Websites kann, das prinzipiell nicht existieren, kann man nicht zur Indexierung auf die Inhalte der Seite mit bloß einem Request zugreifen. Als Beispiel wurde erwähnt, dass die Suchmaschine auch dynamisch implementiert wurde [5].
Wie es im Artikel "*Information retrieval in web crawling: A survey*" beschrieben wurde, unterteilt sich das IR System auf zwei Arten als "*Traditional IR*" sowie "*Automated IR*" [26]. Das Web Crawling wurde als einen Teil von "*Automated IR*" genannt. Das wurde auch geklärt, dass die Beziehungen zwischen den Dokumenten in traditionelles System nicht klar dargestellt werden, da das traditionelles IR spezifische Informationen für die Suche repräsentiert, darüber hinaus hat keine Kenntnisse von anderen Daten oder deren Existenz [26]. Als traditionelles IR Modell wurde folgende Punkte laut "*Chandni Saini und Vinay Arora*" genannt [26]:

- "***Boolisches Retrieval***"
  Es wurde erklärt, dass dieses Modell anhand der Booleschen Algebra funktioniert [26]

- "***Vektor-Raum Modell***"
  Für Vektor-Raum Modell wurde beschrieben, dass es auf Vektoren und Vektor-Operationen basiert ist, sodass zwischen "*Dokument Vektor*" und "*Anfrage Vektor*" ein Ähnlichkeit-maß definiert wird [26].

- "***Probabilitisches Modell***"
  Dieses Modell wurde beschrieben, dass es auf dem Prinzip des Wahrscheinlichkeitsrankings basiert ist [26].

Von *Chandni Saini und Vinay Arora* wurde auch beschrieben, dass automatisierte Systeme eingesetzt werden um den "*Information Overload*" zu verringern. Und diese Art von IR-System wird auch im Bereich Web Crawling eingesetzt [26].

Web-Crawling-Strategien in IR

Anhand der Beschreibung im "*Information retrieval in web crawling: A survey*", sind die Strategien der IR beim Web Crawling in Folgende Punkte unterteilt [26]:

- "***Focused Web Crawler***"
  "Focused Web Crawler" wurde zur Sammlung der Webseiten vorgeschlagen, die einige Eigenschaften erfüllen, dadurch die Crawler-Grenze priorisiert und das Hyperlink-Entdeckung-Prozess verwaltet wird [26].

- "***distributed Web Crawler***"
  "distributed Web Crawler" wurde als verteilte Computing Software beschrieben, damit durch Web Crawling werden bei der Suchmaschinen mit Hilfe des Internets mehrere Computer zur Indizierung der Inhalte vom Web verwendet [26].

- "***incremental Web Crawler***"
  Der Prozess des erneuten Durchsuchens von URLs wurde als "incremental Web Crawler" bezeichnet [26].

- "***hidden Web Crawler***"
  Das "Deep Web" oder "Hidden Web" bezieht sich nicht auf das Abrufen der Hyperlinks, sondern durch Interaktion der Webdaten mit einer webbasierten Suchformular [26].

Der *Focused Web Crawler* kann als Basis-Crawler für verschiedene Darstellungen verwendet werden. Als Beispiel wurden für den "*keywordbasierten, den auf exemplarischen Dokumenten basierenden, den ontologiebasierten, den linksemantischen , den Data-Mining-basierten Ansätze*" genannt [26].
Infolge der Definitionen kommt bei unserem Kontext der keywordbasierte Ansatz zum Einsatz wird zum Parsen der Informationen aus dem Web verwendet. Im Artikel vom Chandni Saini und Vinay Arora [26] wurde so beschrieben, dass die Extraktion von URLs auf der Basis von Schlüsselwörtern oder Suchkriterien ist. Dadurch werden die URLs von den Webseiten extrahiert und als signifikant betrachte, welche das gesuchte Schlüsselwort in Ihrem Inhalt enthalten. Die anderen werden ignoriert und als irrelevante Dokumente eingestuft [26].

Web Scraping Tools

Anhand der Beschreibung im Buch "*Practical Web Scraping*[5] *for Data Science: Best Practices and Examples with Python*"[13] Webbrowser kommuniziert mit einem Server im WWW. Da wurde es auch geklärt, dass die Kernkomponente im Informationsaustausch ist eine HyperText Transfer Protocol (HTTP) Request Message an einen Webserver, der eine HTTP Response als Antwort zurückgibt [13]. "*Seppe vanden Broucke*" und "*Bart Baesens*" beschrieben, dass ein Client bzw. Browser sendet Anfragen an den Server, und der Server sendet Antworten zurück [13]. Der Header soll im richtigen Format sein, und wenn alles damit in Ordnung wäre, wird der Webserver unsere Anfrage verarbeiten und eine "*HTTP-Antwort*" mit der Nummer "*200*" als "*status result*" sowie "*OK*" als "*status message*" mit anderen Infos, wie in Abbildung 2.6 gezeigt, zurücksenden [13].

[5] Wie es von Sarah Redlich [27] beschrieben wurde, kommt *Web-Scraping* zur Extraktion der Informationen aus WWW im Einsatz. In ihrem Atrikel wurde *Web-Scraping* neben den anderen Begriffen wie *Web-Crawling* sowie *Web-Mining* als Synonym verwendet [27].

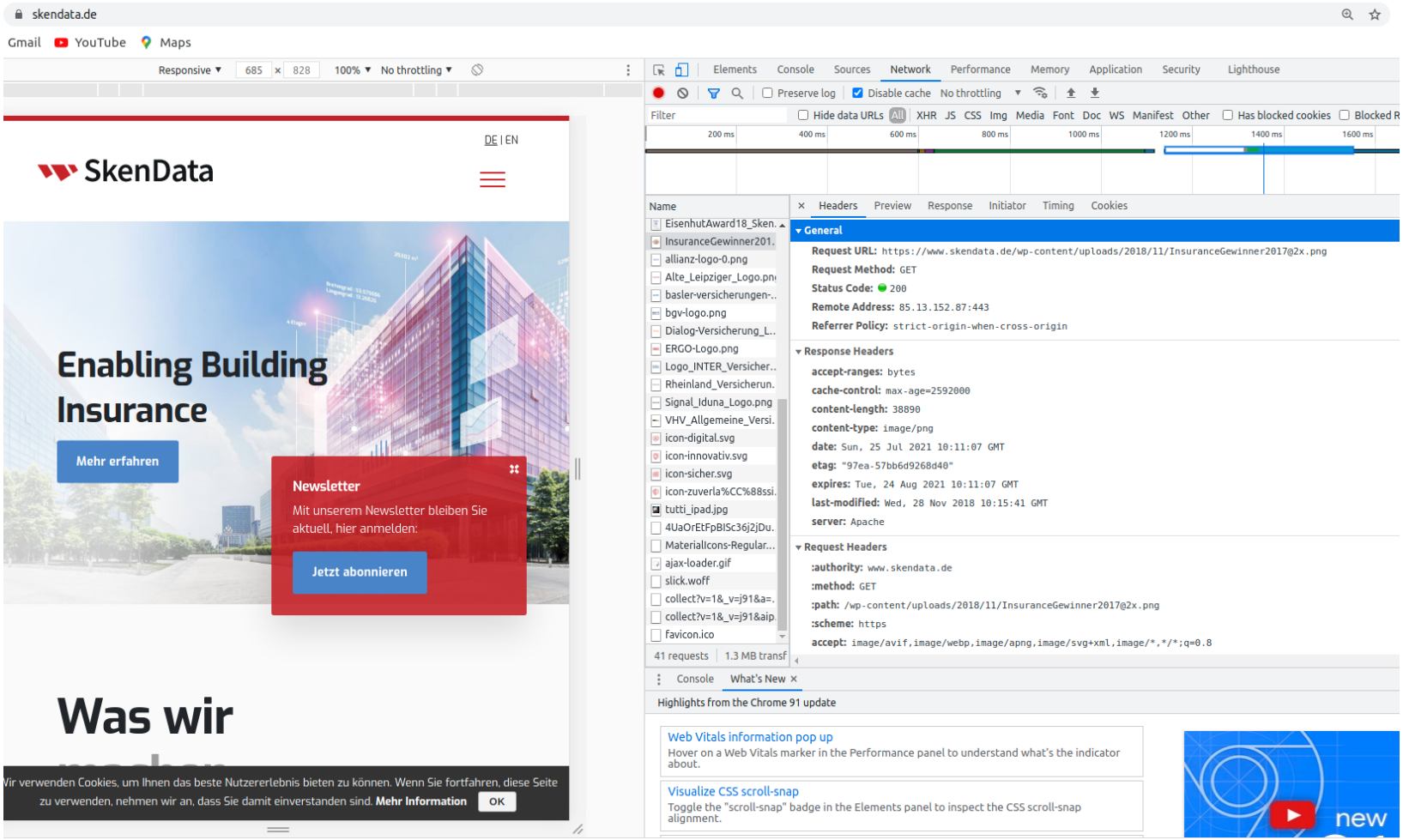

Abbildung 2.6: HTTP request in Chrome
Antwort vom HTTP Request in Chrome nach [13]

Abfrage-String in URLs und Headers beim Request

Nach Erklärung vom "*Seppe vanden Broucke und Bart Baesens*" [13] wird der optionale "?..."-Teil in URLs als "*Query String*" bezeichnet und ist zum Enthalten der Daten gedacht die nicht in die normale hierarchische Pfadstruktur einer URL gehören [13]. Als ein Beispiel dafür anhand des Besuches von Seppe vanden Broucke und Bart Baesens wird in Folgende angezeigt [13]:

"https://www.google.com/search?q=Skendata+GmbH"

Anhand der Beschreibung in Google-Seite wurde die Google-Suchanfrage als ein "*Standard-HTTP-GET-Befehl*" beschrieben, das eine Sammlung von Parametern beinhaltet, die für die Abfragen relevant sind, wobei diese durch "&" Zeichen getrennte Parametern in der Anfrage-URL als "*Name-Wert-Paare*" enthalten sind. Die Parameter enthalten Daten wie die "Such-anfrage" und eine individuelle "Engine-ID (cx)", welche die Engine identifiziert und die HTTP-Anfrage durchführt [28].
von diesem Beispiel kann es so beschrieben werden, dass durch diesen URL man an SERP von Google Webseite anhand der Abfrage "SkenData GmbH" gelandet werden kann und "SkenData GmbH" wird als ein "*URL-Parameter*" von Abfrage gerechnet [13].
Infolge der Erklärungen im Buch "*Practical Web Scraping for Data Science: Best Practices and Examples with Python*" über Request auf HTTP, sind die Webseiten in der Lage, Crawlern zu verhindern, auf die Inhalte zuzugreifen. Dafür wurde geklärt, dass die Webseite verschiedene Möglichkeiten haben, um ein solches System zu entwickeln. Bei einige der genannten Entwicklungen könnte es durch Sendung eines "benutzerdefinierten Header" in "*headers:*" Argument vom Request auf HTTP gelöst werden [13, 29].

Beautiful Soup

Anhand der Beschreibung im Buch von Seppe vanden Broucke und Bart Baesens [13] wurden die meisten Webseiten mit der der HTML formatiert. Darüber hinaus sollte man bei Bedarf Informationen aus solchen Seiten extrahieren können. Die drin auch Cascading Style Sheets (CSS) zur Formatierung bzw. zum Stilisieren moderner Webseiten verwendet wird. Nach einem Request auf Http hat man die Möglichkeit auf HTML Inhalte zuzugreifen [13]. "*Beautiful Soup*" [30]-Bibliothek wurde dafür als eine Möglichkeit vorgeschlagen, die man dadurch mit "*html.parser*""*(a built-in Python parser)*" und bei einige Fällen mit "*lxml*" "*(not built-in Python Modul)*" die Inhalte anhand der definierte bzw. geschriebene "Tags" sowie "Elemente" in HTML-Seite indexieren kann [13].

Wie beschrieben, bietet “Beautiful Soup” die Möglichkeit an, die Wandlung der HTML-Inhalte in eine Baum basierte Darstellung, und dafür wurden als Beispiel “*find*” und “*find_all*” Methode zur Verfügung gestellt [13].
Außerdem wurde es auch beschrieben, dass Verwendung von “CSS-Selector” neben den anderen angebotenen Methoden von “Beautiful Soup” zum Adressieren bzw. Selektieren zur Verfügung gestellt wurde, sodass man dafür einfach “CSS selector rule” als ein “string” einsetzen kann [13]. Durch die verschiedene Methode zur Adressierung in “Beautiful Soup” anhand [13], wurde es beschrieben, dass man in der Lage ist, verschiedene Teile vom “HTML-Tree” zu indexieren [13]. Wie es im Artikel “ *A Study of Web Information Extraction Technology Based on Beautiful Soup*” [31] ersichtlich ist, wurde das Extrahieren der Web-Informationen durch “Beautiful Soup” durchgeführt wurde, und in folgende wurde beschrieben, dass der Web-Crawler eine Genauigkeit von über 95% erreicht und damit die Anforderungen für kommerzielle Anwendungen erfüllt hat [31]. Von Sanya Goel et al. im Artikel “*Web Crawling-based Search Engine using Python*” [32] wurde zum Web-Crawling das “urllib.request” zur Öffnung von URLs sowie “Beautiful Soup” zum Extrahieren von Namen und URLs aus der HTML-Seite verwendet [32].

Selenium

Laut “*Wu Hejing et al.*” im Artikel “*Application Research of Crawler and Data Analysis Based on Python*” kann “*Selenium*” für dynamisch implementierte Website als eine Lösung gerechnet werden [33]. Im Buch “*Practical Web Scraping for Data Science: Best Practices and Examples with Python*” wurde “Selenium” als eine leistungsfähiges Tool für Web-Scraping vorgeschlagen, sodass es zum Laden einer Webseite mit der Automatisierung von Browsern arbeiten kann. damit der Inhalt der Website abgerufen und Aktionen ausgeführt werden können. Es wurde auch erwähnt, dass “Selenium” selbst keinen eigenen Webbrowser hat. sondern benötigt es einen “*WebDriver*”, um mit einer dritten Partei zu interagieren. Außerdem wurde es beschrieben, dass ein Browser-Fenster auf dem Bildschirm öffnet, und die Aktionen anhand des eingesetzten Codes ausführt, wenn ein “WebDriver” von einem modernen Browser wie zum Beispiel Chrome, Firefox usw. verwendet ist [13]. Genau wie es im Buch geklärt wurde, kann man die “Selenium” anstatt der Verwendung von “Requests” Bibliotheken sowie “Beautiful Soup” installieren, um die Inhalte der HTML- Seite zu scrapen bzw. crawlen [13]. Außerdem wurde auch geklärt, dass durch die implementierte Methode in “Selenium” ist man in der Lage einfach die Inhalte zu adressieren bzw. indexieren und auch zu sammeln. Als Beispiele wurde folgende Funktionen im Buch “*Practical Web Scraping for Data Science: Best Practices and Examples with Python*” vorgeschlagen [13].

- “*find_element_by_id*”

- "*find_element_by_name*"
- "*find_element_by_xpath*"
- "*find_element_by_link_text*"
- "*find_element_by_partial_link_text*"
- "*find_element_by_tag_name*"
- "*find_element_by_class_name*"
- "*find_element_by_css_selector*"

Wie es zu sehen ist, hat man über "Selenium" die Möglichkeit, die Elemente durch "xpath-Selector" zu indexieren, sodass es so beschrieben wurde, dass so eine Aufgabe bei "Beautifulsoup" durch "lxml" gehandelt werden kann [34]. Wenn Element durch "Selenium" nicht gefunden werden kann, wird eine Meldung als "*NoSuchElementException*" geraist, sodass bei "Beautiful Soup" in diesem Fall "None" ausgegeben wird, laut Seppe vanden Broucke und Bart Baesens [13].
Neben den genannten Funktionen vom "Selenium" wurde andere interessante Funktionen vorgeschlagen. Damit man mit den dynamischen Webseiten interagieren kann. Als Beispiel wurden "*click*" zum Klicken auf einem Button, "*send_keys*" zur Eingabe eines "string" sowie "*screenshot*" um ein Bildschirmfoto zu machen usw. genannt [13, 35].
Wie es im Artikel "*Application Research of Crawler and Data Analysis Based on Python*" beschrieben wurde, könnten bei Bedarf die Bibliotheken wie "Selenium" und "Beautiful Soup" zusammen verwendet werden [33]. Wie es im Artikel "*Automated management of green building material information using web crawling and ontology*" [36] beschrieben wurde, wurden die beiden "Selenium" und "Beautiful Soup" Bibliotheken zur Sammlung der Informationen, und "*Pandas*"-Bibliothek zum Dokumentieren der Informationen verwendet [36].

Scrapy

"*Scrapy*" wurde auch als eine andere Bibliothek zum Crawlen der Website und Extrahieren strukturierter Daten aus Websites vorgeschlagen [13]. Als ein Vorteile für Scrapy wurde beschrieben, dass es einfach zum Verwenden ist, und es bietet viele vernünftige Vorgaben für paralleles Crawling, Datensammlung usw. an. Darüber hinaus wurde geklärt, dass es zweckmäßig sein kann, wenn einen robusten Crawler eingesetzt werden muss [13]. Als ein Nachteil kann der Umgang vom JavaScript mit dieser Bibliothek aufwändig sein, da es selbst keinen Browser-Stack simuliert. Das

wurde aber geklärt, dass es zwar ein “Plug-in” existiert, dass trotz seines komplizierte Ansatzes bei der Einrichtung und Wartung kann den “*JavaScript-Rendering-Dienst*” “Splash” mit “Scrapy” koppeln kann [13].
Aber als ein Vergleich zwischen den “requests”, “scrapy” und “selenium” wurde es durch Workshop “*Data Workflows mit Python*” [37] geklärt, dass die Geschwindigkeit beim “requests” mehr, und die Einsetzung einfacher als andere Bibliotheken durchgeführt werden kann, wobei “Selenium” kann ein guter Ansatz für die dynamisch basierte Seite sein, sagte Yevgen Papernyk [37].

Pyppeteer

Wie es im Artikel “*Privacy policies over time: Curation and analysis of a million-document dataset*” beschrieben wurde, wurde der Cawler basiert auf “*Pyppeteer*” zur Aufbereitung des Datensatzes implementiert [38]. Darüber hinaus wurde “Pyppetter” als eine andere Bibliothek zum Web Crawling vorgeschlagen, das wie “*Puppeteer*” in Javascript mit einige Unterschiede implementiert wurde. Wie es geklärt wurde, beim “Pyppeteer” wird einen Chrome-Browser geladen, damit man die dynamische Webseite crawlen kann [39]. Als Voraussetzung dafür wurde “*Python 3.5+*” empfohlen [38, 39].
In unserer Umsetzung werden *Pyppeteer* zum Laden eines Browser sowie *Beautiful Soup* zum Parsen der Daten aus der HTML-Seite der Webseiten verwendet. Da wie es in “*developpaper*”[6] erwähnt wurde, muss man die benötigten Browser, wie Chrome zum Verwenden der *Selenium*-API installieren, und dann auf die öffentliche Website gehen, um die passenden Driver herunterzuladen.

2.2 Information Extraktion

Wie es im “*A survey of web information extraction systems*”[40] beschrieben wurde, wurden viele Arbeiten im Bereich IE zur Umwandlung von Input-Seiten in strukturierte Daten durchgeführt. Es wurde von Chang, Chia-Hui geklärt, dass relevante Dokumente aus einer Dokumentensammlung durch IR identifiziert werden können, und Gegensatz dazu erzeugt IE strukturierte Daten bereit zum postprocessing, was für viele Anwendungen von Web-Mining wichtig geklärt wurde [40]. Wie geklärt im “*Information Extraction from Text*” [41] wurde IE als eine Aufgabe genannt, damit strukturierte Informationen aus unstrukturiertem oder semi-strukturiertem Texte gefunden werden können [41]. Außerdem wurde geklärt, dass es sich um eine Aufgabe beim Text-Mining handelt, die in verschiedenen Forschungsbereichen

[6] `https://developpaper.com/pyppeteers-usage-of-python-crawler/` aufgerufen am 15.10.2021

wie Natural Language Processing (NLP), IR und das Web-Mining untersucht wurde [41]. Als zwei grundlegende Tasks zur IE wurde "*Named Entity Recognition (NER)*" [41] vorgeschlagen, dass die Methode so beschrieben wurde, dass es sich auf die Suche nach Namen von Entitäten wie Personen, Organisationen, Orten usw. bezieht [41].

2.2.1 Rule-Based-System

"*Rule-Based*" [42] wurde so beschrieben, dass wir notwendige Rules finden sollen, damit die Entitäten anhand dieser definierten Rules extrahiert werden können [42]. Wie es im Artikel "*Named entity recognition in persian texts*" [43] zur persischsprachigen Texten zu sehen ist, wurde NER-Task durch traditionelle Rule-based-Algorithmus durchgeführt [43]. Grover et al. [42] haben auch ein Rule-Based-System zur Erkennung der Person- sowie Ort-Namen in "*digitalisierten Aufzeichnungen britischer Parlamentsverfahren*" eingesetzt [42]. Sie haben als Evaluation-Ergebnis über F1-Score von 70,35% bis 76,94% berichtet [42].

2.2.2 Pre-Trained Models und Netzwerk-Architekturen

Ein Sprachmodell wurde als eine Wahrscheinlichkeitsverteilung über eine Folge von Wörtern definiert [44]. Wie beschrieben im Artikel "*Pre-trained models for natural language processing: A survey*" [45], wurden mit der Entwicklung von "*Deep Learning (DL)*" [45] verschiedene neuronale Netze wie "*Convolutional Neural Networks (CNNs)*" [46] und "*Recurrent Neural Networks (RNNs)*" [47] und "*Attention-Mechanismus*" [48] zur Lösung von NLP Aufgaben präsentiert, damit das Problem von "*Feature Engineering*" gelöst wurde [45].

Es wurde geklärt, dass nicht-neuronale NLP-Methoden stark auf die manuellen Features basieren, die von Hand erstellt werden, wobei neuronale Methoden normalerweise verwenden "*verteilte Repräsentation*" [45], um die syntaktischen oder semantischen Merkmale der Sprache darzustellen [45]. Darüber hinaus wurde es als eine vereinfachte Methode für NLP-Tasks vorgeschlagen [45].
Es wurde auch erwähnt, dass soweit umfangreiche Arbeiten gezeigt haben, dass Pre-Trained Modelle auf einem großen Korpus universelle Sprach-Repräsentationen trainiert werden können, sodass die nützlich für ein Downstream-Task sind, und das Training eines neuen Modells von Grund nicht nötig werden können [45].

Hidden Markov Models (HMMs)

Wie es im Artikel "*Inducing Hidden Markov Models to Model Long-Term Dependencies*" [49] beschrieben wurde, können HMMs beim "IE"-Task im Einsatz kommen [49]. HMMs wurde aber als einen Non-Neuronalen Language Modell eingesetzt, dass wegen dem Problem beim "*Long Term Dependency*" kompliziert zum Einsetzen geklärt wurde [50]. Wegen diesem Problem wurde bei verschiedenen Aufgaben wie "*Automatic Speech Recognition (ASR)*"[7], verschiedene Weiterentwicklungen wie im Artikel "*Combining Neural Networks and Hidden Markov Models for Speech Recognition*" durchgeführt. in dem genannten Artikel wurden zum Beispiel zwei verschiedenen Architekturen kombiniert, um die Herausforderung bei der Arbeit zu bewältigen [51].

Fully Connected Networks (FCNs)

FCNs wurde im Artikel "*Probabilistic Neural Language Model*" [44] im Jahr 2003 von "*Bengio et al.*" eingesetzt. Dadurch wurde ein Wort anhand der vorherigen Wörter gefunden. Das wurde als ein "*N-Gram Language Modell*" genannt, da wie erwähnt und angezeigt in 2.7, wird das kommende Wort durch der (n-1) vorherigen Wörter erkannt. Wie es sichtbar ist, wurde ein "*Matrix C*" [44] zur Vektorisierung der Wörter eingesetzt [44]. Aus der Abbildung kann man merken, dass da drei Layer's stehen, sodass das dritte Layer mit anderen zwei Layer verbunden ist [44]. Als das Problem von "FCNs" wurde die Höhe Anzahl der Dimensionen bzw. Parameter geklärt [44, 45, 52].

[7] ASR wurde als eine Aufgabe geklärt, die sich zur Klassifizierung von Sequenzen akustischer Features behandelt, die aus dem Sprachsignal extrahiert werden [51].

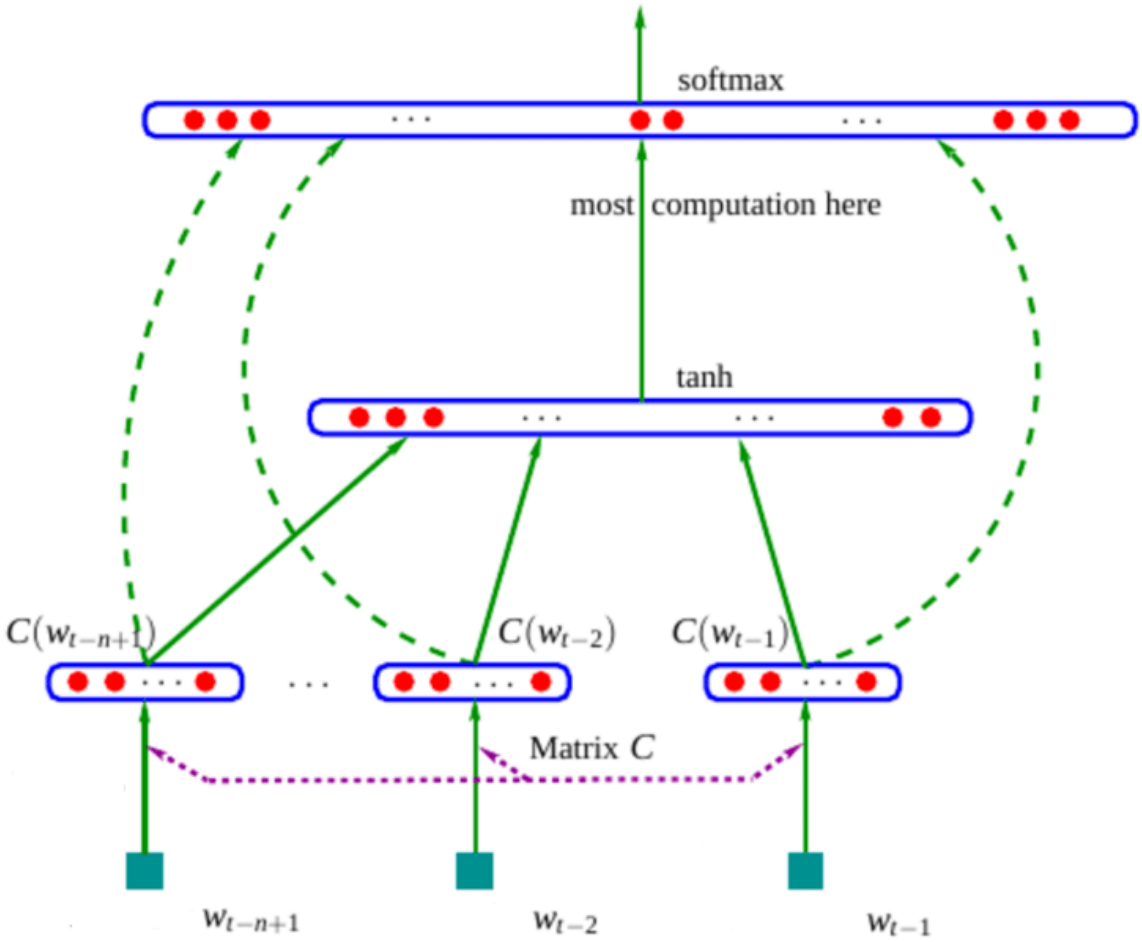

Abbildung 2.7: Neuronale Netzwerk Architektur von FCNs nach [44]

Recurrent Neural Networks (RNNs)

Auf der anderen Seite wurde beschrieben, dass bei RNNs die gleichen Parameter in viele verschiedene Layer verwendet werden. Darüber hinaus benötigt wird weniger Komplexität, Speicherplatz sowie auch weniger Datensatz zu Trainieren im Vergleich zu FCNs [4]. Es wurde geklärt, dass RNNs nehmen die Repräsentation-Plätze der Wörter durch einem "Short Memory" wie "*Long Short Term Memory (LSTMs)*" [45]. Es wurde so definiert, dass dadurch die Informationen von beiden Seiten eines Wortes gesammelt werden, Aber besteht dabei noch das Performance-Problem beim "*Long-Term Dependency*" [45]. LSTMs wurde als einen Ansatz in der Sequenzmodellierung genannt [48]. Um genauer zu sein, wurde es so beschrieben, dass "Seq2Seq"-Architekturen RNNs-Architekturen verwenden, sodass dadurch die Rechnung der aktuellen Output's in jeder Zeit die vorherige History bzw. "*Time-Step's*" [4] sehen benötigen. Das heißt, dass die Wartung auf letzte Output's er-

forderlich wäre, und deswegen gibt es keine parallele Durchführung [4, 45]. Der "*ELMO*" [2] als ein Pre-Trained-Sprach-Modell verwendet LSTMs bzw. RNNs [2].
Im Artikel "*Named Entity Recognition with Long Short-Term Memory*" [53] wurde LSTMs Netzwerk für ein NER-Task trainiert [53]. Die Einsetzung wurde für sowohl deutsch- als auch Englisch-Sprachige Texte durchgeführt, und das gesamte F1-Score-Ergebnis bei beiden unter 73% angezeigt wurden [53].
Neben dem Problem von RNNs wurde es so beschrieben, dass es leicht zu trainieren ist, und in der Lage ist, gute Ergebnisse für verschiedene NLP-Aufgaben zu liefern [45].

Transformer

Als eine unkomplizierte Art wurde "*Fully-connected self-attention model*" [45] zum Verwenden empfohlen [45]. "Transformer" wurde als ein "*breakthrough*" [45] von DL genannt [45]. Es wurde beschrieben, dass bei Transformer keine sequenzielle Durchführung benötigt ist, und damit Parallelization wird mehr als RNNs. Darüber hinaus wurde beschrieben, dass die Training-Zeit weniger wird [45, 4]. Wie es von Azam Rabiee [4] und QIU XiPeng et al. [45] beschrieben wurde, wird FCNs von Transformer verwendet, und gibt es keine "*Cross-Attention*" zwischen "*Encoder*" und "*Decoder*" [45], sondern das "*self-attention*" bzw. "*Multi-Head-Attention*" durchgeführt wird [45, 48, 4]. QIU XiPeng et al. [45] haben gemeint, dass bei Transformer ein "*Positional-Encoding*"[45] durchgeführt wird [45]. Von Azam Rabiee [4] wurde geklärt, dass es bei FCNs keine Reihenfolge der Wörter gab, deswegen kam RNNs. Aber durch "*Fully-connected self-attention model*" [45] und Positional-Encoding werden die Reihenfolge der Wörter berücksichtigt [4, 45, 48].

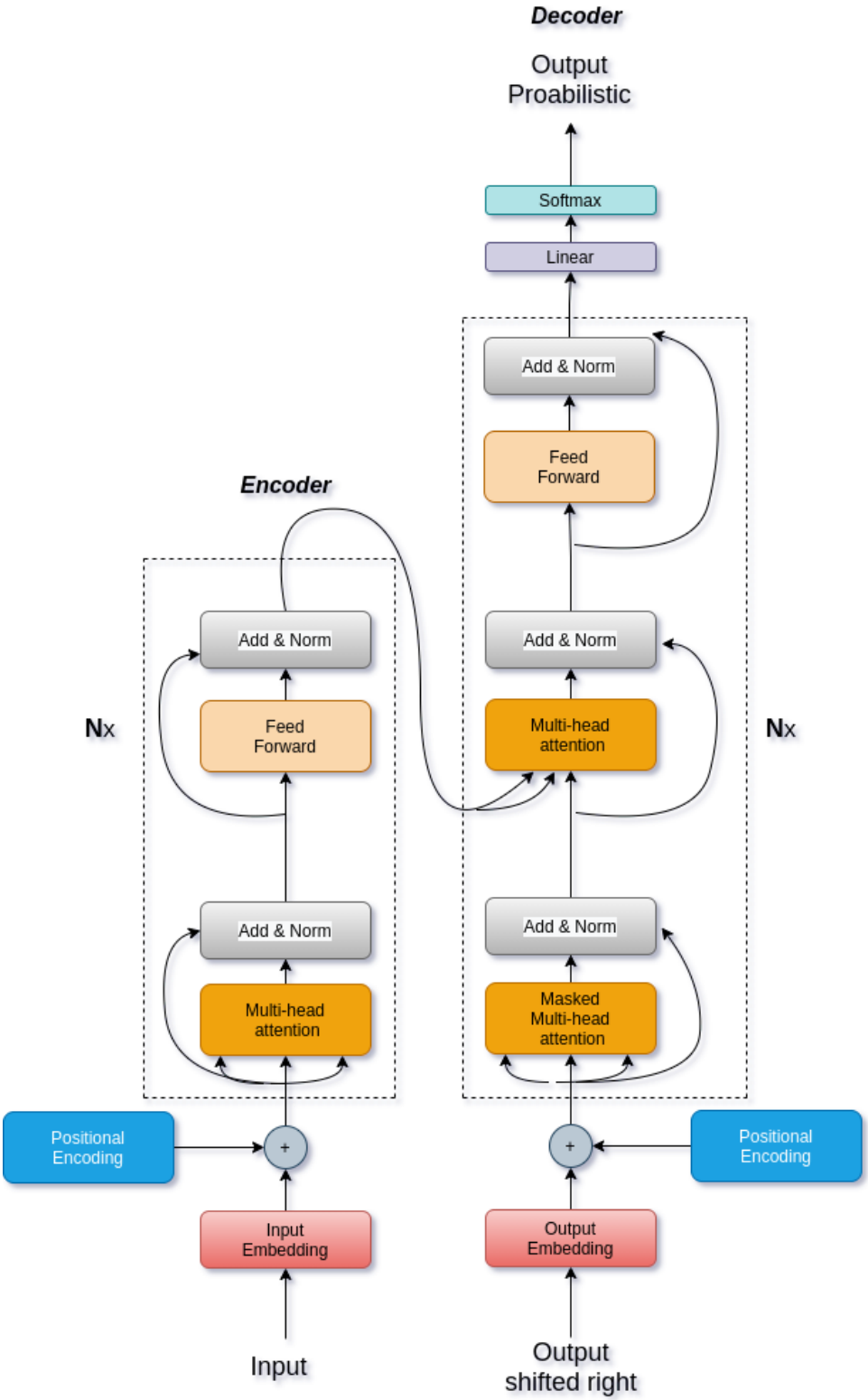

Abbildung 2.8: Transformer-Architektur nach [48]

Bidirectional Encoder Representations from Transformers (BERT)

Jacob Devlin et al. [1] haben "*BERT*" im Jahr 2018 entwickelt [1]. BERT wurde als eine Entwicklung für Transformer-basiertes Maschine-Learning-Technik für NLP pre-training durch Google genannt [1]. Wie es beschrieben wurde, verwendet Bert nur den Encoder-Teil von Transformer [1]. Wie beschrieben, gibt es andere Pre-Trained-Sprach-Modell wie "*ELMO*" [2] und "*GPT*" [3], sodass LSTM wird von ELMO und Transformer von GPT und BERT verwendet. Im Vergleich zu GPT wurde es so beschrieben, dass BERT den Encoder-Teil aber GPT den Decoder-Teil von Transformer verwenden [1, 2, 3]. Wie es im Artikel "*Bert: Pre-training of deep bidirectional transformers for language understanding*" erwähnt wurde, können elf NLP-Tasks wie zB. "*Summerization, Sentiment-Analisis, Quastion-Answering*" [1] usw. durch Bert behandelt werden [1]. Wie es im 2.2 anhand den Artikels von BERT, ELMO und GPT [1, 2, 3] und den Workshop von Azam Rabiee [4] ersichtlich ist, ELMO und BERT sind Bi-directional mit dem Unterschied dass, der BERT hat zwei verbundene Uni-directional Funktion, wobei der ELMO hat zwei Uni-directional nebeneinander. Wie es geklärt wurde, durch Bi-directional-Innovation zu genaueren Wort-Repräsentation führen können[8] [1, 2, 4].

Tabelle 2.2: Vergleich der Sprach-Modellen nach [1, 2, 3, 4]

Modell	Uni/Bi-directional	Nezwerk-Architekture
ELMO	Bi-directional (shallow)	LSTM
GPT	Uni-directional	Transformer-Decoder
BERT	Bi-directional	Transformer-Encoder

Fine-Tuning BERT for NER

Wie erwähnt im Artikel "*Beheshti-NER: Persian named entity recognition Using BERT*", ist BERT auf 104 Sprachen wie Deutsch, Englisch, Persisch, Französisch usw. vor-trainiert [54, 1]. In diesem Artikel wurde auf ein BERT-Pretrained-Modell das Fine-Tuning für ein Spezielles NER-Task durchgeführt. Die Evaluation-Metriken dazu wurde "*CONLL 2003 Score*" berichtet, sodass als das Ergebnis 88.4% F1-Score bei Wort-level und 83.5% F1-score bei phrase-level erreicht wurde [54].

[8] `https://datascience.stackexchange.com/questions/68155/what-are-some-key-strengths-of-bert-over-elmo-ulmfit` aufgerufen am 15.09.2021

Kai Labusch et al. [55] haben ein Pre-Trained-BERT-Modell zum deutsch-sprachigen NER-Task trainiert [55]. Sie haben "*bert_based_Multilingual_Cased*" [1, 55] zur Entwicklung verwendet [55]. Die Evaluation wurde unter "*cross validation*" durchgeführt [55]. "unsupervised pre-training auf DC-SBB Daten" sowie "supervised pre-training auf contemporary NER ground truth" wurden zur Untersuchung verwendet [55]. Wie beschrieben wurde das Training durch sieben Epochen durchgeführt, sodass bei Evaluation-Ergebnisse unter 89% erreicht wurde [55].

Ein anderer NER-Task wurde von Kai Hakala & Sampo Pyysalo [56] im Artikel "*Biomedical Named Entity Recognition with Multilingual BERT*" eingesetzt [56]. Da wurde geklärt, dass sie auch ein Multilingual-BERT zur Forschungsarbeit verwendet haben, und wurde in folgende 88% bei der Entwicklung-Daten und 87% F-Score bei der Test-Datensatz erreicht [56]. Es wurde geklärt, dass die Annotation des Datensatzes für "*Protein, Chemical(+), Chemical(-) und Other*" gemacht wurde [56]. Im Vergleich zu anderen erwähnten Forschungen in dieser Arbeit, wurde es geklärt, dass die Anzahl der Annotation weniger als die anderen vorherigen Untersuchungen war, trotzdem wurde durch die Arbeit die genannten Evaluation-Ergebnisse erreicht [56].

Von Charlene Chambliss [57] wurde auch ein NER-Task durchgeführt [57]. Wie sie es von ihr beschrieben wurde, wurde ein Fine-Tuning von Bert für englische und russische Texte durchgeführt [57]. Es wurde beschrieben, dass dieses Modell als ein Teil in einem Projekt von "*Maschine-Übersetzung*" [57] eingesetzt wurde, damit die Einschätzung der "*MT-Qualität*"[57] zwischen englisch und russische Satzpaaren verbessert wurde [57]. Zum Fine-Tuning englisch-sprachige Texte wurde den "*bert-base-cased*"-Modell verwendet [57]. Anhand "*Hugging Face*"[9] ist dieses Modell zur Case-Sensitive-Projekte eingesetzt [58]. Es wurde beschrieben, dass "*Das BERT-Modell wurde mit dem BookCorpus trainiert, einem Datensatz, der aus 11.038 unveröffentlichten Büchern und der englischen Wikipedia besteht (ohne Listen, Tabellen und Headers)*" [58]. Für die rusisch-sprachige Texte wurde "*bert-base-multilingual-cased*" von Charlene Chambliss [57] in ihrer Einsetzung verwendet [57]. Wie es auch in "*Hugging Face*"[10] beschrieben wurde, ist es als ein Pre-Trained-Modell, den für 104 verschiedene Sprachen durch "*large Wikipedia*" trainiert wurde [58].
Das Fine-Tuning von Bert wurde wie Folgende von Charlene Chambliss in sechs Schritte zusammenfassend beschrieben [59]:

- Preprocessing und Aufbereitung des Datensatzes

[9] `https://huggingface.co/bert-base-cased` aufgerufen am 15.09.2020
[10] `https://huggingface.co/bert-base-multilingual-cased` aufgerufen am 15.09.2020

- Einsetzung der Hyper-Parameter in "*PyTorch*"[11]
- Konvertieren der Daten in Tensoren und das Laden der Tensoren in Dataloader in PyTorch
- Training, Evaluation sowie Speicherung von Modell, empfohlen wurde auf 3-5 Epochen, sodass nach jede Epoche das Evaluation-Ergebnis durch Confution-Matrix und Klassifikation-Report angezeigt wird
- In diesem Schritt wurde so geklärt, dass man sicher werden soll, Ob Modell- und Optimizer-checkpoints richtig gespeichert wurden.
- Prüfen der Modell-Vorhersagen

Laut Charlene Chambliss sollte die Tokens in Training-Datensatz durch den "*IOB*"-Standard [60] annotiert werden, da es für unterschiedliche Plätze der Wörter praktisch geklärt wurde [60]. Als ein Beispiel dazu kann man anhand der geklärten Beschreibung das Annotation-Format wie in Abbbildung2.9 erwähnen [60]:

```
Sie erreichen uns in Mühlendamm,8B mit der Postleitzahl 18059 in Rostock
 O     O       O   O     B-LOC      O   O       O       I-LOC  O  I-LOC
```

Abbildung 2.9: IOB-Standard-Annotation] nach [60]

Charlene Chambliss hat gemeint, dass B, I und O in Annotation-Format standardmäßig die "*Beginning, Inside und Other*" [60] für die Wörter im Text hinweisen [60].
Charlene Chambliss hat zudem erwähnt, dass "*BERT WordPiece-Tokenisierung anstatt Ganz-Wort-Tokenisierung verwendet*" [60]. Damit wurde gemeint, dass die Tokens in Teilwort-Tokens aufgeteilt werden [60]. Zur Behandlung dieser Wörter wurde eine Funktion von ihr als "*tokenize_and_preserve_labels*"[60, 57] durch Implementierung geschrieben, damit die Übertragung von originalem Label eines Wortes während der Tokenisierung auf alle seine Teile durchgeführt wurde [60, 57].
In Folgende wurde es gemeint, dass es während dem Preprocessing zur Token-Klassifikation notwendig ist, ein [PAD] hinzufügen, damit die maximale Sequenzlänge bei den Sätzen berücksichtigt wird [60]. Das wurde beschrieben, dass "*diese [PAD]-Tokens während des Trainings durch eine vorher definierte Attention-Mask von Training-Prozess des Modells maskiert werden, wodurch das Modell angewiesen wird, diese Tokens und ihre Labels bei der Berechnung von Loss in jedem*

[11] `https://github.com/pytorch/pytorch/wiki` aufgerufen am 15.09.2021

Schritt zu ignorieren" [60]. Wie es von ihr durchgeführt wurde, wurden diese [PAD]-Tokens während der Evaluation durch "*Confusion-Matrix*" [60] ignoriert, da sie den Standpunkt vertritt, dass "*die Moral hier ist, dass Sie Ihre Hypothesen über das Verhalten des Modells immer überprüfen sollten*" [60]. Nach ihrer Meinung, ist Confusion-Matrix für alle Klassifikation-Aufgaben empfehlenswert, um sicher zu werden, dass diese [PAD]-Tokens nicht beim Prediction von Modell verwendet worden sind [60].
Es wurde von ihr geklärt, dass sie am Ende ihrer Implementierung 0.95 bei Person-Entity, 0.96 bei Location sowie 0.95 F1-Score bei Organisation als Evaluation-Ergebnis erhalten hat [60].

Ein anderer NER-Task wurde auch von Tobias Sterbak[12] im Jahr 2020 durchgeführt und veröffentlicht [61]. Die Reihenfolge der Arbeit wurde von Tobias Sterbak wie folgt beschrieben [61]:

- Laden der aufbereiteten Trainingsdaten
- Anwendung des Bert-Sprach-Modells
- Aufbereitung der Tokens und ihre Annotationen
- Einrichtung von Bert in PyTorch zum Fine-Tuning sowie Einrichtung der Hyper-Parameter
- Anwendung von Fine-Tuned-Modell für neue Sätze bzw. das Postprocessing

Der Datensatz, den von ihm zum Training verwendet wurde, wurde auch anhand IOB-Standard annotiert [61]. In seiner Arbeit wurde das bert-based-Model für englisch-sprachige Texte verwendet [61]. Zur Evaluation von Modell wurde "*seqeval.metrics*" [62] als eine Evaluation-Bibliothek eingesetzt. Durch seiner zur Verfügung gestellten Source-Code ist es sichtbar, dass eingegebene [PAD]-Tokens zur Evaluation ignoriert wurden [61].

NER mit LSTMs und ELMo

Vorher wurde Ein anderer NER-Task von Tobias Sterbak durchgeführt und veröffentlicht [63]. Während dieser Arbeit wurde die Implementierung durch "*Tensorflow*"-[13] und "*Keras*"-API[14] durchgeführt [63]. Da wurden LSTMs-Netzwerk sowie

[12] `https://www.depends-on-the-definition.com/about/` aufgerufen am 15.09.2021
[13] `https://www.tensorflow.org/api_docs` aufgerufen am 15.09.2021
[14] `https://faroit.com/keras-docs/1.2.0/` aufgerufen am 15.09.2021

"ELMo-Embedding" für ein NER-Task verwendet [63]. Von ihm wurde es beschrieben, dass "*ELMo Embeddings sind Embeddings eines Sprachmodells, das mit dem 1 Billion Word Benchmark trainiert wurde und die Pre-Trained-Version ist auf Tensorflow Hub verfügbar*" [63]. Wie es beschrieben wurde " *verwenden ELMos ein Pre-Trained, mehrschichtiges, bidirektionales, LSTM-basierten Sprach-Modell und extrahieren den Hidden-State von jedem layer für die eingegebene Wortfolge*" [63]. Laut Tobias Sterbak wurde die Implementierung zusammenfassend wie Folgende durchgeführt [63]:

- Aufbereitung des annotierten Datensatzes (der hier auch anhand IOB-Standard annotiert wurde)
- Aufbereitung der tokenisierten Sätze + eingegeben wurden "__*PAD*__" [63] für ausgewählte 50 maximale Länge der Sätze
- Aufbereitung der Tokens und ihre Annotationen in einer Liste von Tupeln
- Herunterladen von ELMo-Modell aus Tensorflow-Hub
- Einrichtung der Parameter in keras
- Vektorisierung der Sätze & Einrichtung LSTM-Netzwerk
- Training & Evaluation von Modell
- Vorhersage

APIs zur Informations- bzw. Entity-Extraktion

Wie es in "*monkeylearn*"-Website [64] beschrieben wurde, gibt es viele APIs, die spezifische Entitäten durch ihre Einsetzung zurück liefern. Als Beispiel wurden Organisation-, Ort-, Person-Namen und Umsatz-Preise erwähnt [64]. Bei manche diese "Software as a Service (SaaS)-APIs" [64] kostet es Geld, um die zu verwenden.

"*MonkeyLearn*" [64] als eine API wurde so beschrieben, dass es bei allen Programmiersprachen durch wenige Zeilen von Code funktionsfähig ist, sodass die Entitäten in einem "JSON"-Format extrahiert werden können [64]. Es wurde geklärt, dass dadurch die Person-, Ort- und Organisation-Namen extrahiert werden können [64].

"*Natural Language API*" [65] von Google bietet auch die Möglichkeit an, um Entitäten zu extrahieren [65]. Außerdem können dadurch paar andere NLP-Aufgaben wie

Sentiment-Analyse und Inhaltsklassifizierung erledigt werden [65]. Neben genannten Möglichkeiten wurde es geklärt, dass man auch dadurch Benutzerdefinierten Entitäten extrahieren lassen kann [65].
Infolge von genannten APIs gibt es anderen wie "AYLIEN" [66], "*IBM Watson*" [67], "*Amazon Comprehend*" [68], um die Informationen aus Texten zu extrahieren.

Wie es in "*Hugging Face*" [69] beschrieben wurde, wurde ein "*API*" für die IE aus deutschsprachige Texte zur Verfügung gestellt, das für Question-Answering-Task eingesetzt wurde [69]. Wie geklärt, wurde dafür "*GELECTRA-Large*"-Modell von deepset.ai zum Fine-Tuning verwendet [69], sodass 77.39% F-Score bei Entwicklung sowie 82.25% bei Test erreicht wurde [69]. Infolge dieser Einsetzung wurde "*Model inferencing*"-Code [69] angeboten, sodass man damit die Entwicklung kostenlos durch einige Zeile zur Verfügung gestellte Code verwenden kann [69]. Das wurde aber geklärt, dass diese API gegen Geld schneller und mit mehr Leistungen funktioniert [69].
In unserer Arbeit wurde am Anfang die kostenlose Version dieser API umgesetzt, um die Informationen aus der Texten der Webseiten zu extrahieren. Wegen langsamer Funktion der API und der unvollständigen Ergebnisse wurde geplant, das Fine-Tuning auf *bert-base-multilingual-cased* basiert auf die beschriebene Sourcecode von Tobias Sterbak durchzuführen, da es aktuell Stand der Technik geklärt wurde und damit einen spezifischen NER-Task umgesetzt wird.

2.3 Informationsintegration

Wie es Am Anfang des Artikels "*Formally Robust Ontology-Based Data and Information Integration*" [70] erwähnt wurde, wurde die Informationsintegration heutzutage als eine der wichtigsten Herausforderungen besonders in Bio-Informatik zum Bewältigen genannt [70]. Wie erwähnt, gibt es Vielzahl von unterschiedlichen Quellen, wo die Informationen stehen, und sie bei Bedarf in viele unterschiedliche Formaten automatisiert bzw. integriert werden sollen [70]. Es wurde erwähnt, dass es viele heterogene Daten in verschiedene Formen gibt [70]. Es kann die strukturierte Daten in relationale Datenbanken, semistrukturierte Daten in "*Extensible Markup Language (XML)*" [70], HTML Hyperlinking oder unstrukturierte Daten in Text-Format sein [70]. Es wurde bisher unterschiedliche Forschungen in diesem Bereich durchgeführt.
Anhand des Artikels "*Integrating Semi-structured Information using Semantic Technologies*" [71] hat heterogener Datenquellen mit zwei Herausforderungen wie "*Heterogeneity und Reconciliation*" zu tun [71].

Laut Alejandra Casas et al. [71] kann Heterogenität in vier folgende Kategorien unterteilt werden [71]:

- "*Verschiedene Schema der Daten*"
- "*Syntaktische Unterschiede, verschiedene Arten der Benennung desselben Objekts*"
- "*systemisch, d.h. verschiedene Plattformen von unterschiedlichen Behörden*"
- "*semantisch, d.h. unterschiedliche Bedeutungen für denselben Begriff oder verschiedene Namen für denselben Begriff*"

In folgende wurde erwähnt, dass Reconciliation sich auf zwei unten genannten Punkte unterteilt [71]:

- "*Schema, das Auffinden von Ähnlichkeiten zwischen Tabellen und Spalten aus verschiedenen Datenquellen, d. h. die Berücksichtigung struktureller Heterogenität*"
- "*Instanz, besteht darin, Instanzen zu identifizieren, die trotz ihrer unterschiedlichen Darstellungen dieselbe Entität in der realen Welt repräsentieren.*"

Anhand des Artikels "*Integration von Informationssystemen*" [72] werden in der Forschungsarbeiten über viele Standards, Konzepten und Techniken diskutiert, die in einzelnen Phasen der Informationssystementwicklung unterschiedliche Gestaltungsoptionen eröffnen [72]. In jeder Entwicklungsphase sollte es laut Alejandra Casas [72] eine Entscheidung getroffen werden, welche Standards, Techniken sowie Konzepten eine Rolle spielen können [72].
Wie beschrieben von Alejandra Casas [72] sollte in der "*fachkonzeptionellen Spezifikation*"-Phase [72] definiert werden, was die Integration der Informationssysteme aus betriebswirtschaftlicher Sicht erreichen soll, sodass diese Phase als eine Aufgabe von "*Requirement Engineering*" gerechnet werden kann [72]. Laut Alejandra Casas kommt nach dieser Phase die nächste als "*Designphase*", sodass dadurch die Bausteine sowie Struktur der Information-System festgelegt wird [72]. Es wurde beschrieben, dass danach in der "*Implementierunsphase*" der Algorithmus und Komponenten "*programmtechnisch*" eingesetzt wird [72].

Es wurde gemeint, dass zur Realisierung der Integration von Informationssystemen das Konzept “*komponentenbasierter Anwendungssysteme*” auch eine wichtige Rolle spielt [72]. Das Ziel von komponentenbasierter Anwendungssysteme wurde so beschrieben, dass die Komponenten der betrieblichen Anwendungssysteme sind im kommerziellen Rahmen zugänglich und können zu komplexen Anwendungssystemen kombiniert werden, und diese auf “*zentrale Workflowmanagementsysteme*” von Betrieb liegt [72].

2.3.1 Ontologie-basierte Informationsintegration

Laut Alejandra Casas et al. [71] können “*Ontologie*”-basierte Methode für die Arbeit mit syntaktischer und semantischer Heterogenität infrage kommen [71]. Wie erwähnt kann z.B. “*MOMIS-Projekt (Mediator environment for Multiple Information Sources)*” [73] die Möglichkeit zur Integration von Daten aus strukturierten und semi-strukturierten Datenquellen anbieten [71, 73], Oder z.B. für “*Schema-Matching*”-Problem kommen heutzutage viele Ontologie-basierte-Methode infrage, da es viele verschiedene nicht formal dokumentierte semantische Aspekte von Schema gibt [71, 72, 73].

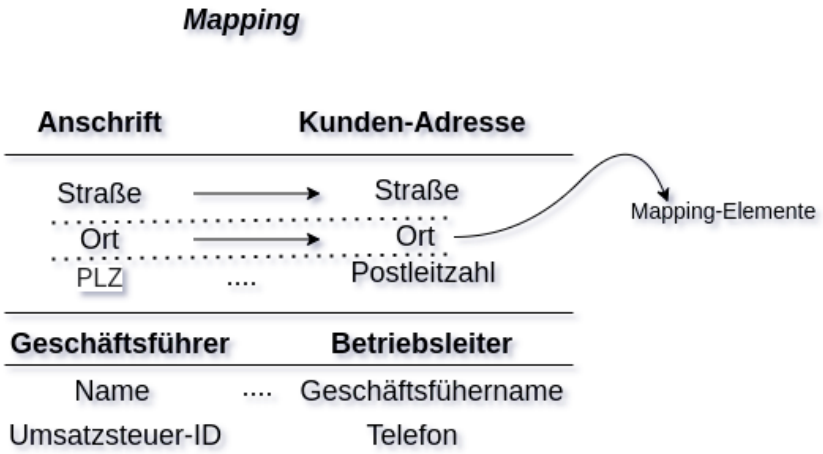

Abbildung 2.10: Beispiel zur Schema-Matching-Architektur nach [72]

2.3.2 Extraction Transformation Load (ETL)-Ansätze

Wie beschrieben, kann das Web als ein globales “*Knowledge-Repository*” [74] gerechnet werden, sodass jede beliebige Information in beliebigen Struktur-Formen in Web-Quellen einsetzen kann [74].

Wie definiert im Artikel "*Role of text mining in information extraction and information management*" [74] werden die Komponenten des Text Mining als IR, Informationsverarbeitung und Information-Integration erläutert [74], die jeder Begriff als ein Teil von ETL-Prozess gerechnet werden kann.
Die Quellen, die von verschiedenen Nutzern im Web erstellt werden, wurde beschrieben, dass im Form vom unstrukturierte-Daten als ein Text-Dokument sein können, die viele Informationen daraus extrahiert werden können [74]. Dazu wurde erklärt, dass verschiedene Text-Mining-Technologien entwickelt wurden [74]. Diese Text-Mining-Technologien wurden als Text Analysis and Knowledge MIning (TAKMI) genannt [74]. Text-Mining wurde dafür erklärt, um Informationen aus verschiedenen Quellen zu extrahieren, und schließlich diese extrahierten Informationen zusammen in einem strukturierten Format zu Automatisieren bzw. zu verknüpfen wie zum Beispiel in einem relationalen Datenbank-System [74].
"*M. Natarajan*"[74] hat gemeint, dass Text-Mining drei wichtige Unterpunkte wie Folgende beinhaltet [74]:

- "***Daten-Aufbereitung***"

  Wie beschrieben, geht dieser Schritt um die Daten-Bereinigung sowie Pre-Processing wie Satz-Tokenization und "*Part-of-Speech-Tagging*" [74], die in der Phase von Feature-Engineering durchgeführt werden können [74, 45]

- "***Daten-Verarbeitung***"

  In diesem Schritt wird geklärt, dass der Prozess auf vorbereitete Daten durchgeführt wird [74]. NLP-Algorithmen spielen in diesem Schritt eine große Rolle [74]. Die Algorithmen, die dazu verwendet werden können, wurden wie zum Beispiel "*decision trees, neural networks, case-based learning, association rules or genetic algorithms*" genannt [74].

- "***Analyse der Daten***"

  Post-Processing, die Evaluationen- bzw. das Prüfen der extrahierten Daten, ob die relevant zu den Anforderungen sind, gehören zu diesem Schritt [74].

Es wurde erklärt, dass verschiedene Algorithmen zum Text-Mining-Applikationen anhand der Betrieb-Anforderungen eingesetzt werden [74]. Zum Beispiel im Bereich Bio-Informatik wurde "ML"-Algorithmus "*SVM*" [74] eingesetzt, sodass bei der Evaluation eine Genauigkeit von über 90% erreicht wurde [74]. Andere Algorithmen anhand verschiedener Anforderungen in diesem Gebiet implementiert wurden, die einige dieser Algorithmen im Abschnitt2.2 erwähnt sind.
Wie es von M. Natarajan [74] beschrieben wurde, wurde Text-Mining als ein Werkzeug, um die Informationen für Patent-Analyse zu extrahieren verwendet [74].

von Tanabe Scherf et al. [75] wurde eine Entwicklung durchgeführt, sodass erstmal die relevante Texte durch Abfrage von "*PubMed*" [74] gesammelt werden, und dann werden die gefundenen Texte nach benutzerdefinierte Relevanz-Kriterien bewertet und gefiltert, dann am Ende wurden die vermutlich interessantesten Dokumente durch eine gestaltete Benutzeroberfläche zur Verfügung gestellt [74, 75].

Im Artikel "*An End-User Pipeline for Scraping and Visualizing Semi-Structured Data over the Web*" [76] wurde eine Visualisierung-Pipeline eingesetzt, wodurch die Integration von semi-strukturierte Daten im Web auf einer grafischen Darstellung auf den Bildschirm des Benutzers durchgeführt wird [76]. In dieser Arbeit von verschiedenen Tools zum Web-Scraping sowie "Web Augmentation Technik zum Unterstützen der interaktiven Visualisierungen" verwendet, ohne dass die Daten geändert werden [76]. Dazwischen wurde sich auf die Extrahierung der Daten von verschieden Web-Seiten mit verschiedenen Strukturen konzentriert [76]. Wie beschrieben die Entwicklung wurde aus Kombination der Web Scraping, Web Augmentation sowie Information-Visualisierung Technik durchgeführt [76]. Wie es beschrieben wurde, ermöglicht Web-Scraping die Umwandlung sowie Speicherung von im Web verfügbaren unstrukturierten Daten, oft im HTML-Format, in strukturierten Datenbank [76]. Dabei wurden auch Daten in Such-Ergebnissen zum Scraping der Webseiten wiederverwendet [76]. von SERP zum Zugreifen auf Homepage wurde von "Gabriela Bosetti" verwendet, und die Implementierung wurde bei 50 ausgewählte Webseiten durchgeführt [76], die dazwischen bei 8 Webseiten aus verschiedenen Gründen wie "*nicht Verfügbarkeit der Website*" nicht Zugriffen [76]. In dieser Arbeit anhand der Anforderung-Analyse wurde geklärt, dass sie die z.B. Tabellen brauchten, deswegen wurde versucht die «tbody», «tr», «ul» Elemente in DOM zu finden, und dadurch die notwendigen Daten zu indexieren [76]. Über die Ausgabe vom Programm wurde es geklärt, dass es in einer JSON-Datei gemappt bzw. integriert wurde, damit es für zukünftige Prozesse verwendet werden kann [76]. In der Pipeline wurde ein Daten-Transformation-Schritt als "Filterung-Schritt"[76] implementiert, da einige Daten aus unterschiedlichen Gründen korrigiert werden sollen, bevor sie in Darstellung-Schritt kommen [76]. Zusammenfassend wurde bei dieser Arbeit drei Problemen wie unten [76] definiert und zur Lösung die Implementierung durchgeführt [76]:

- "*Scraping der semi-strukturierte Daten, um den Datensatz zu erstellen*"
- "*Erhalten der Visualisierung aus dieser Datensätze*"
- "*Erstellung der neuen integrierten kontextbezogenen Visualisierungen für jede Website*"

Bei einiger Arbeiten wurde es so implementiert, dass die extrahierte bzw. geparste Daten aus einer Website wie "*MeatBrain*"-Implementierung anhand der Beschreibung dem Artikel von M. Natarajan [74] direkt zu einer neuen Website gebunden werden [74], oder werden die Daten in einer Datenbank-System gespeichert [74]. Außerdem wurde auch beschrieben, dass die Daten auch in einem strukturierten Format wie JSON-Datei für weitere Prozesse gemappt bzw. umgewandelt werden können [76] .

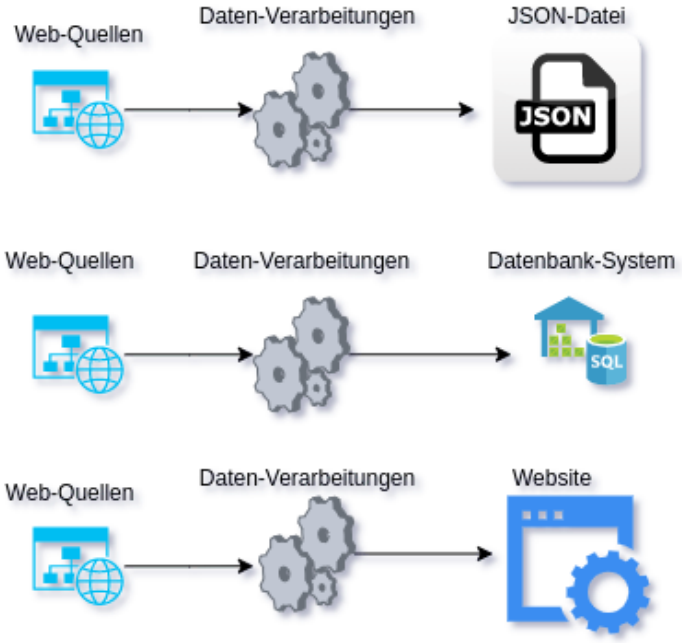

Abbildung 2.11: Einfache Beispiele für die Information-Integration-Struktur nach [70, 74, 76]

Svetla Koeva et al. im Artikel "*Natural Language Processing Pipeline to Annotate Bulgarian Legislative Data*" [77] haben auf einem Projekt "*eTranslation-System der Europäischen Kommission*" gearbeitet [77]. Das Projekt wurde für eine spezielle NLP-Pipeline in Bulgarisch zur Lösung der "*MARCEL*"-Projekt-Anforderungen durchgeführt [77].Es wurde beschrieben, dass im ersten Schritt die Daten zur Transformation im nächsten Schritt aus dem Web (HTML-Format) gecrawlt wurde, und Die Transformation wurde wie Folgende entsetzt [77]:

- "*Änderung der Daten-Formate*"
- "*Organisieren der Daten in Strukturen*"
- "*Anreicherung der Daten mit Linguistik Informationen*"

- *"Analysieren der Daten"*
- *"Erstellung der eindeutigen Verbindungen zwischen Daten-Teilen"*

Nach dem Crawling der Daten, wurde "*NLP-Cube*"-API [78] neben "*Fine-Grained- Version zur Bulgarian Language Processing Chain*" [77] für die Feature-Engineering Phase wie Filterung nach Dokument-Typ, Tokenization, Satz-Tokenization, Tagging und Lemmatization verwendet [77]. Außerdem wurde beschrieben, dass die Entitäten wie Person, Ort, Organisation neben "*IATE-Termen*"[15] sowie "*EuroVoc descriptors*"[16] in Korpus annotiert wurden [77]. Es wurde gemeint, dass "*MongoDB*"[17] zur Speicherung der Daten in diesem Projekt verwendet wurde [77].
Wie es von Svetla Koeva et al. [77] beschrieben wurde, gibt es viele NLP-Bibliotheken für verschiedene Programmiersprachen wie "*Java*"[18], "*Python*"[19]. Für Python-Programmiersprache wurden "*Natural Language Toolkit (NLTK)*"[20], "*spaCy*"[21] genannt [77]. Außerdem wurde die "*TextBlob*"[22] auch als eine Bibliothek von NLP zur Verfügung gestellt. In dieser Arbeit wurde ein Tools mit dem Namen "*TextAnnotator*" [77] zu Annotation der Termen verwendet und diese TextAnnotator wurde zum Finden der Vorkommnisse der Termen in einem Dokument eingesetzt, anhand [77]. Zur Erhöhung der Genauigkeit wurde auch Normalisation-Rules während der Annotation durchgeführt. dadurch wurde z.B. die Ersetzung einiger Symbole eingesetzt [77].
Zur Integration der geparsten sowie extrahierten Informationen in unsere Arbeit werden die Daten auch in einer JSON-Datei für die weitere Prozesse gesammelt. Dann werden alle die Daten mit anderen Daten aus anderen Projekten bzw. Datenquellen in der *Wert14*-App automatisiert.

2.4 Zusammenfassung

In unserem Konzept wird in erstem Schritt eine Methode gebraucht, damit die benötigte sowie aktuelle Attribute in einem JSON-Datei für weitere Prozesse gespeichert werden können. Darüber hinaus wurde es so geplant, dass die IR-Systeme

[15] `https://iate.europa.eu/home`, die Terminologie-Datenbank-API. Es wird für die Erfassung, Verbreitung und Verwaltung von EU-spezifischer Terminologie verwendet. ,aufgerufen am 11.09.2021
[16] `https://marcell-project.eu/links.html` aufgerufen am 11.09.2021
[17] `https://www.mongodb.com/de-de` aufgerufen am 11.09.2021
[18] `https://www.java.com/de/` aufgerufen 15.09.2021
[19] `https://www.python.org/` aufgerufen am 15.09.2021
[20] `https://www.nltk.org/` aufgerufen am 11.09.2021
[21] `https://spacy.io/` aufgerufen am 11.09.2021
[22] `https://textblob.readthedocs.io/en/dev/` aufgerufen am 11.09.2021

von Google sowie Gelbe-Seiten zum Startpunkt und Parsen der Attribute wie *den kompletten Namen* von Unternehmen, *Branche* sowie *Unternehmen-Website-URL* verwendet werden. Genau wie im Artikel "*Natural Language Processing Pipeline to Annotate Bulgarian Legislative Data*" [77] werden diese Attribute aus semistrukturiertem Format bzw. HTML-Seite von den genannten Suchmaschinen gescrapt bzw. gecrawlt.
Wenn wir hier durch den IR-Teil den Website-URL parsen können. Dann werden möglicherweise die komplette Body-Texte der Website aus HTML-Seite von notwendigen Seiten gesammelt.
In dem nächsten Schritt wird eine Methode benötigt, um die weitere notwendige Informationen aus dieser unstrukturierten Datei bzw. Texte der Seiten der Website zu extrahieren.
Da die Verwendung der zur Verfügung gestellten APIs meistens gegen Geld ermöglicht werden, wurde geplant, eine Kostenlose API zu finden. Darüber hinaus wurde am Anfang das "*GELECTRAQA*"-API von "Sahaj Tomar" in unserem Programm eingesetzt [69].
Aber das Problem ist, dass die kostenlose Version von diesem API langsam funktioniert. Auf der anderen Seite ist es notwendig, ein Rechner mit Graphics Processing Unit (GPU) zu haben. Außerdem konnten einige notwendige Informationen dadurch nicht extrahiert werden. Zum Beispiel während der Extraktion der Geschäftsführern-Name konnten einige Namen nicht extrahiert werden, oder zB. Bei der Extraktion der Telefon-Nummern sowie E-Mail-Adresse wird nur ein Attribut von jedem Attribut extrahiert, obwohl es mehr als ein Entity zum Extrahieren gibt.
Aus oben genannten Gründen wurde es geplant, ein Fine-Tuning bei einem Pre-Trained-Sprach-Modell durchzuführen.
Da das BERT-Sprach-Modell aktuell als Stand der Technik vorgeschlagen wurde, haben wir vor, das Fine-Tuning bei "bert_based_multilingual_cased" einzusetzen, da wie es geklärt wurde, funktioniert es auch bei Deutsch-Sprachige Texte.
Wie es bei einigen Artikels geklärt wurde, werden wir danach die Train-Evaluation durch Confusion-Matrix- sowie F1-Score-Metriken durchführen. Wir haben 150 Unternehmen-Daten zur Test-Evaluation. Dadurch werden wir prüfen, wie viele Genauigkeit bei jedem extrahierten Entity anhand unserer Test-Daten erreicht wird. Schließlich werden die Informationen nach der Evaluation in einer JSON-Datei für weitere Prozesse integriert. Auf eine genauere Erklärung über das Konzept der Arbeit werden wir im nächsten Kapitel eingehen.

3 Konzept

In diesem Kapitel wird über das Verfahren und die durchgeführten Schritte der Arbeit beschrieben. Die Programm-Pipeline wurde mit Python als Programmiersprache umgesetzt, wobei eine GPU von Google-Colab[1] für den IE-Teil bzw. zum Fine-Tuning des vortrainierten Sprach-Modells ausgenutzt wurde. Nach dem Training und Speichern des Modells wird es in der Pipeline zum Verwenden aufgerufen.

Zum Ablauf der Arbeit zur Extraktion und Integration heterogener, unstrukturierter und semistrukturierter Daten können folgende Punkte anhand unseres Konzepts erklärt werden:

- Information Retrieval
  Zum Parsen der Informationen aus semistrukturierten Datenquellen bzw. Suchmaschinen werden für jede einzelne Suchmaschine einen Scraper geschrieben, damit die notwendigen Informationen aus der HTML-Seiten der Suchmaschinen extrahiert werden können. Durch einen Crawler werden die Texte der Websites aus den notwendigen Seiten geparst.

- Information-Extraktion
  Um die Informationen aus den unstrukturierten Daten bzw. Texten zu extrahieren, wird das Fine-Tuning auf einem vortrainierten Sprach-Modell, hier z. B. BERT, durchgeführt. Zum Fine-Tuning des Modells wird ein annotiertes Datensatz benötigt. Zur Sammlung dieses Training-Datensatzes werden die Texte von Unternehmens-Website gesammelt. Nach der Sammlung, Tokenisierung sowie Annotation der Daten wird das Fine-Tuning auf dem vortrainierten Modell durchgeführt.

  Nach dem Training sowie die Entwicklung-Mode-Evaluation des trainierten Modells wird die Verfahren wie die Erkennung der Duplikaten sowie Test-Mode-Evaluation anhand dem Test-Datensatz durchgeführt, um zu prüfen, wie genau das Algorithmus funktioniert.

- Informationsintegration
  Nach der Evaluation des Modells werden alle geparsten sowie extrahierten Informationen aus der Suchmaschinen sowie die Texte der Webseiten in einem strukturierten JSON-Datei für weitere Prozesse umgewandelt

Wie es in Kapitel-1 beschrieben wurde, handelt es sich bei diesem Projekt um ein Programm, bei dem der Nutzer den Namen von einem Unternehmen bzw. Geschäft

[1] `https://research.google.com/colaboratory/faq.html` aufgerufen am 20.08.2021

eingibt, und die notwendigen Attribute als Ergebnis in einem strukturierten Datei-Format erhält.
Da wir in der Zukunft viele andere gesammelte Informationen eines Gebäudes von anderen Datenquellen neben diesen Attributen haben werden, werden die extrahierten Informationen in diesem Projekt für weitere Prozesse in einer JSON-Datei gespeichert. Abschließend werden die Informationen über Wert14-App zur Anwendung bereitgestellt.

3.1 WorkFlow

In Abbildung 3.1 wird gezeigt, wie die allgemeine Pipeline eingesetzt wird, und wie das Programm abläuft. Die Attribute, die als Ergebnis bei dieser Arbeit umgewandelt bzw. extrahiert werden, werden aus zwei semi-strukturierten Datenquellen bzw. HTML-Seiten von festgelegten Suchmaschinen sowie unstrukturierten Texten der Websites geparst bzw. extrahiert.

Die erwarteten Attribute aus den Suchmaschinen wurden wie folgt festgelegt:

- Branche des Unternehmens
- Kompletter Name des Unternehmens
- Website-URL des Unternehmens

Die erwarteten Attribute aus den Texten der Unternehmens-Website wurden wie folgt festgelegt:

- Name der Geschäftsführern
- Umsatzsteuer-ID-Nr.
- Anzahl der Mitarbeitern
- Anschrift
- Telefonnummer
- Faxnummer
- E-Mail-Adresse

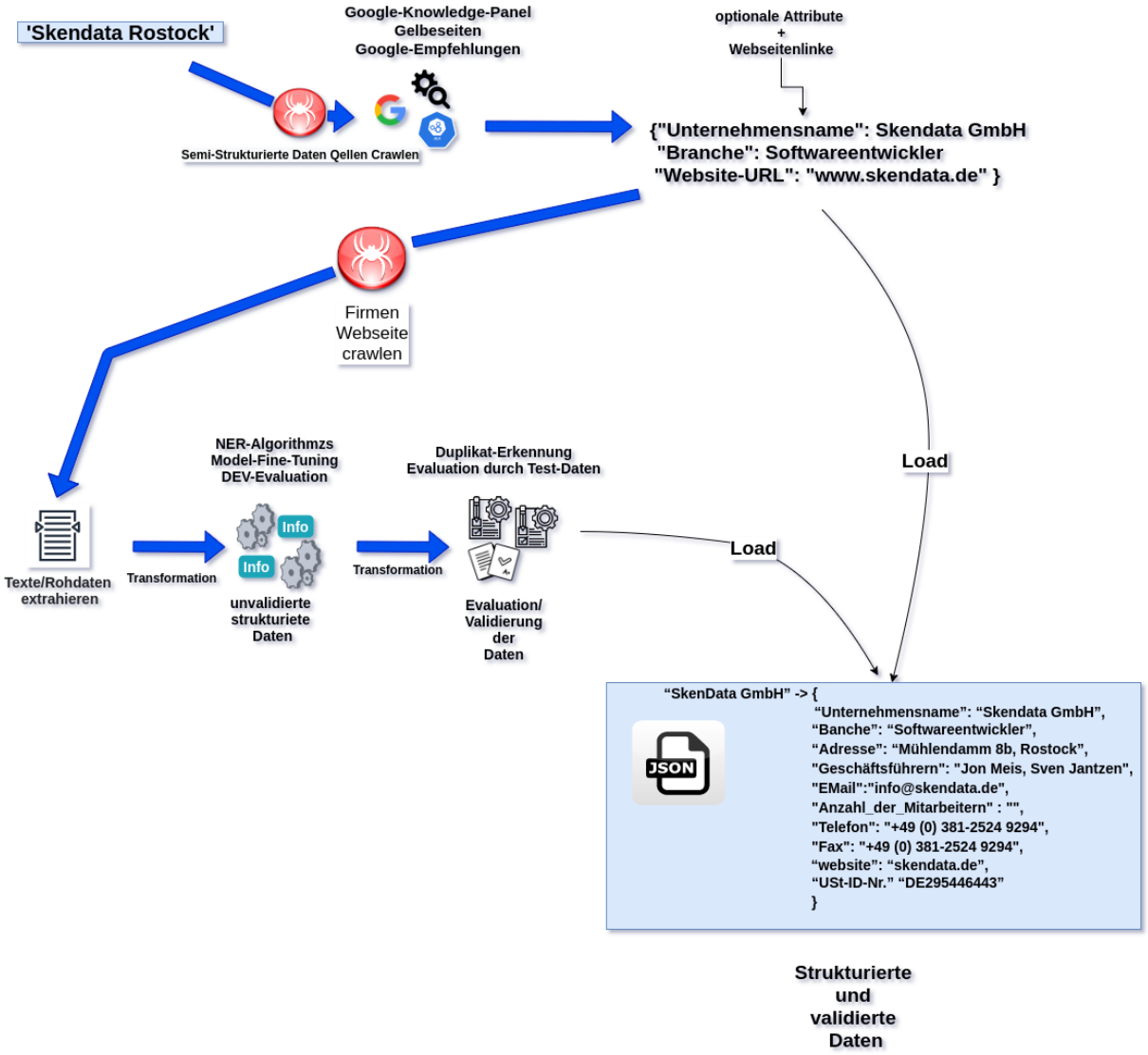

Abbildung 3.1: Allgemeiner Ablauf des Programms

Wie zu sehen ist, unterteilt sich das Programm in den drei unten genannten Schritten:

1) IR aus Suchmaschinen durch die eingesetzten Crawling-Systemen

2) IE aus von Website gesammelte, unstrukturierte Text-Daten und Evaluation der extrahierten Daten

3) Mapping aller gesammelten Daten in einer JSON-Datei

Im Rahmen der Arbeit wurden die Methoden aus dem Stand der Technik verwendet.

3.1.1 IR aus Suchmaschinen

Es gibt viele verschiedene Suchmaschinen, die man als ein IR-System ausnutzen kann, um die notwendigen Informationen zu sammeln und parsen. Zum Beispiel können folgende Suchmaschinen verwendet werden:

- *Open-Street-Map*[2]
- *graphhopper*[3]
- *DuckDuckGo*[4]
- *wego.here*[5]

Die Datenquellen bzw. Suchmaschinen, die für diese Pipeline festgelegt wurden, sind die Folgenden, sodass andere Suchmaschinen bei Bedarf in der Zukunft hinzugefügt werden können:

- Google-Knowledge-Panel
- Gelbe-Seiten
- Erste drei Empfehlungen der Google-Lokal-Suche

Das System funktioniert mittels vier Crawler-Systemen. Es wurde so entwickelt, dass den Crawler-Systeme nach der Eingabe des Nutzers gestartet werden. Im ersten Schritt werden die notwendigen Informationen, wie gezeigt in Abbildung 3.2, aus der HTML-Seite vom Google-Knowledge-Panel zu parsen.

[2] `https://wiki.openstreetmap.org/wiki/DE:Tag:office%3Dcompany` aufgerufen am 01.10.2021
[3] `https://graphhopper.com/maps/` aufgerufen am 01.10.2021
[4] `https://duckduckgo.com/` aufgerufen am 01.10.2021
[5] `https://wego.here.com/` aufgerufen am 01.10.2021

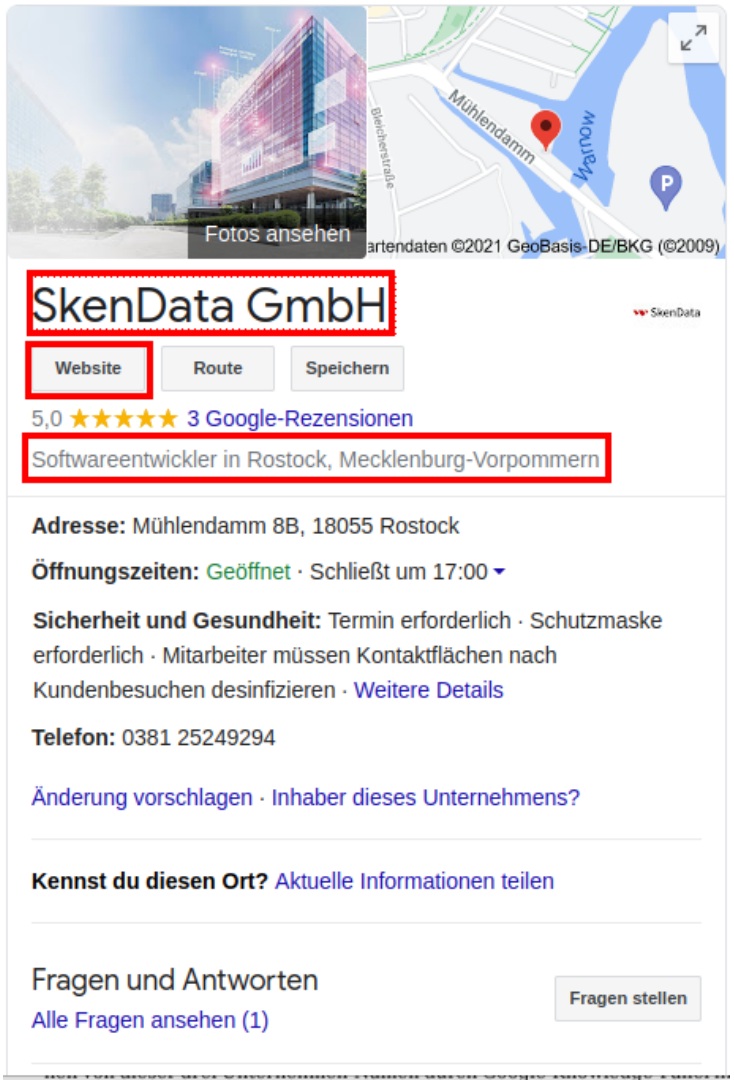

Abbildung 3.2: Rot markiert sind die notwendigen Infos im Google-Knowledge-Panel zum Parsen in unserer Pipeline[6]

Wenn es keine Informationen zum eingegebenen Unternehmensname gibt, werden die notwendigen Informationen aus den Gelben Seiten geparst. Falls in diesem Schritt nur eine Unternehmenswebseite gefunden wird, werden die Attribute der HTML-Seite der Gelbe-Seite gespeichert, ansonsten werden die ersten drei Ergebnissen von Gelbe-Seiten zurückgeliefert, siehe die Abbildungen 3.4 und 3.3.

[6]Mit Respekt auf Google für Bildschirmfoto, aufgerufen am 01.10.2021

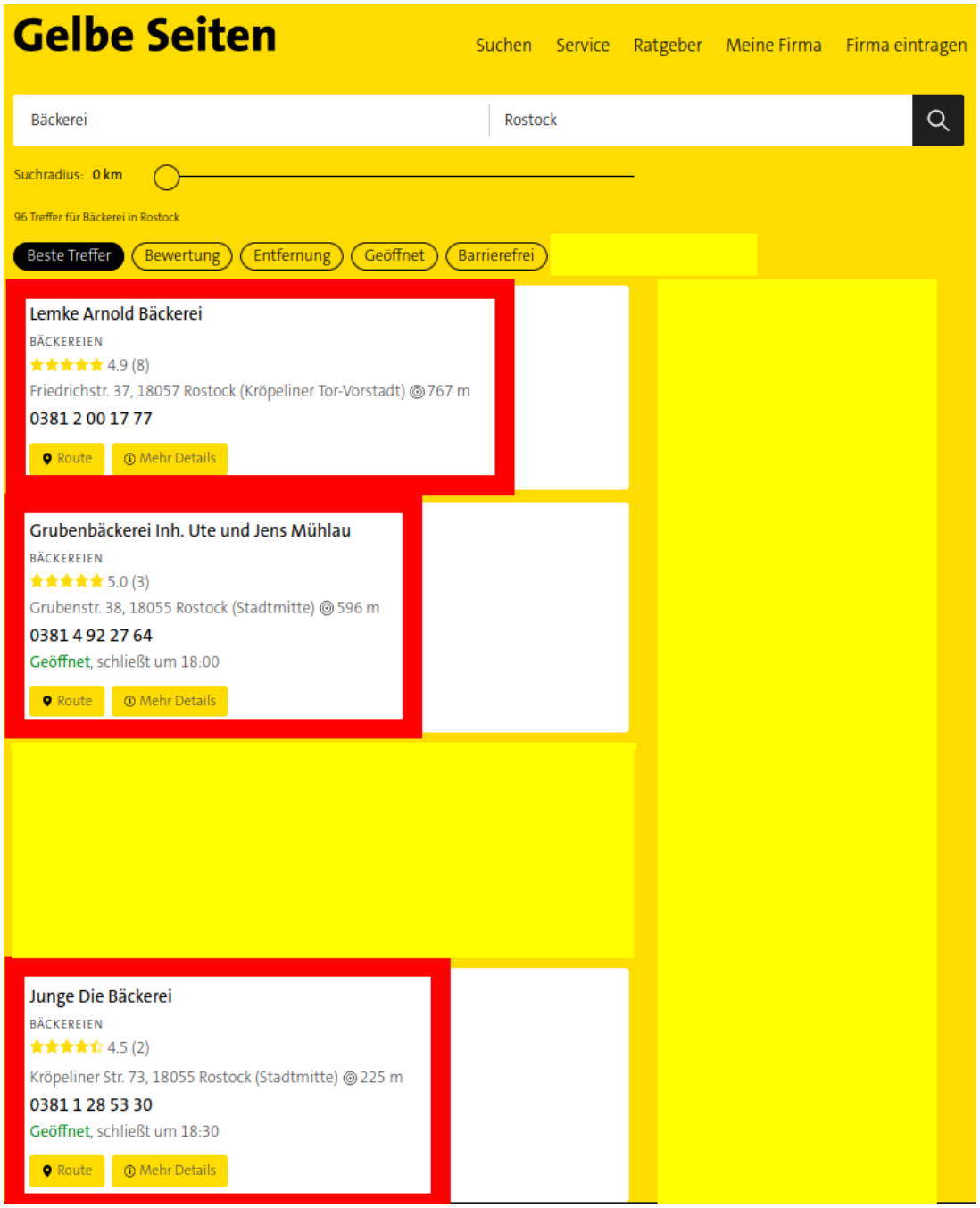

Abbildung 3.3: Rot markiert sind die die ersten drei gelieferten Ergebnissen von Gelbe-Seiten zum Parsen in der Pipeline[7]

[7] Mit Respekt auf Gelbe-Seiten für Bildschirmfoto, aufgerufen am 01.10.2021

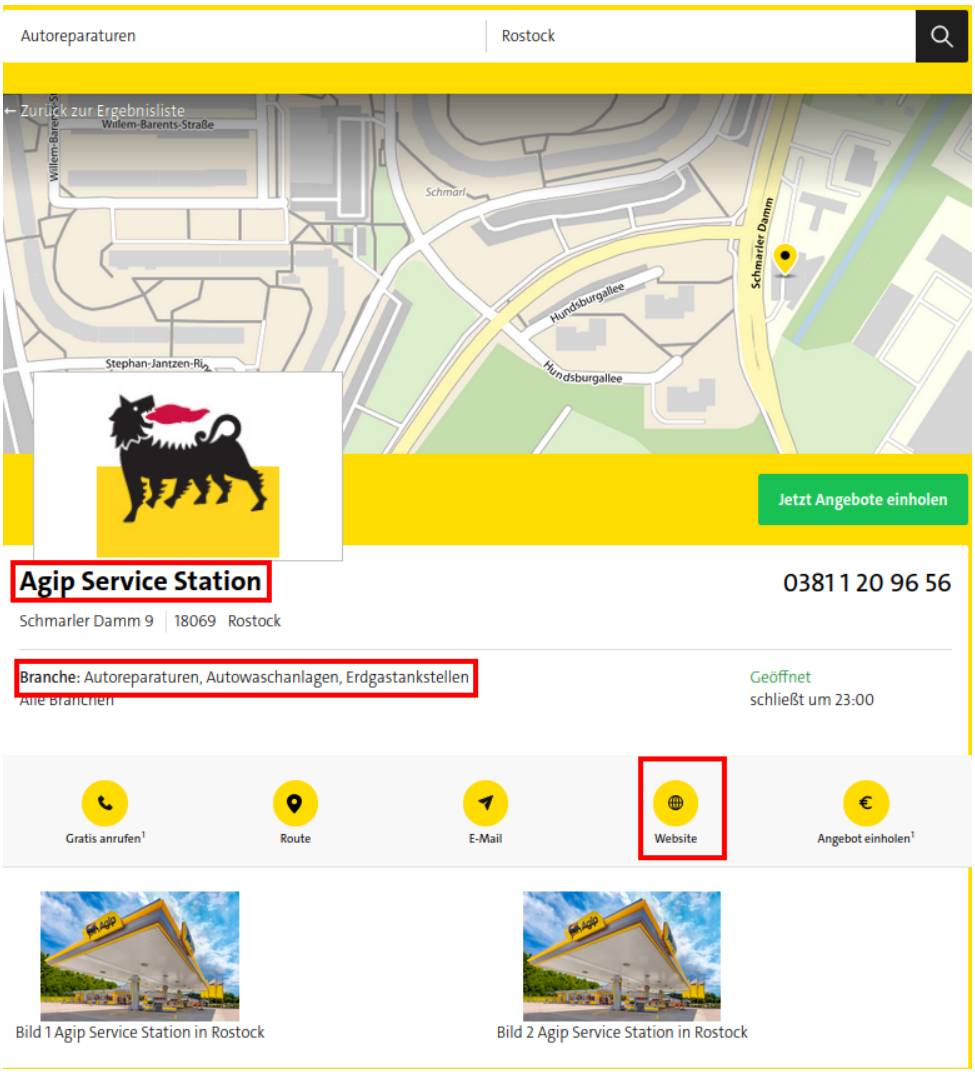

Abbildung 3.4: Beispiel für die drei notwendigen Informationen zum Extrahieren bzw. Speichern aus der Gelbe-Seiten-Suchmaschine[8]

Es kann auch möglich sein, dass es keine Informationen zur eingegebenen Unternehmen-Name in Gelbe-Seite steht. In diesem Fall werden die ersten drei Empfehlungsnamen von der Google-Lokal-Suche, wie gezeigt in Abbildung 3.5 gesammelt, dann werden die Informationen von diesen drei Unternehmensnamen möglicherweise durch das Google-Knowledge-Panel indexiert.

[8]Mit Respekt auf Gelbe-Seiten für Bildschirmfoto, aufgerufen am 01.10.2021

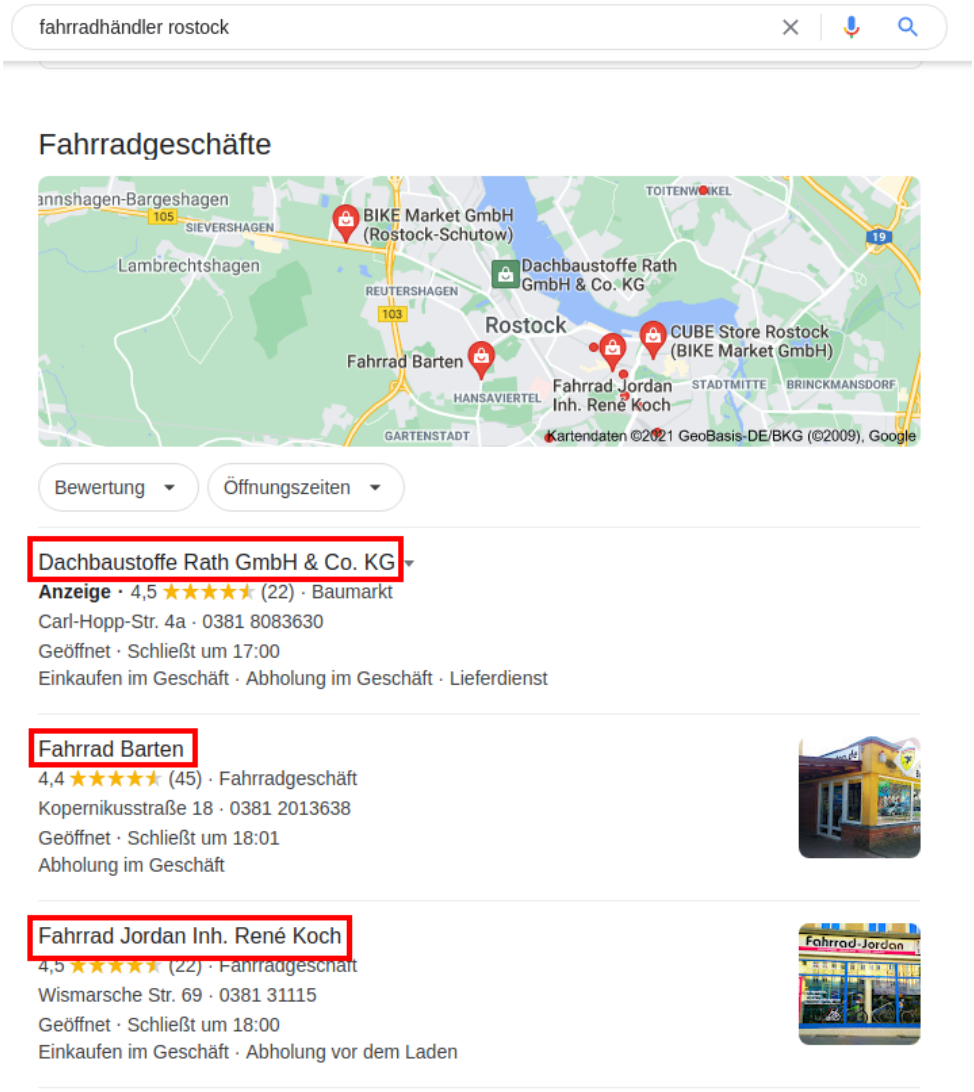

Abbildung 3.5: Beispiel für die ersten drei gelieferten Ergebnissen der Google-Lokal-Suche zum Parsen in der Pipeline[9]

Darüber hinaus werden erste drei Informationen aus Suchmaschinen durch die drei beschriebenen Crawler, wie gezeigt durch Abbildung 3.6, gesammelt.

[9]Mit Respekt auf Google für Bildschirmfoto, aufgerufen am 01.10.2021

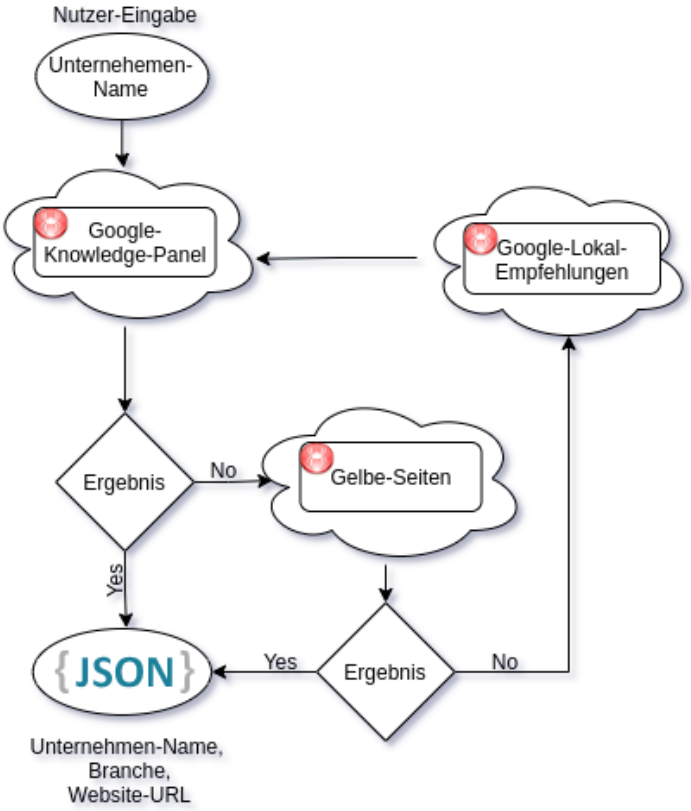

Abbildung 3.6: Crawler-Architektur zur Suchmaschinen-Informationen

Nach der Sammlung der genannten Attributen kommt ein anderer Crawler zum Einsatz, um die Texte aus den notwendigen Seiten der Webseite zu sammeln.
Die Informationen, die wir in diesem Projekt aus unstrukturierten Texten brauchen, findet man normalerweise in Impressum-, Über-Uns- oder Homepage-Seite der Webseiten von Unternehmen. Darüber hinaus wurde es so entwickelt, dass die Texte von Impressum- und Über-Uns-Seite gesammelt werden, und wenn keine Über-Uns-Seite vorhanden ist, dann wird die Texte von Homepage gesammelt. Da über die *Anzahl der Mitarbeiter* in entweder Über-Uns Seite oder in Homepage geklärt wird, wurde es so entschieden. Außerdem kann es bei einiger Webseiten passieren, dass dabei keine Impressum- oder Über-Uns-Seite eingesetzt wurde, deswegen wird es wie in Abbildung 3.7 versucht, die notwendigen Informationen möglichst aus Homepage-Seite zu extrahieren.

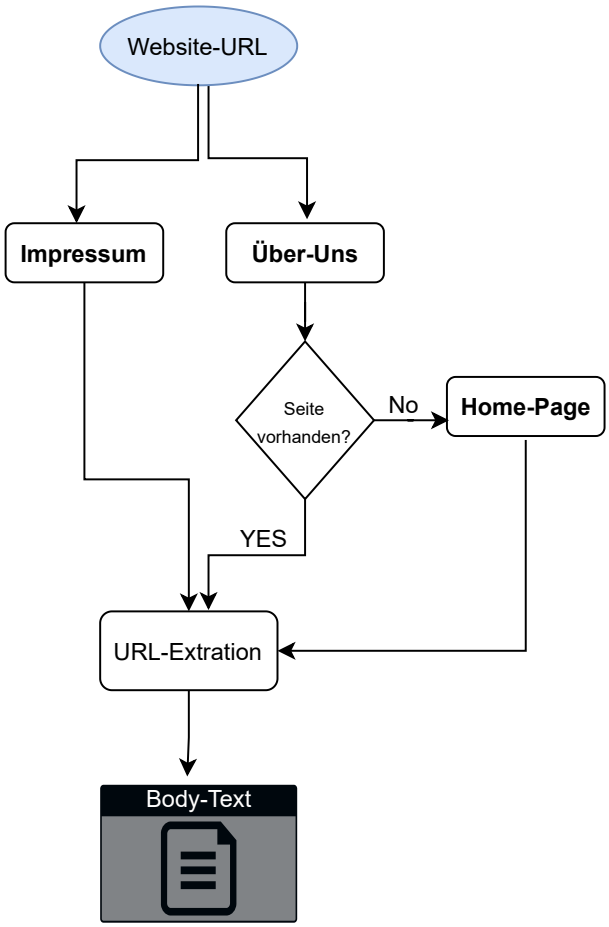

Abbildung 3.7: Crawler-Architektur zur Extraktion der Website-Texte

Nach der Sammlung der notwendigen Text-Dokumenten soll eine Methode zum Extrahieren der festgelegten Attribute bzw. Entitäten zum Einsatz kommen. In diesem Thema werden wir im Abschnitt 3.1.2 eingehen, und beschreiben, wie es gelöst wird.

3.1.2 IE aus der Website-Texten

Wie es im Abschnitt 3.1.1 beschrieben wurde, konnten die Texte der Website durch die Crawler-Systemen extrahiert werden.
Um die notwendigen Entitäten aus den gesammelten Texten zu extrahieren, wurde entschieden, eine BERT-pre-trained-Sprach-Modell zu trainieren bzw. dabei Fine-Tuning durchzuführen, da, wie es in Kapitel 2 beschrieben wurde, ist es der aktuell Stand der Technik und es konnten dadurch viele interessante Ergebnisse mit guten Evaluation-Ergebnissen erreicht werden.

Wie es vorher erwähnt wurde, brauchen wir eine Methode, damit deutschsprachige Entitäten extrahiert werden können. Darüber hinaus sollte einen Pre-Trained-Sprach-Modell ausgewählt werden, der bei deutsch-Sprache Texte funktioniert. Aus diesem Grund wurde "bert-base-multilingual-cased"[10] zum Fine-Tuning ausgewählt.
Fine-Tuning wird durch Google-Colab durchgeführt, da wir dabei GPU-Mode von Colab ausnutzen können.

Wie es im Kapitel 3.1.1 beschrieben wurde, wurde eine NER-Fine-Tuning-Aufgabe von Tobias Sterbak in seiner Arbeit durchgeführt [61]. Wie es von ihm in der "*MIT License*"[11] geklärt wurde, ist es erlaubt, seinen Source-Code zum Fine-Tuning zu verwenden. Deswegen wurde geplant, unseren NER-Task durch seine veröffentlichten Source-Code umzusetzen.

Zwischen der Arbeit von *Tobias Sterbak* und unserem Konzept gibt es die unten genannten Unterschiede:

- Verwendung bert-base-multilingual-cased, wobei bert-base-cased von ihm verwendet wurde, da seine Arbeit sich auf englischsprachige Texte bezieht [61].
- Annotation-Format des Training-Datensatzes
  Die verwendete Daten von Tobias Sterbak wurden durch *IOB*-Standard annotiert, wobei bei unserem Train-Datensatz diesen Standard nicht berücksichtigt wird.
- Training-Evaluation Methode
  Die Bibliotheken sowie die Metriken zur Evaluation bei unserer Arbeit sind unterschiedlich. Das wird in Kapitel 4 genauer beschrieben.
- Umwandelung und Test-Evaluation der extrahierten Informationen bzw. Entitäten durch Post-Prozess

Im Allgemeinen kann es so beschrieben werden, dass sich unsere Umsetzung in diesem Teil der Arbeit auf die unten genannten Unterpunkte bzw. Schritte unterteilt:

- Aufbereitung und Annotation des Training-Datensatzes
- Verwendung des BERT-Pre-Trained-Modells

[10] `https://huggingface.co/bert-base-multilingual-cased` aufgerufen am 20.09.2021
[11] `https://choosealicense.com/licenses/mit/` aufgerufen am 20.09.2021

- Aufbereitung der Sätze und ihre Annotation zum Fine-Tuning
- Einsetzen der Hyper-Parameter
- Trainieren und Evaluation des Modells
- Post-Prozess und Umwandlung der Daten in ein strukturiertes Format
- Evaluation durch die Daten von 150 Unternehmen

Aufbereitung und Annotation des Training-Datensatzes

Zur Sammlung des Training-Datensatzes wurde 230 Unternehmen-Websites durch unsere Crawler-Systeme gecrawlt, um die Texte zu extrahieren. Die Texte können unterschiedliche Struktur haben, aber als Beispiel sehen die notwendigen Attribute in Text wie in Abbildung 3.8.

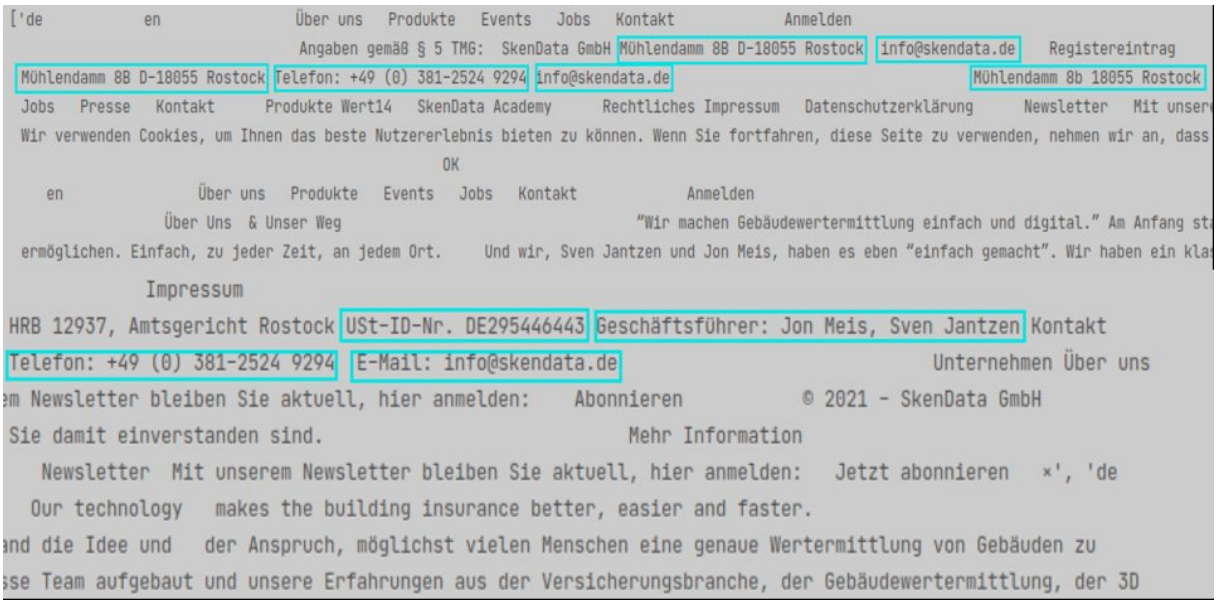

Abbildung 3.8: Beispiel für gesammelte Texte aus der Website und die notwendigen Attribute zur Extraktion

Nach der Sammlung der Texte werden die Texte tokenisiert, und in einem Excel-Datei zur Annotation gespeichert. Die Annotation wird Manuell eingegeben. Da die notwendigen Attribute, die wir brauchen, kommen normalerweise hintereinander wie in Abbildung 3.9, wurde es so geplant, keine "IOB"-Standard zu verwenden, da alle benötigte Tokens anhand unserer Anforderung hintereinander im Text kommen, und damit haben wir mehr Tokens zu annotieren, und dadurch kann unser

Algorithmus genauer trainiert werden, und funktionieren.

```
Anschrift: Mühlendamm 8B  D-18055 Rostock    Geschäftsführer: Jon Meis, Sven Jantzen
   O           LOC    LOC   LOC     LOC             O           BOSS       BOSS
```

Abbildung 3.9: Beispiel zur Annotation

Zur genaueren Beschreibung der Implementierung der Training-Datensatz-Aufbereitung wird in Kapitel 4 betrachtet.

Nach der Aufbereitung sowie Annotation des Datensatzes wird den BERT-Modell aufgerufen. In nächstem Schritt werden die Sätze und Ihre Annotationen in benötigtem Format als Input zum Trainieren in das Modell geschickt, die drüber mehr im Kapitel 4 beschrieben wird.
Nach dem Training und DEV-Evaluation wird dem Modell gespeichert und wieder aufgerufen. In diesem Schritt wird die Vorhersage durch Post-Prozessing durchgeführt, um danach die Test-Evaluation durchzuführen.
Nach der Evaluation, die im Kapitel 4 tiefer betrachtet wird, kommt das Modell in Pipeline zum Einsatz.

3.1.3 Mapping der Daten in ein JSON-Format

Wie es bereits erwähnt wurde, wurden die Attribute wie *Unternehmensname, Branche und Website-URL* aus dem semistrukturierten Format von Suchmaschine gesammelt. Nach der Sammlung der Informationen aus unstrukturierten Texten der Website in nächstem Schritt werden alle indexierte sowie validierte, extrahierte Informationen in einer strukturierten JSON-Datei zur weiteren zukünftigen Prozesse gespeichert.
In Kapitel 4 wird es geklärt, wie die Implementierung abläuft, und mit welchen Methoden die extrahierten Ergebnisse evaluiert werden.

3.2 Datenschutz

In unserer Umsetzung wurden die HTML-Seiten der Suchmaschinen, *Google* und *Gelbe-Seiten* zum Erreichen der Attributen gescrapt. Das wurde von Benjamin Bremert, Harald Zwingelberg [79] geklärt, dass es auch erlaubt wäre, wenn es sich um eine Forschungsarbeit handelt.
In der Regel sollte man aber darauf achten, dass das Scraping der Daten anhand

der Beschreibung von "*Text Mining öffentlich zugänglicher Daten*" [79] nur dann erlaubt ist, wenn man zum Erreichen der Informationen bei Bedarf entweder auf einer Plattform anmeldet oder eine API verwendet.
Infolge werden ein paar Gesetz-Paragrafen zum Web-Crawling sowie zur Verwendung der Personenbezogenen Daten beschrieben:

- "*Öffentliche Zugänglichmachung, § 19a UrhG*"
  "*Das Recht der öffentlichen Zugänglichmachung ist das Recht, das Werk drahtgebunden oder drahtlos der Öffentlichkeit in einer Weise zugänglich zu machen, dass es Mitgliedern der Öffentlichkeit von Orten und zu Zeiten ihrer Wahl zugänglich ist*" [79].

- "*§ 28 Abs. 2 Nr. 3 BDSG*"
  "*Die Übermittlung oder Nutzung für einen anderen Zweck ist zulässig, wenn es im Interesse einer Forschungseinrichtung zur Durchführung wissenschaftlicher Forschung erforderlich ist, das wissenschaftliche Interesse an der Durchführung des Forschungsvorhabens das Interesse des Betroffenen an dem Ausschluss der Zweckänderung erheblich überwiegt und der Zweck der Forschung auf andere Weise nicht oder nur mit unverhältnismäßigem Aufwand erreicht werden kann*" [79].

- "*§ 28 Abs. 1 Satz 1 Nr. 3 BDSG*"
  "*Das Erheben, Speichern, Verändern oder Übermitteln personenbezogener Daten oder ihre Nutzung als Mittel für die Erfüllung eigener Geschäftszwecke ist zulässig, wenn die Daten allgemein zugänglich sind oder die verantwortliche Stelle sie veröffentlichen dürfte, es sei denn, dass das schutzwürdige Interesse des Betroffenen an dem Ausschluss der Verarbeitung oder Nutzung gegenüber dem berechtigten Interesse der verantwortlichen Stelle offensichtlich überwiegt*" [79].

Anhand der Beschreibungen darf man die extrahierten sowie geparsten Informationen speichern, wenn die Daten öffentlich im Internet zur Verfügung gestellt wurden. Darüber hinaus wird es erlaubt, dass wir unseren Datensatz aus der öffentlich zur Verfügung gestellten Informationen im Internet gesammelt und aufbereitet haben.

4 Implementierung

In diesem Kapitel wird auf die Einzelheiten der Implementierung eingegangen. Dabei wird beschrieben, welche Tools bzw. Bibliotheken zur Umsetzung eingesetzt wurden und wie die Implementierung der Methoden erfolgte.

4.1 Allgemein

Wie es am Anfang des Kapitels 3 erwähnt wurde, wird die Implementierung der Pipeline durch die Programmiersprache Python umgesetzt, eine GPU von Google-Colab für den IE-Teil bzw. zum Fine-Tuning des vortrainierten Sprachmodells ausgenutzt wurde. Nach dem Training und Speichern des Modells wird es in der Pipeline zum Verwenden aufgerufen.
Wie beschrieben, unterteilt sich das Projekt auf die drei Unterpunkte IR, IE sowie Informationsintegration, die durch "*objektorientierte Programmierung (OOP)*"[1]-Struktur in der Pipeline umgesetzt wurden. Die komplette Umsetzung wurde unter Verwendung der async/await-Konzept geschrieben.

4.1.1 asyncio — Asynchronous I/O

Anhand der Dokumentation vom "*asyncio*" [80] in Python-Website wurde es als eine Bibliothek zur Entwicklung von parallelen Codes unter Verwendung der async/await-Syntax entwickelt, damit "asyncio" als Grundlage für mehrere asynchrone Python-Frameworks verwendet wird, die damit "*high-performance network*" bereitgestellt wird, damit viele Umsetzungen in Programm verteilt durchgeführt werden können [81].

[1] `https://www.python-kurs.eu/python_OOP.php` aufgerufen am 23.09.2021

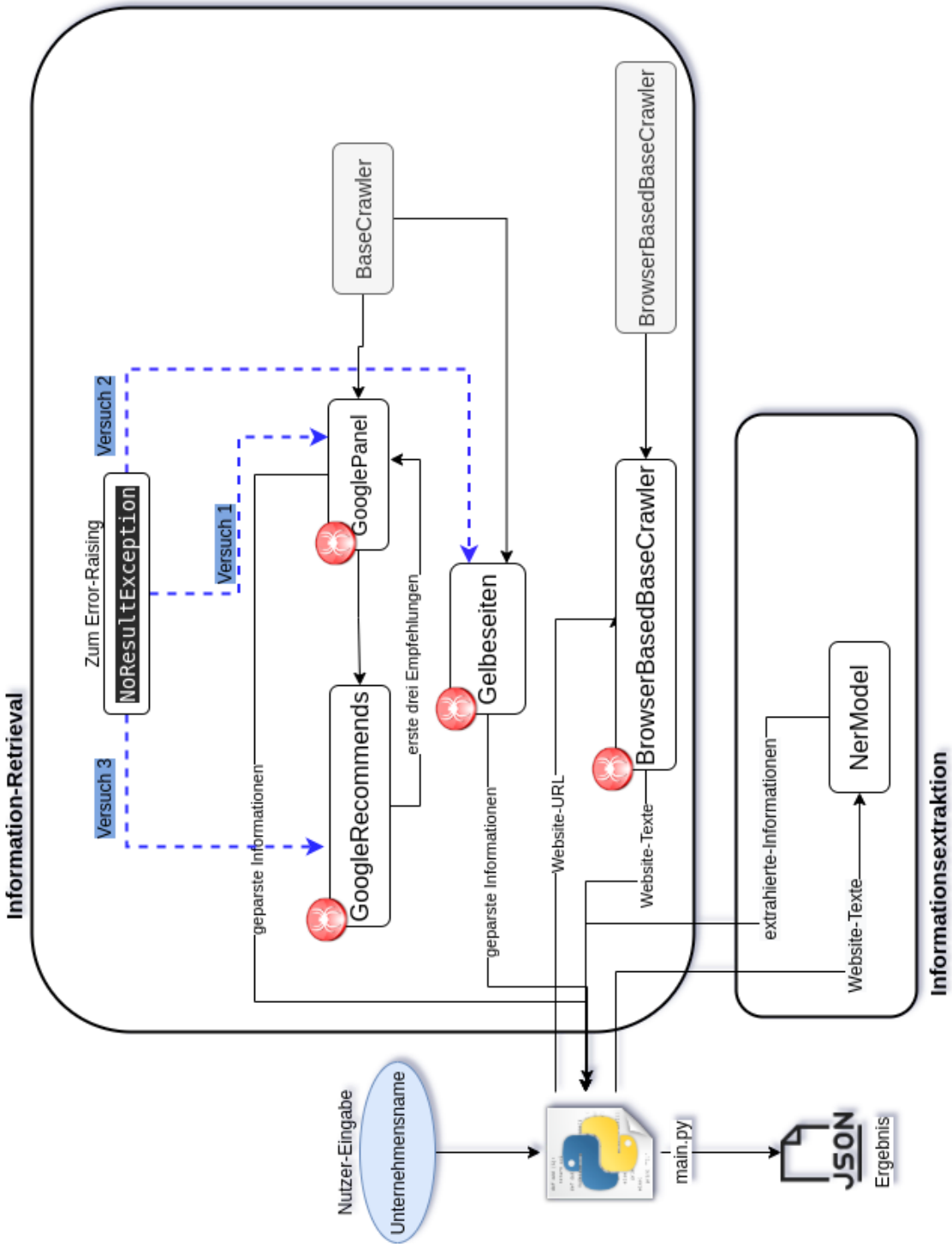

Abbildung 4.1: Verhältnis der wichtigen Klassen zueinander im Programm

4.2 Information-Retrieval

Der IR- bzw. Web-Crawling-Teil unserer Implementierung beinhaltet folgende wichtige Klassen:

- ***BaseCrawler***: Eine Basis-Klasse zum Scraping der festgelegten Datenquellen bzw. für unsere Suchmaschinen-Crawlern
- ***GooglePanel***: Zum Scraping bzw. Parsen der Informationen aus dem Google-Knowledge-Panel
- ***Gelbeseiten***: Zum Crawling bzw. zur Sammlung der Informationen aus Gelbe-Seiten
- ***GoogleRecommends***: Zum Scraping bzw. zur Sammlung der ersten drei Empfehlungsnamen von der Google-Lokal-Suche
- ***BrowserBasedBaseCrawler***: Eine Basis-Klasse für unseren Website-Crawler (***BusinessPlaceWebsite***)
- ***BusinessPlaceWebsite***: Zum Crawling bzw. zur Sammlung der notwendigen Texten der Website

Im ersten Schritt wird eine Anfrage auf der jeweiligen Suchmaschine geschickt. Dafür wurden verschiedene Bibliotheken eingesetzt.

Request-Bibliotheken

Anhand der Erklärungen im Buch *"Practical Web Scraping for Data Science: Best Practices and Examples with Python"*[13] wurde ein Programm entwickelt, damit dieses HTTP-Anfrage behandeln kann. Aber wie geklärt, *"sollte man das Rad nicht nochmal erfinden"* und ein Tool wieder mit gleicher Funktion nochmal entwickeln[13]. Darüber hinaus wurden viele verschiedene Bibliotheken vorgeschlagen, damit diese Aufgabe erledigt werden kann, sowie wie die Kommunikation mit dem HTTP-Server erfolgen kann. In [13] wird der Einsatz folgender Bibliotheken vorgeschlagen:

- *"urllib"*
- *"httplib2"*
- *"urllib3"*
- *"requests"*

- *"grequests"*
- *"aiohttp"*

Es wurde dazu beschrieben, dass "urllib" ein "*built-in Module*" [13] in "Python3" ist, sodass mit allen Schnittstellen in HTTP-Anfragen umgegangen werden kann. Trotzdem ist aber die "requests"-Bibliothek als ein einfaches Modul zum Verwenden vorgeschlagen wurde.
In "urllib3" wurden einige Module bezüglich HTTP-Anfragen verbessert. Trotzdem wurde es wie "urllib" auch nicht viel besser im Vergleich zu "requests" vorgeschlagen [13]. Anhand der Erfahrungen der anderen Nutzer auf "*stackoverflow*"[2] wurde auch die "requests"-Bibliothek als empfehlenswert eingestuft [82].
Infolge der genannten "HTTP" Anfragen-Bibliotheken wurden "grequests" und "aiohttp" genannt. Es wurde so geklärt, dass sie entwickelt wurden, um den HTTP mit Python asynchroner zu machen. Bei diesen zwei Bibliotheken können viele HTTP-Anfragen so schnell wie möglich gestellt werden [13].

4.2.1 *BaseCrawler*-Klasse

In unserer Umsetzung wird eine Basis-Klasse mit dem Namen *BaseCrawler* eingesetzt, bei der *aiohttp*[3] wegen der asynchron-Funktion verwendet wird. Dazu wird eine Funktion geschrieben, sodass diese *URL* als Eingabe bekommt, und schickt diese Anfrage an diesen URL. Wie angesprochen, wurde die Google-Suchmaschine so umgesetzt, dass ein Robot bzw. Crawler auf keine Informationen zugreifen kann. Dafür wird durch unsere Einsetzung ein *Header*[4] während dem Schicken der Anfrage verwendet.
Nach dem Schicken der Anfrage auf Header der Suchmaschine werden die Elemente bzw. Inhalte der HTML-Seite durch *Beautiful-Soup*-Bibliothek[5] geparst. Zum Verwenden von *Beautiful-Soup* sowie der eingesetzten Funktion von *aiohttp* haben wir eine Funktion, die als Eingabe einen URL erhält, und die Inhalte der HTML-Seite zurückliefert.

[2] `https://stackoverflow.com/company` *"Zur Ermöglichung der weltweiten Technologie-Entwicklung durch die Nutzung anderer Wissens wurde beschrieben."* aufgerufen am 25.08.2021
[3] `https://docs.aiohttp.org/en/stable/` aufgerufen am 24.09.2021
[4] `https://stackoverflow.com/questions/38619478/google-search-web-scraping-with-python` aufgerufen am 24.09.2021
[5] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/` aufgerufen am 24.09.2021

4.2.2 *GooglePanel-Klasse*

Diese Klasse erbt von der ***BaseCrawler**-Klasse*, um die *aiohttp*-Anfragen und *Beautiful-Soup*-Funktionen zu verwenden.
Die ***GooglePanel**-Klasse* wurde so umgesetzt, dass der *Untanehemensname* als Eingabe erhält. Nach dem Aufbau des Google-URLs von Unternehmen werden die erwarteten Attribute aus der HTML-Seite vom Google-Knowledge-Panel, wenn vorhanden, geparst. Dann wird das Ergebnis in einem Dictionary-Format für die weiteren Prozesse zurückgeliefert.
Zum Aufbau des URLs wird es anhand der, in Kapitel 2 beschriebenen, Erklärungen im Buch von Seppe vanden Brouckeund Bart Baesens [13], wie in Quellcodeausschnitt 4.1 gezeigt, durchgeführt.

```
1     def __init__(self, name: str):
2         self.name: str = name
3         self.main_url: str =
          f"https://www.google.com/search?q={'+'.join(urllib.parse.quote(self.name,
          encoding='utf-8').split())}"
```

Quellcodeausschnitt 4.1: Abfrage-String in Google-URL für Unternehmensname

Wie es in Quellcodeausschnitt 4.1 sichtbar ist, wird anhand der Erklärung von Paolo Moretti[6] die "*urllib.parse.quote*"[7] zur Behandlung der deutsch-sprachigen Sonder- und Satzzeichen wie "ß" in dem URL-String verwendet.
Zum Indexieren aller drei notwendigen Attributen wie den *kompletten Namen*, *Branche*, und *Website-URL* von Unternehmen wird erstmals in diesem Schritt versucht, ein Pattern in HTML-Seite von Google-Knowledge-Panel zu finden, damit wir jede einzelne notwendige Information parsen können.

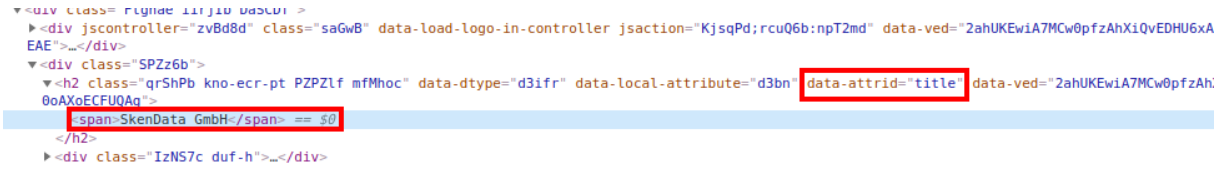

Abbildung 4.2: *Unternehemensname*-Adressierung durch sinnvolle Attribute in HTML-Seite von *Google-Knowledge-Panel*[8]. Den Unternehmensname wird hier anhand der *title*-Variable vom *data-attrid*-Attribut adressiert bzw. geparst.

[6] https://stackoverflow.com/questions/1695183/how-to-percent-encode-url-parameters-in-python aufgerufen am 24.09.2021
[7] https://docs.python.org/3/library/urllib.parse.html aufgerufen am 24.09.2021
[8] Mit Respekt auf https://www.google.com/search?q=skendata+GmbH für Bildschirmfoto

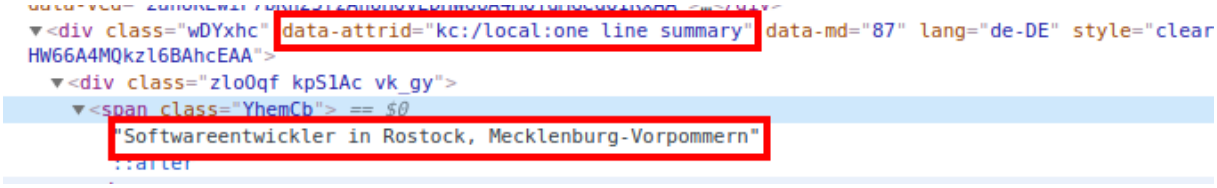

Abbildung 4.3: *Branch*-Adressierung durch sinnvolle Attribute in HTML-Seite von *Google-Knowledge-Panel*[9]. Die Branche wird hier auch anhand der Variablen vom *data-attrid*-Attribut adressiert bzw. geparst.

Abbildung 4.4: *Website-URL*-Adressierung durch sinnvolle Attribute in HTML-Seite von *Google-Knowledge-Panel*[10]. Den href-URL bzw. Website-URL wird hier auch anhand dem *ab button* vom *class*-Attribut in "a"-Tag adressiert bzw. geparst.

Wie es durch Abbildung 4.2 zu sehen ist, wird das Element-Attribute "*data-attrid='title'*" in "h2"-Tag gefunden. Danach wird den Name vom Unternehmen im "span"-Tag durch die "*lxml*"-Eigenschaft[11] von *Beautiful-Soup* indexiert bzw. geparst. Bei den anderen notwendigen Attributen wie *Branche* sowie der *Website-URL* wird es genauso, wie gezeigt in Abbildungen 4.3 sowie 4.4, durchgeführt.
Wenn alle notwendigen Attributen, wie geklärt, geparst sind, werden sie zusammen anhand der Lösung von "*JuniorCompressor*"[12] in einem Dictionary-Format gesammelt.

4.2.3 *Gelbeseiten-Klasse*

Diese Klasse vererbt sich auch von ***BaseCrawler**-Klasse*, um die *aiohttp*-Request- und *Beautiful-Soup*-Funktionen zu verwenden.
Bei Gelbe-Seite wird auch durch ein URL-Rule anhand des eingegeben Unternehmensname durchgeführt.

[9] Mit Respekt auf `https://www.google.com/search?q=skendata+GmbH` für Bildschirmfoto
[10] Mit Respekt auf `https://www.google.com/search?q=skendata+GmbH` für Bildschirmfoto
[11] `https://stackoverflow.com/questions/27790415/set-lxml-as-default-beautifulsoup-parser/41891285` aufgerufen am 24.09.2021
[12] `https://stackoverflow.com/questions/29736027/creating-json-from-lists-using-zip` aufgerufen am 24.09.2021

```
1     def __init__(self, name: str):
2         self.name: str = name
3         self.main_url: str =
          f"https://www.gelbeseiten.de/Suche/{'%20'.join(self.name.split())}"
```

Quellcodeausschnitt 4.2: Abfrage-String in Gelbe-Seite-URL für Unternahmenssname

Bei dieser Suchmaschine wurde es so umgesetzt, dass wenn sich mehr als, wie gezeigt durch Abbildung 3.3, erscheinen, wird versucht, die ersten drei Unternehmen-Ergebnisse zurückzuliefern.
Die Adressierung auf den Seite-URL jeder Unternehmensinformationen-Seiten in HTML-Seite von *Gelbe-Seite* wird durch die *Beautiful-Soup* und "*CSS-Selector*"[13] durchgeführt.

Abbildung 4.5: Übersicht von Hyperlink in Gelbeseiten von einem Unternehmensinformation-Seite-URL[14]. Durch bestehenden relativen URL von *data-realid* wird einen absoluten URL mithilfe des Gelbe-Seiten-URLs zum Treffen auf der Unternehmensinformation-Seite aufgebaut.

Wie es durch Abbildung 4.5 zu sehen ist, wird das "*data-realid*" adressiert, da das "*href*"-Attribut bei allen Unternehmens-Seiten nicht vorhanden ist.
Nach dem Parsing der relativen URL von der Unternehmensseite wird den absolute-URL aufgebaut. Dann werden alle URLs in einer Liste zurückgeliefert.
Eine andere Funktion wird zum Indexieren der drei notwendigen Unternehmensinformationen geschrieben, sodass diese die Inhalte der HTML-Seite erhält, und die Attribute wie den *Namen*, *Branche*, *Website-URL* des Unternehmens auch durch die Verwendung von *CSS-Selector* zurückgibt. Schließlich werden alle indexierten Attribute auch in einem Dictionary-Format gesammelt.

[13] `https://stackoverflow.com/questions/24801548/how-to-use-css-selectors-to-retrieve-specific-links-lying-in-some-class-using-be` aufgerufen am 24.09.2021
[14] Mit Respekt auf Gelbe-Seiten für Bildschirmfoto

4.2.4 *GoogleRecommends-Klasse*

Diese Klasse vererbt sich von ***GooglePanel**-Klasse*, sodass diese, wie erwähnt, auch von ***BaseCrawler**-Klasse* erbt.
Diese Klasse erhält einen "*Unternehmens-Namen*" als Eingabe. Nach dem Aufbau eines Links, wie vorherige Klassen durch den eingegebenen Namen, wird es versucht, die erste drei Unternehmensname-Empfehlungen von Google-Lokal-Suche, wenn wie die Abbildung 3.5 Vorhanden, zu indexieren.
Die Indexierung wird auch durch das Finden der sinnvollen Element-Attributen von HTML-Seite der *Google-Lokal-Suche* durchgeführt. Nach der Indexierung der *Unternehmens-Namen* durch die Vererbung von ***BaseCrawler**-Klasse* werden die drei notwendigen Attributen durch die Vererbung von ***GooglePanel**-Klasse* in einem Dictionary-Format zurückgeliefert.

4.2.5 *BrowserBasedBaseCrawler*-Klasse

Da in diesem Projekt die Texte der Webseiten zur Extraktion der Informationen aus verschiedenen Websites mit unterschiedlichen Strukturen gesammelt werden sollen, wird eine Methode benötigt, damit die Texte aller notwendigen Seiten der Websites gesammelt werden können. Da viele Websites asynchron-basiert[15] implementiert wurden, wurde es geplant, einen Browser zu laden, damit die kompletten Inhalte der HTML-Seite der Website geparst werden können.
Wie es durch Kapitel 2 beschrieben wurde, gibt es die Bibliotheken wie "Selenium"[16] sowie "Pyppeteer"[17] usw., die dazu zur Verfügung gestellt wurden. Da es beschrieben wurde, dass die Pyppeteer einen virtuellen Chrom-Browser lädt, und asynchron funktionieren kann, wird es in unserer Einsetzung verwendet. Darüber hinaus wird eine Funktion in dieser Klasse geschrieben, sodass nach dem Erhalten eines *URL*s ein Browser geladen wird.
Nach dem Laden des Browsers werden die Inhalte der HTML-Seite der getroffenen Seite durch die *Beautiful-Soup* geparst, um die weiteren notwendigen Informationen zu indexieren.
Zum Erhalten der Inhalte der HTML-Seite wird eine Funktion geschrieben, dass als Eingabe einen URL erhält, und drei unten genannten Ausgabe in einem Dictionary-Format zurückliefert.

[15] `https://ladezeit-optimierung.de/asynchrones-laden-von-ressourcen-und-inhalten/` Wie es geklärt wurde, kann die Ladezeit von Ressourcen und Inhalten durch die asynchron-Implementierung verbessert werden, und die Inhalte der Seiten werden dadurch nach dem Laden von "*Document Object Model (DOM)*" ausgeführt. aufgerufen am 24.09.2021
[16] `https://www.selenium.dev/selenium/docs/api/py/api.html` aufgerufen am 24.09.2021
[17] `https://miyakogi.github.io/pyppeteer/reference.html` aufgerufen am 24.09.2021

- Normales Format des HTML-Bodys: Es geht um das Laden der HTML-Inhalte der getroffenen Seite, sodass später die notwendigen Texte zur IE verwendet werden können.
- auf lower-case umgestelltes HTML-Body: Wie es beschrieben wurde, werden die Texte der *Homepage-*, *Über-uns-* sowie *Impressum*-Seite zur IE anhand der Texte von Tags gefunden bzw. geparst. Da diese Texte von Tags in beliebigen Formen von anderen Entwicklern geschrieben wurden, wird es versucht alle diese Texte auf *lower-case* umzustellen, damit die Links der Website durch weniger Zeile von Code gefunden und geparst werden können.
- current-URL: Es gibt Websites, die aus unterschiedlichen Gründen auf anderen URL umgeleitet (redirected) werden können, deswegen werden diese umgeleiteten URL-Adressen zum Treffen auf Website benötigt.

```
    async def html_body(self, url: str):
        #to load the browser
        await self.init_browser()
        try:
            await self.page.goto(url)
        except:
            #try again after 2 second
            await asyncio.sleep(2)
            await self.page.goto(url)
        #wait to completely loading of slowly-websites
        await asyncio.sleep(2)
        current_url = await self.page.evaluate("() => window.location.href")
        res = await self.page.evaluate("document.body.outerHTML", force_expr=True)
        soup_lower_case = BeautifulSoup(res.lower(), "html.parser")
        # to avoid to stick the words in html text
        res_for_context = res.replace("<br>", " ").replace("</br>", " ") \
                        .replace("<br/>", " ").replace("<b>", " ") \
                        .replace("<li>", " ").replace("</li>", " ") \
                        .replace("<span>", "  ").replace("</span>", "  ") \
                        .replace("</p>", " ").replace("<p>", " ") \
                        .replace("</a>", " ").replace("<h2>", " ") \
                        .replace("</h2>", " ").replace("</h3>", " ") \
                        .replace("<h3>", " ").replace("<h5>", " ") \
                        .replace("</h5>", " ").replace("<td>", " ") \
                        .replace("</td>", " ")
        soup = BeautifulSoup(res_for_context, "html.parser")
        return {"soup": soup, "soup_lower_case": soup_lower_case, "current_url":
        current_url}
```

Quellcodeausschnitt 4.3: Funktion zum Parsen der HTML-Inhalte in zwei Formen sowie die umgeleitete URL

Wie es durch Quellcodeausschnitt 4.3 zu sehen ist, wurde einen Variable als *res_for_context* definiert. Es geht darum, dass wir später die komplette Body-Texte zum Extrahieren der Entitäten sammeln werden. Durch diese Variable wird es ermöglicht, um das Zusammenhängen der Wörter der Texte zu verhindern.

4.2.6 *BusinessPlaceWebsite-Klasse*

Durch die drei beschriebenen Crawlern ist unser Programm in der Lage, die ersten drei notwendigen Attribute, wie *Name*, *Branche*, *Hompage-URL* der Website, aus semistrukturierten Datenquellen bzw. HTML-Seiten der Suchmaschinen zu extrahieren.
Wie es vorher beschrieben wurde, brauchen wir andere Attribute, die aus den unstrukturierten Texten der Unternehmen-Website extrahiert werden.
In diesem Schritt nutzen wir den von Suchmaschinen indexierten Website-URL aus, um die Texte des Unternehmens aus festgelegten Seiten der Unternehmens-Website zu sammeln. ***BusinessPlaceWebsite**-Klasse*, wird so funktioniert, dass es sich von ***BrowserBasedBaseCrawler***-Klasse vererbt, damit die Verwendung von *Pyppeteer*- sowie *Beautiful-Soup*-Bibliotheken möglich werden.
Wie gezeigt durch Quellcodeausschnitt 4.4, erhält diese Klasse einen *Website-URL* als Eingabe. Nach dem Crawling des eingegebenen Links liefert die Texte der notwendigen Seiten der Website zurück.
Wie es in Kapitel 3 beschrieben wurde, wird es versucht die Texte der *Impressum-*, *Über-Uns*-Seiten zu extrahieren.
Die Links der genannten Seiten werden anhand der Tag-Text in HTML-Seite gefunden. Dazu wird eine Funktion durchgeführt, sodass *eine Liste von Kewords* als Eingabe bekommt. Dann wird es versucht, den “a”-Tag über auf *lower_case* umgestellte HTML-Seite zu finden, der ein Wort auf dieser Liste in seiner *Tag-Text* beinhaltet. Wenn aus der gefundenen *href-URL* einen *Absolute-URL* aufgebaut werden soll, wird es anhand der Lösung von “*Cédric Julien*”[18] durch “*urljoin*”-Funktion von “*urllib.parse*”-Bibliothek[19] und den gelieferten *current_url* in Listing 4.3 durchgeführt. Seppe vanden Broucke und Bart Baesens [13] haben beschrieben, um verschiedene Seite auf eine Webseite zu treffen, braucht man einen “Absolute-URL” von einem “Relative-URL” aufzubauen, wie in 2.1 gezeigt. Wie geklärt, könnte bei einigen Webseiten “*base_url + relative_url*” die Lösung sein, aber bei einige Fällen um eine richtige Lösung zu haben kommt “*urljoin*” im Einsatz, um einen richtigen “Absolute-URL” einzurichten.

[18] `https://stackoverflow.com/questions/8223939/how-to-join-absolute-and-relative-urls` aufgerufen am 25.09.2021
[19] `https://docs.python.org/3/library/urllib.parse.html` aufgerufen am 25.09.2021

```python
    async def get_link_of_website_pages(self, keywords) -> Dict[str, str]:
        soup: Optional[BeautifulSoup] = await self.html_body(self.main_url)
        for key in keywords:
            items: ResultSet =
            soup.get("soup_lower_case").select(f'a:-soup-contains({key})')
            if items:
                href: str = items[0].get("href")
                # to make an absolute-url if needed
                if href:
                    href: str = href if href.startswith("http") else
                    urljoin(soup.get("current_url"), href)
                    return {"url": href, "name": self.main_url.split(".")[1]}
                break
        return {"note": "Any Link of Pages of Webpage is founded:("}
```

Quellcodeausschnitt 4.4: Funktion zum Parsen der *Seiten-URL* der Website anhand eingegebener *Keywords* wie z.B. *impressum* oder *über-uns*

Nach dem Finden der Seiten-URLS durch gezeigte Funktion in Quellcodeausschnitt 4.4 werden alle *body-Texte* der getroffenen Seiten gesammelt.
Da die komplette HTML-Inhalte auf *lower_case* umgestellt wurde, und, wie recherchiert, viele *Relative-URL* der Seiten mit dem Kapital-Form, wie in Abbildung 4.6, von anderen Entwicklern implementiert wurden, wird es dann erstmals mit dem gefundenen *Seite-URL* versucht, ob ein *response.status* nach dem *Request* als "200", mit der "*OK*"-Bedeutung[20], zurückkommt. Wenn es nicht zurückgeliefert wird, dann wird es Versucht, das gefundene *Keyword* auf großgeschriebenes Form umzustellen und weiter zu scrapen.

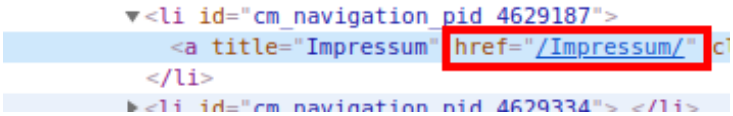

Abbildung 4.6: Beispiel zum großgeschriebenes Form implementiertem *Relative-URL*. Das keyword *Impressum* wurde bei dieser Website großgeschrieben, dann sollte die Anfrage auf den großgeschriebenes Form vom URL zum Scraping geschickt werden.

Als Ausgabe der Klasse wird es so implementiert, wenn keinen URL zur *Über-Uns*-Seite gefunden wird, dann werden die Texte von *Homepage* extrahiert. Da oft die notwendigen Attribute, hier besonders *Anzahl der Mitarbeiter*, auf *Über-Uns*-Seite gefunden werden können.

[20] `https://docs.aiohttp.org/en/latest/web_exceptions.html` aufgerufen am 25.09.2021

Nach der Sammlung der benötigten Texte der Website wird ein Algorithmus benötigt, um die weiteren notwendigen, aktualisierten Attributen aus der Texten zu extrahieren. In Folgende wird es beschrieben, wie ein Fine-Tuning bei BERT-Pre-trained-Sprach-Modell durchgeführt, evaluiert und in Pipeline aufgerufen wird.

4.3 Informationsextraktion aus der Website-Texten

Durch die vorherigen Schritten wurde es geklärt, wie die Texte der Website gesammelt werden. Wie es beschrieben wurde, wurde es geplant, ein NER-Algorithmus zu entwickeln, damit die bestehenden Informationen aus der Texten extrahiert werden können.
Dazu wird die von Tobias Sterbak [61] zur Verfügung gestellten Source-Code zum Fine-Tuning eines BERT-Pre-Trained-Sprach-Modells zum Einsatz kommen.
Wie es geklärt wurde, wird das Fine-Tuning auf *bert-base-multilingual-cased* durchgeführt, da es aktuell Stand der Technik, und bei der deutsch-sprachigen Texten funktionsfähig geklärt wurde.

4.3.1 Datensatz-Aufbereitung

Es wurde in Kapitel 3 erwähnt, dass einen Datensatz zum Fine-Tuning des Pre-Trained-Modells aufbereitet werden soll, sodass dadurch die notwendigen Tokens zum Training annotiert werden.
Zu diesem Zweck wurden die Texte von fast 230 Unternehmens-Websites durch unsere Crawler-Systeme gesammelt, tokenisiert und in Excel-Datei gespeichert. Während der Speicherung der Tokens in Excel-Datei wurden alle Tokens automatisch als “O” , wie Quellcodeausschnitt 4.5, annotiert. Nach der Speicherung wurden alle notwendigen Informationen manuell annotiert. wie die Anzahl der annotierten Tokens in Abbildung 4.8 zu sehen ist.
Zum Satz-Tokenization wurde die “*nltk.sent_tokenize*”[21] verwendet. Dann wurde am Ende jedes Satzes ein *##EOL##* zur Aufbereitung der *Eingabe* von NER-Algorithmus, wie gezeigt in Abbildung 4.7, eingegeben.

```
1     df = pd.read_excel('Firmen-Info.xlsx')
2     async def get_website_text():
3         for name in tqdm(df["Firmen Name"]):
```

[21] https://stackoverflow.com/questions/37605710/tokenize-a-paragraph-into-sentence-and-then-into-words-in-nltk aufgerufen am 25.09.2021

```
4            #to get website-url from search engine
5            for url in await get_website_link(name):
6                website_crawler_obj = BusinessPlaceWebsite(url)
7                context = await website_crawler_obj.extract_contexts()
8
9                tokenized_sentence = ''
10               res = nltk.sent_tokenize(str(text), language='german')
11
12               for sen in res:
13                   tokenized_sentence += sen +' ##EOL## '
14
15               df = pd.DataFrame({'token': str(tokenized_sentence).split(), 'tag':
                 'O'})
16               df.to_excel(f'data_sets//NER_dataset_{name}_{counter}.xlsx',
                 index=False)
17               counter+=1
```

Quellcodeausschnitt 4.5: Sammlung sowie Tokenizierung der Texte der Websiten, sowie das Eingeben des Default-Tags als 'O' für die extrahierten Tokens, und das Speichern in einer Excel-Datei zum Aufbau des Training-Datensatzes

1204	Es	O
1205	gilt	O
1206	ausschließlich	O
1207	das	O
1208	Recht	O
1209	der	O
1210	Bundesrepublik	O
1211	Deutschland.	O
1212	##EOL##	O
1213	www.99-euro-computer.de	O
1214	Daniel	O
1215	Bruhn	O
1216	Enzianstraße	LOC
1217	8	LOC
1218	22297	LOC
1219	Hamburg	LOC
1220	Telefon:	O
1221	040/	TELL
1222	43270672	TELL
1223	Druckversion	O
1224	|	O
1225	Sitemap	O
1226	©	O
1227	Daniel	O
1228	Bruhn	O
1229	WebansichtMobile-Ansicht	O
1230	Logout	O

Abbildung 4.7: Übersicht für das Beispiel der Annotation

Durch die Einsetzung wurden 231299 Tokens annotiert, sodass die Anzahl bei jedem eingegebenen Label wie in Abbildung 4.8 aussieht.

0	222996
LOC	2971
TEL	2541
BOSS	978
FAX	825
MAIL	527
TaxID	360
NumEMPLOYEE	101

Abbildung 4.8: Anzahl bei jedem einzelnen Label

Da es viele deutsche Abkürzungen beliebig im Text gab, die in der Regel bei deutsch-sprachigen Texten einen Punkt nach der Abkürzung kommt, wie zum Beispiel *Inh.*, *Tel.*, *Dipl.-Ing.*, *str.* usw., wurde es geplant, um eine Funktion, wie geklärt im Artikel von "*Svetla Koeva et al.*"[77], zum Normalisation-Rules zur Erhöhung der Genauigkeit zu schreiben, damit das komplette Wort auf seine Abkürzung ersetzt wird. Dazu wurden diese Regeln in einer *statische-Methode*[22] neben den Regeln zur Reinigung der Texten geschrieben.

Wie es zum Fine-Tuning des Sprach-Modells von *Tobias Sterbak* [61] geklärt wurde, wird den aufbereiteten Datensatz geladen. Nach der Verwendung des *bert-base-multilingual-cased*-Modells werden alle Sätze und ihre Label zum Training aufbereitet. Danach wird die Hyper-Parameter unseres NER-Tasks eingerichtet. In nächstem Schritt wird die Train-Evaluation durchgeführt. Schließlich wird den trainierten Modell zum *Post-Processing* sowie Aufrufen in der Pipeline gespeichert.

4.3.2 Laden des Datensatzes

Nach der Installation sowie dem Importieren der notwendigen Packages, wie "*Transformers*"[23], "*Pandas*"[24], "*numpy*"[25], anhand der Arbeit von *Tobias Sterbak*, werden wir den Datensatz laden [61].

[22] `https://docs.python.org/3/library/functions.html` aufgerufen am 26.09.2021
[23] `https://huggingface.co/transformers/` aufgerufen am 26.09.2021
[24] `https://pandas.pydata.org/docs/user_guide/index.html` aufgerufen am 26.09.2021
[25] `https://numpy.org/doc/stable/user/` aufgerufen am 26.09.2021

```
    sentence =[]
    label = []
    sentences = list()
    labels = list()

    for _, row in train_df.iterrows():
        if row['Word'] == '##EOL##':
            sentence.append(row['Word']) #['A','B','C','D','##EOL##']
            label.append(row['Tag'])
            sentences.append(sentence)
            #[['A','B','C','D','##EOL##'],['A','B','##EOL##']]
            labels.append(label)
            sentence = []
            label = []
        else:
            sentence.append(row['Word'])
            label.append(row['Tag'])

    tag_values = train_df.Tag.value_counts().index.tolist()
    tag_values.append("PAD")
    tag2idx = {t: i for i, t in enumerate(tag_values)}
```

Quellcodeausschnitt 4.6: Aufbereitung der Sätzen sowie Labeln

Nach dem Laden des Datensatzes, wie es in Quellcodeausschnitt 4.6 zu sehen ist, werden die Sätze und Labeln in dem gezeigten List-Formen gesammelt. Eine Liste von Labeln wird als *Tag_Values* aufgebaut. Das *'PAD'* wird in *Tags-Liste* eingegeben. Dann wird für jeden *Tag* eine Nummer festgelegt [61]. Nach diesem Schritt werden die Sätze und Labeln genau wie den Einsatz von *Tobias-Sterbak* in *PyTorch*[26] durchgeführt [61]. Die Parameter bzw. Hyper-Parameter, die von ihm eingerichtet wurden, wurden wie im Folgenden berichtet, anhand [61]:

- maximale Länge der Sätze auf 75
- batch_size auf 32
- Epoch auf 3
- Adam-Optimizer wurde mit eingerichtetem Learning-Rate auf 3e-5 verwendet.
- "max-grad-norm" auf 1.0 zum Verhindern des Explodieren von Gradient

Bei uns werden aber die genannten Parameter wie folgt eingerichtet:

- maximale Länge der Sätze auf 200, da wir längere Sätze im Text haben.

[26] `https://pytorch.org/docs/stable/index.html` aufgerufen am 26.09.2021

- batch_size auf 32
- Epoch auf 8
- Adam-Optimizer wurde mit eingerichtetem Learning-Rate auf 5e-5, vorgeschlagen wurde von Turc et al. [83], verwendet.
- “max-grad-norm” auf 1.0 zum Verhindern des Explodieren von Gradient

Die Epoche wurde am Anfang auf 15 eingesetzt. Durch die Visualisierung, anhand der Methode von *Tobias Sterbak* [61], mithife der *Matplotlib*[27]- und *Seaborn*[28]-Bibliotheken ist es sichtbar, dass das Netzwerk sich nach der Epoche 8 nicht wesentlich verbessern kann, sehe Abbildung 4.9.

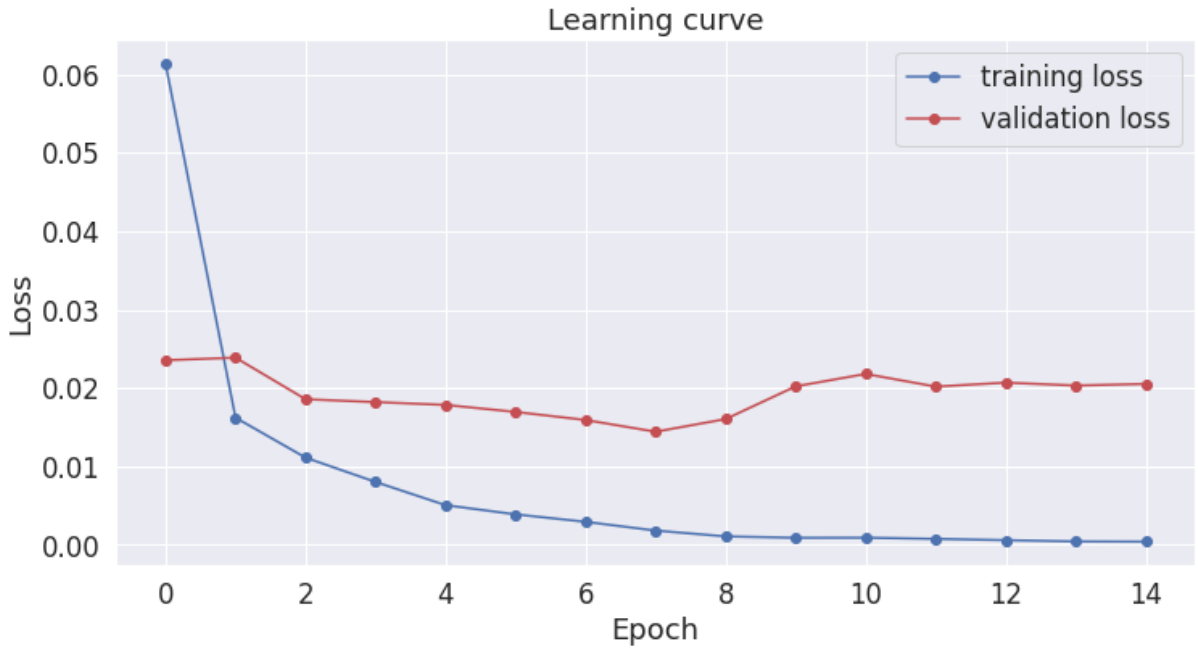

Abbildung 4.9: Learning-Curve-Übersicht bis Epoche 15, gerechnet nach der Methode in [61]

4.3.3 Train-Evaluation

Bei der Einsetzung von *Tobias Sterbak* wurden 10% der Daten zur Evaluation und 90% der Daten zum Training festgelegt, wobei bei unserer Arbeit werden 20% der Daten zur Evaluation und 80% zum Training eingerichtet. Es wurde von

[27] `https://matplotlib.org/stable/contents.html` aufgerufen am 26.09.2021
[28] `https://seaborn.pydata.org/` 26.09.2021

Tobias so eingesetzt, dass nach jedem Epoch, eine Genauigkeit-Evaluation durch "*seqeval.metrics*" [62] angezeigt wurde. Dieses Verfahren wird in unserem Konzept berücksichtigt, aber mit dem Unterschied, dass bei uns die Evaluation, anhand der Arbeit von *Charlene Chambliss* [59], durch die Metriken wie "*confusion_matrix*"[29] sowie "*classification_report*"[30] von "*scikit-learn*"-Bibliothek[31] gerechnet werden [61].
Die Evaluation-Ergebnissen sind durch die Abbildungen 4.10 für *Confusion-Matrix* sowie 4.11 für *Classification-Report* zu sehen.

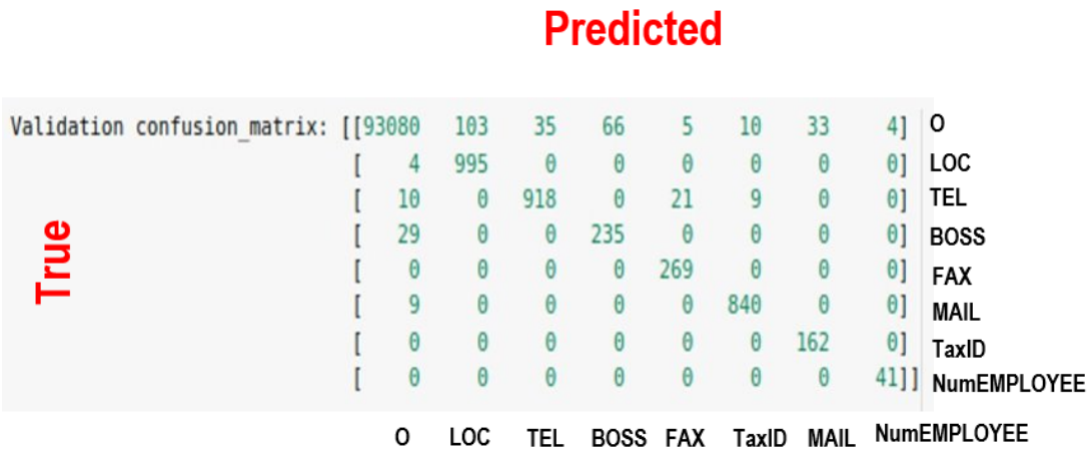

Abbildung 4.10: Confusion Matrix, gerechnet nach der Methode in [59]

Validation classification_report:	precision	recall	f1-score	support
BOSS	0.78	0.89	0.83	264
FAX	0.91	1.00	0.95	269
LOC	0.91	1.00	0.95	999
MAIL	0.98	0.99	0.98	849
NumEMPLOYEE	0.91	1.00	0.95	41
O	1.00	1.00	1.00	93336
TEL	0.96	0.96	0.96	958
TaxID	0.83	1.00	0.91	162

Abbildung 4.11: Classification Report, gerechnet nach der Methode in [59]

[29] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html aufgerufen am 21.09.2021
[30] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html aufgerufen am 21.09.2021
[31] https://www.kite.com/python/docs/sklearn aufgerufen am 21.09.2021

4.3.4 Post-Prozesses

Durch die veröffentlichen Code von *Tobias Sterbak* [61], konnten wir nach der Behandelung der tokenizierten Sätzen durch *bert-base-multilingual-cased* das Ergebnis, wie in Abbildung 4.12 bei einem Test-Satz erhalten.

Da bei den extrahierten Texten aus verschiedenen Websites durch unsere Crawler-Systeme längere Texte in dem Modell zur Vorhersage eingehen, wobei, wie auch erwähnt von Mahmoud, "*der BERT eine maximale Länge von Token=512 akzeptiert*"[32]. Darüber hinaus wird es so implementiert, dass wir die Sätzen durch genannte *nltk.sent_tokenize* tokenizieren, bevor die Sätzen in den Modell gehen. Aber da wir alle *Body-Texte* sammeln, wurde es gezeigt, dass wir immer noch die Texte länger als 512 haben, dass keine Punkte zum Tokenisieren haben. Deswegen wurde es geplant, wenn solche Texte nach dem *sent_tokenize* durch *nltk* gefunden sind, werden die Segmentierung auf 512-Länge bei diesen Sätzen durchgeführt, und danach in dem Modell eingehen.
Die extrahierten Attribute werden durch eine Pre-Prozess-Methode in der Listen gesammelt.

[32] `https://stackoverflow.com/questions/58636587/how-to-use-bert-for-long-text-classification` aufgerufen am 26.09.2021

```
O       -
O       ID
O       -
O       Nr
O       .
TaxID   DE295446443
O       Geschäftsführer
O       :
BOSS    Jon
BOSS    Meis
BOSS    ,
BOSS    Sven
BOSS    Jantzen
O       Kontakt
O       '
O       '
LOC     Mühlendamm
LOC     8B
LOC     D
LOC     -
LOC     18055
LOC     Rostock
O       Telefon
O       :
TEL     +
TEL     49
TEL     (
TEL     0
TEL     )
TEL     381
TEL     -
TEL     2524
TEL     9294
O       '
O       '
MAIL    info
MAIL    @
MAIL    skendata
MAIL    .
MAIL    de
LOC     Mühlendamm
LOC     8b
LOC     18055
LOC     '
LOC     '
LOC     Rostock
O       Telefon
O       :
TEL     +
TEL     49
TEL     (
TEL     0
TEL     )
TEL     381
TEL     -
TEL     2524
TEL     9294
O       E
O       -
O       Mail
O       .
O       :
```

Abbildung 4.12: Die Vorhersage anhand der *Tobias Sterbak*-Source-Code nach [61]

Duplikateerkennung

Bei der Ausgabe von Modell wurde es gezeigt, dass wir Duplikate bei den extrahierten Attributen haben. Dazu wurde eine Methode eingesetzt, dass eine *Liste von Attributen* als Eingabe erhält. Nach dem Finden der Duplikate zwischen den Attributen wird eine gefilterte Liste zurückgeliefert. Diese Duplikate-Behandlung wird bei unserer Einsetzung durch "*Levenshtein Distance*"[33] nach der Lösung von Jean-François Corbett[34] gehandelt.

```
1     @staticmethod
2     async def filter_duplicate(list_of_strings):
3         filtered_list = []
4         for i in range(len(list_of_strings)):
5             flag = True
6             for j in range(i + 1, len(list_of_strings)):
7                 if Levenshtein.distance(list_of_strings[i], list_of_strings[j]) < 1:
8                     flag = False
9             if flag:
10                filtered_list.append(list_of_strings[i])
11        return filtered_list
```

Quellcodeausschnitt 4.7: Duplikate-Behandlung durch die *Levenshtein-Distance*-Verwendung

4.3.5 Speicherung des Modells

Bei der Arbeit von *Tobias Sterbak* wurde das Modell nicht gespeichert [61]. Da es aber ein Teil unserer Arbeit gerechnet wird, müssen wir das Modell zum Aufrufen in der Pipeline speichern. Die Speicherung sowie das Laden des Modells in *Google-Colab* werden anhand der Lösung von "*dontloo*"[35] sowie "*shwin Geet D'Sa*"[36] durchgeführt. Außerdem wurde es in Post-Prozess-Teil der Source-Code von *Tobias* so eingesetzt, dass vorher gespeicherte Dictionary-Format von *tag2idx*-Variablen (Zum Festlegen einer Nummer für jeden Label in Dictionary-Format), wieder zur Vorhersage aufgerufen werden. Darüber hinaus ist es erforderlich die *Keys* dieses

[33] `https://pypi.org/project/python-Levenshtein/` aufgerufen am 26.09.2021
[34] `https://stackoverflow.com/questions/21276721/find-near-duplicates-of-comma-separated-lists-using-levenshtein-distance` aufgerufen am 26.09.2021
[35] `https://stackoverflow.com/questions/42703500/best-way-to-save-a-trained-model-in-pytorch` aufgerufen am 26.09.2021
[36] `https://stackoverflow.com/questions/58417374/how-to-load-the-saved-tokenizer-from-pretrained-model` aufgerufen am 26.09.2021

Variables zu indexieren. Die Speicherung dieses Objekts wird durch die "*pickle*"[37] behandelt.
Schließlich wird unseren Modell in der Pipeline gespeichert. Dazu wird eine Klasse als ***NerModel**-Klasse* geschrieben, sodass eine *context* als Eingabe erhält. Dann werden die extrahierten Attributen aus dem Kontext in einem Dictionary-Format zurückliefern, sodass das Type von Values in diesem Objekt eine Liste von Attributen ist.

Durch die beschriebenen Punkte der Arbeit sind wir in der Lage, alle notwendigen Attributen, wenn vorhanden, aus der Texte der Website zu extrahieren. Dazu werden wir unsere Crawling-Systeme und NER-Methode in einer Pipeline bringen. Darauf werden wir in nächstem *Informationsintegration*-Teil eingehen.

4.4 Informationsintegration

Wie es erwähnt wurde, sollen jetzt alle extrahierten Attributen aus semistrukturierten Datenquellen durch die Crawling-Systeme sowie unstrukturierten Texten der Website in einem strukturierten Format gesammelt werden.
Da die extrahierten Attribute durch diese Masterarbeit-Projekt mit anderen Attributen aus anderen Datenquellen für Endbenutzer durch *Wert14*-APP visualisiert werden sollen, wurde es geplant, alle diese Informationen in einem JSON-Format für die weiteren Prozesse in der Zukunft zu speichern.

Dazu werden für jede Suchmaschinen-Crawling-Systeme eine Funktion mit dem Namen als *resolve* geschrieben, sodass die ersten drei notwendigen Attributen in einem Dictionary-Typ in einer Liste zurückliefern, wenn es die Informationen des gesuchten Unternehmens bei der Suchmaschine gibt, ansonsten wird eine Liste ohne keine Inhalte zurückgegeben, dass in diesem Fall bei der Funktion ein *NoResultException* geraist wird.

Dann werden diese eingesetzten Klassen in der *main.py*-Datei von unserem "*Virtual Environments*"[38] unseres Programms durch eine Funktion mit dem Namen als *resolve* instanziert und iteriert, wie in Quellcodeausschnitt 4.8 gezeigt, um die ersten drei Attribute des Unternehmens von den Suchmaschinen zu haben.

```
1     async def resolve(query):
```

[37] `https://python.readthedocs.io/en/stable/library/pickle.html` aufgerufen am 26.09.2021, *Das wurde so beschrieben, dass "Pickling" der Prozess ist, eine Python-Objekthierarchie in einen Byte-Stream konvertiert wird.*
[38] `https://docs.python-guide.org/dev/virtualenvs/` aufgerufen am 26.09.2021

```
2         for resolver in (GooglePanel(query), Gelbeseiten(query),
          GoogleRecommends(query)):
3             try:
4                 #Color-Class to print and show the info of data-source in color
5                 if await resolver.resolve():
6                     print(f"{Color.GREEN}Extracted INFO from
                      {Color.YELLOW}{resolver.__module__} {Color.GREEN}for
                      {Color.YELLOW}{resolver.name} :) {Color.END} ")
7                 return await resolver.resolve()
8             except NoResultException:
9                 print(f"{Color.CYAN}NO INfO from {Color.RED}{resolver.__module__}
                  {Color.CYAN}for {Color.RED}{resolver.name} :/ {Color.END}")
10                pass
```

Quellcodeausschnitt 4.8: Funktion zum Instanziieren und Iterieren der Suchmaschinen-Crawler-Klassen

Wie es durch Quellcodeausschnitt 4.8 sichtbar ist, werden die Informationen über getroffenen Datenquellen farbig durch eine eingesetzte ***Color**-Klasse* nach der Lösung von "*Bouba*"[39] angezeigt.
Wie geklärt, werden wir die Website-URL Zwischen den indexierten Attributen erhalten. Zum Instanzieren unserer eingesetzten Klassen bzw. der Sammlung der notwendigen Informationen wurde eine Funktion mit dem Namen *extract_infos* geschrieben, der einen *Unternehmenssname* als Eingabe erhält, und einen Dictionary-Type, wie in Abbildung 4.13 gezeigt, zurückliefert.

[39] `https://stackoverflow.com/questions/8924173/how-do-i-print-bold-text-in-python/20210807` aufgerufen am 26.09.2021

```
please insert the name of business place: Skendata Rostock
Extracted INFO from crawler.google_knowledge_panel for Skendata Rostock :)
200
https://www.skendata.de/impressum/
200
https://www.skendata.de/ueber-uns/
[{'Boss': 'Jon Meis , Sven Jantzen',
  'Branch': 'Softwareentwickler in Rostock, Mecklenburg-Vorpommern',
  'Company name': 'SkenData GmbH',
  'E-Mail': 'info@skendata.de',
  'Fax': '',
  'Location': 'Mühlendamm 8B D - 18055 Rostock',
  'Number of Employee': '',
  'Telefon': '+49(0)381-25249294',
  'Umsatz-St-ID': 'DE295446443',
  'Website': 'http://www.skendata.de/'}]
```

Abbildung 4.13: validiertes sowie strukturiertes Ergebnis von Programm]

Durch Abbildung 4.13 ist die Ausgabe von “Skendata Rostock” in Entwicklung-Modus zu sehen.

4.5 Test-Evaluation

Zum Test-Evaluation-Verfahren haben wir 150 Unternehmens-Daten in einem Excel-Datei-Format, um die Genauigkeit von unserem NER-Algorithmus zu zeigen. Diese Informationen in unserem Test-Datensatz waren bei einigen Unternehmen alt, deswegen wurde es zuerst versucht, bei jedem Unternehmen, die Informationen manuell abzugleichen und zu aktualisieren.

Nach der richtigen Funktion unserer Einsetzung in der Pipeline, sodass der Nutzer einen Unternehmens-Name eingibt, und die extrahierten Informationen in einem JSON-Datei-Format erhält, wird die Test-Evaluation durchgeführt.

Crawling-Systeme

Zur Evaluation werden 150 Unternehmens-Webseite durch unsere Crawler-Systeme gecrawlt. 130 Unternehmens-Website konnten dadurch erfolgreich gefunden bzw. gecrawlt werden. 20 Websites konnten aus verschiedenen unten genannten Gründen nicht gecrawlt werden:

- Die Website ist aktuell nicht verfügbar.
- Die gesuchten Unternehmen stehten nicht in der festgelegten Datenquelle.
- Es gibt Unternehmens-Informationen in der festgelegten Datenquellen, wobei dort keine Website-URL vorhanden ist.
- verschiedene Struktur der Hyperlinks. Um genauer zu sein, wenn über den relativen URL-String in HTML-Seite der getroffenen Website noch ein anderes Wort außer unseres gesuchtes *keyword*, wie in Abbildung 4.6 zu sehen, in *großgeschriebenes Form* steht, wird die Inhalte dieses Links durch unsere ***BusinessPlaceWebsite**-Klasse* nicht gecrawlt, da wir am Anfang komplette Inhalte der HTML-Seite auf *lower-case* umgestellt haben, und nur bei Bedarf wird das Keyword auf Kapital-Format umgestellt wird.

Test-Daten

Für alle Attribute aus den Websites können mehr als eine Information extrahiert werden. Es gibt viele Unternehmens, die mehr als eine Telefon-Nummer, E-Mail-Adresse sowie Fax-Nummer usw. haben, wobei in unserem Test-Datensatz nur eine Information für jeden Attributen steht.
Das ist auch zu erwähnen, dass es passiert kann, dass die Informationen im Datensatz syntaktische Unterschiede mit den extrahierten Informationen, wie gezeigt durch Abbildung 4.14 haben, wobei sie semantisch gleich sind.

```
E-Mail:
beipiel@skendata.de
beispiel(at)skendata.de
beispiel[at]skendata[punkt]de

Telefon:
038125249294
0381-25249294
0049 381 25249294
+49(0)381-25249294
```

Abbildung 4.14: Beispiel zur verschiedenen Formen der Informationen

Als Ergebnis von unserem Algorithmus werden die extrahierten Informationen in einer Liste gesammelt.
Darüber hinaus wird die Evaluation so durchgeführt, wenn es die bestehende Information auch in der extrahierten Information-Liste steht, dann wird manuell als *TRUE* in Excel-Datei gelabelt. Wenn es aber keine- oder falsche Informationen extrahiert werden, wobei diese Informationen in den Texten der Website stehen, wird

in diesem Fall als *FALSE* gelabelt. Schließlich sieht der Test-Datensatz wie in Abbildung 4.15 aus.

	Firma1	LOC_ACTUAL	LOC_pred	LOC_founded	TEL_actual	TEL_pred	TEL_founded
0	WIKING Sicherheit und Service GmbH	Schulzenweg 34 , 19061 Schwerin	['Salvador - Allende - Straße , 80 A 12559 Ber...	True	0385-745100	['033439 / 17 6 17 [UNK]', '030 / 411 984 11',...	True

Fax_actual	FAX_pred	FAX_founded	BOSS_actual	BOSS_pred	BOSS_founded
0385-74510120	NaN	False	Herr Jörg- P. Althaus Herr Jörg Schultz ...	['Jörg - P . Althaus', 'Maik Althaus']	True

MAIL_actual	MAIL_pred	MAIL_founded
info@wiking-sicherheit.de	['datenschutz [at] wiking - sicherheit . de'...	True

Abbildung 4.15: Beispiel zur Übersicht des aufbereiteten Test-Datensatzes sowie der Vorhersagen von Programm

Nach der Aufbereitung des annotierten Test-Evaluation-Datensatzes wird die Evaluation durch *"accuracy_score"*[40] von *sklearn*-Bibliothek durchgeführt. Dadurch werden alle Attributen im Datensatz als *TRUE* betrachtet, und mit den extrahierten und gelabelten Attributen, durch Quellcodeausschnitt 4.9 verglichen bzw. evaluiert.

```
1   names=["LOC_founded", "TEL_founded", "FAX_founded", "BOSS_founded",
    "MAIL_founded", "TaxID_founded", "NumEMPLOYEE_founded"]
2   acc=[]
3   for name in names:
4     y_actual=[True]*129
5     y_pred=df_eval[name]
6     acc.append(accuracy_score(y_actual, y_pred))
7     print(name+": "+str(accuracy_score(y_actual, y_pred)))
```

Quellcodeausschnitt 4.9: Test-Evaluation

Dann sehen die Ergebnisse für jedes Attribut, wie in Quellcodeausschnitt 4.9 gezeigt, wie folgt aus:

[40] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html aufgerufen am 29.09.2021

```
LOC_founded: 0.9767441860465116
TEL_founded: 0.9689922480620154
FAX_founded: 0.8992248062015504
BOSS_founded: 0.9689922480620154
MAIL_founded: 0.9767441860465116
TaxID_founded: 0.9689922480620154
NumEMPLOYEE_founded: 0.9224806201550387
```

Abbildung 4.16: Das Ergebnis bei Test-Evaluation

5 Fazit und Ausblick

In dieser Arbeit geht es um die dynamische IE und Informationsintegration heterogener unstrukturierter und semistrukturierte Daten für die Tarifierung der gewerblichen Gebäude-Versicherungen bei der *SkenData GmbH*. Integriert werden die Daten letztendlich in einer strukturierten JSON-Datei.

5.1 Zusammenfassung

Zum Finden, Integration sowie Extraktion dieser notwendigen Informationen wurden zwei Suchmaschinen sowie die Website von gewerblichen Gebäuden als semistrukturierte- und unstrukturierte Datenquellen geplant.
Die Informationen, die durch Suchmaschinen indexiert werden, sind wie der *komplette Name*, *Branche* sowie die *Website* des Unternehmens. Dann werden die anderen Informationen wie den *Geschäftsführer-Namen*, *Anschrift*, *E-Mail-Adressen*, *Telefonnummern*, *Faxnummern*, *Anzahl an Mitarbeitern*, *Umsatzsteuer-ID* aus den Texten der Seiten von indexierter *Unternehmens-Website* wie *Impressum-*, *Über-Uns-* und bei Bedarf von der *Homepage*-Seite extrahiert. Als Datenquellen wurden die HTML-Seiten von *Google-Knowledge-Panel*, *Gelbe-Seite* sowie *Google-Lokal-Suche* festgelegt. Zum Parsen der notwendigen Attributen aus diesen drei Datenquellen wurden drei Crawler-Systeme eingesetzt. Im ersten Schritt werden, die Daten aus dem *Google-Knowledge-Panel* zu extrahieren. Wenn es aber kein *Knowledge-Panel* von *Google* zum gesuchten Namen geliefert wird, wird es im zweiten Schritt versucht, die Informationen aus *Gelbe-Seiten* zu parsen. Es kann bei einigen eingegebenen Unternehmensnamen passieren, dass mehr als ein Ergebnis zurückgeliefert wird. in diesem Fall werden die Informationen der ersten drei Ergebnissen gesammelt. Wenn keine Informationen von diesen zwei Datenquellen geliefert werden, dann wird versucht, die erste drei empfohlenen Namen von *Google-Lokal-Suche* zu sammeln. Danach wird versucht, die Informationen dieser drei Namen aus dem *Knowledge-Panel* zu parsen.

Nach der Indexierung und Sammlung der Informationen aus einer Datenquelle kommt ein anderes Crawler-System in Einsatz. Dieser Clawler erhält den indexierten *Website-URL* und liefert die *Body-Texte* der HTML-Seite von *Impressums-* sowie *Über-Uns*-Seiten aus der Website zurück. Wenn es aber keine *Über-Uns*-Seite bei der betreffenden Website vorhanden ist, dann werden die *Body-Texte* von *Hompage*-Seite gesammelt.

Um weiteren Informationen aus gesammelten Texten der Website zu sammeln, wurde zunächst eine zur Verfügung gestellte API von *Sahaj Tomar* [69] in der Pipeline eingesetzt. Die wichtigsten Probleme mit bei der kostenlosen Version dieser API waren, dass es bei uns sehr langsam funktionierte und nicht die kompletten Informationen eines Attributes zurückliefern konnte. Zum Beispiel konnte nur ein Geschäftsführername extrahiert werden, wenn es mehr als einen Namen im Text gab.

Da wir einen möglichst kostenlosen Algorithmus zur Extraktion der notwendigen Attribute aus den Texten benötigen, wurde geplant, das Fine-Tuning auf einem Pre-Trained-Sprach-Modell durchzuführen. Als Sprach-Modell wurde *BERT-multilingual-base-model-cased* ausgewählt, da es aktuell Stand der Technik geklärt wurde. Außerdem funktioniert es auch bei deutsch-sprachigen Texten.
Darüber hinaus wurden die Texte von fast 230 Unternehmens durch unsere Crawler-System gesammelt, und als Train-Datensatz annotiert. Nach der Aufbereitung des Datensatzes haben wir die zur Verfügung gestellte Source-Code von *Tobias Sterbak* [61] zum Fine-Tuning des Modells ausgenutzt.
Nach dem Training des Modells wurde Dev-Evaluation durchgeführt, sodass die Evaluation-Ergebnissen für *Confusion-Matrix* sowie *Classification-Report*, wie in Abbildungen 4.10 sowie 4.11, erreicht werden konnten. Nach dem Postprozess und Duplikate-Erkennung wurden alle Informationen in einem Dictionary-Format umgewandelt. Dann wurde das trainierte und gespeicherte Modell in der Pipeline aufgerufen.

Im nächsten Schritt wurde die Test-Evaluation anhand 150 Unternehmens-Daten als Test-Daten durchgeführt. Von diesen 150 Unternehmens konnten die Texte von 130 Unternehmens durch unsere Crawler-Systeme gecrawlt bzw. gesammelt werden.
Nach der Sammlung der Information von 130 Unternehmens wurde die Genauigkeit bei jedem einzelnen Attribut, wie in Abbildung 4.16 zu sehen, ausgewertet. Als durchschnittliche Genauigkeit anhand dieser Attribute konnten wir 95% erreichen.
Nach der Evaluationsphase wurden alle gesammelten sowie extrahierten Informationen in einer JSON-Datei, wie in Abbildung 4.13 zu sehen ist, für weitere zukünftige Prozesse durch die *SkenData GmbH* bzw. die *Wert14*-APP integriert.

5.2 Ausblick

Wie erwähnt wurde, sammeln wir alle *Body-Texte* der Webseiten, um die notwendigen Informationen zu extrahieren. Darüber hinaus können viele irrelevante Infor-

mationen neben relevanten Informationen extrahiert werden. Es kam vor, dass im Text der Website auf anderen Unternehmenssinformationen verwiesen wurde. Diese Informationen können z.B. die *E-Mail-Adresse*, *Anschrift* sowie *Telefon-Nummer* von Designer der Website, der die Website extern implementiert hat, sein.
In diesem Fall wird eine Methode gebraucht, damit nur die relevanten Paragraphen der Texten in unseren NER-Modell eingehen, um relevante Informationen zu extrahieren. Diese Methode kann definitiv zur Optimierung des Verfahrens im Einsatz kommen.

Literaturverzeichnis

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep Contextualized Word Representations," *arXiv preprint arXiv:1802.05365*, 2018.

[3] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-training," *https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf*, 2018.

[4] A. Rabiee, "Deep Learning for NLP, Workshop." https://www.aparat.com/v/n5r7q?playlist=683474, 2021. Zuletzt aufgerufen am 07.08.2021.

[5] R. Baeza-Yates, B. Ribeiro-Neto, *et al.*, *Modern Information Retrieval*, vol. 463. ACM press New York, 1999.

[6] S. Ceri, A. Bozzon, M. Brambilla, E. Della Valle, P. Fraternali, and S. Quarteroni, *Web Information Retrieval*. Springer Science & Business Media, 2013.

[7] J. S. Culpepper and A. Moffat, "Efficient set Intersection for Inverted Indexing," *ACM Transactions on Information Systems (TOIS)*, vol. 29, no. 1, pp. 1–25, 2010.

[8] N. Duhan, A. Sharma, and K. K. Bhatia, "Page Ranking Algorithms: a Survey," in *2009 IEEE International Advance Computing Conference*, pp. 1530–1537, IEEE, 2009.

[9] M. Sanderson and J. Zobel, "Information Retrieval System Evaluation: Effort, Sensitivity, and Reliability," in *Proceedings of the 28th annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 162–169, 2005.

[10] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon, "Novelty and Diversity in Information Retrieval Evaluation," in *Proceedings of the 31st annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 659–666, 2008.

[11] Seo-Nerd, "Seo-Nerd." https://www.seo-nerd.com/de/news/absolute-relative-urls, 2021. Zuletzt aufgerufen am 26.07.2021.

[12] A. Shen, *Algorithms and Programming: Problems and Solutions*. Springer Science & Business Media, 2011.

[13] S. vanden Broucke and B. Baesens, *Practical Web Scraping for Data Science: Best Practices and Examples with Python*. Apress, 2018.

[14] C. Hahnel, F. Goldhammer, U. Kröhne, and J. Naumann, "The Role of Reading Skills in the Evaluation of Online Information gathered from Search Engine Environments," *Computers in Human Behavior*, vol. 78, pp. 223–234, 2018.

[15] Statcounter, "Statcounter Global Status." https://gs.statcounter.com/search-engine-market-share/desktop/worldwide/, 2021. Zuletzt aufgerufen am 26.07.2021.

[16] D. Lewandowski, *Suchmaschinen Verstehen*. Springer, 2015.

[17] Klicksafe, "Die verschiedenen Suchmaschinentypen." https://www.klicksafe.de/suchmaschinen/die-verschiedenen-suchmaschinentypen/, 2021. Zuletzt aufgerufen am 26.07.2021.

[18] G. Seiten, "Gelbe Seiten." https://www.gelbeseiten.de/gsservice/kooperationspartner, 2021. Zuletzt aufgerufen am 26.07.2021.

[19] M. Kejriwal, "What Is a Knowledge Graph?," in *Domain-Specific Knowledge Graph Construction*, pp. 1–7, Springer, 2019.

[20] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler, "Introduction: What is a Knowledge Graph?," in *Knowledge Graphs*, pp. 1–10, Springer, 2020.

[21] A. Uyar and F. M. Aliyu, "Evaluating Search Features of Google Knowledge Graph and Bing Satori: Entity Types, List Searches and Query Interfaces," *Online Information Review*, 2015.

[22] Google, "Google Knowledge Graph Search API." https://developers.google.com/knowledge-graph, 2021. Zuletzt aufgerufen am 26.07.2021.

[23] Bing, "Bring rich knowledge of People, Places, Things and Local Businesses to your Apps." https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps, 2021. Zuletzt aufgerufen am 26.07.2021.

[24] Google, "How Google determines local ranking." https://support.google.com/business/answer/7091?hl=en#zippy=%2Crelevance%2Cdistance%2Cprominence, 2021. Zuletzt aufgerufen am 26.07.2021.

[25] C. Olston and M. Najork, *Web Crawling*. Now Publishers Inc, 2010.

[26] C. Saini and V. Arora, "Information Retrieval in Web Crawling: A Survey," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2635–2643, IEEE, 2016.

[27] S. Redlich, "Web Scraping zur Gewinnung von Testdaten für administrative Register," *WISTA-Wirtschaft und Statistik*, 2020.

[28] Google, "XML API Reference." `https://developers.google.com/custom-search/docs/xml_results#WebSearch_Request_Format`, 2021. Zuletzt aufgerufen am 26.07.2021.

[29] Ben, "Google Search with Python Requests Library." `https://stackoverflow.com/questions/22623798/google-search-with-python-requests-library`, 2021. Zuletzt aufgerufen am 26.07.2021.

[30] L. Richardson, "Beautiful Soup." `https://github.com/waylan/beautifulsoup`, 2021. Zuletzt aufgerufen am 26.07.2021.

[31] C. Zheng, G. He, and Z. Peng, "A Study of Web Information Extraction Technology Based on Beautiful Soup," *J. Comput.*, vol. 10, no. 6, pp. 381–387, 2015.

[32] S. GOEL, M. BANSAL, A. K. SRIVASTAVA, and N. ARORA, "Web Crawling-based Search Engine using Python," in *2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 436–438, IEEE, 2019.

[33] W. Hejing, L. Fang, Z. Long, S. Yabin, and C. Ran, "Application Research of Crawler and Data Analysis based on Python," *Associate Editor-in-Chief*, p. 64, 2020.

[34] M. Pieters, "Can we use xpath with beautifulsoup?." `https://stackoverflow.com/questions/11465555/can-we-use-xpath-with-beautifulsoup`, 2021. Zuletzt aufgerufen am 26.07.2021.

[35] B. Muthukadan, "Selenium with Python." `https://selenium-python.readthedocs.io/`, 2021. Zuletzt aufgerufen am 26.07.2021.

[36] S.-H. Hong, S.-K. Lee, and J.-H. Yu, "Automated Management of Green Building Material Information using Web Crawling and Ontology," *Automation in Construction*, vol. 102, pp. 230–244, 2019.

[37] Y. Papernyk, "Data Workflows mit Python." https://www.enterpy.de/python_data_workflows.php#programm, 2021. Zuletzt aufgerufen am 26.07.2021.

[38] R. Amos, G. Acar, E. Lucherini, M. Kshirsagar, A. Narayanan, and J. Mayer, "Privacy Policies over Time: Curation and Analysis of a Million-Document Dataset," in *Proceedings of the Web Conference 2021*, pp. 2165–2176, 2021.

[39] Miyakogi, "Pyppeteer's Documentation." https://github.com/miyakogi/pyppeteer, 2021. Zuletzt aufgerufen am 27.07.2021.

[40] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A Survey of Web Information Extraction Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1411–1428, 2006.

[41] J. Jiang, "Information Extraction from Text," in *Mining text data*, pp. 11–41, Springer, 2012.

[42] C. Grover, S. Givon, R. Tobin, and J. Ball, "Named Entity Recognition for Digitised Historical Texts," in *LREC*, 2008.

[43] P. Mortazavi and M. Shamsfard, "Named Entity Recognition in Persian Texts," in *15th National CSI Computer Conference*, 2009.

[44] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A Neural Probabilistic Language Model," *The Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[45] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained Models for Natural Language Processing: A Survey," *Science China Technological Sciences*, pp. 1–26, 2020.

[46] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional Sequence to Sequence Learning," in *International Conference on Machine Learning*, pp. 1243–1252, PMLR, 2017.

[47] P. Liu, X. Qiu, and X. Huang, "Recurrent Neural Network for Text Classification with Multi-Task Learning," *arXiv preprint arXiv:1605.05101*, 2016.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[49] J. Callut and P. Dupont, "Inducing Hidden Markov Models to Model Long-Term Dependencies," in *European Conference on Machine Learning*, pp. 513–521, Springer, 2005.

[50] S. El Hihi and Y. Bengio, "Hierarchical Recurrent Neural Networks for Long-Term Dependencies," in *Advances in Neural Information Processing Systems*, pp. 493–499, 1996.

[51] E. Trentin and M. Gori, "Combining Neural Networks and Hidden Markov Models for Speech Recognition," *Neural Nets WIRN VIETRI-98*, pp. 63–79, 1999.

[52] A. G. Schwing and R. Urtasun, "Fully Connected Deep Structured Networks," *arXiv preprint arXiv:1503.02351*, 2015.

[53] J. Hammerton, "Named Entity Recognition with Long Short-Term Memory," in *Proceedings of the seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003.

[54] E. Taher, S. A. Hoseini, and M. Shamsfard, "Beheshti-NER: Persian Named Entity Recognition Using BERT," *arXiv preprint arXiv:2003.08875*, 2020.

[55] K. Labusch, P. Kulturbesitz, C. Neudecker, and D. Zellhöfer, "BERT for Named Entity Recognition in Contemporary and Historical German," in *Proceedings of the 15th Conference on Natural Language Processing, Erlangen, Germany*, 2019.

[56] K. Hakala and S. Pyysalo, "Biomedical Named Entity Recognition with Multilingual BERT," in *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks*, 2019.

[57] C. Chambliss, "Multilingual-NER." `https://github.com/chambliss/Multilingual_NER`, 2021. Zuletzt aufgerufen am 15.09.2021.

[58] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *CoRR*, vol. abs/1810.04805, 2018.

[59] C. Chambliss, "How to Fine-Tune BERT for Named Entity Recognition." `https://gab41.lab41.org/how-to-fine-tune-bert-for-named-entity-recognition-2257b5e5ce7e`, 2021. Zuletzt aufgerufen am 15.09.2021.

[60] C. Chambliss, "Lessons Learned from Fine-Tuning BERT for Named Entity Recognition." `https://gab41.lab41.org/lessons-learned-fine-tuning-bert-for-named-entity-recognition-4022a53c0d90`, 2021. Zuletzt aufgerufen am 15.09.2021.

[61] T. Sterbak, "Named Entity Recognition with Bert." https://www.depends-on-the-definition.com/named-entity-recognition-with-bert/, 2021. Zuletzt aufgerufen am 15.09.2021.

[62] H. Nakayama, "seqeval: A python framework for sequence labeling evaluation," 2018. Software available from https://github.com/chakki-works/seqeval Zuletzt aufgerufen am 01.10.2021.

[63] T. Sterbak, "State-of-the-Art Named Entity Recognition with Residual LSTM and ELMo." https://www.depends-on-the-definition.com/named-entity-recognition-with-residual-lstm-and-elmo/, 2021. Zuletzt aufgerufen am 15.09.2021.

[64] Monkeylearn, "MonkeyLearn's Entity Extraction API & Other Tools." https://monkeylearn.com/blog/entity-extraction-api/, 2021. Zuletzt aufgerufen am 07.08.2021.

[65] Google, "Natural Language API." https://cloud.google.com/natural-language/, 2021. Zuletzt aufgerufen am 07.08.2021.

[66] aylien, "Transform the World's News Into Real-Time Business Insight." https://aylien.com/, 2021. Zuletzt aufgerufen am 07.08.2021.

[67] IBM, "IBM Watson is AI for smarter Business." https://www.ibm.com/watson, 2021. Zuletzt aufgerufen am 07.08.2021.

[68] AMAZON, "Amazon Comprehend." https://aws.amazon.com/de/comprehend/, 2021. Zuletzt aufgerufen am 07.08.2021.

[69] S. Tomar, "Sahajtomar GELECTRAQA." https://huggingface.co/Sahajtomar/GELECTRAQA, 2021. Zuletzt aufgerufen am 07.08.2021.

[70] W. Ceusters, B. Smith, and J. M. Fielding, "Formally Robust Ontology-Based Data and Information Integration," in *International Workshop on Data Integration in the Life Sciences*, Springer, 2004.

[71] A. Casas-Bayona and H. G. Ceballos, "Integrating Semi-structured Information using Semantic Technologies An Evaluation of Tools and a Case Study on University Rankings Data," 2014. https://www.scitepress.org/Papers/2014/50042/50042.pdf.

[72] R. Holten, "Integration von Informationssystemen," *Wirtschaftsinformatik*, vol. 45, no. 1, pp. 41–52, 2003.

[73] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini, "The momis Approach to Information Integration," 2001.

[74] M. Natarajan, "Role of Text Mining in Information Extraction and Information Management," *DESIDOC Journal of Library & Information Technology*, vol. 25, no. 4, 2005.

[75] L. Tanabe, U. Scherf, L. Smith, J. Lee, L. Hunter, and J. Weinstein, "MedMiner: an Internet Text-Mining Tool for Biomedical Information, with Application to Gene Expression Profiling," *Biotechniques*, vol. 27, no. 6, pp. 1210–1217, 1999.

[76] G. Bosetti, S. Firmenich, M. Winckler, G. Rossi, U. C. Fandos, and E. Egyed-Zsigmond, "An End-User Pipeline for Scraping and Visualizing Semi-Structured Data over the Web," in *International Conference on Web Engineering*, pp. 223–237, Springer, 2019.

[77] S. Koeva, N. Obreshkov, and M. Yalamov, "Natural Language Processing Pipeline to Annotate Bulgarian Legislative Documents," in *Proceedings of The 12th Language Resources and Evaluation Conference*, pp. 6988–6994, 2020.

[78] T. Boroș, S. D. Dumitrescu, and R. Burtica, "NLP-Cube: End-to-End Raw Text Processing With Neural Networks," in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, (Brussels, Belgium), pp. 171–179, Association for Computational Linguistics, October 2018.

[79] H. Z. Benjamin Bremert, "Text Mining öffentlich zugänglicher Daten." `https://www.datenschutzzentrum.de/uploads/projekte/itesa/Bremert_Zwingelberg_Text-mining.pdf`, 2021. Zuletzt aufgerufen am 15.10.2021.

[80] Python, "asyncio — Asynchronous I/O." `https://docs.python.org/3/library/asyncio.html`, 2021. Zuletzt aufgerufen am 26.07.2021.

[81] K. Stanley, ":mod:'asyncio' — Asynchronous I/O." `https://github.com/python/cpython/blob/3.9/Doc/library/asyncio.rst`, 2021. Zuletzt aufgerufen am 26.07.2021.

[82] Hutch and M. Pieters, "What are the differences between the urllib, urllib2, urllib3 and requests Module?." `https://stackoverflow.com/questions/2018026/what-are-the-differences-between-the-urllib-urllib2-urllib3-and-requests-modul`, 2021. Zuletzt aufgerufen am 26.07.2021.

[83] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, “Well-read students learn better: On the importance of pre-training compact models,” *arXiv preprint arXiv:1908.08962v2*, 2019.

Selbstständigkeitserklärung zur Masterarbeit

Ich, Sina Ghashghaie, erkläre, dass ich die vorliegende Arbeit zum Thema „*Extraktion und Integration heterogener unstrukturierter und semistrukturierter Daten für die Tarifierung von gewerblichen Gebäudeversicherungen*“ selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 18. Oktober 2021

Ort, Datum

Unterschrift