VERY LARGE
BUSINESS APPLICATIONS
Carl von Ossietzky Universität Oldenburg

Einsatz von Large Language Models im Record-Linkage Prozess

Bachelorarbeit

Bachelorstudiengang: Informatik

Themensteller:	Datenschmiede.ai GmbH
Betreuer:	Dr. Felix Kruse Christoph Schröer Jan-Philipp Awick
Vorgelegt von:	Andrej Cepygin Semesteranschrift PLZ Wohnort: 27612 Loxstedt Telefonnummer: 01522 6572035 andrej.cepygin@uni-oldenburg.de
Abgabetermin:	16. Juni 2024

Inhaltsverzeichnis

Glossar

Few-Shot
Eine Technik aus dem maschinellen Lernens (Machine-Learning, ML), bei denen die Modelle eine begrenzte Menge an Beispielen bekommen. Die Modelle sollen Vorhersagen anhand einer begrenzten Menge an Beispiels-Datensätzen treffen (Rouse, 2024).

Zero-Shot
Eine Technik aus dem maschinellen Lernens (Machine-Learning, ML), bei denen die Modelle keine Beispiele bekommen. Die Modelle sollen Vorhersagen anhand der Beziehungen von bereits bekannten Klassen treffen (Rouse, 2024).

Mergen
Das Zusammenführen von Datensätzen oder Einträgen.

Dataframe
Eine Tabellenartige Struktur von Dateieinträgen, welche mit Spaltennamen gekennzeichnet werden können. Wird aufgrund der leichten Implementierung als Darstellungsart der Datensätze benutzt.

Threshold
Ein Schwellenwert, der als Grenze für Dateneinträge dient. Wenn der Ähnlichkeitswert von zwei Einträgen über dem Schwellenwert liegt, wird diese Paar als Treffer (Match) angesehen, sonst sind es keine Treffer (no Match).

Score
In dieser Arbeit bezeichnet der Score die Kosinusähnlichkeit von Wort-Vektoren.

Embedding
Wort-Vektor

Vectorstore
Vektordatenbank

Externe Links

Verweis zum GitLab-Repository: https://gitlab.uni-oldenburg.de/goda7323/record-linkage

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

In der heutigen Welt steigt die Menge an generierten Daten immer weiter. Dabei kann es sich um Unternehmensdaten, Patientendaten oder generell um Kundendaten handeln. Die Datensätze können von Organisationen intern oder extern, von beispielsweise Werbepartnern, erstellt werden. Daten können sogar aus Webseiten wie Wikipedia und Social-Media Plattformen wie Facebook extrahiert werden. Diese Datensätze besitzen meistens unkontrollierbare Qualität wie falsche Einträge, fehlende Einträge und Selektionsverzerrungen (Christen, 2019). Deshalb ist es wichtig, die Datensätze zu verarbeiten, um eine Grundlage für erfolgreiche Datenintegration zu schaffen. Dieser Prozess der Datenverarbeitung ist bekannt als Data Linkage, Record-Linkage, Data Matching oder Entity Resolution (Christen, 2019). Record-Linkage ist ein "rechnerisches Verfahren" (Domingo-Ferrer, 2018, S. 3128–3129), um jeden Eintrag aus mehreren Datenquellen so zu kombinieren, dass eine korrespondierende Entität entsteht (Gruenheid, 2018, S. 1–5). Record-Linkage besteht aus den fünf Schritten: Data Pre-Processing, Blocking/Indexing, Comparison, Classification und Evaluation (Christen, 2012b, S. 23–35), diese werden in Abb.1 visualisiert.

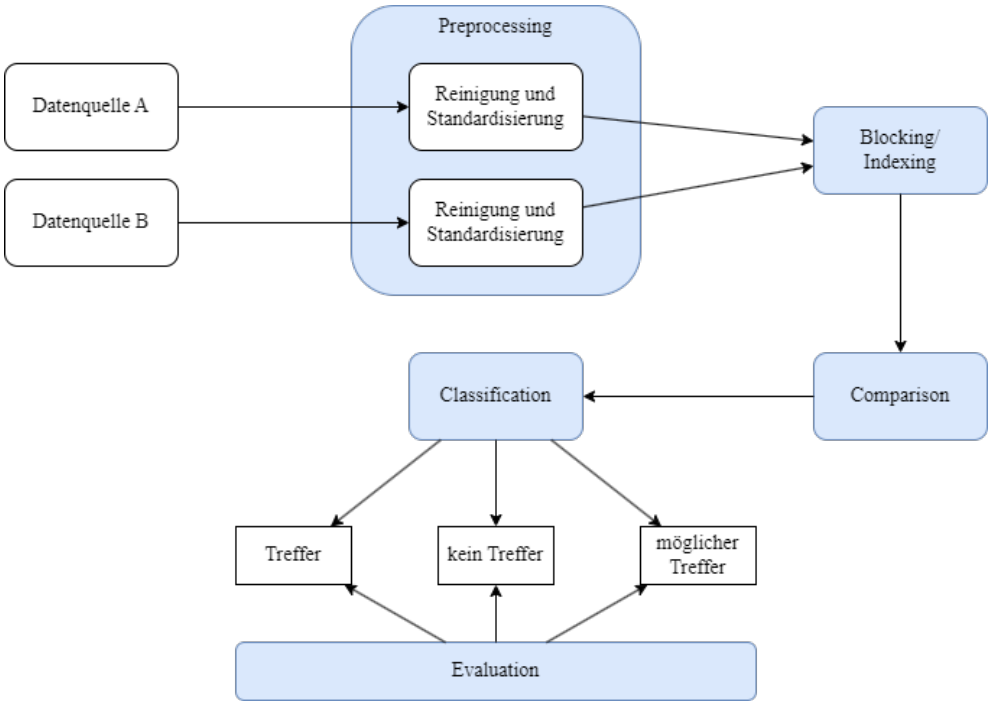

Abbildung 1: Record-Linkage Process, eigene Darstellung, angelehnt an Peter Christen, Data Matching, 2009, S. 7

Es gibt hauptsächlich zwei Arten von Algorithmen für das Record-Linkage: die deterministischen und die probabilistischen Algorithmen. Die Wahl der Algorithmus-Art hängt unter anderem von den Faktoren Zeit, Ressourcen, Quantität und Qualität der Einträge in den Datenquellen sowie von dem Anwendungsfall ab (Dusetzina SB, 2014). Bei den deterministischen Algorithmen müssen die Einträge beider Datenquellen in jeder passenden Variabel exakt gleich sein, damit eine korrespondieren Entität gebilden werden kann. Deterministische Algorithmen werden häufig benutzt, wenn eine identifizierende Variabel, zum Beispiel eine ID-Nummer[1], in beiden Datenquellen vorkommt (Shlomo, 2019, S. 47–65). Bei dem probabilistischem Record-Linkage wird für jeden Eintrag aus den Datenquellen ein Paar gebildet, welches in einem Index gespeichert wird und anschließend ein Ähnlichkeitswert (Score) des Paares berechnet. Dieser Score wird mit einem vorgegebenen Schwellenwert (Threshold) verglichen und das Paar in die Kategorien Treffer, kein Treffer oder möglicher Treffer klassifiziert. Dabei muss der mögliche Treffer von Menschenhand geprüft werden, da diese Gruppe nicht in Treffer oder kein Treffer klassifiziert werden kann (Domingo-Ferrer & Torra, 2002, S 208).
Der Einsatz von Record-Linkage erweist sich als immer wichtiger für Datenintegration in Unternehmen. Trotz state-of-the-art Ergebnissen des Record-Linkage werden diese aufgrund des hohen manuellen Aufwandes und fehlenden erforderlichen Fachwissens nicht verwendet (Kruse, Schröer, Awick, Reinkensmeier & Gómez, 2022). Um den manuellen Aufwand von Record-Linkage zu verringern, gibt es bereits Studien für den Einsatz von Large-Language Models (LLMs) im Record-Linkage auf Unternehmensdaten. Ein LLM dient dabei als Kommunikationsmittel um die Interaktion zwischen Mensch und Maschine zu erleichtern und übernimmt komplexe Sprachaufgaben wie Übersetzung, Zusammenfassung, Informationsabfrage etc. (Naveed et al., 2023). LLMs beziehen sich auf Transformer language models, welche mehrere Milliarden von Parametern aufweisen und auf einer großen Textdatei traniert wurden (Zhao et al., 2023, S. 4). Die Größe eines LLMs wird über dessen Parameteranzahl repräsentiert und ist ein Identifikator über die Performance eines Modells (Zhao et al., 2023, S. 4). Bei Modellen mit dem selben Aufgabenbereich liefert das größere Modell ein besseres Ergebnis (Naveed et al., 2023, S. 9), kann aber von kleineren Modellen, welche gefine-tuned wurden, überholt werden (Zhao et al., 2023, S. 5). Peeters und Bizer erreichten bei ihrer Untersuchung eines Chat basierten LLMs eine Präzision von bis zu 0.95. Dabei wird das Daten-Paar zusammen mit einer Anweisung, ob diese zusammengehören, in eine Prompt kombiniert, woraus das LLM eine Antwort generiert (Abb.2) (Peeters & Bizer, 2023, S. 1). Eine andere Einsatzmöglichkeit eines LLMs ist die Implementierung in einer Record-Linkage Pipeline. Ein erfolgreiches Modell für den Ein-

[1] Ausweisnummer, Steueridentifikationsnummer, Matrikelnummer etc.

satz in einer Record-Linkage Pipeline ist Ditto, welches von Yuliang Li und seinem Team trainiert wurde und eine 96 Prozentige Trefferquote auf Unternehmensdaten aufweist (Li, Li, Suhara, Doan & Tan, 2020).

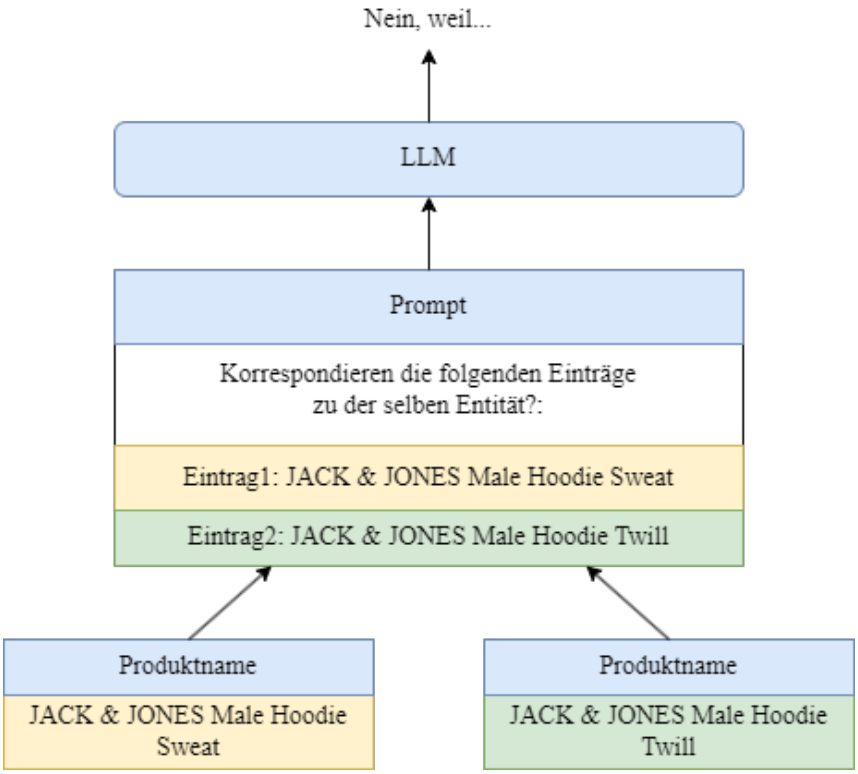

Abbildung 2: Bildung einer Prompt mit zwei Einträgen, die einem LLM als Anweisung übergeben wird, eigene Darstellung, angelehnt an (Peeters & Bizer, 2023)

1.2 Problemstellung

Aus der Motivation erschließt sich, dass der Einsatz von LLMs auf Datensätze im Record-Linkage vielversprechend ist. LLMs wurden in den vergangenen Jahren immer weiter entwickelt und state-of-the-art Modelle wie GPT3.5 und GPT4 können spezifische Anweisungen problemlos ausführen (Johns, 2023). In dieser Arbeit wird untersucht, welche Open-Source Modelle im Record-Linkage angewandt werden können. Dadurch beschränkt sich die Auswahl auf die Modelle, welche sich auf der Internetplattform Hugging Face befinden. Desweiteren steht eine begrenzte Hardware zur verfügung, was den Einsatz von großen LLMs unmöglich macht. Zum Beispiel benötigt das Modell GPT-3 175B (175 Milliarden Parameter) mindestens 5x80GB A100 GPUs und 350GB Arbeitsspeicher wenn man es lokal

verwenden möchte (Naveed et al., 2023, S. 20). Peeters und Bizer verwenden in ihrer Studie zwar Open-Source LLMs (Peeters & Bizer, 2023, S. 2), diese brauchen jedoch bis zu 80GB Arbeitsspeicher[2]. Das Ditto Modell liefert bereits in der Basisversion bessere Ergebnisse als state-of-the-art Verfahren im Record-Linkage (Li et al., 2020, S. 3), es wird jedoch nicht erwähnt, welche Hardware Ansprüche für das Modell benötigt werden. Die eingesetzten LLMs werden auf einer Maschine mit viel kleinerer Hardwareleistung laufen, was die Qualität der Ausgabe verringert wird. LLMs werden für bestimmte Aufgabenbereiche (tasks) trainiert und die allgemeine Performance eines LLMs ist neben der Parametermenge auch davon abhängig, ob diese für die Aufgaben benutz werden, auf denen sie trainiert wurden (Zhao et al., 2023). Sollten die verwendeten LLMs, wie in Abb.2 Chat-basiert sein, ist das richtige Einsetzen der Prompts eine weitere Hürde. Jedes LLM hat eine unterschiedliche Sensitivität bezüglich den Prompts (Peeters & Bizer, 2023, S. 3). Dadurch kann sich die Qualität der Ausgabe mit der richtigen Formulierung der Prompt ebenfalls verändern. Wie bereits erwähnt hängt Performance eines LLM standardmäßig von dessen Parametern ab. Um den kompletten Record-Linkage Prozess durchzuführen, muss das Modell in der Lage sein, alle Schritte selbstständig durchzuführen. Mit einem kleinerem Modell wird erwartet, dass die Ausgabe des LLMs, wenn dieses auf den gesamten Prozess angewendet wird, im vergleich zu herkömmlichen Algorithmen qualitativ niedriger sein kann und somit keinen Mehrwert bieten wird (Ding et al., 2023). Von daher muss überprüft werden, ob ein LLM, das auf der verfügbaren, lokalen Maschine läuft, effizientere Ergebnisse liefert, wenn dieses nur auf bestimmte Bereiche des Record-Linkage angewandt wird.

[2]HuggingFace SOLAR-0-70b-16bit

1.3 Zielsetzung und methodisches Vorgehen

Um eine Aussage über die Verwendung von LLMs im Record-Linkage Prozess zu machen, wird folgende Hauptforschungsfrage beantwortet:

Hauptforschungsfrage: Wie können LLMs in Record-Linkage von Unternehmensdaten eingesetzt werden?

Diese Frage wird in zwei Teilfragen aufgeteilt:

Teilfrage 1: Welches Large-Language Model kann im Record-Linkage eingesetzt werden?

Hierzu wird eine Literaturrecherche durchgeführt, um zu identifizieren, welche LLMs im Record-Linkage eingesetzt werden können. Die Internetplatform Hugging Face[3] stellt eine vielzahl von Open-Source LLMs zur Verfügung, welche für eine spezifische Aufgabe (Task) trainiert wurden. Die verwendeten Modelle (SOLAR und Beluga2) von Peeters und Bizer wurden beide für text-generation traniert (Peeters & Bizer, 2023, S. 2). Andere Modelle, welche im Record-Linkage zum Einsatz kommen, wurden für text-classification (RoBERTa) und entity-resolution (Ditto) trainiert (Li et al., 2020). Nachdem mögliche LLM-Kandidaten ausgewählt wurden, werden diese in der nächsten Teilfrage untersucht.

Teilfrage 2: Wo können Large-Language Models im Record-Linkage eingesetzt werden?

Nachdem eine Auswahl der Modelle in Teilfrage 1 getroffen wurde, wird der Einsatzort untersucht. Dabei wird analysiert, in welchem Schritt des Record-Linkage Prozesses die Verwendung von LLMs sinnvoll erscheint und welche Methoden als Open-Source zur Verfügung stehen.

[3] Hugging Face Models

2 Theoretischer Hintergrund

2.1 Large-Language-Models

Large-Language Models sind Algorithmen, welche auf Basis von Deep Learning Modellen arbeiten, um Aufgaben in Textform zu verstehen und auszuführen. In den letzten Jahren gab es bedeutsame Fortschritte bei der Erforschung von Sprachmodellen, welche der Transformer-Architektur zuzuschreiben ist und LLMs hervorbringen kann, welche Ergebnisse auf dem menschlichen Leistungsniveau bei verschiedenen Tasks erzielen können (Naveed et al., 2023, S. 1). Der Prozess der Informationsverarbeitung aus natürlicher Sprache wird Natural-Language-Processing genannt (NLP) (Grigoleit, 2019). LLMs haben ihren Ursprung im Statistical-Language-Modeling (SLM) (Naveed et al., 2023, S. 2), welches probabilistisch das nächste Wort in einer Sequenz vorhersagen kann, basierend auf den vorherigen Wörtern (Chelba et al., 2014). Daraus entstanden Neural-Language-Models (NLM), welche mit Neurale Netzwerken statt probalistisch arbeiten (Jing & Xu, 2019). Beide Modell-Arten werden Task-Spezifisch in einer überwachten (supervised) Umgebung trainiert. Die nächste Evolution von Sprachmodellen werden in einer selbstüberwachten (self-supervised) Umgebung trainiert und dienen als Grundmodelle in modernen NLP Anwendungen (Wei, Wang, Wang & Kuo, 2023). Diese Modelle werden als Pre-trained-Language-Models (PLMs) betitelt und sollen ein breites Spektrum von Spachaufforderungen bearbeiten können (Naveed et al., 2023, S. 2). Das PLM wird weiter für spezifischere Aufgaben gefine-tuned, was zu einem Performance Anstieg im Vergleich zu herkömmlichen Modellen (SLMs und NLMs) führt. Dabei werden die Modell-Parameter und Trainingsdaten signifikant erhöht und es erfolgt ein Übergang von PLM zu einem LLM (Naveed et al., 2023, S. 2). Pre-trained LLMs können einfachere Tasks mit zero-shot Aufforderungen ausführen, scheitern jedoch beim verstehen der Benutzerabsichten und weisen eine schlechtere Performance als in einem few-shot-setting auf (Naveed et al., 2023, S. 2). Um die zero-shot Performance zu steigern, wird das Modell mit Aufgaben-Anforderungen (task instruction data) gefine-tuned (Wang et al., 2022, S. 6) und auf eine Präferenz abgestimmt, die der Anforderungen des Menschlichen benutzers entspricht, wie zum Beispiel einem Chat Modell, welches nur mit Emojis antwortet (Touvron et al., 2023, S. 16). Folglich werden Begriffe und Eigenschaften von LLMs erklärt, um einen Überblick zu schaffen, womit gearbeitet wird und welche Eigenschaften sich bei unterschiedlichen Modellen unterscheiden.

Parameter

Die Parameter in einem Neuralen-Netzwerk, auf den LLMs basieren, sind entscheidend für die Ergebnisse, die das Netzwerk generiert. Diese werden in die Kategorien Gewicht (weights) und Neigung (bias) unterteilt und werden in den Knoten des Netzwerkes gespeichert (Bailer-Jones, Gupta & Singh, 2001, S. 4). Während des Trainings werden die zuvor generierten Gewichte verändert, um die gewollte Antwort zu erzielen, während die Knoten mit den Bias-Werten konstant bleiben (Bailer-Jones et al., 2001, S. 5). Dadurch, dass Parameter zuständig für die Antwortgenerierung eines Netzwerkes sind, ist die Menge von Parametern in einem Modell neben der Menge von Trainings-Datensätzen und Trainings-Einstellungen aussagekräftig über die allgemeine Performance eines LLM (Zhao et al., 2023, S. 4).

Tasks

Als Tasks werden die Aufgabenbereiche im NLP bezeichnet. Die Internetplattform Huggingface stellt auf ihrer Webseite 11 NLP Tasks[4], in die LLMs eingeordnet werden können. Während Modelle wie ChatGPT als universelle Task-Löser dienen können, gibt es Anwendungen, bei denen Modelle, die für diesen Task gefine-tuned wurden, bessere Ergebnisse erzielen (Qin et al., 2023, S. 9). Zu den Tasks zählen Aufgabenbereiche wie Text Generation, Sentence Similarity oder Translation.

Tokens

Tokenisierung (Tokenization) ist der Prozess, bei dem ein Text in typographische Einheiten (Mielke et al., 2021, S. 7) unterteilt wird, um als Eingabe (input) für das LLM zu dienen (Zhao et al., 2023, S. 19). Diese können aus Wörtern, Teilwörtern, Symbolen und Buchstaben, abhängig von dem Tokenizierungs Prozess, bestehen (Naveed et al., 2023, S. 4). Einer der bekanntesten Tokenizer ist Byte-Pair-Encoding (BPE), bei dem das häufigste Bytepaar in einer Sequenz iterativ durch ein einzelnes, nicht verwendetes Byte ersetzt wird (Sennrich, Haddow & Birch, 2016, S. 3). Dieses Verfahren kann jedoch zu inkonsistenten Ergebnissen führen, wenn Zahlen oder Satzzeichen im Text auftauchen (Sun et al., 2023, S. 2).

[4]Huggingface Tasks

Embeddings

Wort-Vektoren (Embeddings) sind numerische Vektoren, welche die syntaktische und semantische Bedeutung von Wörtern in Maschinenverständlicher Schreibweise darstellen. Embeddings werden grundsätzlich in zwei Kategorien unterteilt: Count-Based-Models, welche für das Zählen von Wörtern eingesetzt werden und Prediction-Based-Models, welche für das Vorhersagen von Wörtern benutzt werden (Almeida & Xexéo, 2023, S. 3). Der Name Embedding leitet sich von den NLMs ab, da es den Vorgang der Projizierung des Wort-Vektors in die erste Schicht des Modells, dem Embedding-Layer, angibt (Almeida & Xexéo, 2023, S. 3). Als Beispiel wird in Abb. 11 ein Embedding für das Wort "dog" erstellt. Mit der Berechnung der Kosinusähnlichkeit zweier Embeddings können Scores gebildet werden, welche die Ähnlichkeit zweier Einträge angibt. Zum Beispiel können die Embeddings von Firmennamen auf ihre Ähnlichkeit überprüft (Tab. 1) und numerisch dargestellt werden.

Firma A	Firma B	Score
MÜLLER & EGERER BÄCKEREI & KONDITOREI	MUELLER & EGERER BAECKEREI UND KONDITOREI GMBH	0.99
ERICH HINSCHE GASTRO-RENT GMBH	ERICH HINSCHE GMBH & CO. KG	0.33
ALI BEY H. ELMA E.K.	ALI BEY, H. ELMA E.K.	0.89

Tabelle 1: Scores für Firmennamen

Prompts

Prompts sind Anweisungen, welche dem LLM übergeben werden, um spezifische Regeln, Verhaltensweisen oder Qualitätsanforderungen im generierten Output festzulegen (White et al., 2023). Die Prompt legt den Kontext der Unterhaltung fest und gibt dem Modell vor, welche Informationen wichtig sind und welche Form der gewünschte Output haben soll. Das Programmieren von LLMs anhand der Prompt wird Prompt-Engineering genannt (White et al., 2023, S. 1).

Vectorstores

Vektordatenbanken (Vectorstores) sind Datenbanken, in denen Vektoren (Embeddings) von unstrukturierten Dateneinträgen gespeichert werden. Diese werden während einer Anfrage (query) mit den unstrukturierten Embeddings der query verglichen um die Vektoren mit der meisten Ähnlichkeit zu den query-Embeddings zu erhalten (DeWilde, o. J.). Diese Ähnlichkeit wird mit Algorithmen wie k-Nearest Neighbor (kNN) oder Approximane Nea-

rest Neighbor (ANN) durchgeführt und liefert im Vergleich zur herkömmlichen Schlüsselwortsuche relevantere Ergebnisse in kürzerer Zeit (Elastic, o. J.).

Transformer

Die Transformer Architektur besteht aus einem Encoder, welcher die Inputs erhält und eine Repräsentation der Eigenschaften (features) bildet und einem Decoder, welcher diese features erhält und eine Sequenz daraus bildet (Abb.3)(HuggingFace, o. J.).

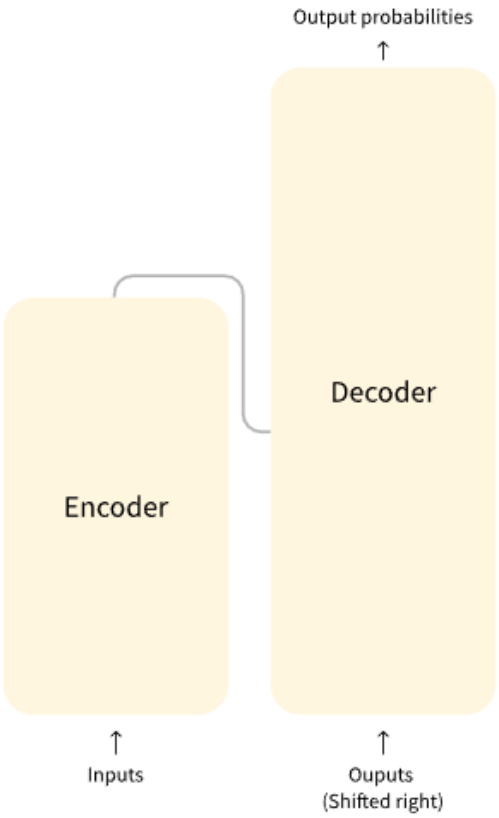

Abbildung 3: Encoder-Decoder Architektur, entnommen aus (HuggingFace, How do Transformers work?, 02.06.2024)

Das Hauptmerkmal von Transformer Modellen ist das Attention-Layer, welches bestimmten Wörtern in einem Text mehr Aufmerksamkeit bietet als anderen. Dadurch ist das Modell in der Lage, komplexere Sätze und Anweisungen zu verstehen (Vaswani et al., 2023).

Je nach Task können Encoder und Decoder unabhängig eingesetzt werden (HuggingFace, o. J.):

- **Encoder Modelle** eignen sich für Tasks, die sich mit dem Verstehen von Inputs befasst, wie sentence-classification und named-entity-recognition
- **Decoder Modelle** eignen sich für generative Tasks, wie text generation
- **Encoder-Decoder-Modelle** eignen sich für generative Tasks, welche einen Input erfordern, wie translation oder summarization

2.2 Record-Linkage-Process

Record-Linkage wird zur Verbindung von unterschiedlichen Datensätzen zu einer Entität genutzt (Christen, 2019). Wenn die Datensätze einen gemeinsamen Identifikator, zum Beispiel eine Interne Firmen-ID, haben, wird das deterministische Record-Linkage eingesetzt, welches die Einträge anhand dieses Identifikators miteinader verbindet (Shlomo, 2019). Diese Arbeit beschäftigt sich mit Datensätzen unterschiedlicher Unternehmen und Datenquellen, die keinen gemeinsamen Identifikator aufweisen. Deshalb wird sich auf das probabilistische Record-Linkage fokussiert. Damit dieser Vorgang automatisch durchgeführt werden kann, werden Bibliotheken (libraries) und Toolkits, wie Pandas und Numpy, für Python benutzt. Das Python Record Linkage Toolkit[5] ist eine Bibliothek, welche Blocking-Methoden und Compare-Functions zur Verfügung stellt, um Record-Linkage auf Datenquellen durchzuführen (De Bruin, 2023). Dieses Toolkit dient als Leitfaden für die Implementierung von Record-Linkage in Python. Folglich werden die Record-Linkage Schritte und derer Umsetzung im Record-Linkage Toolkit erklärt.

Preprocessing

Der erste Schritt beim Record-Linkage ist das Data Pre-Processing, bei dem jeder Eintrag aus den Datenquellen in seine Variablen zerlegt und in eine einheitliche Form gebracht wird (Christen, Churches & Zhu, 2002, S. 99–108). Das Ziel von Preprocessing ist die Homogenisierung von Einträgen, um ein möglichst genaues Matching zu erzielen (Christen, 2012c, S. 39–67). Dieser Schritt ist notwendig, da jede Datenquelle ihre eigene Struktur für die Speicherung ihrer Dateneinträge hat (Tab. 2). Dadurch, dass die Einträge fehlerhaft sein können, ist *Data Pre-Processing* ein wichtiger Schritt, um ein qualitativ hochwertiges Ergebnis des *Record-Linkage* zu erhalten (Christen, 2012c, S. 39–67).

Adresse aus Datenquelle A	Adresse aus Datenquelle B
Posthalterweg	CLOPPENBURGER STR. 457
Würzburger Str.	SUEDWEG 12 A
Dietrichsweg	CAECILIENPLATZ 1

Tabelle 2: Darstellung unterschiedlicher Adresseinträge aus unterschiedlichen Datenquellen

Das Preprocessing ist Abhängig von den Spalteneinträgen der Datensätze. Numerische Einträge sollen im selben Format dargestellt werden, während in Texteinträge Satzzei-

[5] Python Record Linkage Toolkit

chen und Zahlen entfernt werden. Das Record-Linkage Toolkit enthält mehrere Tools, welche sich im Submodul *recordlinkage.preprocessing*[6] befinden und zum Preprocessing von Einträgen benutzt werden können. Das Toolkit benötigt eine Reihe von Einträgen (pandas.Series), die in einem Pandas Dataframe als Spalteneinträge liegen können. Als Standard Preprocessing-Funktion dient *recordlinkage.preprocessing.clean()*, welche Zahlen, Klammern und weitere Zeichen aus den Spalteneinträgen entfernen kann (De Bruin, 2023), so dass nur der gewünschte Textabschnitt in Groß- oder Kleinschreibung in den Dataframe-Spalten bestehen bleibt (Tab. 3).

Adresse	Bereinigte Adresse
CLOPPENBURGER STR. 457	cloppenburger str
SUEDWEG 12 A	suedweg a
CAECILIENPLATZ 1	caecilienplatz

Tabelle 3: Bereinigte Adressen mit Python Record Linkage Toolkit

Neben dem Bereinigen und Homogenisieren von Einträgen, können diese phonetisch dargestellt werden (Tab, 4), dazu bietet das Toolkit die Methoden "`soundex`", "`nysiis`", "`metaphone`" und "`match_rating`"

Bereinigte Adresse	soundex	nysiis	metaphone	match_rating
cloppenburger str	C415	CLAPANBARGARSTR	KLPNBRJRSTR	CLPSTR
suedweg a	S320	SADWAG	STWK	SDWG
caecilienplatz	C245	CACALANPLAT	KSLNPLTS	CCLLTZ

Tabelle 4: Phonetisch kodierte Adresseinträge

Phonetische Kodierung wird genutzt um die Typografischen Ungleichheiten in Texteinträgen zu eliminieren. Der erste Kodierungs-Algorithmus für die Englische Sprache war SoundEx, welcher für die Koderung von Namen nach Klang eingesetzt wurde. Dadurch, dass Namen wie "Smith", "Smithe" und "Smyth" die selbe Aussprache haben, kodiert SoundEx alle mit dem Code "S530" (Vykhovanets, Du & Sakulin, 2020). Phonetisches Kodieren führt zu einer besseren Performance bei den Schritten Blocking und Comparison (De Bruin, 2023).

[6] Record Linkage Toolkit Preprocessing

Indexing

Record-Linkage vergleicht jeden Eintrag aus jeder Datenquelle, dadurch steigt die Anzahl der benötigten Rechenoperationen exponentiell an (Michelson & Knoblock, 2006, S. 440). Indexing beschreibt den Schritt der Erstellung von Paaren aus angegeben Datenquellen. Diese Paare werden als Kanditaten (candidate links/candidate matches) bezeichnet (Christen, 2012b) und verringern die Rechen-Komplexität des Record-Linkage-Prozesses (Christen & Goiser, 2007). Das Record Linkage Toolkit bietet einige Indexing-/Blocking-Algorithmen wie "Full", "Block" und "SortedNeighbourhood" und befinden sich im Submodul *recordlinkage.index*[7]. Die Klasse *recordlinkage.index.Full* generiert einen Index mit dem Kartesischen Produkt von beiden Dataframes. Es beinhaltet alle möglichen Kombinationen von Paaren. Mit *recordlinkage.index.Block* werden Kandidaten-Paare erstellt, welche den selben Wert in einer oder mehreren Variablen haben. Der Algorithmus für SortedNeighbourhood, welcher sich in der Klasse *recordlinkage.index.SortedNeighbourhood* befindet, erstellt Kandidaten-Paare, welche den selben Wert in einer Schlüsselvariable (sorting key) aufweisen und Kandidaten-Paare welche sich in derer Umgebung befinden. Dieser Algorithmus eignet sich, wenn sich die Einträge syntaktisch zu stark von einander unterscheiden (zum Beispiel wegen Rechtschreibfehler) und der Blocking-Algorithmus deswegen zu viele Einträge außen vor lässt (De Bruin, 2023). Die Anzahl der Paare im SortedNeighbourhood-Algorithmus wird mit dem "`window`" Parameter bestimmt, wobei "`window = 1`" der Blocking-Index ist. In Tab. 5 wurden die drei erwähnten Indexing-Algorithmen auf die Datensätze aus Tab.14 angewandt. Als Schlüsselvariable dienten die zuvor bereinigten Adresseinträge.

Anz. Datenquelle A	Anz. Datenquelle B	Full	Blocking	SortedNeighbourhood
15652	68163	998641462	258566	419301

Tabelle 5: Indexing von Einträgen aus zwei Datenquellen, SortedNeighbourhood window=5

Die Auswirkung von phonetischer Kodierung auf die Menge der gebildeten Paare ist in Tab. 6 dargestellt.

[7]Record Linkage Toolkit Indexing

Kodierung	Blocking	SortedNeighbourhood
`soundex`	861551	1701940
`nysiis`	287582	454972
`metaphone`	295510	458269
`match_rating`	666770	868985

Tabelle 6: Indexing von Einträgen aus zwei Datenquellen mit unterschiedlichen phonetischen Kodierungen, SortedNeighbourhood window=5

Comparison

Nachdem die Paare gebildet wurden, folgt der Vergleich, oder Comparison. Das Paar wird anhand mehrerer vorhandener Variablen verglichen und ein Vektor mit numerischen Werten erstellt, welcher die Ähnlichkeit beider Paar-Einträge darstellt (Christen, 2012e, S. 101–127).Die Klasse *recordlinkage.Compare*[8] bietet Methoden für das Vergleichen dieser Eigenschaften. Die Methoden für das Coparing beinhalten unter anderem den Vergleich von Datum, Geographischer Koordinaten, Numerischer Vergleich und exakter Vergleich (De Bruin, 2023). Die wichtigste Klasse für diese Arbeitet ist *recordlinkage.compare.String*, welche für das Vergleichen von Strings zuständig ist, da beide Datenquellen zukünftig auf drei String-Variablen verglichen werden. Für den Vergleich von Strings stell die Klasse die Algorithmen "`jaro`", "`jarowinkler`", "`levenshtein`", "`damerau_levenshtein`", "`qgram`" und "`cosine`". Alle Algorithmen geben die Ähnlichkeit als einen Float von 0 (keine Ähnlichkeit) bis 1 (absolute Ähnlichkeit) an. Der Unterschied bei der Ähnlichkeitszuweisung der Algorithmen auf ein Kandidaten-Paar (Tab.7) wird in (Tab.8) aufgezeigt.

Adresse	Firmenname aus Datenquelle A	Firmenname aus Datenquelle B	Paar-Nr.
posthalterweg	NMS NORDWEST MANAGEMENT SERVICE GMBH	MEDIA MARKT TV-HIFI- ELEKTRO GMBH	1
dietrichsweg	HBO VERWALTUNGS GMBH	HBO VERWALTUNGS GMBH	2

Tabelle 7: Blocking-Paare von zwei Datenquellen, welche auf Adressen geblockt wurden

`jaro`	`jarowinkler`	`levenshtein`	`damerau_levenshtein`	`qgram`	`cosine`	`Paar-Nr.`
`0.623`	`0.623`	`0.25`	`0.25`	`0.243`	`0.279`	`1`
`1.0`	`1.0`	`1.0`	`1.0`	`1.0`	`1.0`	`2`

Tabelle 8: Anwendung der Comparison Algorithmen den Firmennamen zweier Paare, welche auf Adressen geblockt wurde

[8] Record Linkage Toolkit Comparing

Während die Paar-Nr.2 von allen Algorithmen mit einem Wert von 1.0 als gleich erkannt wird, gibt es bei der Paar-Nr.1 Diskrepanzen bezüglich des Ähnlichkeits-Wertes. Die Comparison-Methoden besitzen einen Threshold-Parameter, der individuell angegeben werden kann. Durch den Threshold werden die Ähnlichkeits-Werte nicht als Float dargestellt, sondern nur noch in die Kategorien 0 für nicht ähnlich (unterhalb des Thresholds) und 1 für absolut ähnlich (oberhalb des Thresholds) eingeteilt.

Classification

Bei der Classification werden die Werte des Vektors genutzt, um zu entscheiden, ob das Paar zur selben Entität gehört (Christen, 2012a, S. 129–162). Dabei werden die Kandidaten Paare in Matches, non-Matches und possible Matches unterteilt (Christen, 2012b). Die Klassifizierungs-Algorithmen, welche beim Record Linkage Toolkit vertreten sind[9], können in supervised (mit Trainingsdaten) und unsupervised (ohne Trainingsdaten) unterteilt werden. Zu den supervised Classifiern zählen der LogisticRegressionClassifier, welcher auch als deterministischer Record-Linkage Algorithmus bekannt ist, der NaiveBayesClassifier und der SVMClassifier (De Bruin, 2023). Der ECMClassifier und KMeansClassifier gehören zu den unsupervised Classifiern.

Evaluation

Das Ergebnis des gesamten Record-Linkage Prozesses wird im Schritt Evaluation bewertet. Dabei spielt die Qualität der Einträge eine entscheidende Rolle. Sie lässt sich daran messen, wie viele der abgeglichenen Daten-Paare sich auf dieselbe Entität beziehen und wie viele der bekannten übereinstimmenden Daten-Paare erkannt und als Treffer klassifiziert wurden.(Christen, 2012d, S. 163–184). Die Bewertung des Prozesses teilt sich in folgende Auswertungen auf (De Bruin, 2023):

- **True Positives**: Die Menge der korrekt gematchten Paare wird True Positives (TP) genannt
- **True Negatives**: Die Menge der korrekt zugeordneten non-matches werden True Negatives (TN) bezeichnet
- **False Positives**: Die Menge der falsch zugeordneten non-matches (Paare, die keine Matches sind, aber als Matches eingeordnet werden) werden als False Positives (FP)

[9] Record Linkage Toolkit Classifier

bezeichnet

- **False Negatives**: Die Menge der falsch zugeordneten Matches (Paare, die Matches sind, aber als non-Match zugeordnet werden) werden False Negatives (FN) genannt
- **Precision**: Die Precision gibt das Verhältnis von gebildeten Paaren zu korrekten Paaren an und wird mit $TP/(TP + FP)$ berechnet
- **Recall**: Der Recall gibt das Verhältnis von gebildeten Paaren zu der menge an existierenden Matches an und wird mit $TP/(TP + FN)$ berechnet
- **Accuracy**: Die Accuracy gibt das Verhältnis von richtig eingeordneten Paaren zu der Menge aller paare an und wird mit $(TP+TN)/(TP+FP+TN+FN)$ berechnet
- **F-Score**: Der F-Score ist das harmonische Mittel von Precision und Recall und wird mit $2 * (Precision * Recall)/(Precision + Recall)$ berechnet

3 Umsetzung und Implementierung

In diesem Kapitel wird die Umsetzung des Record Linkage mit LLMs, sowie die benutzten Bibliotheken, Module und Language-Models erklärt. Als Basis-Struktur für den Linkage-Prozess wird das Python Record Linkage Toolkit, welches in Kapitel 2.2 eingeführt wurde. Dessen Schritte werden in Abb. 4 blau dargestellt und die Code implementierung in Abb. 15. Die Ergebnisse des Toolkits werden als Referenz genommen, mit der die Performance des LLM-Einsatzes in Kapitel 5 verglichen wird. Der gesamte Code mit seinen Anwendungen befindet sich in dem GitLab-Repository (Link Hier).

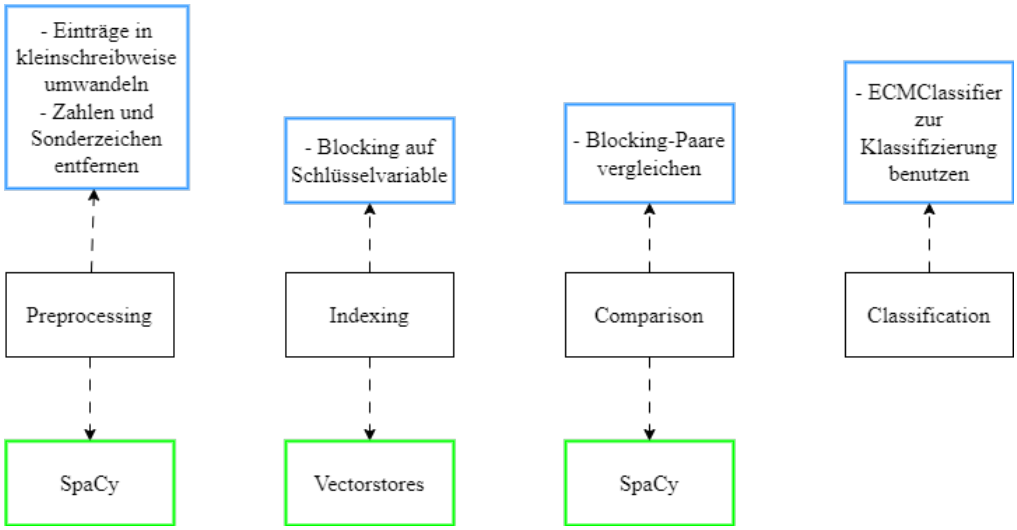

Abbildung 4: Umsetzung des Record-Linkage Prozesses, blau ist die Umsetzung mit dem Record Linkage Toolkit, grün ist die geplante Umsetzung mit LLMs

Der Evaluations-Schritt des Record-Linkage wird manuell, ohne Einsatz von Toolkits, gemacht und wird in Kapitel 5.1 erklärt. Die Implementierung der LLMs verläuft in jedem Schritt gekapselt, so dass eine Implementierung des Record-Linkage Toolkits mit den LLM-Methoden gewährleistet ist und eine Evaluation der LLM-Methode für jeden Schritt separat erfolgen kann. Die Auswahl der Taskspezifischen LLMs ist abhängig von der Methode der Implementierung und wird in Kapitel 3.4 behandelt.

3.1 SpaCy

SpaCy ist eine Open-Source Bibliothek, welche eine vielzahl von Funktionen für Text-Processing bietet (Honnibal & Montani, 2017). Ihre nlp Pipeline bietet Funktionen wie Tokenization, Similarity und Text Classification. In Abb. 5 wird der Ablauf der Pipeline

dargestellt. Wenn die nlp-Pipeline auf eine Texteingabe angewendet wird, wird der Text in Tokens gespalten und ein Doc Objekt erstellt, welches in mehreren schritten weiter verarbeitet wird. Die Pipeline enthält normalerweise einen Tagger, Lemmatizer, Parser und einen Entity-Recognizer. Jede Pipeline-Komponente gibt das verarbeitete Doc-Objekt aus, welches zur nächsten Komponente weitergereicht wird (Honnibal & Montani, 2017).

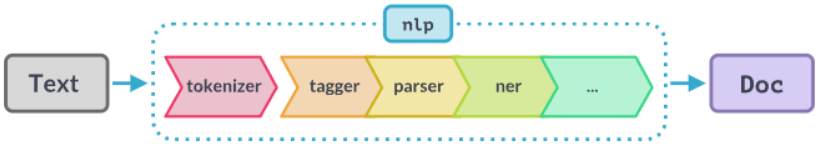

Abbildung 5: Processing Pipeline von SpaCy, Entnommen aus: Language Processing Pipelines

SpaCy bietet, neben der Möglichkeit eigene Pipelines zu trainieren, bereits trainierte Pipelines für 25 unterschiedliche Sprachen (Honnibal & Montani, 2017). Die nlp-Pipeline kann für das Untersuchen von Text, als auch für das Vergleichen von Text verwendet werden und wird im Record-Linkage beim Preprocessing und Comparison implementiert.

3.2 LangChain-Vectorstores

LangChain ist ein Open-Source Framework zum entwickeln von LLM-basierender Anwendungen (LangChain, 2024). Es bietet Module für unterschiedliche Bereiche der Arbeit mit LLMs, wie zum Beispiel das Bereitstellen mehrerer Chat-Modell-Packages[10], Document-Loaders für RAG[11] und How-to's für unterschiedliche Anwendungen von LLMs[12], welche regelmäßig geupdated werden. Neben den Document-Loaders und Document-Splitters besitzt LangChain eine Sammlung von Vectorstores, welche für das erstellen von RAG(Retrieval-Augmented-Generation)-Chatbots benutzt werden können. LangChain wird aufgrund der einfachen Implementationsmöglichkeit von Vectorstores verwendet.

3.3 LinkTransformer

Die 2023 veröffentlichte Bibliothek LinkTransformer befasst sich mit dem Zusammenführen (Mergen) und Deduplizierung von Daten mit Large-Language-Models. Die Bibliothek ist auf die Benutzung von Transformer-Modellen ausgelegt und kann 1:1, 1:m und m:1 Merges,

[10] LangChain Chat Models
[11] LangChain Document loaders
[12] LangChain How-to's

basierend auf den Ähnlichkeiten der Embeddings, vornehmen und können neben dem Mergen von Daten weitere Schritte des Record-Linkage, so wie Blocking und Preprocessing, übernehmen (Arora & Dell, 2023, S. 1). Um die Anwendung von LLMs im RL zu vereinfachen, damit jeder ohne eine Vielzahl von Berührungspunkten mit Deep-Learning einen leichteren Einstieg hat, bietet LinkTransformer eine vielzahl von Eigenschaften, so wie ein eigenes Repository von pre-trained semantic similarity Modellen, eine einfache Integration von beliebigen Transformer-Modellen, APIs, die weitere RL Schritte übernehmen können und Tools für das fine-tunen von Modellen (Arora & Dell, 2023, S. 2). Record-Linkage wird von LinkTransformermit als ein knn-Retrieval-Task, bei der der nächste Nachbar für jede Entität in einem Datasatz aus einem Schlüssel-Embedding (key embedding) Datensatz abgerufen wird, behandelt. Dabei wird die Kosinusähnlichkeit benutzt, welche in einem FAISS back-end implementiert wird (Arora & Dell, 2023, S. 3). In Abb. 6 wird die LinkTransformer Architektur visualisiert.

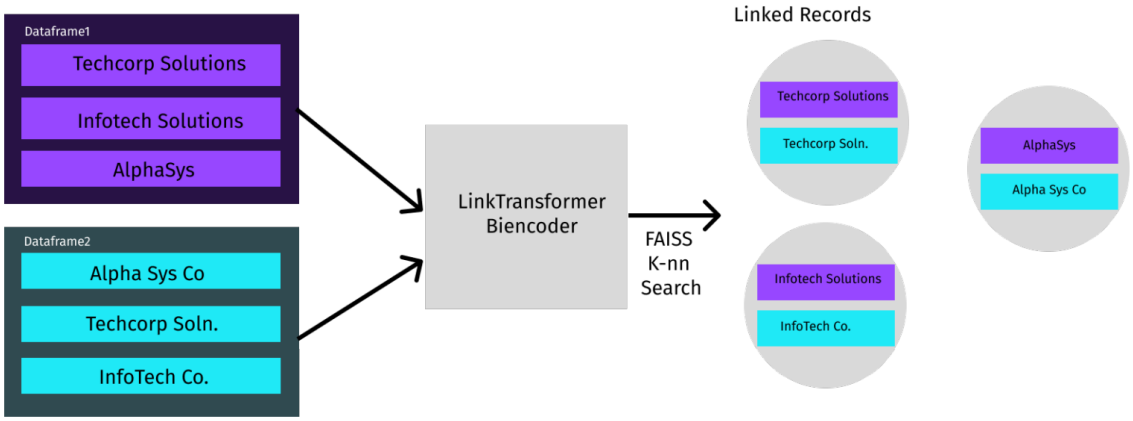

Abbildung 6: Visuelle Darstellung der LinkTransformer Architekture, Entnommen aus: (Arora & Dell, 2023, S. 3)

Dadurch, dass LinkTransormer eigenständig Paare bildet und einen Score der Paare wiedergibt, wird diese Bibliothek, ohne das Verwenden eines Blocking-Verfahrens, für den Comparison-Schritt benutzt.

3.4 Auswahl der LLMs

Jede Implementationsmöglichkeit besitzt eigene Language-Models, welche für den jeweiligen Task, oder Implementation, ausgelegt sind. Folglich werden die gewählten Modelle aufgezeigt:

- **SpaCy**:
  - `en_core_web_sm`: standardmäßige Pipeline für die Englische Sprache, wird für das Preprocessing verwendet
  - `en_core_web_lg`: die größere Pipeline für das Vergleichen von Embeddings
- **Vectorstores**: Die Vektorsuche verläuft über die Embeddings, die durch ein Text Embedding Model generiert wurden. Eine Handvoll Embedding-Modelle wurde aus dem Massive Text Embedding Benchmark (MTEB) Leaderboard (Muennighoff, Tazi, Magne & Reimers, 2022) ausgewählt:
  - `sentence-transformers/all-MiniLM-L6-v2`
  - `BAAI/bge-base-en-v1.5`
  - `BAAI/bge-m3`
  - `BAAI/bge-large-en-v1.5`
  - `intfloat/multilingual-e5-large`
- **LinkTransformer**: Dadurch, dass LinkTransformer mit jedem Transformer-Modell funktioniert, wurden neben den ausgewählten Embedding-Modellen auch die von LinkTransformern bereitgestellten Modelle untersucht:
  - `dell-research-harvard/lt-wikidata-comp-multi`
  - `dell-research-harvard/lt-wikidata-comp-de`
  - `dell-research-harvard/lt-wikidata-comp-en`

Die Auswahl der Embedding-Modelle wurde auf Grund der Score-Werte(Tab. 9) der einzelnen Modelle getroffen, welche auf dem Huggingface MTEB-Leaderboard[13] zu finden sind. Dabei ist der Name des Modells die vollständige Huggingface-Kennzeichnung.

[13] Huggingface MTEB-Leaderboard

Modell	Average	Benchmark	STS-DE	STS-EN-DE	STS-DE-EN
sentence-transformers/all-MiniLM-L6-v2	78.9	82.03	31.04	35.82	44.04
BAAI/bge-base-en-v1.5	83.11	86.42			
BAAI/bge-large-en-v1.5	83.11	87.52			
BAAI/bge-m3		84.85			
intfloat/multilingual-e5-large	81.56	87.29	56.59	86.16	56.6

Tabelle 9: MTEB-Scores der Embedding Modelle

4 Einsatzgebiete von LLMs

4.1 Preprocessing

Damit ein Vergleich von den Methoden des Record Linkage Toolkits zu der Implementierung von SpaCy gemacht werden kann, werden in beiden Anwendungen die selben Preprocessing Schritte gemacht. Diese beinhalten das Konvertieren der Einträge in Kleinschreibung und das Entfernen von Zahlen und Sonderzeichen. Mit der nlp-Pipeline und dem Modell `en_core_web_sm` wird jedes Token aus dem Text verglichen, um festzustellen, ob es sich um ein Sonderzeichen handelt (Abb. 12).

Um nicht direkt auf der nlp-Pipeline zu arbeiten, kann die Python Bibliothek textacy verwendet werden, die auf SpaCy basiert (DeWilde, 2023). Mit textacy können Dokumente bearbeitet werden, wobei die Schritte der tokenization, part-of-speech tagging und dependency parsing ausgelagert werden. Dies kann unter Umständen zu einer einfacheren Implementierung des Preprocessing von Texten führen. Die Implementierung von textacy ist in Abb. 13 zu sehen.

4.2 Blocking

Für das Blocking werden Vectorstores verwendet, welche mit Hilfe von LangChain[14] implementiert werden. Der Kerngedanke ist, die Einträge einer Datenquelle A, welche als Schlüsselvariablen für das Blocking dienen sollen, als Embeddings in einen Vectorstore zu speichern und eine Ähnlichkeitssuche auf die zu blockenden Schlüsselvariablen von Datenquelle B durchzuführen. Eine einfache Darstellung einer Vektorsuche wird in Abb. 7 dargestellt.

[14]LangChain Vectorstores

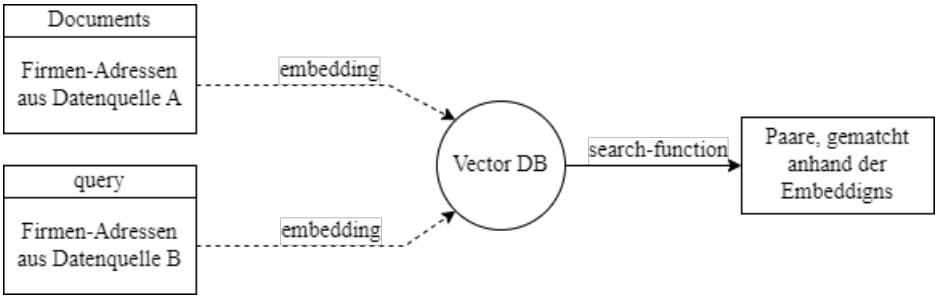

Abbildung 7: Darstellung einer Vektorsuche mit einem Vectorstore (VectorDB), eigene Darstellung

Dabei dienen die Adress-Variablen in den Documents als Schlüsselvariable, mit denen die Adress-Variablen aus der query verglichen werden. Als Vectorstore wird ChromaDB verwendet, aufgrund der einfacheren Bedienung dank LangChain. Die Embeddings werden mit dem Modell `BAAI/bge-base-en-v1.5` generiert. Für die search-functon wird die von LangChain bereitgestellte Funktion

`similarity_search_by_vector_with_relevance_scores`

verwendet, die folglich als similarity-function bezeichnet wird, welche die query-Embeddings mit den Document-Embeddings vergleicht und das Ergebnis zusammen mit einem Score zurückgibt. Da die similaritiy-function nur die Embeddings vergleicht, welche in der query übergeben werden, muss diese über die gesamte Länge der Datensätze angewandt werden, um die größtmögliche Menge an Blocking-Paaren zu erhalten. Dadurch werden matches gebildet, die keine True Matches sind. Der von der Funktion ermittelte Score ist der Kosinusabstand beider Embeddings, was eine Ähnlichkeit bei kleineren Werten und eine Unähnlichkeit bei größeren Werten darstellt. Um die Menge von den False Positve Matches zu verringern, werden nur die Paare als Blocking-Paare genommen, welche einen bestimmt Kosinusabstand nicht übersteigen, was zu einer höheren Chance führt, dass die Paare True Matches sind.

4.3 Comparison und Classification

Bei den Schritten Comparison und Classification wird anhand des Scores entschieden, ob zwei Einträge zur selben Entität gehören. Es werden zwei mögliche Vorgehen behandelt, wie LLMs eingesetzt werden können. Das Erste Vorgehen benutzt die Open-Source Python Bibliothek SpaCy, welche ebenfalls im Schritt Preprocessing benutzt wird. Das zweite Vorgehen befasst sich mit dem Einsatz von LinkTransformern. Es wird der selbe Classifier

benutzt wie bei dem herkömmlichen Vorgehen, da ein LLM-Classifier mit dem selben Prinzip funktionieren würde.

4.3.1 Einsatz von SpaCy

Anders als bei Preprocessing wird jetzt das Modell `en_core_web_lg` verwendet. Der Grund ist, laut eigener Aussage, dass die kleinen Packages (`sm`) nur Kontext-sensitive Tensoren enthalten um kompakt und schnell zu sein. Daher muss das größere Package (`lg`) installiert werden, welches die Embeddings enthält[15]. Das Modell wird mit SpaCy geladen und mit Hilfe der Methode *entry1.similarity(entry2)* erhält man die Ähnlichkeit der Einträge entry1 und entry2. Die Methode berechnet die Kosinusähnlichkeit der Embeddings für entry1 und entry2 und gibt einen Score aus (Honnibal & Montani, 2017). Die Scores werden mit einem Threshold gefilter, damit der Classifier die selbe Struktur eines Dataframes bekommt wie bei der herkömmlichen Record-Linkage Implementierung.

4.3.2 Einsatz von LinkTransformer

Für Comparison und Classification wird sich auf die Merge-Funktionalität von LinkTransformer fokussiert. Diese funktioniert ähnlich zu der Dataframe.merge() Methode aus der Pandas Bibliothek (Abb. 10).

Damit die LinkTransformer Methode eine Zusammenführung durchführen kann, muss der Spaltenname auf dem Zusammengeführt wird identisch sein. Nach dem Einsatz der Methode erhält man ein Dataframe mit den Paaren und einen dazugehörigen Score, welcher ähnlich zu Kap. 4.3.1, dazu genutzt werden kann, um die Paare zu Klassifizieren. Dadurch, dass LinkTransformer, außer den identischen Spaltennamen, kein Preprocessing und Indexing benötigt, erhält man im Fall eines direkten Merges, wie in Abb. 10, die selbe Anzahl an Paaren, wie die Länge des zuerst angegebenen Dataframes (df1) beträgt.

[15] SpaCy's vector similarity

5 Evaluation

5.1 Ergebnisfindung und Erklärung

Die Untersuchung des Einsatzes von LLMs im Record-Linkage soll mit echten Firmendaten erfolgen, welche von der Datenschmiede.ai GmbH zur Verfügung gestellt wurden. Diese beinhalten neben den Firmeneinträgen (wie Firmennamen, Adressen, Postleitzahlen oder URL-Links) auch die Matches, welche zuvor mit erstellt wurden. Diese Matches dienen als Grundlage für die Auswertung des RL-Prozesses und werden Ground Truths genannt. Als Datenquellen wurden `Databyte` (db) und `Bureau van Dijk` (bvd) gewählt, da dessen Ground Truth mit insgesamt 2213 Einträgen (Tab. 10) eine ausreichende Menge von True Positives und True Negatives zur Evaluation des RL aufweist, mit einer überschaubaren Anzahl der Datensätze in den einzelnen Datenquellen (Tab. 14).

Anz. Einträge	Anz. TP	Anz. TN
2213	1225	988

Tabelle 10: Informationen der Ground Truth

Der Aufbau der Ground Truth wird in Tabelle 11 gezeigt. Dabei dienen die Label 0 für TN und 1 für TP. Die Kommentar-Spalte gibt Informationen bezüglich der TP-TN Zuweisung, wird für die Auswertung des RL-Prozesses, nicht weiter genutzt.

databyte_id	bvd_id	label	comment
14483	58108	0	(NULL)
1097	31886	1	Standort

Tabelle 11: Aufbau der Ground Truth

Der Aufbau der Datenquellen ist in Tab. 12 und Tab. 13 zu sehen.

id	Firma	Strasse	Hausnr	Plz	...
14483	REWE Markt Oldenburg	Hauptstr.	89-95	26131	...
1097	Kühne Verwaltungsgesellschaft mbH	Steinkamp	4	26125	...

Tabelle 12: Aufbau der Databyte Datenbank

id	companyname	address	addresspostalcode	...
58108	REWE-MARKT SCHEUNER OHG		37181.0	...
31886	KUEHN VERWALTUNGS GMBH		21640.0	...

Tabelle 13: Aufbau der Bureau van Dijk Datenbank

Neben den unterschiedlichen Eintragungsarten der Firmennamen und Postleitzahlen ist die Abwesenheit von Adress-Einträgen in der bvd-Datenbank zu beachten. Dies hat einen Einfluss auf das Blocking und Comparison der jeweiligen Einträge. Die Spalten-Einträge der Adressen und Postleitzahlen wurden neben den Firmennamen als Untersuchungsmerkmale ausgewählt, da diese nicht nur Buchstaben sondern auch Zahlen in den Einträgen besitzen und somit eine Untersuchung auf das Verständnis von LLMs zu Einträgen mit gemischtem Text gemacht werden kann.

Datenquelle	Anz. Datensätze	Leere Adresseinträge	Leere PLZ-Einträge
Databyte	15652	24	0
Bureau van Dijk	68163	41214	865

Tabelle 14: Datenquellen und Informationen

Vor dem Ausführen eines Record-Linkage Schrittes werden die Spaltennamen der Datenbanken einheitlich benannt (Tab. 15). Dieser Prozess wird bei dem Erstellen der Dataframes, in denen die Einträge aus den Datenquellen eingetragen werden, für beide Datenquellen durchgeführt. Der Grund ist die Implementierung der LinkTransformer, welche voraussetzt, dass der Spaltenname der zwei Dataframes, auf dem die Merge-Methode ausgeführt wird, gleich ist.

CompanyName	CompanyPostalCode	CompanyAdresses	id
...	...	...	...

Tabelle 15: Einheitliche Spaltenbeschriftungen der Datenquellen

Nachdem der Record-Linkage Prozess durchgeführt und die Matches erhalten wurden, wird die Ground Truth als ein Dataframe geladen und die Matches werden mit der Ground Truth anhand der id's zusammengeführt. Dadurch entsteht ein Dataframe bei dem die True Positives eindeutig identifiziert werden können.

Als Classifier wird der ECMClassifier aus dem Record Linkage Toolkit verwendet. Dieser verwendet den Expectation-Maximization-Algorithmus (De Bruin, 2023), welcher den

Mittelpunkt eines Datenclusters anhand der Menge der Dateneinträge bestimmten kann (Moon, 1996). Für Record-Linkage bedeutet das, wenn die Comparison-Methode zwei Attributen den Score 1 und einem Attribut den Score 0 verleiht, wird das Paar als ein Match klassifiziert. Der ECMClassifier kann für das Klassifizieren von den LLM-Comparisons genutzt werden, da diese ebenfalls einen Score ausgeben. Für die Auswertung der LinkTransformer Matches wird ein eigenes Vorgehen benutzt, da das Ergebnis-Dataframe, welches LinkTransformer nach einem Matching ausgibt, nur einen Score beinhaltet und der ECMClassifier dafür ungeeignet ist. Die LinkTransformer Merge-Methode gibt das Dataframe mit den Kandidaten und deren Scores. Diese Kandidaten werden als Matches angesehen. Das Ergebnis-Dataframe wird dann anhand der `databyte_id` und `bvd_id` mit der Ground Truth zusammengeführt, um die TPs und TNs zu erhalten. Anschließend wird ein Threshold für die Scores gesucht, bei denen die höchste Präzision vorliegt. Der Gedanke hierbei ist, einen Threshold-Wert zu finden, der eine hohe Präzision mit dem kleinsten Verlust des Recalls aufweist, ohne dass der Wert manuell ermittelt werden muss.
Die Evaluation verläuft für den ECMClassifier mit der Methode

`generate_match_calculation()`

welche im File `MatchEval.py` Implementiert ist und für LinkTransformer mit der Methode

`generate_match_eval_calculation()`

welche in `MatchEval_Calc.py` definiert wird. Beide Methoden Berechnen die TP, FP, Precision, Recall, F1 und speichern die Ergebnisse als Excel-Datei. Die Namensgebung der Dateien ist wie folgt:

- `bvd_db_exp{Experiment Nr.}` für alle Matches
- `bvd_db_exp{Exp. Nr.}_neg` für die True Negatives
- `bvd_db_exp{Exp. Nr.}_pos` für die True Positives
- `bvd_db_exp{Exp. Nr.}_eval` für die Evaluationsberechnungen wie Precision, Recall und F1
- `blocked_bvd_db_exp{Exp. Nr.}` für die geblockten Paare mit dem Vectorstore

Die Dateien `bvd_db_exp{Experiment Nr.}` sind im Ordner

`LinkTransformer/Matched_Dataframes/DB_BVD`

und die Restlichen Dateien unter

`Compared-Dataframes/DB_BVD_COL`

zu finden.

5.2 Darstellung der Ergebnisse

5.2.1 Record Linkage Toolkit

Um einen Anhaltspunkt über die Performance für die herkömmlichen Record-Linkage Vorgehen zu erschaffen, wurde der Prozess mit Firmennamen, Adressen und Postleitzahlen als Schlüsselvariablen für das Blocking durchgeführt (Tab. 16) und Recall, Precision und F1-Score berechnet, welche als Vergleichswerte dienen sollen. Die Comparison-Methode für String ist `jarowinkler` mit einem Threshold von 0.8 und die Methode für das Vergleichen der Postleitzahlen `compare.exact()`. Als Referenz werden die Experimente `bvd_db_exp{Experiment Nr.}` fortlaufend mit `exp{Experiment Nr.}` betitelt.

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
258,566	Adresse	1,365	503	862	722	0.41	0.37	0.39	exp154
258,566	Adresse	61,332	511	60,821	714	0.42	0.01	0.02	exp156
827	Name	507	444	63	781	0.36	0.88	0.51	exp157
827	Name	504	441	63	784	0.36	0.88	0.51	exp159
2,506,133	Plz	1,482	629	853	596	0.51	0.42	0.46	exp160
2,506,133	Plz	1,482	629	853	596	0.51	0.42	0.46	exp161

Tabelle 16: Record-Linkage angewand auf Firmendaten mit dem Record Linkage Toolkit

Die in Tab. 16 durchgeführten Matchings wurden zuerst auf Firmennamen und Adressen (z.B. exp154) und anschließend auf Firmennamen, Adressen und die Postleitzahl (PLZ) verglichen (z.B. exp 156). Dieses Matching-Verfahren wird über den gesamten Evaluations-Verlauf eingehalten. In Tab. 17 werden die Matching-Abläufe visualisiert.

Blocking Paare	Blocking	Matches	EXP	Comparison
258,566	Adresse	1,365	exp154	Name, Adresse
258,566	Adresse	61,332	exp156	Name, Adresse, PLZ
827	Name	507	exp157	Name, Adresse
827	Name	504	exp159	Name, Adresse, PLZ
2,506,133	Plz	1,482	exp160	Name, Adresse
2,506,133	Plz	1,482	exp161	Name, Adresse, PLZ

Tabelle 17: Comparison-Ablauf für die Evaluation

5.2.2 Preprocessing mit LLMs

In Tab. 18 wurden die Dataframes mit den LLM-gereinigten Einträgen in die Record Linkage Toolkit Methoden übergeben, dabei blieben Matching-Methoden und Einstellungen

wie in Kapitel 5.2.1

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
702	Adresse	6	3	3	1,222	0	0.50	0	exp164
702	Adresse	414	3	411	1,222	0	0.01	0	exp165
1,200	Name	429	387	42	838	0.32	0.9	0.47	exp162
1,200	Name	426	384	42	841	0.31	0.9	0.47	exp163
2,506,133	Plz	1,502	626	876	599	0.51	0.42	0.46	exp166
2,506,133	Plz	1,502	626	876	599	0.51	0.42	0.46	exp167

Tabelle 18: Record-Linkage angewand auf Firmendaten mit textacy für das Preprocessing

5.2.3 Blocking mit LLMs

Für Comparison und Classification wurden weiterhin die Methoden aus dem Toolkit verwendet. Das Blocking erfolgt in der ChromaDB mit der in Kapitel 4.2 beschriebenen similarity-function. Für das Preprocessing wird sowohl die `preprocessing.clean()` Methode als auch textacy verwendet. Dadurch, dass die Methode den Kosinusabstand als Wert ausgibt, wurden nur Blocking-Paare mit einem Wert $\leq$0.30 genommen, um die Kandidaten mit dem wenigsten Abstand zu erhalten (Tab. 19 und Tab. 20).

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
3,649	Adresse	14	8	6	1,217	0.01	0.57	0.01	exp168
3,649	Adresse	526	8	518	1,217	0.01	0.02	0.01	exp169
2,142	Name	475	438	37	787	0.36	0.92	0.52	exp170
2,142	Name	639	589	50	636	0.48	0.92	0.63	exp171
1,293	Plz	0	0	0	0	0	0	0	exp172
1,293	Plz	0	0	0	0	0	0	0	exp173

Tabelle 19: Blocking mit der ChromaDB als Vectorstore, Preprocesseing mit dem Record Linkage Toolkit

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
3,722	Adresse	0	0	0	0	0	0	0	exp174
3,722	Adresse	16	10	6	1,215	0.01	0.63	0.02	exp175
2,244	Name	0	0	0	0	0	0	0	exp176
2,244	Name	0	0	0	0	0	0	0	exp177
1,293	Plz	0	0	0	0	0	0	0	exp178
1,293	Plz	0	0	0	0	0	0	0	exp179

Tabelle 20: Blocking mit der ChromaDB als Vectorstore, Preprocesseing mit textacy

5.2.4 Comparison mit LLMs

Genau wie bei den vorherigen Schritten bleibt der Einsatz der Classification gleich. Neben der Auswahl der Preprocess Möglichkeiten, kommt jetzt die Auswahl der unterschiedlichen Blocking-Verfahren hinzu. Tab. 21 und Tab. 22 zeigen die Comparison-Ergebnisse mit den unterschiedlichen Preprocess Verfahren, dabei wurde das Blocking auf PLZ als Schlüsselvariable aufgrund der langen Comparison-Dauer übersprungen. (Blocking auf die PLZ ergibt 2,506,133 Paare, das Comparing von 258,566 Paaren dauert mit SpaCy 90 Minuten)

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
258,566	Adresse	13,034	453	12,581	772	0.37	0.03	0.06	exp180
258,566	Adresse	4,041	450	3,591	775	0.37	0.11	0.17	exp181
827	Name	437	379	58	846	0.31	0.87	0.46	exp182
827	Name	430	373	57	852	0.30	0.87	0.45	exp183

Tabelle 21: Comparison mit dem Record Linkage Toolkit für Preprocessing und Blocking

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
1,200	Adresse	57	1	56	1,224	0.00	0.02	0.00	exp186
1,200	Adresse	35	1	34	1,224	0.00	0.03	0.00	exp187
702	Name	248	209	39	1,016	0.17	0.84	0.28	exp188
702	Name	243	205	38	1,020	0.17	0.84	0.28	exp189

Tabelle 22: Comparison mit textacy für Preprocessing und dem Record Linkage Toolkit für das Blocking

Die Tabellen 23 und 24 zeigen die Ergebnisse für Comparison mit dem ChromaDB-Vectorstore als Blocking verfahren mit unterschiedlichen Preprocessing Methoden.

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
3,649	Adresse	169	8	161	1,217	0.01	0.05	0.01	exp192
3,649	Adresse	601	8	593	1,217	0.01	0.01	0.01	exp193
2,142	Name	382	354	28	871	0.29	0.93	0.44	exp194
2,142	Name	652	594	58	631	0.48	0.91	0.63	exp195
1,293	Plz	0	0	0	0	0	0	0	exp196
1,293	Plz	0	0	0	0	0	0	0	exp197

Tabelle 23: Comparison mit dem Record Linkage Toolkit für Preprocessing und ChromaDB für das Blocking

Blocking Paare	Blocking	Matches	TP	FP	FN	Recall	Precision	F1	EXP
3,722	Adresse	0	0	0	0	0	0	0	exp198
3,722	Adresse	554	10	544	1,215	0.01	0.02	0.01	exp199
2,244	Name	254	229	25	996	0.19	0.90	0.31	exp200
2,244	Name	657	589	68	636	0.48	0.90	0.63	exp201
1,293	Plz	0	0	0	0	0	0	0	exp202
1,293	Plz	0	0	0	0	0	0	0	exp203

Tabelle 24: Comparison mit textacy für Preprocessing und ChromaDB für das Blocking

5.2.5 Einsatz von LinkTransformer

Der Vorteil von LinkTransformern, im Vergleich zu dem Record Linkage Toolkit, ist die Einfachheit der Durchführung des Record-Linkage Prozesses. Deshalb wurde zuerst untersucht, wie sich die ausgewählten Modelle aus Kapitel 3.4 ohne ein Preprocessing verhalten (Tab. 25).

Modell	Recall	Precision	F1	EXP
sentence-transformers/all-MiniLM-L6-v2	0.811	0.067	0.124	exp6
BAAI/bge-base-en-v1.5	0.841	0.070	0.129	exp14
BAAI/bge-m3	0.783	0.065	0.120	exp22
BAAI/bge-large-en-v1.5	0.841	0.070	0.129	exp30
intfloat/multilingual-e5-large	0.806	0.067	0.124	exp38
dell-research-harvard/lt-wikidata-comp-multi	0.630	0.053	0.098	exp46
dell-research-harvard/lt-wikidata-comp-de	0.520	0.043	0.079	exp54
dell-research-harvard/lt-wikidata-comp-en	0.814	0.068	0.126	exp62

Tabelle 25: Untersuchte Modelle mit LinkTransformer, gematched auf Firmennamen

Daraufhin wurde mit der in Kapitel 5.1 eingeführten Formel ein Threshold ermittelt, bei dem die minimale Anzahl von False Positive Matches exisitiert (Tab. 26)

Modell	Recall	Precision	F1	EXP	Threshold
sentence-transformers/all-MiniLM-L6-v2	0.346	0.998	0.514	exp6	1
BAAI/bge-base-en-v1.5	0.337	0.998	0.504	exp14	1
BAAI/bge-m3	0.143	0.994	0.120	exp22	0.89
BAAI/bge-large-en-v1.5	0.396	0.998	0.567	exp30	1
intfloat/multilingual-e5-large	0.189	1.000	0.318	exp38	0.96
dell-research-harvard/lt-wikidata-comp-multi	0.056	1.000	0.106	exp46	0.91
dell-research-harvard/lt-wikidata-comp-de	0.064	0.963	0.120	exo54	0.95
dell-research-harvard/lt-wikidata-comp-en	0.318	1.000	0.483	exp62	1

Tabelle 26: Untersuchte Modelle mit LinkTransformer, gematched auf Firmennamen, Ergenisse eingeschränkt auf den Threshold

In beiden Fällen (Tab. 25 und Tab. 26) wurden die Ergebnisse für das Matching auf den

Firmennamen aufgezeigt. In Tab. 27 werden die Ergebnisse mit den jeweils besten Werten und derer Matching-Schlüsselvariablen aufgezeigt.

Modell	Recall	Precision	F1	TP	Matched on	EXP
sentence-transformers/all-MiniLM-L6-v2	0.348	0.998	0.516	426	Name	exp10
BAAI/bge-base-en-v1.5	0.342	0.998	0.509	419	Name	exp18
BAAI/bge-m3	0.275	1	0.431	337	Name + PLZ	exp28
BAAI/bge-large-en-v1.5	0.404	0.998	0.575	495	Name	exp34
intfloat/multilingual-e5-large	0.189	1	0.318	232	Name	exp38
dell-research-harvard/lt-wikidata-comp-multi	0.072	0.989	0.134	88	Name + PLZ	exp52
dell-research-harvard/lt-wikidata-comp-de	0.064	0.963	0.120	79	Name	exp54
dell-research-harvard/lt-wikidata-comp-en	0.321	1	0.486	393	Name	exp66

Tabelle 27: Untersuchte Modelle mit LinkTransformer, gematched auf die Variablen in der "Matched on" Spalte, Ergenisse eingeschränkt auf den Threshold

Aus den Tabellen wird deutlich, dass das Modell `BAAI/bge-large-en-v1.5` die besten Ergebnisse liefert, somit wird sich im weiteren Verlauf auf dieses Modell fokussiert. Die Ergebnisse für dieses Modell sind in Tab. 28 und Tab. 29 einzusehen.

Blocking Paare	Recall	Precision	F1	TP	Matched on	EXP
14,651	0.84	0.07	0.13	1,030	Name	exp30
14,651	0.80	0.07	0.12	970	Name, Adresse	exp31
14,651	0.86	0.07	0.13	1,052	Name, PLZ	exp32
14,651	0.84	0.07	0.13	1,032	Name, Adresse, PLZ	exp33
68,162	0.81	0.01	0.03	997	Name	exp34
68,162	0.79	0.01	0.02	974	Name, Adresse	exp35
68,162	0.82	0.01	0.03	1,008	Name, PLZ	exp36
68,162	0.82	0.01	0.03	1,002	Name, Adresse, PLZ	exp37

Tabelle 28: Matching-Ergebnisse mit LinkTransformer mit dem Modell `bge-large-en-v1.5` auf verschiedenen Schlüsselvariablen (Matched on)

Blocking Paare	Recall	Precision	F1	TP	Matched on	EXP
486	0.39	0.99	0.57	485	Name	exp30
155	0.13	1.0	0.22	155	Name, Adresse	exp31
161	0.13	1.0	0.23	161	Name, PLZ	exp32
193	0.16	1.0	0.27	193	Name, Adresse, PLZ	exp33
496	0.40	0.99	0.58	495	Name	exp34
177	0.14	1.0	0.25	177	Name, Adresse	exp35
158	0.13	1.0	0.23	158	Name, PLZ	exp36
220	0.18	1.0	0.30	220	Name, Adresse, PLZ	exp37

Tabelle 29: Matching-Ergebnisse mit LinkTransformer mit dem Modell `bge-large-en-v1.5` auf verschiedenen Schlüsselvariablen (Matched on), eingeschränkt auf den Threshold

Nach der Auswahl des Modells, wurde untersucht wie sich das Bearbeiten der Adressen auf die Performance ausschlägt (Tab. 30)

Blocking Paare	Matched on	TP Rohe Adressen	EXP	TP Bereinigte Adressen	EXP
14,651	Name	485	exp30	491	126
14,651	Name, Adresse	155	exp31	231	127
14,651	Name, PLZ	161	exp32	169	128
14,651	Name, Adresse, PLZ	193	exp33	239	129

Tabelle 30: Matching-Ergebnisse mit LinkTransformer mit dem Modell `bge-large-en-v1.5` auf verschiedenen Schlüsselvariablen (Matched on), bereinigte und unbereinigte Adresseinträge, eingeschränkt auf den Threshold

Um den maximalen F1-Score zu erhalten wurde nun untersucht, wie sich die Ergebnis-Dataframes der LinkTransformer auf das Mergen mit sich selber reagierten. Dabei ist die Idee, dass zwei Ergebnis-Frames mit der Pandas.DataFrame.merge Methode vereint werden und der höhere Scores übernommen wird. Dies hat zur Folge, dass ein neues Dataframe entsteht, welches die Fehlenden Matches eines der vorherigen Dataframes auffüllt (Abb. 8). Das Mergen verläuft rechtsseitig (`how="right"`).

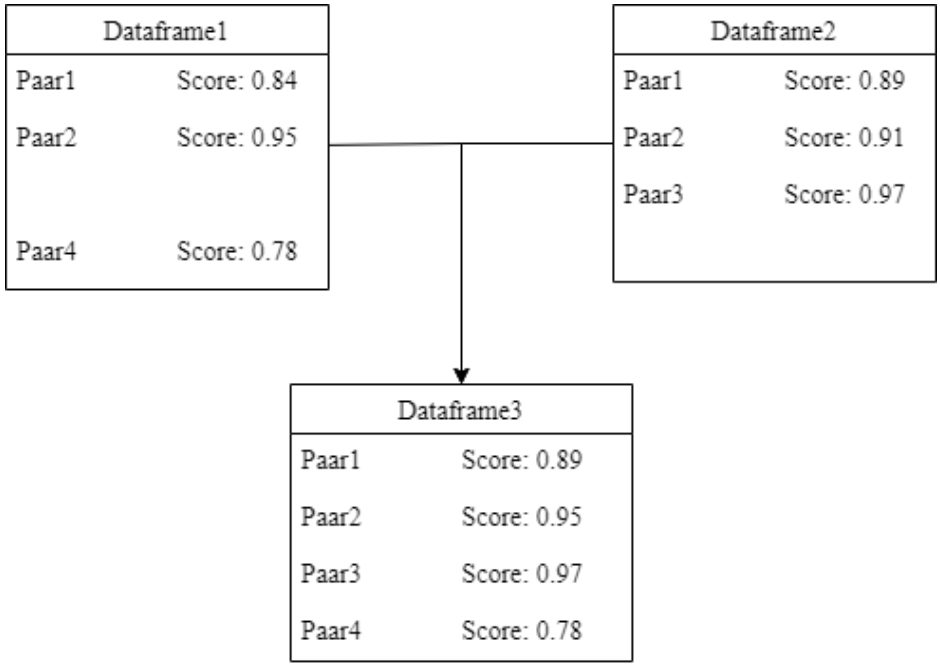

Abbildung 8: Mergen der Ergebnis-Dataframes

Das Ergebnis des Merge-Vorgangs ist in Tab.31 dargestellt, dabei gibt "Merge order" die Reihenfolge der Dataframes beim Benutzen der Merge-Methode an, zum Beispiel wurde das Ergebnis von exp133 durch den Vorgang

```
plz_address_frame = plz_frame.merge(address_frame, how="right")
plz_address_name_frame = plz_address_frame.merge(name_frame, how="right")
```

gebildet. Die Ergebnisse nach dem der Threshold ermittelt wurde sind in Tab. 32 dargestellt.

Blocking Paare	Merge order	TP	EXP
14,651	PLZ, Adresse, Name	1,001	exp133
14,651	Name, PLZ, Adresse	1,051	exp134
14,651	PLZ, Name, Adresse	1,053	exp135

Tabelle 31: Gemergte Ergebnis-Frames der LinkTransformer

Blocking Paare	TP	Recall	Precision	F1	EXP
537	536	0.44	0.99	0.61	exp133
498	497	0.41	0.99	0.58	exp134
349	349	0.28	1.0	0.44	exp135

Tabelle 32: Zusammengeführte Ergebnis-Frames der LinkTransformer, eingeschränkt auf den Threshold

5.3 Auswertung und Diskussion

Aus Tab. 18 lässt sich entnehmen, dass das Preprocessing mit textacy ungenauer als mit der `.clean()`-Methode ist. Dies ist auf die ungründliche Anwendung der preprocessing-Methoden (Abb. 13) zurückzuführen, welche in der Implementierungszeit nicht aufgefallen sind. Dies ist auch in Tab. 20 bemerkbar, bei der die mit textacy bereinigten Einträge fast keine Ergebnisse liefern. Das Blocking mit einem Vectorstore (Tab. 19) liefert im Vergleich zum herkömmlichen Blocking bessere Ergebnisse, wenn diese auf die Firmennamen geblockt werden (exp170, exp171). Dies liegt daran, dass die Datenquelle Bureu van Dijk eine hohe Anzahl an Firmeneinträgen ohne Adressen besitzt und der Vectorstore den nächstbesten Eintrag als Paar nimmt. Mit einem kleineren Threshold für den Kosinusabstand könnte die Precision zu Kosten des Recalls erhöht werden. Das Comparison mit SpaCy liefert in beiden Fällen vom Blocking mit dem Record Linkage Toolkit schlechtere Ergebnisse (Tab. 21 und Tab. 22) als das reine Anwenden der herkömmlichen Record-Linkage Methoden, wohingegen es die besten Ergebnisse liefert, wenn das Blocking auf einem Vectorstore verläuft (Tab. 23, exp195 und Tab. 24, exp201). Der Einsatz von LinkTransformer hat gezeigt, dass LLMs im Stande sind, den Record-Linkage Prozess auf unbereinigte Dateneinträge durchzuführen und teilweise Bessere Ergebnisse liefern als das Record Linkage Toolkit, wenn man Paare mit einem Score von 1.0 als Matches auswählt (Tab. 26). Dies macht LinkTransformer zu einem Präferierten Kandidaten für das Record-Linkage, denn deren Einsatz lässt sich auf 4 Schritte runterbrechen:

1. Dateneinträge in Dataframes laden
2. Den Spaltennamen, auf dem das Matching erfolgen soll in beiden Dataframes gleich benennen
3. Dataframes in die LinkTransformer.merge()-Methode mit dem gewünschten Embedding-Modell einfügen
4. Die Paare aus dem Ergebnis-Frames auswählen, die einen Wert von 0.99 bis 1.0

besitzen

Durch ein geschicktes Mergen von Ergebnis-Frames, welche zuvor mit dem LinkTransformer auf unterschiedliche Schlüsselvariablen gematcht wurden, können bessere Ergebnisse für das Record-Linkage erzielt werden (Tab. 27). Diese Optimierung der RL-Ergebnisse wird im nächsten Unterkapitel 5.4 weiter untersucht.

5.4 Optimierung des LLM-Einsatzes

5.4.1 Optimierung von Preprocessing

In den Fällen vom Blocking mit Vectorstores und dem Comparison mit SpaCy werden die Ähnlichkeiten der Einträge anhand der Entfernung der Embeddings im Vektorraum verglichen. Um zu verstehen, welche Preprocessing Schritte für die Embeddings am wichtigsten sind, muss untersucht werden, welche Auswirkungen Zahlen, Symbole und Satzzeichen auf die Ähnlichkeit von Embeddings haben.

Kandidat 1	Kandidat 2	Score
2145	2144	0.83
2145	2145.0	0.89
2145	2145,0	0.89

Tabelle 33: Kosinusähnlichkeiten (Score) von unterschiedlichen Darstellungen der Postleitzahlen

Kandidat 1	Kandidat 2	Score
Georg-Bölts-Str.	GEORG-BOELTS-STR. 2 -8	0.69
Georg-Bölts-Str.	Georg Bölts Str.	0.96
Georg-Bölts-Str.	Georg-Boelts-Str.	0.79
Georg-Bölts-Str.	Georg Boelts Str.	0.72
Georg Bölts Str,	Georg Boelts Str,	0.75

Tabelle 34: Kosinusähnlichkeiten von Adressen mit Umlauten

Kandidat 1	Kandidat 2	Score
Bremer Heerstr.	Bremer Heerstr.	1.0
Bremer Heerstr.	BREMER HEERSTR. 280	0.77
Bremer Heerstr.	Bremer Heerstr	0.93
Bremer Heerstr.	bremer heerstr	0.93
Bremer Heerstr.	Bremer + Heerstr.	0.89
Bremer Heerstr.	BREMER HEERSTR.	1.0
Bremer Heerstr.	BREMER + HEERSTR.	0.89
Bremer Heerstr.	BremerHeerstr.	0.81
Bremer Heerstr.	BremerHeerstr	0.74

Tabelle 35: Kosinusähnlichkeiten von Adressen mit unterschiedlichen Schreibweisen

Kandidat 1	Kandidat 2	Score
Bloherfelder Str.	BLOHERFELDER STR. 216 A	0.75
Bloherfelder Str.	BLOHERFELDER STR. A	0.93
Bloherfelder Str.	BLOHERFELDER STR. A	0.93
Bloherfelder Str.	Bloherfelder Straße	0.88
Bloherfelder Str.	Bloherfelder Straße A	0.80
Bloherfelder Str.	BLOHERFELDER STR, A	0.95
Bloherfelder Str.	BLOHERFELDER STR,	0.98
Bloherfelder Str	Bloherfelder Straße	0.86
Bloherfelder Str	Bloherfelder Straße A	0.74
Bloherfelder Str	Bloherfelder Str,	0.94
Bloherfelder Str,	Bloherfelder Straße	0.89
Bloherfelder Str,	Bloherfelder Straße A	0.82
Bloherfelder Str,	Bloherfelder Straße,	0.94
Bloherfelder Str,	Bloherfelder Straße A,	0.89

Tabelle 36: Kosinusähnlichkeiten von Adressen mit Adresszusätzen

Aus den Tabellen 34, 35 und 36 liest sich ab, dass folgende Schritte gemacht werden müssen, um ein effizienteres Vergleichen der Embeddings zu erschaffen:

- Zahlen aus den Einträgen entfernen
- Sonderzeichen (so wie `+,-,#` etc.) entfernen
- Umlaute umschreiben

- Adresszusätze entfernen
- Punkte durch Kommas ersetzen
- Kommas an das Ende des Textes setzen

Dadurch, dass die Ähnlichkeit von Embeddings mit Zahlen bei kleinster Abweichung schwankt (Tab. 33), werden keine Matches mit Postleitzahlen untersucht, da dessen ergebnisse unzuverlässig sind. Die Embeddings und Kosinusähnlichen wurden mit dem Modell `BAAI/bge-large-en-v1.5` berechnet. Für das Preprocessing wurde eine eigene Funktion erstellt (Abb. 16), die die aufgezählten Schritte ausführt.

5.4.2 Ergebnis und Auswertung

Für das Comparison wird sowohl SpaCy all auch `BAAI/bge-large-en-v1.5` genommen, alle anderen Parameter, wie der Threshold für die Kosinusdistanz von dem Blocking der Vectorstore und die Kosinusähnlichkeit bei dem Comparison, wurden gleichgelassen um die selben Bedingungen zu schaffen.

Blocking Paare	Recall	Precision	F1	TP	Matched on	EXP
4,050	0.01	0.33	0.02	11	Name, Adresse	exp205
4,050	0.01	0.00	0.00	11	Name, Adresse, PLZ	exp206
2,337	0.35	0.91	0.51	434	Name, Adresse	exp207
2,337	0.35	0.90	0.51	434	Name, Adresse, PLZ	exp208

Tabelle 37: Comparison mit dem Modell `BAAI/bge-large-en-v1.5`

Blocking Paare	Recall	Precision	F1	TP	Matched on	EXP
4,050	0.01	0.01	0.01	10	Name, Adresse	exp209
4,050	0.01	0.04	0.01	10	Name, Adresse, PLZ	exp210
2,337	0.59	0.43	0.50	724	Name, Adresse	exp211
2,337	0.48	0.90	0.63	588	Name, Adresse, PLZ	exp212

Tabelle 38: Comparison mit dem SpaCy Modell `en_core_web_lg`

Ein gründliches Bereinigen der Einträge hat nicht nur für höhere F1-Scores bei dem Einsatz von LLMs geführt (Tab. 37 und Tab. 38), sondern auch bei dem Einsatz vom Record Linkage Toolkit (Tab. 39). LinkTransformer bietet ähnliche Ergebnisse wie das Blocking mit Vectorstores (Tab 41 und Tab. 38) mit anschließendem LLM-Comparison, was den Aufbau der LinkTransformer-Pipeline wiederspiegelt. Ebenfalls gibt es durch die Erhöhung des Thresholds bei der Evaluation von linkTransformern keinen Verlust der

Blocking Paare	Recall	Precision	F1	TP	Matched on	EXP
1,266	0.00	0.50	0.00	3	Name, Adresse	exp213
1,266	0.00	0.60	0.0	3	Name, Adresse, PLZ	exp214
817	0.38	0.91	0.54	466	Name, Adresse	exp215
817	0.38	0.91	0.53	463	Name, Adresse, PLZ	exp216

Tabelle 39: Matching mit dem Record Linkage Toolkit, geblockt auf Adressen (exp213 und exp214) und Namen (exp215 und exp216) als Schlüsselvaraiblen

Blocking Paare	TP	Matched on	EXP
14,651	559	Name	exp217
14,651	351	Name, Adresse	exp218
68,126	565	Name	exp219
68,126	347	Name, Adresse	exp220

Tabelle 40: Matching mit LinkTransformer

Blocking Paare	Recall	Precision	F1	TP	Matched on	EXP
560	0.46	0.99	0.63	559	Name	exp217
351	0.29	1.0	0.45	351	Name, Adresse	exp218
566	0.46	0.99	0.63	565	Name	exp219
347	0.28	1.0	0.44	347	Name, Adresse	exp220

Tabelle 41: Matching mit LinkTransformer, Ergebnisse mit Threshold = 1.0

True Positive Matches (Tab. 40 und 41). Außerdem lässt sich ein niedriger F1-Score bei dem Matching auf Name und Adresse mit jedem Matchingverfahren festellen, was auf die fehlenden Adress-Einträge aus der Bureau van Dijk Datenbank zurückzuführen ist.

Um eine Parallelität zu dem LinkTransformer-Matching zu schaffen, wurde auch das Matching auf nur Firmennamen als Variable durchgeführt. Da der ECMClassifier bei dem Comparison auf eine Variable keine Ergebnisse liefert, wurde ein eigenes Classification-System entwickelt, welches nur Einträge über einem bestimmten Similarity-Threshold als Matches ansieht. Für das Blocking wurden die Paare genommen, welche mit ChromaDB auf den Firmennamen geblockt wurden.

Somit erschließt sich, dass das Matching mit dem `bge-large-en-v1.5` Modell für Comparison den besten F1-Score liefert, wenn das Matching nur auf Firmennamen mit einem Threshold von 0.99 erfolgt (Tab. 43), wobei das herkömmliche Vergleichen von Strings mit `jarowinkler` und dem Record Linkage Toolkit ähnliche Ergebnisse liefert (Tab. 44).

Blocking Paare	Matches	Recall	Precision	F1	TP	Threshold	EXP
2,337	1,683	0.59	0.43	0.50	724	0.80	exp221
2,337	1,231	0.56	0.56	0.56	688	0.99	exp222
2,337	886	0.52	0.72	0.60	637	1.0	exp223

Tabelle 42: Matching mit dem SpaCy Modell `en_core_web_lg` auf eine Schlüsselvariable

Blocking Paare	Matches	Recall	Precision	F1	TP	Threshold	EXP
2,337	2,155	0.61	0.34	0.44	743	0.80	exp224
2,337	692	0.52	0.92	0.67	640	0.99	exp225
2,337	621	0.47	0.93	0.63	578	1.0	exp226

Tabelle 43: Matching mit `bge-large-en-v1.5` auf eine Schlüsselvariable

Blocking Paare	Matches	Recall	Precision	F1	TP	Threshold	EXP
2,337	1,762	0.61	0.42	0.50	743	0.80	exp227
2,337	653	0.49	0.93	0.65	606	0.99	exp228
2,337	616	0.47	0.94	0.63	579	1.0	exp229

Tabelle 44: Matching mit dem Record Linkage Toolkit auf eine Schlüsselvariable

Die finale Implementierung ,welche die besten Record-Linkage-Ergebnisse aufweist, wird in Abb. 9 dargestellt. Dabei ist die Umsetzung mit herkömmlichen Methoden/Methoden ohne LLMs blau dargestellt und die Schritte, bei denen LLMs zum Einsatz kommen grün.

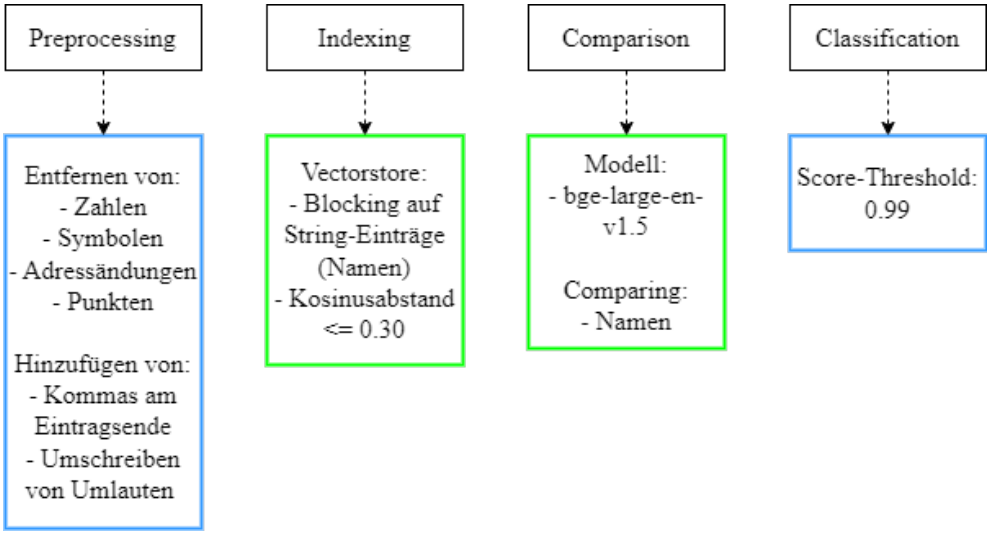

Abbildung 9: Finale Implementierung für die besten Ergebnisse

Die Hauptforschungsfrage lässt sich folgend beantworten:

Teilfrage 1: Es können sentence-transformer-Modelle eingesetzt werden, welche für den Task sentence similarity trainiert wurden. Wenn man statt den Huggingface-Modellen SpaCy benutzen möchte, dann können alle von SpaCy zur Verfügung stehenden Modelle benutzt werden.

Teilfrage 2: Der Einsatz in den Schritten Indexing und Comparison erweist sich als der optimalste Ort für den Einsatz von LLMs im Record-Linkage.

6 Fazit und Ausblick

6.1 Limitierungen dieser Arbeit

Die Auswahl von LLMs wurde nur auf die Open-Source Modelle beschränkt, wobei sich auf die Ergebnisse aus dem MTEB-Leaderboard (Muennighoff et al., 2022) verlassen wurde. Es gibt Modelle, welche höhere Werte erzielen als `bge-large-en-v1.5`, diese erfordern einen API-Zugriff (z.B. `voyage-large-2-instruct` und `google-gecko.text-embedding-preview-0409`) und wurden deshalb außer acht gelassen. Es wurde nur ChromaDB als Vectorstore untersucht und es wurden keine Untersuchungen bezüglich einem NearestNeighbourhood-Indexing gemacht. Ebenfalls wurde nicht untersucht, ob andere Classifier (NaiveBayesClassifier, SVMClassifier) besser für das Klassifizieren von Matches auf einer oder zwei Schlüsselvariablen gewesen wären. Die Datenquellen wurden nicht verändert und es gibt keine Ergebnisse für das Untersuchen anderer Datenquellen mit anderer Darstellung von Namen und Adressen. Ein Versuch andere Datenquellen mit deren Ground Truth zu Untersuchen wurde zwar gemacht, jedoch war die Struktur der Datenbank von der Ground Truth und der Datenquellen so unterschiedlich zu den bereits vorhandenen Datenquellen, dass ein Ändern der gesamten Code-Base nötig wäre, um diese vernünftig auszuwerten.

6.2 Fazit und Ausblick

In dieser Arbeit wurde der Einsatz von LLMs auf verschiedene Schritte des Record-Linkage Prozesses untersucht und mit den Ergebnissen von herkömmlichen Methoden, welche mit dem Record Linkage Toolkit implementiert wurden, verglichen.

LLMs liefern ähnliche, wenn nicht sogar bessere, Ergebnisse wie herkömmliche Verfahren, wenn ein ausreichendes Preprocessing durchgeführt wurde. Python bietet bereits Möglichkeiten für das Bearbeiten von Strings, welche, im Vergleich zu LLMs, leichter Umzusetzen sind, wodurch der Einsatz von LLMs eher behinderlich wirkt. LLMs bieten die besten Ergebnisse, wenn diese beim Blocking mit Vectorstores und dem Comparison von Einträgen auf deren Kosinusähnlichkeit angewandt werden. LinkTransformer bieten eine einfache Alternative, da sie kaum Aufwand für das erreichen von Ergebnissen, mit ähnlichem F1-Score wie die herkömmlichen Methoden, benötigen. Untersuchungen von unterschiedlichen Vectorstores wie zum Beispiel FAISS, ElasticSearch oder AnalyticDB, welche mit LangChain implementiert werden können, könnte Aufschlüsse bezüglich der besseren Variante zu ChromaDB liefern. Genauso wie ein anderes Embedding-Modell, welches in Kap. 6.1 angesprochen wurde, könnte ebenfalls bessere Ergebnisse zu den benutzten Mo-

dellen liefern. Der Einsatz von LLMs in Record-Linkage könnte weiter optimiert werden, indem unterschiedliche Comparison-Optionen, wie zum Beispiel das Mischen von LLMs mit herkömmlichen Comparison Methoden, untersucht werden. Ebenfalls wäre die Untersuchung von fine tuning eines Modells mit Firmendaten naheliegend.

Anhang

```python
import linktransformer as lt
# Load data frame
df1 = pd.create_frame("Datenquelle 1")
df2 = pd.create_frame("Datenquelle 2")
# Define language model
model = "sentence-transformers/all-MiniLM-L6-v2"
# merge frames with language model
df_matched = lt.merge(df1, df2,
                      merge_type ='1:m',
                      on=["Varname"],
                      model=model)
```

Abbildung 10: Implementierung der merge-Methode von LinkTransformer

```
import spacy

nlp = spacy.load("en_core_web_sm")
tokens = nlp("dog")
print(tokens.vector)
✓ 0.3s
[-1.6806675  -1.2663747  -0.71255565  0.22143888  0.28581634  0.23924345
  1.2992647   1.0683641  -0.1666507  -0.593021    0.20207635 -0.71124184
 -0.5710875  -0.2685267  -0.5052826   0.60505986 -1.5851773  -1.6874862
  0.7026561   0.60366225  0.3043416   1.3963956  -0.056483   -0.6299367
  0.09717859  0.5463655   0.36506647  0.73901325 -0.16906115  0.35410628
  0.33770823 -0.7352046   1.6755302   0.48371333  0.0184648  -0.92315257
  0.6245377   0.11393103  0.8193037  -0.01115507 -0.49064368 -0.30198318
  0.43095675 -0.05127436 -0.11000359 -0.64060974 -1.4619632   0.85834503
 -0.4855454   0.01614086 -0.10124743  1.2471893   0.7936216  -0.49573362
  1.0994403  -0.22723481  1.289906   -0.8966851  -0.07580797 -0.6877714
 -0.9954758  -0.70957065 -0.3484952  -0.35015944  0.6045856   0.21346438
 -0.22741812  0.12088367  0.8521288   0.11694434  0.42555034  0.8777968
  1.9081106  -0.62818205  0.2991566  -0.26263964 -0.4627701  -0.11653326
  0.6030469  -1.1594303  -0.3226232  -0.1124312  -1.5564214  -0.44001883
  0.8246197   1.0218511   0.31111258 -0.46998724 -2.129951    1.1728595
 -1.2842656  -1.1179041   1.596463    0.51193094  0.22032768  1.6200272 ]
```

Abbildung 11: Wort-Vektor generiert mit dem Modell `en_core_web_sm`.

```python
import spacy

nlp = spacy.load("en_core_web_sm")

def clean_address(text):
    doc = nlp(text)
    cleaned_tokens = [token.text for token in doc if
                      not token.is_digit
                      and not token.is_punct]
    return ' '.join(cleaned_tokens).lower()
```

Abbildung 12: Preprocessing mit SpaCy

```python
from textacy import preprocessing

# define a normalization function
def normalize(text):
    text = preprocessing.remove.punctuation(text)
    text = preprocessing.replace.numbers(text, repl='')
    return text.lower()

df2[COMPANY_ADDRESSES] = df2[COMPANY_ADDRESSES].map(normalize)
```

Abbildung 13: Preprocessing mit textacy

```python
import pandas as pd
import spacy

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

# Sample DataFrame with address entries
data = {'Address': ['123 Main St, Apt 45', '456 Elm St, #201', '789 Oak Avenue']}
df = pd.DataFrame(data)

def clean_address(text):
    doc = nlp(text)
    cleaned_tokens = [token.text for token in doc if not token.is_digit]
    return ' '.join(cleaned_tokens)

# Apply the clean_address function to the DataFrame
df['Cleaned_Address'] = df['Address'].apply(clean_address)
```

Abbildung 14: Beispielhafte Anwendung der nlp-Pipeline auf ein Dataframe mit dem Modell `en_core_web_sm`

```python
import recordlinkage
from recordlinkage.preprocessing import clean

# Preprocessing
cleaned_frame1_names = clean(df1[COMPANY_NAME], lowercase=True, replace_by_none=r"[^ \-\_A-Za-z]+")
cleaned_frame2_names = clean(df2[COMPANY_NAME], lowercase=True, replace_by_none=r"[^ \-\_A-Za-z]+")
cleaned_frame1_address = clean(df1[COMPANY_ADDRESSES], lowercase=True, replace_by_none=r"[^ \-\_A-Za-z]+")
cleaned_frame2_address = clean(df2[COMPANY_ADDRESSES], lowercase=True, replace_by_none=r"[^ \-\_A-Za-z]+")

df1[COMPANY_NAME] = cleaned_frame1_names
df2[COMPANY_NAME] = cleaned_frame2_names
df1[COMPANY_ADDRESSES] = cleaned_frame1_address
df2[COMPANY_ADDRESSES] = cleaned_frame2_address

# Indexation step
indexer = recordlinkage.Index()
indexer.block(COMPANY_ADDRESSES)
candidate_pairs = indexer.index(df1, df2)

# Comparison step
compare = recordlinkage.Compare()
compare.string(COMPANY_NAME, COMPANY_NAME,method='jarowinkler', threshold = 0.8, label=COMPANY_NAME)
compare.string(COMPANY_ADDRESSES, COMPANY_ADDRESSES, method='jarowinkler', threshold=0.8, label=COMPANY_ADDRESSES)

comparison_vectors = compare.compute(candidate_pairs, df1, df2)

# Classification step
classifier = recordlinkage.ECMClassifier()
classifier_results = classifier.fit_predict(comparison_vectors)
```

Abbildung 15: Code-Implementierung des Python Record Linkage Toolkits

```python
import re

def translate(text):
    # remove numbers
    pattern = r'[0-9]'
    text = re.sub(pattern,'', text)
    # remove special characters
    pattern2 = r'[- + () / "]'
    text = re.sub(pattern2,' ', text)
    # remove address extensions
    try:
        if text[-2] == ' ':
            text = text[:-1]
    except:
        text = text
    # remove dots
    text = text.replace(".", "")
    # add commas to end of entry
    text = text + ","
    # replace umlauts
    special_char_map = {ord('ä'):'ae', ord('ü'):'ue', ord('ö'):'oe', ord('ß'):'ss',
                        ord('&'):'und'}
    return text.translate(special_char_map)

df1[COMPANY_ADDRESSES] = df1[COMPANY_ADDRESSES]. map(translate)
df2[COMPANY_ADDRESSES] = df2[COMPANY_ADDRESSES].map(translate)
df1[COMPANY_NAME] = df1[COMPANY_NAME]. map(translate)
df2[COMPANY_NAME] = df2[COMPANY_NAME].map(translate)
```

Abbildung 16: Eigene Funktion für das Preprocessing

```python
docs = generate_documents(frame1)
db = Chroma.from_documents(docs, embedding_function, persist_directory=output_path)

answertuple = []
for entry in tqdm(range(len(frame2))):
    query = frame2[blocking_variable].iloc[entry]
    embedding_vector = embedding_function.embed_query(query)
    answer = db.similarity_search_by_vector_with_relevance_scores(embedding_vector, k=1)
    if (answer[0][1] * 100) <= 30:
        try:
            answertuple.append({"CompanyName_x": frame1[COMPANY_NAME].loc[df1["id"] == int(answer[0][0].metadata['id'])].iloc[0],
                    "CompanyAddresses_x": frame1[COMPANY_ADDRESSES].loc[df1["id"] == int(answer[0][0].metadata['id'])].iloc[0],
                    "CompanyPostalCode_x": frame1[COMPANY_POSTAL_CODE].loc[df1["id"] == int(answer[0][0].metadata['id'])].iloc[0],
                    "id_x":int(answer[0][0].metadata['id']),
                    "CompanyName_y": frame2[COMPANY_NAME].iloc[entry],
                    "CompanyAddresses_y": frame2[COMPANY_ADDRESSES].iloc[entry],
                    "CompanyPostalCode_y": frame2[COMPANY_POSTAL_CODE].iloc[entry],
                    "id_y":frame2["id"].iloc[entry],
                    "blocking_score": answer[0][1]})
        except:
            continue

return pd.DataFrame(answertuple)
```

Abbildung 17: Code für das Blocking mit der ChromaDB auf eine Blocking-Variable

Literatur

Almeida, F. & Xexéo, G. (2023). *Word embeddings: A survey.*

Arora, A. & Dell, M. (2023). *Linktransformer: A unified package for record linkage with transformer language models.*

Bailer-Jones, C. A. L., Gupta, R. & Singh, H. P. (2001). *An introduction to artificial neural networks.*

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P. & Robinson, T. (2014). *One billion word benchmark for measuring progress in statistical language modeling.*

Christen, P. (2012a). Classification. In *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection* (S. 129–162). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_6` doi: 10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_6

Christen, P. (2012b). The data matching process. In *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection* (S. 23–35). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_2` doi: 10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_2

Christen, P. (2012c). Data pre-processing. In *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection* (S. 39–67). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_3` doi: 10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_3

Christen, P. (2012d). Evaluation of matching quality and complexity. In *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection* (S. 163–184). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_7` doi: 10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_7

Christen, P. (2012e). Field and record comparison. In *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection* (S. 101–127). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_5` doi: 10Γ"2E1007/978Γ"2D3Γ"2D642Γ"2D31164Γ"2D2_5

Christen, P. (2019, nov 1). Data Linkage: The Big Picture. *Harvard Data Science Review*,

1 (2). (https://hdsr.mitpress.mit.edu/pub/8fm8lo1e)

Christen, P., Churches, T. & Zhu, J. (2002). Probabilistic name and address cleaning and standardisation. In *Ausdm* (S. 99–108).

Christen, P. & Goiser, K. (2007). Quality and complexity measures for data linkage and deduplication. In F. J. Guillet & H. J. Hamilton (Hrsg.), *Quality measures in data mining* (S. 127–151). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D540Γ"2D44918Γ"2D8_6` doi: 10Γ”2E1007/978Γ”2D3Γ”2D540Γ”2D44918Γ”2D8_6

De Bruin, J. (2023). *Python record linkage toolkit: A toolkit for record linkage and duplicate detection in python.* Zenodo. Zugriff auf `https://recordlinkageΓ"2Ereadthedocs Γ"2Eio/en/latest/indexΓ"2Ehtml` (Accessed: 02.06.2024) doi: 10Γ”2E5281/ zenodoΓ”2E8169000Γ”2E

DeWilde, B. (o. J.). *Vector stores.* Zugriff auf `https://jsΓ"2ElangchainΓ"2Ecom/ v0Γ"2E1/docs/modules/data_connection/vectorstores/#:~:text=A% 20vector%20store%20takes%20careΓ"2Cperforming%20vector%20search% 20for%20youΓ"2E` (Accessed: 01.06.2024)

DeWilde, B. (2023). *textacy: Nlp, before and after spacy.* Zugriff auf `https://textacyΓ"2EreadthedocsΓ"2Eio/en/latest/#textacyΓ"2Dnlp Γ"2DbeforeΓ"2DandΓ"2DafterΓ"2Dspacy`

Ding, T., Chen, T., Zhu, H., Jiang, J., Zhong, Y., Zhou, J., ... Liang, L. (2023). *The efficiency spectrum of large language models: An algorithmic survey.*

Domingo-Ferrer, J. (2018). Record linkage. In L. Liu & M. T. Özsu (Hrsg.), *Encyclopedia of database systems* (S. 3128–3129). New York, NY: Springer New York. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D1Γ"2D4614Γ"2D8265Γ"2D9_1504` doi: 10Γ”2E1007/978Γ”2D1Γ”2D4614Γ”2D8265Γ”2D9_1504

Domingo-Ferrer, J. & Torra, V. (2002). Validating distance-based record linkage with probabilistic record linkage. In M. T. Escrig, F. Toledo & E. Golobardes (Hrsg.), *Topics in artificial intelligence* (S. 207–215). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dusetzina SB, M. A. e. a., Tyree S. (2014). An overview of record linkage methods. *Linking Data for Health Services Research: A Framework and Instructional Guide [Internet]..* Zugriff auf `Availablefrom:`

Elastic. (o. J.). *Was bedeutet vektorsuche?* Zugriff auf `https://wwwΓ"2EelasticΓ"2Eco/ de/whatΓ"2Dis/vectorΓ"2Dsearch` (Accessed: 01.06.2024)

Grigoleit, S. (2019). *Natural language processing.* Zugriff auf `https://publica Γ"2EfraunhoferΓ"2Ede/handle/publica/260578` doi: 10Γ”2E24406/publica

Γ"2DfhgΓ"2D260578

Gruenheid, A. (2018). Record linkage. In S. Sakr & A. Zomaya (Hrsg.), *Encyclopedia of big data technologies* (S. 1–5). Cham: Springer International Publishing. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D319Γ"2D63962Γ"2D8_19Γ"2D1` doi: 10Γ"2E1007/978Γ"2D3Γ"2D319Γ"2D63962Γ"2D8_19Γ"2D1

Honnibal, M. & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* Zugriff auf `https://spacyΓ"2Eio/usage/spacyΓ"2D101` (To appear)

HuggingFace. (o. J.). *How do transformers work?* Zugriff auf `https://huggingfaceΓ"2Eco/learn/nlpΓ"2Dcourse/chapter1/4` (Accessed: 01.06.2024)

Jing, K. & Xu, J. (2019). *A survey on neural network language models.*

Johns, R. (2023). Openai might be leading the race, but what are the best llms?

Kruse, F., Schröer, C., Awick, J.-P., Reinkensmeier, J. & Gómez, J. M. (2022). Towards an automated record linkage process for datasource-independent company matching. In *2022 3rd international conference on next generation computing applications (nextcomp)* (S. 1-7). doi: 10Γ"2E1109/NextComp55567Γ"2E2022Γ"2E9932178

LangChain. (2024). *Introduction.* Zugriff am Accessed: 05.06.2024 auf `https://pythonΓ"2ElangchainΓ"2Ecom/v0Γ"2E2/docs/introduction/`

Li, Y., Li, J., Suhara, Y., Doan, A. & Tan, W. C. (2020). Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, *14*, 50 - 60. Zugriff auf `https://apiΓ"2EsemanticscholarΓ"2Eorg/CorpusID:214743579`

Michelson, M. & Knoblock, C. A. (2006). Learning blocking schemes for record linkage. In *Aaai* (Bd. 6, S. 440–445).

Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., ... Tan, S. (2021). *Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp.*

Moon, T. (1996). The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, *13* (6), 47-60. doi: 10Γ"2E1109/79Γ"2E543975

Muennighoff, N., Tazi, N., Magne, L. & Reimers, N. (2022). Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*. Zugriff auf `https://arxivΓ"2Eorg/abs/2210Γ"2E07316` doi: 10Γ"2E48550/ARXIVΓ"2E2210Γ"2E07316

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., ... Mian, A. (2023). *A comprehensive overview of large language models.*

Peeters, R. & Bizer, C. (2023). *Entity matching using large language models.*

Qin, C., Zhang, A., Zhang, Z., Chen, J., Yasunaga, M. & Yang, D. (2023). *Is chatgpt a general-purpose natural language processing task solver?*

Rouse, M. (2024). *Zero-shot-, one-shot-, few-shot-lernen.* Zugriff auf `https://wwwΓ"2EtechopediaΓ"2Ecom/de/definition/zeroΓ"2DshotΓ"2DoneΓ"2Dshot Γ"2DfewΓ"2DshotΓ"2Dlernen`

Sennrich, R., Haddow, B. & Birch, A. (2016). *Neural machine translation of rare words with subword units.*

Shlomo, N. (2019). Overview of data linkage methods for policy design and evaluation. In N. Crato & P. Paruolo (Hrsg.), *Data-driven policy impact evaluation: How access to microdata is transforming policy design* (S. 47–65). Cham: Springer International Publishing. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1007/978Γ"2D3Γ"2D319Γ"2D78461Γ"2D8_4` doi: 10Γ”2E1007/978Γ”2D3Γ”2D319Γ”2D78461Γ”2D8_4

Sun, K., Qi, P., Zhang, Y., Liu, L., Wang, W. Y. & Huang, Z. (2023). *Tokenization consistency matters for generative models on extractive nlp tasks.*

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... Scialom, T. (2023). *Llama 2: Open foundation and fine-tuned chat models.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2023). *Attention is all you need.*

Vykhovanets, V. S., Du, J. & Sakulin, S. A. (2020). An overview of phonetic encoding algorithms. Zugriff auf `https://doiΓ"2Eorg/10Γ"2E1134/S0005117920100082` doi: 10Γ”2E1134/S0005117920100082

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., ... Khashabi, D. (2022). *Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks.*

Wei, C., Wang, Y.-C., Wang, B. & Kuo, C. C. J. (2023). *An overview on language models: Recent developments and outlook.*

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., ... Schmidt, D. C. (2023). *A prompt pattern catalog to enhance prompt engineering with chatgpt.*

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... Wen, J.-R. (2023). *A survey of large language models.*