# Digitale Harmonie aus historischer Dissonanz

Extraktion, Ordnung und Analyse
unstrukturierter Archivdaten
des Männerchor Murg

Sven Burkhardt

0009-0001-4954-4426
17-056-912
15.08.2025

University of Basel
Digital Humanities Lab

University of Basel
Digital Humanities Lab
Switzerland

Diese Arbeit befasst sich mit dem Archiv des *Männerchor Murg* in den Jahren des Zweiten Weltkrieges. Hierfür wird eine automatisierte Pipeline auf Basis von LLMs und Pattern-matching vorgestellt, mit deren Hilfe Named Entities extrahiert und weiterverarbeitet werden. Der Hauptfokus der Weiterverarbeitung liegt auf der Ausgestaltung des Entity Matching, um erkannte Entitäten mit einer Groundtruth-Tabelle abzugleichen. Ziel ist es, dieses Archiv digital zugänglich und die beteiligten Personen sowie deren Netzwerke sowie die geographische Ausdehnung sichtbar zu machen.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Ziel und Relevanz der Arbeit

## 1.2 Formulierung der Forschungsfrage

## 1.3 Aufbau der Arbeit

## 1.4 Geografischer und historischer Kontext

Die vorliegende Arbeit stützt sich auf Unterlagen aus dem Archiv des „Männerchor Murg“ dessen Nachfolge im Jahr 2021 durch die „New Gospelsingers Murg“ angetreten wurde. Murg ist eine deutsche Gemeinde am Hochrhein, rund 30 km Luftlinie von Basel entfernt. Der Ort liegt am gleichnamigen Fluss Murg, der in den Rhein mündet. Beide Gewässer bildeten über Jahrhunderte hinweg den wirtschaftlichen Motor der Region: Die Wasserkraft der Murg begünstigte früh die Ansiedlung von Mühlen, Hammerwerken und Schmieden entlang des Bachlaufs, während der Rhein mit seiner Drahtseil-Fähre eine bedeutende Verkehrs- und Handelsverbindung bot, die bis zum Ersten Weltkrieg privat betrieben wurde.

Mit dem Ausbau der Landstrasse, der heutigen Bundesstrasse 34, sowie dem Anschluss an die Bahnstrecke Basel–Konstanz entwickelte sich Murg im 19. Jahrhundert von einer landwirtschaftlich geprägten Siedlung zu einer Gewerbe-, Handels- und Industriegemeinde. Die Wasserkraft wurde dabei zu einem entscheidenden Standortfaktor: Die Ansiedlung der Schweizer Textilfirma *Hüssy & Künzli AG* im Jahr 1853[1] trug wesentlich zum wirtschaftlichen Wachstum der Gemeinde bei. Zahlreiche Arbeitskräfte, vor allem aus der benachbarten Schweiz, machten Murg zu einem wichtigen Standort der regionalen Textilindustrie.

Die Gründung des *Männerchor Murg* im Jahr 1861 durch Schweizer Textilarbeiter belegt diesen engen Zusammenhang zwischen wirtschaftlicher Migration, Industrialisierung und lokalem Vereinswesen. Diese historische Verflechtung bildet eine zentrale Grundlage für die vorliegende Untersuchung.

[1] vgl. *Geschichte Gemeinde Murg* 2025.

## 2 Forschungsstand und Forschungslücke

Die vorliegende Arbeit knüpft hauptsächlich an zwei Vorarbeiten an, die in den Jahren 2019 und 2022 am Departement Geschichte der Universität Basel durchgeführt werden. In drei Digital History-Seminaren[2] mit Fokus auf Transkribus werden erste Teilbestände der „Männerchor Akten 1925–1944“ erschlossen und in einem Korpus von 137 Einzeldokumenten zusammengeführt.[3] Ein kleinerer Korpus von rund 50 Dokumenten wird mit Metadaten versehen. Erfasst werden unter anderem die genaue Position im Ordner auf Seitenebene, Kurztitel und Entstehungsdatum. Diese Metadaten bilden die Grundlage für eine erstmalige systematische Erschliessung.

Während in einem frühen Projektschritt vorrangig häufig genannte Personennamen („Carl Burger“, „Fritz Jung“) dokumentiert werden, richtet sich der Fokus im zweiten Schritt auf die Feldpost. Ziel ist es, über die Auswertung der Feldpostnummern Rückschlüsse auf beteiligte Militäreinheiten, deren Stationierungen und Verlagerungen während des Zweiten Weltkriegs zu ziehen.

Für diese Recherchen kommen einschlägige Fachliteratur zu den jeweiligen Fachgebieten zum Einsatz. Hier sind besonders die Bücher von Alex Buchner[4], Christian Hartmann[5], Werner Haupt[6], Christoph Rass[7], Georg Tessin[8] und Christian Zentner[9] zu nennen.

Darüberhinaus werden eigene Recherchen in den Beständen des *Bundesarchivs – Militärarchiv Freiburg*[10] durchgeführt. Ergänzende Recherchen stammen aus den Suchlisten des *Deutschen Roten Kreuzes (DRK)*[11]. Hinzu kommen philatelistische Übersichts-Websites[12], die bei der Entzifferung von Briefmarken und Stempeln helfen. Absoult essentiell für den Erfolg dieser Recherchen sind Citizen-Science-Foren[13]. Sie ergänzen und validieren eigene Forschung.

Parallel zur inhaltlichen Erschliessungen entsteht 2022 eine erste digitale Storymap mit

[2] vgl. Hodel 2019; Serif 2019; Serif 2022.
[3] vgl. Burkhardt 2022.
[4] vgl. Buchner 1989.
[5] vgl. Hartmann 2010.
[6] vgl. Haupt 1982.
[7] vgl. Rass und Rohrkamp 2009.
[8] vgl. Tessin 1977.
[9] vgl. Zentner 1983.
[10] Hollmann 2025.
[11] *DRK Suchdienst | Suche per Feldpostnummer* 2025.
[12] *Feldpost Number Database | GermanStamps.net* 2025.
[13] vor Allem werden verwendet: *Forum der Wehrmacht* (*Forum Geschichte der Wehrmacht* 2025) und das *Lexikon der Wehrmacht* (Altenburger 2023).

*ArcGIS*, die zentrale Ergebnisse des Projekts öffentlich zugänglich macht. Grundlage bildet die Sichtung, konservatorische Aufbereitung und Digitalisierung von zunächst rund 30 der etwa 800 Seiten Vereinsakten. Der Teilkorpus wird entheftet, gescannt und mit Metadaten wie Absender, Datum, Feldpostnummer und Einheit versehen. Da jedes Dokument einen anderen Verfasser aufweist, erfolgt die Transkription manuell. Eine automatische Handschriftenerkennung ist aufgrund der heterogenen Schriftbilder nicht praktikabel. Am Beispiel einzelner Sänger wie *Emil Durst* lässt sich durch die Rechercheergebnisse mithilfe der Feldpostnummern und ergänzender Kartenmaterialien der Aufenthaltsort bis auf Gebäude oder wenige Meter genau rekonstruieren. Diese Erkenntnisse werden mit historischen Karten, Luftbildern und Ortsrecherchen verknüpft und in einer interaktiven ArcGIS-Karte visualisiert, die Stationierungen, Märsche und Frontverschiebungen der Chormitglieder anschaulich darstellt.

Die in diesen Vorprojekten erarbeiteten Listen, Geodaten, Transkriptionen und Visualisierungen fliessen in die vorliegende Arbeit ein und bilden eine wesentliche Grundlage für die erweiterte, automatisierte Pipeline, die im Folgenden vorgestellt wird. Dazu gehören beispielsweise auch die Verbandsabzeichen, Taktische Zeichen[14] der jeweiligen Einheiten, die auch in die Groundtruth der vorliegenden Arbeit inkorporiert werden.

Abgesehen von diesen Vorarbeiten ist der Quellenkorpus wissenschaftlich unerschlossen. Mit dieser Arbeit liegt erstmals eine umfassendere wissenschaftliche Auswertung vor.

Mit der notwendigen manuellen Recherche in oben dargelegten Datenbankstrukturen wird zugleich sichtbar, wie sehr es an Brücken fehlt, um unterschiedliche Klassifikationen, fachspezifische Ordnungslogiken und semantische Webtechnologien nachhaltig miteinander zu verbinden. Ein verhältnismässig einfaches Webscraping nach Informationen zu diesem Korpus ist nahezu unmöglich. Ausgeführt werden diese Probleme beispielsweise bei Smiraglia und Scharnhorst (2021)[15], die anhand konkreter Fallstudien verdeutlichen, wie fragmentiert semantische Strukturen bislang entwickelt werden und welche Hürden bei der praktischen Verknüpfung heterogener Wissensorganisationen bestehen. Dabei benennen sie insbesondere die Herausforderungen bei der Übersetzung historisch gewachsener Klassifikationen in standardisierte semantische Formate, die Notwendigkeit dauerhafter technischer Wartung und die Abhängigkeit von nachhaltigen Infrastruktur-Partnern[16].

[14] vgl. Haupt 1982, S.64-66.
[15] vgl. Richard und Andrea 2022.
[16] vgl. ebd., Kap. 2 und 5.

Für eine Einordnung zu historischen Netzwerkanaylsen sei auf Gamper&Reschke[17] verwiesen. Der Sammelband *Knoten und Kanten III* verdeutlicht, dass die historische Netzwerkanalyse zwar von einem interdisziplinär etablierten Methodenkanon profitiert, jedoch nach wie vor vor erheblichen Herausforderungen steht. Dazu zählen die Fragmentierung historischer Quellen, der hohe manuelle Erfassungsaufwand und methodische Desiderate im Umgang mit zeitlichen und räumlichen Dimensionen. Erschwerende Faktoren einer systematische Erfassung relationaler Strukturen. Dennoch eröffnen netzwerkanalytische Verfahren – besonders im Zusammenspiel mit relationaler Soziologie und Figurationsansätzen – neue Perspektiven auf Macht, Abhängigkeiten und Akteurskonstellationen in historischen Gesellschaften.

# 3 Korpus

Aus dem Bestand des Ordners *"Männerchor Akten 1925–1944"* werden für diese Arbeit ausschliesslich Akten verwendet, die während des Zweiten Weltkriegs verfasst wurden. Der Analysezeitraum erstreckt sich dementsprchend zwischen dem 01. September 1939 und dem 8. Mai 1945, dem Tag der bedingungslosen Kapitulation Deutschlands.

Die zeitliche Eingrenzung ist notwendig, um die Funktionalität der im Folgenden beschriebenen Pipeline in einem klar definierten historischen Kontext demonstrieren zu können. Gleichzeitig führt TeXshop?diese Auswahl zu einer bewussten Reduzierung der potenziell erfassten Akteurinnen und Akteure, Orte und Organisationen. Diese Fokussierung ist insbesondere im Hinblick auf die Erstellung einer verlässlichen Groundtruth bedeutsam, die durch ergänzende Archivrecherchen mit historischen Metadaten angereichert wird.

Die Kombination aus einer präzise definierten Quellengrundlage und der digitalen Anreicherung dient dazu, das Potenzial der computergestützten Auswertung historischer Dokumente exemplarisch aufzuzeigen. Zugleich unterstreicht sie, dass die Qualität der Ergebnisse wesentlich von der sorgfältigen Eingrenzung des Korpus und der manuellen Validierung und Anreicherung abhängt.

[17] Gamper und Reschke 2015.

## 3.1 Quellen

### 3.1.1 Quellentradierung

In den Lagerräumen der New Gospel Singers Murg, dem Nachfolgeverein des Männerchors Murg, wird im Jahr 2018 mehrere je ca. 800 Seiten umfassende Ordner mit historischen Unterlagen gefunden. Für diese Arbeit wird ein Ordner mit der Aufschrift *„Männerchor Akten 1925–1944“* gewählt, da er neben dem Ordner *„Männerchor Akten 1946–1950“* den grössten Zeitraum abdeckt. Darüberhinaus bietet er das Potential, aufschlussreiche Einblicke in das Vereinsleben in der Zeit vor und während des Nationalsozialismus, insbesondere des Zweiten Weltkrieges, zu geben.
Der Ordner umfasst insgesamt 780 Seiten und deren Inhalt kann als „Protokoll“, „Brief“, „Postkarte“, „Rechnung“, „Regierungsdokument“, „Noten“, „Zeitungsartikel“, „Liste“, „Notizzettel“ oder „Offerte“ kategorisiert werden.

Die Unterlagen könnten bereits direkt nach ihrer Entstehung in die Ordner eingelegt worden sein. Einzelne Akten sind mit einem „Heftstreifen“, auch „Aktendulli“ genannt, zusammengefehtet. In der Plastikversion, wie er in diesen Akten vorliegt, wurde er bereits 1938 patentiert[18]. Wer die Akten so archiviert hat lässt sich nicht mehr sagen. Der sogenannte „Archival-Bias“ des Archivars, also die Grundeinstellung, weshalb etwas aufbewahrt oder vernichtet wurde, lässt sich damit nicht mehr feststellen.

[18] vgl. *Heftstreifen* 2023.

### 3.1.2 Quellenbeschrieb

Für diese Arbeit wurde ein Korpus selektiert, dessen Auswahl in Korpus näher beschrieben wird. Erfasst, benannt und tabellarisch mit groben Metadaten versehen werden sämtliche Unterlagen aus dem Ordner *„Männerchor Akten 1925–1944“*. Diese Auflistung in der Datei *Akten_Gesamtübersicht.csv* erlaubt die Zuordnung zu folgenden Kategorien: Briefe, Postkarten, Protokolle, Regierungsdokumente, Zeitungsartikel, Rechnungen und Offerten. Die Verteilung ist ungleichmässig: Briefe bilden mit 282 von 381 Seiten die grösste Kategorie; Rechnungen und Offerten sind jeweils nur einseitig vertreten.

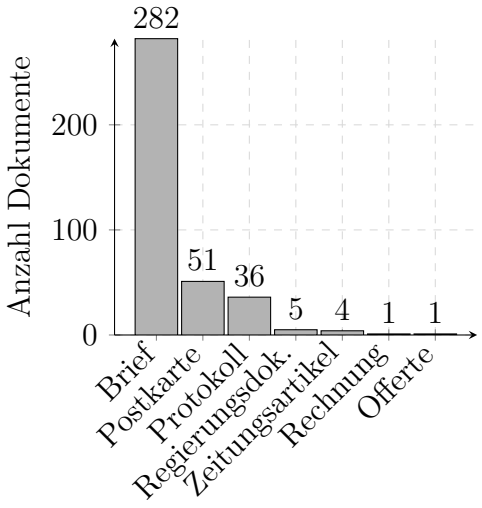



Abbildung 1: Verteilung der Dokumententypen im untersuchten Bestand (150 Akten – 381 Seiten).

Auf Grundlage der oben erwähnten, händisch erstellten Gesamtliste der Akten können durch die systematische Benennung der Dokumente auch Rückschlüsse auf den bislang nicht untersuchten Teil des Bestandes gezogen werden. Mithilfe des Sprachmodells von OpenAI wurde eine grobe Näherung zur Zusammensetzung des restlichen Korpus erarbeitet, wie sie in der rechtsstehenden Darstellung visualisiert ist.

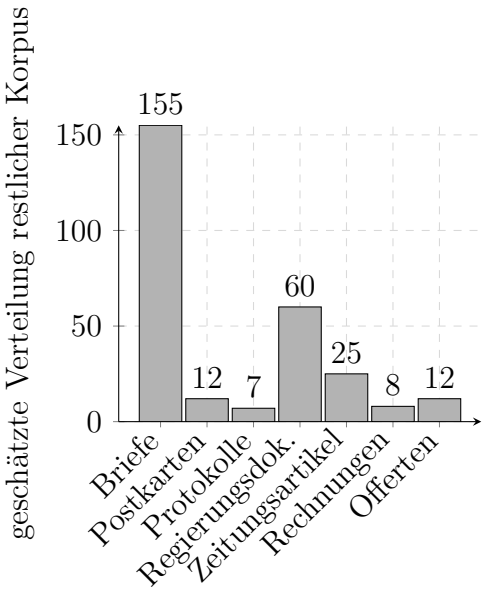



Abbildung 2: geschätzte Verteilung der Dokumententypen im restlichen Bestand.

Diese Übersicht erhebt keinen Anspruch auf Vollständigkeit oder Genauigkeit, sondern dient der Veranschaulichung, dass die Verteilung der Quellengattungen im Zeitraum vor dem Zweiten Weltkrieg möglicherweise deutlich anders gestaltet ist. Eine vertiefte Untersuchung dieser bislang unbearbeiteten Bestände erscheint daher notwendig und markiert eine zentrale Forschungslücke,

die im Rahmen dieser Arbeit erstmals systematisch benannt wird.

### 3.1.3 Sichtung & Kategorisierung in Akten

Für diese Arbeit werden alle Seiten in dem Ordner *„Männerchor Akten 1925–1944“* zwei mal gesichtet und gelesen. Beim ersten Durchgang werden explorativ nach zusammenhängenden Unterlagen gesucht, die im Anschluss zu einer Akten gefasst werden können.

Ausschlaggebend für eine Zusammenfassung in einer Akte sind folgende Faktoren:

- Historische Gliederung durch Bindung (Büroklammern, Heftstreifen, etc)
- gleiche Autorenschaft in direkt aufeinanderfolgenden Seiten
- gleiches Datum [" " " "]
- gleiches Thema [" " " "]

Auf dieser Grundlage wird eine Aktenübersicht[19] im CSV-Format erstellt. Sie seztt sich zusammen aus der Aktennummer, die die Reihenfolge innehalb des ursprünglichen Ordners beschreibt. In diesem ersten Schritt gilt die Lage als Identifikator für die Unterlagen. Jeder Nummer wird darüber hinaus ein beschreibender Titel, und das Erstellungsdatum zugewiesen.

Vorgreifend soll auch die zweite Quellensichtung beschrieben sein, in der diese Daten in der CSV um Metadaten auf Seitenebene und aus Transkribus ergänzt werden. Die Kategorisierung findet also in einem parallelen Prozess mit der Digitalisierung statt. Hierfür werden die Akten auf Seitenebene genau ausgebaut. Dem zugrunde liegt eine internen Seiten-ID, die den Aktennamen und die Position innerhalb der Akte kombiniert (Bsp: Akt_078_S001.jpg). Ab dem Zeitpunkt des Uploads bei Transkribus wird diese jedoch durch Traskribus-Dokument-ID abgelöst. Beide IDs werden zu besseren Nachvollziehbarkeit in der CSV notiert.

Auch inhaltlich wird nochmals schärfer kategorisiert. Mit Tags in Apple-Dateien und der CSV wird nun folgendes erfasst:

- Handschrift
- Maschinenschrift

[19] genannt Akten_Gesamtübersicht.csv; in den Projektdaten

- Bild
- Signatur

Die in Quellenbeschrieb dargestellten Kategorisierungen werden nun als Groundtruthdaten in die CSV aufgenommen, um sie später in der Pipeline auszuwerten.

## 3.2 Digitalisierung der Quellen

Überlieferte analogen Dokumente müssen zunächst fachgerecht für das Projekt und den Digitalisierungsprozess aufbereitet werden. Hierzu werden die Akten aus ihren ursprünglichen Ablagesystemen entnommen und sorgfältig von Heftklammern, Büro- und Gummibändern befreit. Diese konservatorischen Massnahmen sind notwendig, um die langfristige Materialerhaltung zu gewährleisten, da insbesondere Korrosionsspuren ehemaliger Metallklammern die Papierfasern nachhaltig schädigen können. Zudem finden sich häufig Anzeichen von Säurefrass, sofern nicht säurefreies Archivmaterial verwendet wurde.

Für die eigentliche Digitalisierung kommt die native „Dateien"-Applikation von Apple[20] zum Einsatz. Diese bietet neben einer vergleichsweise hochauflösenden Erfassung die Möglichkeit zur direkten Speicherung in einem Cloud-basierten Speichersystem sowie eine automatische Texterkennung (OCR). Ziel dieser Vorgehensweise ist es, die digitalisierten Inhalte möglichst schnell durchsuchbar zu machen und standortunabhängig für das Projekt zugänglich zu machen.

Die Aufnahme der Dokumente erfolgt mithilfe eines Tablets, das auf einem stabilen Stativ exakt im rechten Winkel (90°) über dem zu digitalisierenden Schriftgut positioniert wird. Diese einfache, jedoch effiziente Konfiguration gewährleistet eine gleichbleibend hohe Bildqualität bei gleichzeitig hoher Verarbeitungsgeschwindigkeit. Die digital erfassten Dateien werden konsistent benannt und folgen einer vorab definierten Gesamtübersicht der Bestände. Mehrseitige Konvolute werden dabei als zusammengehörige Akteneinheiten geführt, während Einzeldokumente entsprechend separat erfasst werden. Die Archivierung erfolgt sowohl analog als auch digital auf Seitenebene, um eine möglichst feingranulare Erschliessung zu ermöglichen.

Die initiale Speicherung erfolgt dabei standardmässig im PDF-Format. Für die anschliessende Verarbeitung mit den unten dargestellten Transkriptionswerkzeugen müssen die

[20] vgl. Apple Support: Dateien-App

Dokumente jedoch in das JPEG-Format konvertiert werden. Die Umwandlung erfolgt automatisiert mithilfe eines eigens erstellten Python-Skripts, wie in Anhang A.1 beschrieben.[21] Es extrahiert die Seiten, speichert im geeigneten Format ab und ergänzt die Dateinamen systematisch um eine dreistellige, führend nullengefüllte Seitennummer.

## 3.3 Transkription

Nach der Digitalisierung und Konvertierung der Dokumente beginnt die eigentliche Transkription. Wie im Kapitel Transkriptionen (Methodenvergleich) dargestellt, entscheidet sich dieses Projekt bewusst für Transkribus als zentrale Plattform. Ausschlaggebend sind insbesondere die Möglichkeit, ein eigenes HTR-Modell auf Basis einer projektspezifischen Groundtruth zu trainieren, sowie die integrierten Funktionen zur Annotation von Named Entities direkt im Transkriptionsprozess. Für die effiziente Transkription soll im Folgenden der Workflow beschrieben werden, der einen Mixed Method Ansatz verfolgt.

Wie das Beispiel Abbildung Beispiel für handschriftlichen Text in Akte_076 rechts verdeutlichen soll, sind viele der Akten schwer entzifferbar. Auch Transkribus kommt wegen der kleinen Sets unterscheidlicher Autoren und Autorinnen für spezifische Handschrift an seine Grenzen. Mit Expertise sind diese nur mit hohem zeitlichen Aufwand transkribierbar. Die Baselines[22] ist in diesem Beispiel verhältnismässig homogen. Schwierigkeiten bereiten jedoch Postkarten oder Zeitungsartikel, die mit einer komplexeren Schrift-Setzung einen höheren Aufwand in der Transkription benötigen.

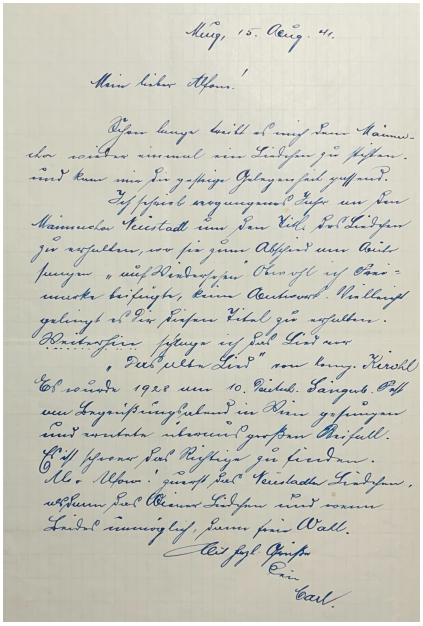


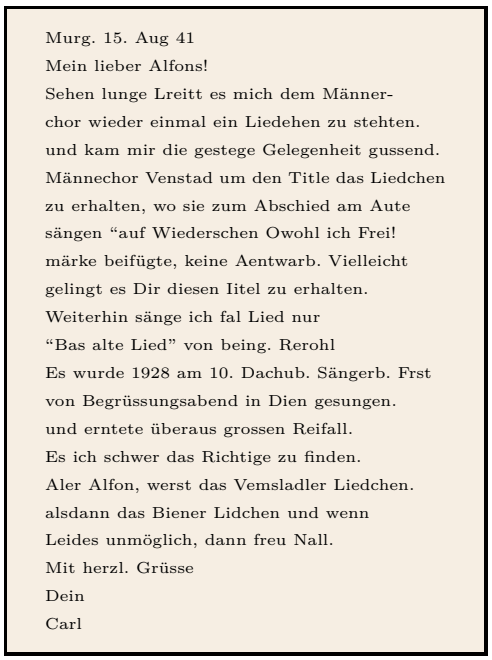
Abbildung 3: Beispiel für handschriftlichen Text in Akte_076

Ausgangspunkt für dieses Projekt ist das generischen Modell *The German Giant I*, das mit einer CER[23] von 8,30% zunächst auf 70 Akten angewendet wird. Sie umfassen 158 Seiten mit insgesamt 22.155 Wörtern. Die Ergebnisse sind dabei jedoch sehr unpräzise, wie Abbildung LLM-Version von Abbildung 3 veranschaulicht. In insgesamt vier Durchläufen

[21] Burkhardt 2025b.
[22] Baselines = Schriftausrichtung
[23] **CER** (Character Error Rate): Kennzahl für die Anzahl falsch erkannter Zeichen.

über diese Selektion wird daher manuell eine Groundtruth für ein eigenes Modell erstellt und gleichzeitig regelbasiert und strukturiert Personen, Orte, Daten und Organisationen getaggt. Zwar erweist sich ChatGPT in der direkten Texterkennung aus Bilddateien (OCR) als nicht ausreichend zuverlässig.

Wie oben ausgeführt wird zur Erstellung des Groundthruth-Korpus bei der manuellen Korrektur OpenAIs ChatGPT-4o-Modell für die Rechtschreibprüfung verwendet. Die vermeintliche Schwäche bei der Transkription, passende Begriffe zu halluzinieren, stellen sich als besonders hilfreich heraus. Insbesondere in der Rekonstruktion fehlender Worte oder Satzteile aus dem semantischen Kontext heraus Wörter oder Satzteile. In Kombination mit der philologischen Expertise bei der Entzifferung einzelner Buchstaben entsteht so ein kollaborativer Transkriptionsprozess, bei dem Maschine und Mensch sich wechselseitig ergänzen. Die automatische Transkription wird in Abbildung 4 dargestellt, die überarbeitete LLM-Version folgt in Abbildung 5.



Murg. 15. Aug 41
Mein lieber Alfons!
Sehen lunge Lreitt es mich dem Männer-
chor wieder einmal ein Liedehen zu stehten.
und kam mir die gestege Gelegenheit gussend.
Männechor Venstad um den Title das Liedchen
zu erhalten, wo sie zum Abschied am Aute
sängen “auf Wiederschen Owohl ich Frei!
märke beifügte, keine Aentwarb. Vielleicht
gelingt es Dir diesen Iitel zu erhalten.
Weiterhin sänge ich fal Lied nur
“Bas alte Lied” von being. Rerohl
Es wurde 1928 am 10. Dachub. Sängerb. Frst
von Begrüssungsabend in Dien gesungen.
und erntete überaus grossen Reifall.
Es ich schwer das Richtige zu finden.
Aler Alfon, werst das Vemsladler Liedchen.
alsdann das Biener Lidchen und wenn
Leides unmöglich, dann freu Nall.
Mit herzl. Grüsse
Dein
Carl

Abbildung 4: LLM-Version von Abbildung 3

Die so entstehnde Groundtruth wird für das Training des Modells (ModelID: 287793)[24] verwendet. Dieses trainierte Modell erreicht eine CER von 6,58% und kommt anschliessend für die automatische Transkription der übrigen Dokumente zum Einsatz. Auch hier ist eine manuelle Überprüfung der durch das eigens trainierte Modell erstellten Transkription unabdingbar, die knapp 1.7% geringere CER macht sich jedoch beim Korrekturaufwand bereits bemerkbar. Gleichzeitig wird diese Korrektur für das Taggen benutzt, das folgend beschrieben werden soll.

Murg, 15. Aug. 41
Mein lieber Alfons!
Schon lange treibt es mich, dem Männerchor wieder einmal ein Liedchen zu stiften, und kam mir die günstige Gelegenheit gelegen.
Ich schrieb vergangenes Jahr an den Männerchor Venstad, um den Titel des Liedchens zu erhalten, das sie zum Abschied am Auto sangen: „Auf Wiedersehen, o wohl ich frei!“
Ich fügte eine Frankierung bei, erhielt jedoch keine Antwort. Vielleicht gelingt es Dir, diesen Titel zu erhalten.
Weiterhin sang ich das Lied nur „Das alte Lied von Wien“. Obwohl es am 10. Dezember 1928 beim Sängerbund-Fest von Begrüssungsabend in Wien gesungen wurde und überaus grossen Beifall erntete, ist es schwer, das Richtige zu finden.
Aber Alfons, zuerst das Venstadler Liedchen, dann das Wiener Liedchen und wenn beides unmöglich, dann Fröhlichsein.
Mit herzlichen Grüssen
Dein
Carl

Abbildung 5: Transkription durch ChatGPT von Abbildung 4

### 3.3.1 Tagging mit Transkribus und LLM

Während der manuellen Korrektur der Transkriptionen erfolgt parallel die Annotation zentraler Entitäten. Transkribus bietet hierfür ein flexibles Tagging-System, mit dem sowohl strukturelle als auch semantische Informationen direkt im Dokument markiert werden können. Im Zentrum stehen dabei Tags für Personen, Orte, Organisationen und Datumsangaben. Diese Kategorien sind für die spätere Analyse besonders relevant, etwa für die Modellierung historischer Netzwerke oder die Kontextualisierung von Ereignissen.

Ein Mixed-Method-Verfahren kommt dort zum Tragen, wo die Transkription an ihre Grenzen stösst: Fehlende Buchstaben, fehlerhafte Worttrennungen oder unleserliche Handschriften lassen sich durch die Kombination aus Modellwissen und menschlichem Quellenverständnis rekonstruieren. ChatGPT liefert hier auf Basis des Kontexts plausible Vorschläge, die von einer historisch geschulten Bearbeitung geprüft und übernommen oder verworfen werden. Dieser kollaborative Vorgang verbessert nicht nur die Lesbarkeit, sondern erhöht auch die semantische Genauigkeit der rekonstruierten Passagen.

Ein Beispiel zeigt die schrittweise Entwicklung einer Transkription: Ausgehend von einem

[24] (vgl. *Transkribus* 2024)

gescannten Originalbrief (Abbildung 3) wird zunächst eine maschinelle Transkription erstellt (Abbildung 4), die anschliessend durch ein LLM geglättet und lesbarer gemacht wird (Abbildung 5).

In einem letzten Schritt erfolgt die manuelle Annotation mit Transkribus-Tags (**??**): Hier werden etwa `Murg` und `Wien` als Orte, `Alfons`, `Carl` und `Kirchl` als Personen sowie das `Deutsch. Sängerb. Fest` als Ereignis markiert. Auch der Liedtitel „Das alte Lied von Wien“ wird in seinen Bestandteilen zwischen Person, Ort und kulturellem Kontext aufgeschlüsselt. Für unklare oder unleserliche Textstellen, wie etwa das Fragment `auf Wiederschen`, kommt das Tag `unclear` zum Einsatz – häufig auf Grundlage einer Vorschlagsformulierung durch ChatGPT.

Zusätzlich zur semantischen Markierung ermöglicht Transkribus auch die Kennzeichnung struktureller Eigenschaften. So wird beispielsweise die Abkürzung `V.D.A.` – für *Verein für das Deutschtum im Ausland* – mit dem Tag `abbrev` versehen, auch wenn diese Tags in der XML-Exportstruktur teilweise nicht vollständig erhalten bleiben (vgl. Kapitel 4.5).

Für die spezifischen Anforderungen dieses Korpus wird das Tagging-Schema gezielt erweitert, etwa um den benutzerdefinierten Tag `signature`, der handschriftliche Unterschriften maschinenlesbar ausweist. Das zweite Beispiel – ein poetischer Brief an Otto (**??**, unten) – zeigt die Anwendung dieses Verfahrens in lyrischer Sprache. Auch hier werden alle erwähnten Personen (u. a. `Otto`, `Lina Fingerdick`, `Otto Bollinger`, `Alfons Zimmermann`), Orte (`Murg`, `Laufenburg (Baden)`, `Rhina`) sowie Organisationen (`Männerchor`) mit den entsprechenden Tags versehen. Die adressierte Funktion `Vereinsführer des Männerchor` wird dabei als Organisationseinheit erfasst und semantisch vom Personenbezug getrennt.

Fehlerhafte oder fehleranfällige Passagen – insbesondere historisch bedingte Schreibungen oder Transkriptionsunschärfen – werden mit dem Tag `sic` versehen. In diesen Fällen folgt die standardisierte Lesart unmittelbar auf das markierte Original, wodurch ein differenzierter Umgang mit dem Quelltext sichergestellt ist.

Alle Tags werden während des Transkriptionsprozesses konsistent dokumentiert und in einem projektspezifischen Regelwerk festgehalten. Dieses dient nicht nur der internen Nachvollziehbarkeit, sondern auch als Grundlage für die spätere Verarbeitung durch Sprachmodelle, die auf die gleichen semantischen Kategorien angewiesen sind. Das strukturierte Tagging bildet somit die Brücke zwischen manueller Quellenarbeit und automatisierter Weiterverarbeitung.

# 4 Methodisches Vorgehen

Digitale Methoden spielen für die Durchführung dieser Arbeit eine zentrale Rolle. Von der Digitalisierung der Quellen über die Transkription bis hin zur Auswertung durchlaufen die Daten zahlreiche Prozessschritte, die mithilfe von Large Language Models, Deep-Learning-Modellen und anderen digitalen Werkzeugen verarbeitet und visualisiert werden. Die Auswahl der Tools orientierte sich dabei an Kriterien wie Verfügbarkeit (Open Source vs. proprietär), Kompatibilität, Community-Support, erforderlichem Arbeitsaufwand und selbstverständlich dem konkreten Mehrwert für die Forschungsfragen.

In diesem Kapitel werden sowohl Werkzeuge vorgestellt, die tatsächlich eingesetzt wurden, als auch solche, die sich im Verlauf des Projekts als ungeeignet erwiesen. Transparenz ist hierbei ein wesentlicher Aspekt: Ein grosser Teil der Methodik entwickelte sich erst im Forschungsprozess selbst. Da sich Large Language Models rasant weiterentwickeln, ist nicht immer von Beginn an klar, ob ein Tool für den eigenen Anwendungsfall geeignet ist. Um diese Unsicherheiten zu dokumentieren, werden hier auch gescheiterte Versuche dargestellt.

## 4.1 LOD – Linked Open Data

Linked Open Data (LOD) bezeichnet einen dezentral organisierten Ansatz zur Veröffentlichung und Verknüpfung strukturierter Daten im Web. Ziel ist es, Datensätze verschiedener Institutionen und Akteure maschinenlesbar zugänglich zu machen und über standardisierte Formate wie RDF und SPARQL miteinander zu verbinden[25]. Wesentliches Merkmal der LOD-Cloud ist dabei die Nutzung semantischer Beziehungen, insbesondere Äquivalenzen einzelner Daten. Hierfür wird häufig das Prädikat `owl:sameAs` genutzt, um z.B. mit *:Choir owl:sameAs wd:Q131186* eine eigene Instanz als identisch mit der Wikidata-Entität für einen Chor zu deklarieren. Klassen oder Instanzen können so aus unterschiedlichen Datenquellen eindeutig identifiziert und zusammengeführt werden.

Die OWL Web Ontology Language, entwickelt vom World Wide Web Consortium (W3C), ist damit ein zentrales Werkzeug für die Realisierung von LOD.[26] Mit ihr lassen sich Ontologien definieren, die Domänen über Klassen, Individuen und deren Relationen formal beschreiben. Sie ermöglichen, logische Schlussfolgerungen zu ziehen, um verteilte Daten-

[25] vgl. Garoufallou und Ovalle-Perandones 2020, S. VI und 13f.
[26] vgl. *OWL Guide* 2004.

bestände zu verknüpfen und maschinenlesbar auszuwerten. Besonders relevant ist dabei `owl:sameAs`, das als Identitätsrelation fungiert: Es deklariert Instanzen, die in unterschiedlichen Quellen unter verschiedenen URIs[27] geführt werden, als dasselbe reale Objekt[28] und ermöglicht so eine präzise Zusammenführung von Informationen — ein Grundpfeiler für die Interoperabilität im Semantic Web. Die OWL-Spezifikation baut auf RDF[29] auf und erweitert es um zusätzliche Konzepte. Die RDF-Daten werden häufig im Turtle-Format (TTL) serialisiert, einer textbasierten Notation für RDF, die eine kompakte, leicht lesbare Schreibweise bietet. Dieses Format eignet sich besonders für den Austausch und die manuelle Bearbeitung von RDF-Tripeln. Die Sprache liegt in drei Varianten vor[30], die sich im Grad ihrer Ausdrucksstärke unterscheiden.[31] Insbesondere OWL DL bietet einen praktikablen Mittelweg zwischen hoher Ausdruckskraft und vollständigem, entscheidbarem Schliessen (Reasoning) und ist daher für viele LOD-Anwendungsfälle geeignet.

Trotz ihres Potenzials wird diese Form der Datenverknüpfung bislang jedoch nicht von allen Websites konsequent umgesetzt.[32]. Für die technische Umsetzung für diese Arbeit werden zwei zentrale Werkzeuge genutzt: Protégé zur Modellierung der Ontologie und GraphDB für deren Verwaltung und Abfrage.

### 4.1.1 Protégé

Zur praktischen Modellierung der Ontologie kam *Protégé* zum Einsatz. Protégé ist eine weit verbreitete Open-Source-Software zur Erstellung, Visualisierung und Verwaltung von Ontologien. Die grafische Oberfläche unterstützt eine intuitive Klassendefinition, Relationserstellung und Instanzverwaltung. Mit Hilfe von Plugins können darüber hinaus logische Konsistenzprüfungen durchgeführt und Ontologien direkt im OWL-Format exportiert werden, um sie in LOD-Workflows einzubinden. Die initiale Version der Ontologie für dieses Projekt entstand zuerst im Codeeditor *Visual Studio Code* wurde aber schnell vollständig in Protégé überarbeitet. Damit bildet das Programm die Grundlage für erste Experimente mit Abfragen in SPARQL.

[27] Abk. **URI** Uniform Resource Identifier
[28] vgl. *OWL Guide* 2004, 2.3. Data Aggregation and Privacy.
[29] Abk. **RDF** Resource Description Framework
[30] OWL Lite, OWL DL und OWL Full
[31] vgl. ebd., 1.1. The Species of OWL.
[32] vgl. Garoufallou und Ovalle-Perandones 2020, S. 14.

### 4.1.2 GraphDB

Für die Speicherung und Abfrage der Ontologie wurde *GraphDB* verwendet. GraphDB ist eine spezialisierte RDF-Triplestore-Datenbank, die es ermöglicht, grosse Mengen an semantisch verknüpften Daten effizient zu verwalten. Mit der integrierten SPARQL-Schnittstelle können Benutzer gezielt nach Instanzen, Klassen und Relationen suchen und komplexe Muster in den Datenbeständen erkennen. Im Rahmen dieser Arbeit diente GraphDB als Backend, um die in Protégé entwickelte Ontologie zu testen und mit realen Entitäten aus den untersuchten Quellen abzugleichen.

### 4.1.3 LOD-Ontologie

Ein wichtiger Aspekt dieser Arbeit ist die Unstrukturiertheit relevanter Informationen. Aus diesem Grund wurde auf der Basis der oben beschriebenen Semantik begonnen, eine eigene Ontologie zu entwickeln, die die identifizierten Entitäten systematisch erfasst. Beim Schreiben dieser initialen Ontologie aus rund 2000 Zeilen Code erweist sich schnell ein neues Problem. Die Datengrundlage aus den geschilderten Vorprojekten (vgl. Forschungsstand und Forschungslücke) ist zu klein, um daraus eine aussagekräftige Netzwerkanalyse zu machen. Hierfür erweisen sich die Unterschiede der Daten zusätzlich als zu gross und damit aufwendig. Der Fokus der Arbeit verschiebt sich dementsprechend von der Ontologieentwicklung auf die Extraktion von Entitäten.

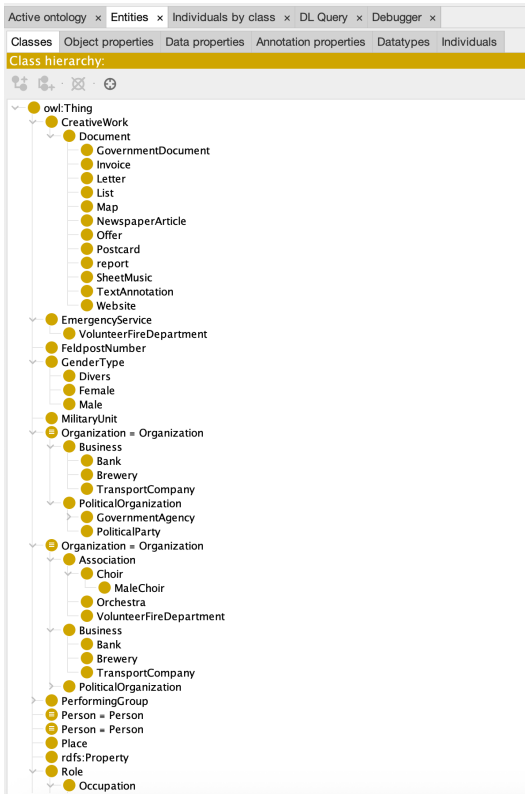



Abbildung 6: Ausschnitt der TTL-Ontologie.

Hinzu kommen externe Quellen, und deren Zugänglichkeit. Die Hauptquellen für Informationen über militärische Einheiten und deren Feldpostnummern sind das „Forum der Wehrmacht“[33] und der „Suchdienst des DRK“[34]. In beiden Fällen liegen die Daten jedoch nicht als LOD vor, sondern im Forum als einfache Strings und beim Deutschen

[33] vgl. Altenburger 2023.
[34] vgl. *DRK Suchdienst | Suche per Feldpostnummer* 2025.

Roten Kreuz als OCR-PDF[35] historischer Suchlisten aus der Nachkriegszeit. Ein manuelles Recherchieren dieser Daten scheint zu diesem Zeitpunkt den Rahmen der Arbeit zu sprengen. Daher wird im Januar 2025 entschieden, den Aufbau einer LOD Datenbank in dieser Form abzubrechen, und einen Fokus auf die Extraktion von Named Entites und das Entity Matching zu legen. Die bis zu diesem Schritt geleistete Vorarbeit beim Sortieren und Klassifizieren von Entitäten, wird in späteren Prozessschritten wieder aufgegriffen[36]. Auch die Recherche der IDs in Wikidata und Geonames, wie im Folgenden beschrieben, wird weiterverwendet.

## 4.2 Wikidata

*Wikidata*[37] ist eines der zentralen Repositorien für Linked Open Data, und bietet eine hohe Interoperabilität durch standardisierte URIs, SPARQL-Endpunkte und offene APIs zu den Entitäten. Jede Entität erhält dabei eine eindeutige, persistente URI (z.B. `wd:Q131186` für einen Chor), die in LOD-Szenarien als stabiler Referenzpunkt dient. Neben anderen betonen Martinez & Pereyra Metnik (2024) beispielsweise:
*„Wikidata stands out for its great potential in interoperability and its ability to connect data from various domains.“*[38]

Wikidata entspricht, ebenso wie das nachfolgend beschriebene GeoNames, den FAIR-Prinzipien: Die Daten sind **F***indable* und **A***ccessible*, **I***nteroperable* und **R***eusable*[39].

Im Rahmen dieser Arbeit dient Wikidata als zentrale externe Referenz, um lokal erhobene Entitäten mit international etablierten Datenobjekten zu verknüpfen und so ihre Interoperabilität sicherzustellen. Die Plattform ermöglicht eine eindeutige Identifizierung sowie die maschinenlesbare Anreicherung um zusätzliche Informationen.

Die praktische Umsetzung zeigt jedoch eine strukturelle Einschränkung. Für diese Arbeiteigens angelegter Einträge auf Wikidata werden trotz systematischer Verknüpfung mit anderen dort verwalteten Entitäten, etwa mit Armeen, Militäreinheiten, Orten und Personen, entfernt die Community–Moderation etwa 70% dieser Einträge. Das zeigt einerseits hohe internen Qualitätsanforderungen auf, andererseits werden diese jedoch nicht klar kommuniziert. Mit regidem Löschen neuer Einträge wird die Verlässlichkeit und den

[35] OCR = Optical Character Recognition
[36] siehe Abschnitt Nodegoat
[37] vgl. *Wikidata* 2025.
[38] Martinez und Metnik o. D.
[39] vgl. Wilkinson u. a. 2016, S. 2.

Nutzen der geleisteten Arbeit erheblich begrenzt. Aufwand und Unsicherheit über die Persistenz der Einträge machen den ursprünglich vorgesehenen LOD–Ansatz in dieser Form nicht praktikabel. Es findet innerhalb der Pipeline daher optionale Anwendung. Wenn einzelne Entitäten eine Wikidata-ID haben, wird diese in die Nodegoat-Datenbank aufgenommen, und für das Entity Matching verwendet.

## 4.3 GeoNames

Ebenso wie Wikidata bietet *GeoNames*[40] eine Open-Source-Plattform für interoperable Daten. GeoNames fokussiert sich hierbei auf geografische Informationen und stellt eine umfassende Datenbank mit über 25 Millionen Ortsnamen und rund 12 Millionen eindeutigen geografischen Objekten bereit. Die Plattform integriert Daten zu Ortsnamen in verschiedenen Sprachen, Höhenlagen, Bevölkerungszahlen und weiteren Attributen aus unterschiedlichen nationalen und internationalen Quellen. Sämtliche Geokoordinaten basieren auf dem WGS84–System[41] und können über frei zugängliche Webservices oder eine API abgerufen werden. Darüber hinaus erlaubt GeoNames registrierten Nutzenden, bestehende Datensätze über eine Wiki–Oberfläche zu bearbeiten oder zu ergänzen, wodurch eine kollaborative Qualitätssicherung gewährleistet wird.

GeoNames wird in dieser Arbeit intensiv zur Referenzierung von Ortsnamen verwendet und bildet die Basis für die Groundtruth, wie sie in den Kapiteln Nodegoat und place_matcher.py beschrieben ist. Im Gegensatz zu Wikidata wurde hier von Beginn an darauf verzichtet, eigene Ortsdatensätze zu ergänzen. Dies liegt einerseits an den klar kommunizierten Community-Guidelines und andererseits daran, dass der Datensatz bis auf wenige, sehr lokale Flurnamen als nahezu vollständig gelten kann[42].

Historische Gebäude wie Gaststätten oder Spitäler fehlen folgerichtig in der GeoNames–Datenbank. Diese Lücke ist erwartbar, aber erwähnenswert, da GeoNames ansonsten eine nahezu vollständige und ausgesprochen detaillierte Datengrundlage bietet.

[40] vgl. *GeoNames* 2025.

[41] *WGS84: geodätische Grundlage des Global Positioning System (GPS)* ;vgl. *WGS84 | Landesamt für Geoinformation und Landesvermessung Niedersachsen* 2025.

[42] der „Totenbühl“ in Murg ist beispielsweise ein solcher Flurname

## 4.4 Nodegoat

Nachdem sich die Implementierung von Linked Open Data (LOD) für das vorliegende Projekt aufgrund des hohen zeitlichen Aufwands als nicht realisierbar erwiesen hat, wird mit **Nodegoat**[43] eine praktikable und zugleich forschungsnahe Alternative eingeführt. Im Folgenden soll das Tool näher beschrieben, und seine Anwendung für das Projekt erläutert werden.

Nodegoat ist eine webbasierte Plattform, die sich besonders in den digitalen Forschungsprojekten in den Geisteswissenschaften etabliert hat. Es unterstützt Forschende in der Modellierung, Verwaltung, Analyse und Visualisierung komplexer Datenbestände. Ein zentrales Merkmal von Nodegoat ist die grafische Benutzeroberfläche, die eine vergleichsweise niedrige Einstiegshürde bietet. Auch Forschenden ohne tiefgehende Programmierkenntnisse wird so die Möglichkeit eröffnet, eigene Datenmodelle zu definieren, zu pflegen und weiterzuentwickeln. Kritisiert werden muss an Nodegoat jedoch die fehlende Dokumentation. Viele Informationen finden sich nur über Drittparteien[44] oder, wie unten geschildert, auf konkrete Nachfrage bei den Entwicklern. Der dann geleistete Support ist jedoch ausgesprochen zeitnah und umfassend erfolgt. Die Plattform folgt einem modularen Prinzip: Über das UI[45] können beliebig viele Datenmodelle erstellt werden, die sich flexibel an die spezifischen Forschungsfragen anpassen lassen. Diese hohe Individualisierbarkeit der Datenstrukturen erlaubt es, innerhalb kürzester Zeit projektspezifische Datenbanken zu konzipieren und fortlaufend zu erweitern oder an sich ändernde Bedürfnisse anzupassen.

Die Grundstruktur eines Modells unterscheidet in Nodegoat zwischen sogenannten `Object Descriptions` und `Sub-Objects`. Erstere legen die grundlegenden Merkmale eines Objekts fest, etwa Zeichenketten, Zahlen oder Verweise auf andere Einträge. So kann eine `Person` in diesem Projekt neben einem Feld für Vor- und Nachname auch eine Referenz auf ein `Gender`-Objekt enthalten, das eine eindeutige Geschlechtszuordnung ermöglicht.

Für das hier behandelte Forschungsvorhaben übernimmt Nodegoat die zentrale Verwaltung der Groundtruthdaten, die später als CSV-Export in die Pipeline integriert werden. Für die Groundtruth werden folgende Entitäten modelliert:

[43] Kessels und Bree 2013.
[44] beispielsweise durch Schulungsunterlagen von Universitäten, hier besonders: Gubler 2025.
[45] Abk.:**UI** User-Interface

| | |
|---|---|
| Personen | Orte |
| Organisationen | Ereignisse |
| Dokumente | Rollen |
| Gender | |

Tabelle 1: Übersicht der erfassten Entitäten

Die Auflistung ist dabei so geordnet, dass die Entitäten mit der grössten Anzahl oder Vielfalt an zugehörigen Sub-Objects zuerst genannt werden. `Sub-Objects` erlauben eine noch vielfältigere Abbildung abhängiger Informationen. Das können in dieser Arbeit zum Beispiel Quellenbelege, sowie temporale oder lokale Attribute, die einem Hauptobjekt zugeordnet werden. Lokale Attribute werden in Verknüpfung mit GeoNames und für Länder, Flurnamen oder Gewässer im Geojson-Format[46]. So lassen sich beispielsweise für `Persons` in den Sub-Objekten `Geburt` oder `Tod` das Datum und der Ort modelieren, sowie die Todesursache notieren.

Die in den Kapiteln Transkriptionen und Methoden, Forschungsstand und Forschungslücke sowie Unmatched Logger beschriebenen Verarbeitungsschritte bilden die Grundlage für die in Nodegoat hinterlegten Entitäten. Die strukturierte Erfassung dient dabei nicht nur der internen Qualitätssicherung, sondern ermöglicht auch eine nachhaltige Referenzierbarkeit durch eindeutig vergebene Identifikatoren, die beispielsweise beim Abgleich von Personendaten genutzt werden, um Textstellen präzise mit den zugehörigen Datenbankeinträgen zu verknüpfen.

Darüber hinaus wird Nodegoat über seine API[47] mit einer eigens entwickelten Webanwendung verknüpft. Diese Webanwendung fungiert als publikumsorientierte Such- und Präsentationsplattform für die erarbeiteten Quellen. Die in den strukturierten JSON-Daten enthaltenen Nodegoat-IDs verknüpfen hier ebenfalls jede identifizierte Named Entity eindeutig mit dem zugehörigen Objekt in der Nodegoat-Datenbank.[48]

[46] Weiterführend: Thomson, Wilde und Roach 2017.
[47] Programmierschnittstelle
[48] an dieser Stelle sei ein Rückverweis auf die Kapitel GeoNames und Wikidata gegeben.

## 4.5 Transkriptionen (Methodenvergleich)

### 4.5.1 Tesseract

Da bereits zu Beginn des Projekts klar ist, dass ein Grossteil der Datenverarbeitung mit Python-Code erfolgen soll, wird gezielt nach Werkzeugen gesucht, die eine automatische Transkription von gescannten Dokumenten ermöglichen. Als besonders etabliert erweist sich die OCR-Engine Tesseract, ein Open-Source-Projekt zur Texterkennung in Bilddateien. Tesseract wird seit den 1980er-Jahren entwickelt, zunächst von Hewlett-Packard, später von Google weitergeführt, und ist über GitHub öffentlich zugänglich.[49]

Die Software basiert seit Version 4 auf einem LSTM-basierten[50] neuronalen Netzwerk, das besonders bei der Erkennung von zusammenhängenden gedruckten Textzeilen eine hohe Genauigkeit bietet. Tesseract unterstützt neben modernen Schrifttypen auch historische Schriftsätze wie Fraktur, was es besonders geeignet für den Einsatz in digitalisierten Archiven macht.[51]

Vorbereitend für den Einsatz von Tesseract müssen alle gescannten PDF in das JPEG Format umgewandelt werden, wofür ein kurzes Python-Script verwendet wird[52]. Im praktischen Einsatz scheiterte die Integration von Tesseract jedoch an der Heterogenität des Korpus: uneinheitliche Layouts, wechselnde Schrifttypen, maschinen- und handschriftliche Texte sowie komplexe Textverläufe. Beispielhaft sollen hier überlagerte oder mehrspaltig angeordnete Passagen auf Postkarten und Zeitungsartikeln genannt werden, die zu massiven Erkennungsfehlern führten. Auch mit angepassten Segmentierungsparametern konnte keine zufriedenstellende Texterkennung erzielt werden.

Tesseract wurde daher nicht weiterverwendet.

### 4.5.2 LLM

Analog zum Einsatz von Python[53] stellt die Integration von Large Language Models (LLMs) von Beginn an einen zentralen Bestandteil der Projektkonzeption dar. Aus diesem

[49] vgl. Weil, Pugin und Dovev 2025.
[50] Abk.: **LSTM** steht für *Long Short-Term Memory*; Architektur der frühen Generation rekurrenter neuronaler Netzwerke *RNNs*. LSTMs wurden entwickelt, um Sequenzdaten zu verarbeiten und dabei sowohl kurzfristige als auch langfristige Abhängigkeiten in der Datenfolge zu erfassen – ein typisches Beispiel sind Texte, Sprache, Zeitreihen oder Handschrift. ;vgl. Beck u. a. 2020, p.1-2.
[51] vgl. Weil, Pugin und Dovev 2025.
[52] vgl. Burkhardt 2025b.
[53] siehe Abschnitt **??**

Grund wird in einer frühen Phase auch der Einsatz von LLMs, spezifisch ChatGPT[54], bei der Transkription der Unterlagen erprobt.

Der Einsatz von LLMs wie ChatGPT für die Transkription historischer Quellen erweist sich als ambivalent. Während die Modelle nach gezielter Anleitung eine erstaunlich präzise Rekonstruktion von Layoutstrukturen und maschinell erfassten Textdaten leisten, bestehen erhebliche Einschränkungen. Im Detail sind das erhebliche Probleme bei der semantischen Genauigkeit. Das LLM beginnt sehr schnell mit sinnverändernden Halluzinationen, die unklare Textpassagen aus dem gelernten Kontext stimmig auffüllt. Eine genaue Transkription, mit forcierter Notation von unklaren Stellen[55] gelingt in der Regel nicht. Die Verarbeitung handschriftlicher Dokumente scheitert weitgehend und führt zu stark spekulativen oder fehlerhaften Inhalten.

Hinzu kommen zu Projektbeginn technische Begrenzungen: Da ein API-Zugang zu OpenAI noch nicht verfügbar ist, erfolgt der Zugriff über die Weboberfläche. Diese stösst bei umfangreichen Eingaben rasch an Kapazitätsgrenzen; Sitzungen brechen häufig ab oder lassen sich nicht zuverlässig fortsetzen. Das kann zu inpersistenzen in der Promptstrukturierung und dem Kontext des LLMs führen, was wiederum direkten Einfluss auf die Verarbeitung der Unterlagen hat.

Zum Zeitpunkt der exploratien Nutzun stelt das zentrales Hindernis der integrierte Content-Filter der Modelle dar. Inhalte mit Bezug zum Nationalsozialismus führen zu einem sofortigen Abbruch der Verarbeitung. Beispielhaft sollen etwa Grussformeln wie „*Heil Hitler*“ genannt sein. Auffällig ist jedoch, dass sich diese Filtermechanismen durch alternative Schreibweisen in den Quellen umgehen lassen. Die schreibweise „*Heil – Hitler*“ umgeht den Filter komplett und wird ohne Einschränkung transkribiert.

Ohne den Zugang zur API und dem damit notwendigen Umweg über den Webclient zeigt sich zudem, dass LLMs ohne persistente Promptstrukturierung dazu neigen, wichtige Hintergrundinformationen zu vergessen. Eine Kombination aus Ground-Truth-gestützter Anleitung und manuellem Review ist daher notwendig, um eine verlässliche Transkription zu gewährleisten. Sie wird in dem Abschnitt Large Language Models näher ausgeführt.

Aus den genannten Gründen kommt auch eine Transkription mittels generativem LLM nicht zum Einsatz.

[54] eine detaillierte Erläuterung findet sich in OpenAI – ChatGPT
[55] Beispielsweise durch das Einfügen von „[...]“

### 4.5.3 Transkribus

Transkribus ist eine webbasierte Plattform zur automatisierten Handschrifterkennung (HTR) und Texterkennung (OCR), die sich seit ihrer Entwicklung im EU-Projekt READ (Recognition and Enrichment of Archival Documents)[56] als Standardwerkzeug in den digitalen Geschichtswissenschaften etabliert hat[57]. Betrieben wird Transkribus durch die READ-COOP SCE, einer europäischen Genossenschaft.

Die Plattform bietet zwei zentrale Zugriffsmöglichkeiten: einerseits die schlanke Webanwendung *Transkribus Lite*, andererseits den *Expert Client*, eine umfangreiche Desktopsoftware zur Bearbeitung und Verwaltung grosser Dokumentenkorpora. Beide Varianten ermöglichen die Transkription von gescannten Dokumenten, die Annotation von strukturellen und semantischen Einheiten sowie den Export in verschiedenen Dateiformaten.

Die Nutzung des Expert Clients erlaubt darüber hinaus eine detaillierte Kontrolle über Transkriptionsprozesse und das zugrunde liegende Datenmanagement. Über integrierte Schnittstellen lassen sich grosse Datenmengen effizient verwalten. Auch externe FTP-Clients können zur Anbindung an das interne Dateisystem verwendet werden, um beispielsweise umfangreiche Digitalisate in strukturierter Form einzubinden.

Ein zentrales Merkmal von Transkribus ist die Möglichkeit, *Tags* zu vergeben. Diese umfassen sowohl strukturelle Merkmale wie Abkürzungen, Unklarheiten oder Layout-Elemente, als auch semantische Einheiten wie Personen, Orte, Organisationen und Daten. Tags können individuell erweitert oder angepasst werden und werden im XML-Export maschinenlesbar dargestellt.

Die Exportfunktion von Transkribus erlaubt den Download der Transkriptionen im standardisierten PageXML-Format. Dieses Format ist auf die langfristige Nachnutzung struktureller Informationen ausgelegt und bildet die Grundlage für weiterführende Auswertungsschritte etwa in Digital Humanities-Projekten.

In der praktischen Handhabung zeigt sich jedoch eine teils deutliche Diskrepanz zwischen den im Interface sichtbaren Informationen und der tatsächlichen XML-Ausgabe. So werden beispielsweise benutzerdefinierte Abkürzungsauflösungen oder Listenstrukturen nicht zuverlässig im XML ausgegeben. Informationen, die manuell innerhalb der Transkriptionsumgebung gepflegt wurden, gehen im strukturierten Export unter Umständen verloren.

[56] vgl. *Recognition and Enrichment of Archival Documents | READ | Projekt | Fact Sheet | H2020* 2025.
[57] vgl. Mühlberger 2019.

Insbesondere bei Listenobjekten, etwa für Personenverzeichnisse oder Inventare, bleibt die XML-Struktur häufig leer. Eine Möglichkeit zur systematischen Nachbearbeitung oder maschinellen Extraktion steht bislang nicht bereit.

Diese Einschränkungen wurden auch in aktuellen Studien festgestellt. So verweisen Capurro et al.[58] im Rahmen ihrer Analyse mehrsprachiger Handschriftenkorpora auf signifikante Herausforderungen bei der automatisierten Layoutanalyse sowie bei der Verarbeitung komplexer Dokumentstrukturen. Sowohl beim Tagging als auch bei der Postcorrection sei weiterhin eine umfangreiche manuelle Nachbearbeitung notwendig, um konsistente und weiterverwendbare Datenformate zu erzeugen.

Trotz dieser Limitierungen liegt der methodische Mehrwert von Transkribus insbesondere in der Möglichkeit, ein eigenes HTR-Modell auf Basis einer spezifischen Groundtruth zu trainieren. Dies erlaubt es, auf charakteristische Eigenschaften eines konkreten Korpus einzugehen und so die Character Error Rate (CER) gegenüber generischen Modellen deutlich zu reduzieren. Darüber hinaus kann durch strukturierte Annotation eine Grundlage für die spätere Modellbewertung oder den Vergleich mit LLM-basierten Verfahren geschaffen werden.

Insgesamt stellt Transkribus eine leistungsfähige Plattform zur initialen Bearbeitung und Annotation historischer Quellen dar. Die automatisierte Erkennung unterstützt den Einstieg in umfangreiche Korpora, ersetzt jedoch nicht die editorische Kontrolle und Nachbearbeitung. Gerade für forschungsorientierte Projekte mit Fokus auf strukturierte, semantisch angereicherte Daten bleibt eine kritische Auseinandersetzung mit den technischen Grenzen unerlässlich.

## 4.6 Large Language Models

Ein zentrales Werkzeug bei der Verarbeitung der historischen Quellen ist die weiter unten näher beschriebene Python-Pipeline, die auf der Verarbeitung von XML-Dateien basiert. Vorgreifend sei erwähnt, dass diese XML-Verarbeitung ein Large Language Model (LLM) zum Custom-Tagging nutzt. Nebst dem Tagging stellt das Programmieren dieser Pipeline eine der Kernherausforderungen dieses Forschungsprojekts dar. Für das Tagging und die Entwicklung der Pipeline werden verschiedene Large Language Models intensiv getestet und eingesetzt.

[58] Capurro, Provatorova und Kanoulas 2023.

### 4.6.1 Msty

Um ein dafür geeignetes LLM zu evaluieren, werden zu Beginn des Projektes beispielhafte Prompts erstellt und deren Ergebnisse systematisch verglichen. Um diesen Vergleich zu erleichtern, wird die Desktop-Anwendung Msty[59] eingesetzt. Zu den zentralen Funktionen gehören parallele Chatinterfaces („Parallel Multiverse Chats“), eine flexible Verwaltung lokaler Wissensbestände („Knowledge Stacks“)[60], sowie eine vollständige Offline-Nutzung ohne externe Datenübertragung. Msty dient dazu, verschiedene Modelle zu testen, durch die Parallel Multiverse Chats Antworten zu vergleichen und Konversationen strukturiert zu verzweigen und auszuwerten.

Hervorzuheben ist, dass dies kein klassisches Benchmarking auf Basis vergleichbarer Resultate ist. Es wird zu diesem frühen Projektzeitpunkt weder systematisch überprüft, welche Qualität der jeweilige Codeteil hat, noch wird gemessen, wie viel Prozent der Named Entities jeweils richtig erkannt werden. Der direkte Vergleich der getesteten LLMs liefert jedoch schnell ein Bild, welches Modell sich für die gleiche Aufgabe besser eignet. Erprobt werden zu Beginn des Projektes im November die folgenden Anbieter und Modelle:

- Alphabet – Gemini
- Anthopic – Claude
- OpenAI – ChatGPT

Sie sollen nachfolgend eingeordnet und deren Verwendung erläutert werden.

### 4.6.2 Alphabet – Gemini

Google Gemini[61] wird im Dezember 2023[62] von Google zunächst einer eingeschränkten Userzahl verfügbar gemacht und gilt als direkte Antwort auf OpenAIs ChatGPT. Es nimmt in dieser Arbeit keinen grossen Raum ein, da es sich im direkten Vergleich mit ChatGPT zum Zeitpunkt des Tests im Winter 2024 und Frühjahr 2025 als weniger präzise bei der Annotation von XML-Files, und weniger zuverlässig beim coden in Python herausstellte. Zur Falsifizierung dieser Annahme gelegentlich durchgeführte Überprüfungen im April und Juni 2025 brachten das selbe Ergebnis. Auch aus ökonomischen Gründen

[59] vgl. *Msty - Using AI Models made Simple and Easy* 2025.
[60] vgl. ebd.
[61] ehemals Google Bard
[62] *Gemini – unser größtes und leistungsfähigstes KI-Modell* 2023.

wurde auf ein zusätzlich zu OpenAI abgeschlossenes Abonnement verzichtet.

### 4.6.3 Anthropic – Claude Code

Claude Code wird am 22. Mai 2025 offiziell veröffentlicht[63] und ist bereits ab dem 24. Februar 2025 in einer öffentlichen Betaphase verfügbar. In diesem Projekt erfolgt die Integration des Tools bereits sehr früh. Am 3. April 2025 wird es erstmals eingesetzt, um komplexe Aufgaben bei der Entwicklung der Verarbeitungs- und Analysepipeline direkt in VS-Code[64] zu übernehmen.

In den ersten Tagen erzielt Claude Code beeindruckende Resultate. Es unterstützt erfolgreich bei der Generierung von Programmcode, insbesondere bei strukturierenden Aufgaben bei der Initialisierung von Python-Funktionen. Einzelne Funktionsbausteine lassen sich durch das Tool effizient erstellen, was zunächst zu einer spürbaren Beschleunigung des Entwicklungsfortschritts führt.

Im weiteren Verlauf zeigen sich jedoch klare Begrenzungen. Bereits im April ist der Kontextumfang des Projekts so gross, dass Claude Code Schwierigkeiten hat, über mehrere Module hinweg konsistent zu arbeiten. Eine zentrale Schwäche besteht darin, dass projektweite Variablen nicht zuverlässig erkannt und übergeben werden. So kann etwa die Variable `mentioned_persons` aus dem Modul `person_matcher.py` nicht als wichtig erkannt und korrekt in `letter_matcher.py` eingebunden werden.

Ein weiteres Problem ist die tiefgreifende Veränderung bestehender Funktionsstrukturen. Claude Code verändert mitunter voll funktionsfähige Module, ohne auf deren interne Abhängigkeiten Rücksicht zu nehmen. Diese Eingriffe führen häufig dazu, dass vormals stabile Komponenten nicht mehr korrekt ausgeführt werden. Variablennamen werden ohne Anzeige modifiziert, Dictionaries durch Listen ersetzt. Der Aufwand für das anschliessende Debugging übersteigt in vielen Fällen den Nutzen der automatisierten Generierung. Zwar können einzelne Änderungsschritte von Claude Code angezeigt werden, Veränderungen können aber so subtil sein, dass sie oft unbemerkt bleiben.

Hinzu kommen erhebliche Nutzungskosten. Aufgrund der Projektgrösse entstehen bereits im April für einzelne Prompts Kosten von bis zu vier US-Dollar. Durch präzises Prompting[65] lässt sich dieser Betrag zwar begrenzen, doch unterschreitet der Preis pro Anfrage

[63] *Claude Code* 2025.
[64] **Abk.:** *Visual Studio Code*, die verwendete Programmierumgebung
[65] Beispielsweise durch Angabe von Dateipfaden und Zeilennummern

selten 0,40$. Die Arbeit mit Claude Code ist damit nicht nur zeitintensiv durch den notwendigen Korrekturaufwand, sondern auch kostenintensiv im operativen Betrieb.

Nach einer intensiven Probephase von etwa fünf Wochen wird der Einsatz von Claude Code im Projekt vor dem Public Release beendet. Die Entscheidung basiert auf einer kritischen Abwägung von Nutzen, Aufwand und Nachhaltigkeit im Hinblick auf die langfristige Wartbarkeit des Codes.

### 4.6.4 OpenAI – ChatGPT

Im Verlauf der Arbeit wird ChatGPT intensiv in mehreren Funktionen genutzt. Dazu zählen Generierung und Überarbeitung von Quellcode, Fehleranalyse in Python- und LaTeX-Skripten, strukturierte Formulierung von Dokumentationsinhalten[66] sowie die semantische Annotation von Personen, Orten und Ereignissen im Transkriptionskorpus. Zum Einsatz kommen insbesondere die Modelle *GPT-4o*, *GPT-4.5*, *o3-mini* sowie deren Varianten *mini high* und *GPT-4.1 mini*[67]. Bereits vor der eigentlichen Konzeptionsphase erfolgen erste Versuche mit dem Vorgängermodell *GPT-3.5*, um grundlegende Anwendungsbereiche für historische Datenverarbeitung auszuloten. Im Folgenden sollen die drei Einsatzfelder **Custom-GPT, Custom-Projekt** und **API** beschrieben und eingeordnet werden.

#### *Custom-GPT*

Ein speziell für das Projekt konfiguriertes *Custom-GPT*[68] wird in dieser Frühphase eingerichtet und dient als Initialer Funktionstest für den Einsatz nicht-domänenspezifischer multimodaler Sprachmodelle. Dieses Modell[69] basiert auf projektspezifischen Anweisungen, Orts- und Personenlisten sowie Groundtruth-Daten aus dem Korpus des Männerchors. Es stellt damit eine Vorform der heute verfügbaren „Projekte“-Funktion im Webinterface von ChatGPT dar. Konfiguriert wird es auf die Verarbeitung und Analyse historischer Dokumente. Zum Einsatz kommen in der Vorphase des Projektes erste Transkriptionen, Metadaten aus der Akten_Gesamtübersicht.csv und ein erster kleiner Korpus sowie ein Manuskript von einem Chormitglied[70]. Getestet wird, ob das Modell erkannte

[66] in der Code-Dokumentation, Modulzusammenfassung oder beim Überarbeiten dieses Textes
[67] vgl. *Model Release Notes* 2025.
[68] **Abk.: GPT** = **G**enerative **P**retrained **T**ransformer
[69] vgl. *GPT Modell forschung-mannerchor-murg* 2025.
[70] Durst 1948, Dieses autobiografische Manuskript wurde anfangs noch in den Korpus zu integrieren versucht, später jedoch entfernt. Es kommt aber in der Verifikation der Archivforschung (bspw. im Militärarchiv Freiburg) zum Einsatz.

Personennamen mit einer bereitgestellten Namensliste vergleichen kann und die Verarbeitung diverser Formate (CSV, Excel, PDF, Bilddateien, Klartext) unterstützt.

Der Abgleich mit den hinterlegten Namenslisten erweist sich als unzureichend. Die Ergebnisse stellen sich als methodisch unzuverlässig bzw. nicht reproduzierbar heraus. Auch eine Weiterverarbeitung durch eine vermeintlich direkte Abfrage von Wikidata zur semantischen Anreicherung historischer Entitäten bleibt ohne belastbare Resultate. Wikidata-IDs werden willkürlich generiert, so verweist die angebliche ID für München auf die Golden Gate Bridge. Gleichwohl zeigen sich punktuelle Stärken bei der automatisierten Erkennung von Strings, etwa bei der Identifikation potenzieller Personen- oder Ortsnamen. Dies legt den Grundstein für den Preprocessing–Schritt[71] und die später automatisierte NER Pipeline[72]. Im Dezember 2024 führt OpenAI „Custom-Projekten“ ein. Ab diesem Zeitpunkt erfolgt der Wechsel zur Projektfunktion, die im Folgenden beschrieben wird.

***Custom-Projekt***

Projekte ermöglichen eine persistente, thematisch fokussierte Interaktion mit dem Modell über längere Zeiträume hinweg. Dabei wird kontextuell relevantes Hintergrundwissen – etwa über Datenstrukturen, verwendete Tools, Entitätenlisten oder Groundtruth-Dateien – dauerhaft im System gespeichert. Diese Informationen stehen in allen Konversationen innerhalb des Projekts zur Verfügung, ohne erneut übergeben werden zu müssen. Im Unterschied zu den oben genannten *Custom GPTs*, bei denen ein Modell über eine Benutzeroberfläche gezielt mit Systemmeldungen und Beispieldaten konfiguriert wird, basiert das Projektkonzept nicht auf einer einmaligen statischen Initialisierung, sondern auf fortlaufender Kontexterweiterung durch Nutzerinteraktionen. Projekte sind nicht öffentlich verfügbar, nicht durch Dritte nutzbar, und bieten keine explizite Konfigurationsmaske. Für dieses längerfristige Forschungsvorhaben mit sich entwickelnden Anforderungen eignet sich die Projekt-Funktion gut. Sich entwickelnde Änderungen im Code können nicht nur durch regelmässigen Uploads neuer Dateien abgebildet werden, sondern werden fortlaufend durch die Interaktion mit dem Modell erweitert, ergänzt und angepasst. So erhält das Projekt auch einen Überblick über andere Module, an denen gerade gearbeitet wird, und schliesst sie in die Antworten zu einem gewissen Grad ein. Insgesamt werden in dem Custom-Projekt „Masterarbeit“ 17 Dateien hochgeladen, darunter fallen die Groundtruth-Dateien und die einzelnen Python-Module. Der Hauptfokus des Projektes liegt auf der

[71] vgl. Vorverarbeitung
[72] vgl. Hauptmodul – Transkribus_to_base

Unterstützung beim Coden der im Kapitel Module im Detail erläuterten Codeteile.

### *API*

Für die automatisierte Verarbeitung und Annotation historischer Dokumente wird in diesem Projekt die Programmierschnittstelle (API) von OpenAI verwendet. Die API ermöglicht eine präzise Steuerung zentraler Parameter wie den Modelltyp, die Kontextlänge und die Temperatur[73]. Die Nutzung der API erleichtert die Integration in automatisierte, skalierbare Workflows und erlaubt eine Wiederverwendung von Prompts über große Dokumentenmengen hinweg. Gleichwohl ist die tatsächliche Reproduzierbarkeit der Ergebnisse nur eingeschränkt gegeben. Selbst bei identischen Eingaben und konstanten Parametern kann es zu leichten Variationen in den Ausgaben kommen, insbesondere bei höheren Temperatureinstellungen oder bei Modellen mit probabilistischer Antwortgenerierung. Reproduzierbarkeit ist daher in diesem Kontext primär als funktionale Replizierbarkeit von Abläufen, weniger als identische Ergebniswiedergabe zu verstehen. So ist eine der Grundannahmen dieser Arbeit, dass ChatGPT den Prompt exakt so ausführt und den Inhalt der historischen Texte komplett unverändert lässt. Dies soll in einem späteren Projektschritt durch eine automatisierte Pipeline überprüft werden. [74]

Ein zentrales Anwendungsfeld der API im Projekt ist die Named Entity Recognition, die über prompt-basierte Anfragen realisiert wird. Dabei kann auf ein aufwendiges Modelltraining und damit einhergehende Groundthruth-Generierung verzichtet werden, da LLMs wie GPT-4 auch ohne Fine-Tuning eine kontextsensitive Entitätserkennung ermöglichen. Bereits 2020 verweisen Brown et al. darauf, dass sich die Lösung nicht explizit trainierte Tasks mit zunehmender Grösse des Models verbessert:

*Each increase has brought improvements in text synthesis and/or downstream NLP tasks, and there is evidence suggesting that log loss, which correlates well with many downstream tasks, follows a smooth trend of improvement with scale [KMH+20]. Since in-context learning involves absorbing many skills and tasks within the parameters of the model, it is plausible that in-context learning abilities might show similarly strong gains with scale.*[75]

Dementsprechend wird OpenAIs LLM in diversen konzeptionellen und operativen Phasen dieses

[73] Die Temperatur steuert den Grad an Zufälligkeit in der Textgenerierung. Bei niedrigen Werten (z.B. 0,2) bevorzugt das Modell sehr wahrscheinliche Ausgaben; bei höheren Werten (bis 1,0) steigt der Anteil weniger wahrscheinlicher, "kreativer" Antworten.

[74] Beispielsweise durch stichprobenartigen Abgleich einzelner Page-XMLs vor und nach der Verarbeitung, einem Stripping aller Tags und einem direkten Stringvergleich mit Levenshtein und einer ausgegebenen Check-Summe. Aufgrund zeitlicher Limitationen ist dies jedoch aktuell nicht der Fall

[75] vgl. Brown u. a. 2020, S.4.

Projektes verwendet. Konkrete Beispiele hierfür finden sich unter anderem in dem Kapiteln Vorverarbeitung.

## 4.7 Webtool

Die Planung sieht vor, oben beschriebene Verknüpfung zu nutzen, um aus der Webanwendung heraus bei Bedarf Detailinformationen zu den einzelnen Entitäten abzurufen. Dazu wird über spezifische API-Requests die jeweilige Objektbeschreibung geladen. In einem exemplarischen Anwendungsfall wird etwa eine Abfrage an eine URL der Form gesendet. Hierbei steht die

`https://api.nodegoat.dasch.swiss/data/type/11680object/ngEL9c68pELQqGVuoFN49t/`

Type-ID *11680* im Beispiel für den jeweiligen Modelltyp „Organisation“ und die Zeichenkette am Ende für die eindeutige Nodegoat-ID des Objekts[76].

Die dabei zurückgelieferte JSON-Antwort enthält die gespeicherten Metadaten (z.B. Namensvarianten, Zugehörigkeiten, Quellenbelege). Um diese Informationen benutzerfreundlich darzustellen, wird angestrebt, den aus Nodegoat bekannten „Object Detail View“ in Form eines iFrames direkt in die Webanwendung einzubetten. Dabei kann über gezielte CSS-Regeln gesteuert werden, dass lediglich der gewünschte Objektbereich angezeigt wird, ohne die übrigen Bestandteile der öffentlichen Nodegoat-Oberfläche zu übernehmen.

In der technischen Umsetzung wurde zudem erörtert, ob die interne Object-ID oder die plattformübergreifende Nodegoat-ID als Referenz verwendet werden sollte. Die Rückmeldung bei den Nodegoat-Entwicklern [77] legt auf Anfrage nahe, dass beide ID-Typen im Prinzip austauschbar sind: Die Object-ID gewährleistet eine eindeutige Identifikation innerhalb einer spezifischen Nodegoat-Instanz, während die Nodegoat-ID eine konsistente Referenz über mehrere Installationen hinweg ermöglicht — auch im Hinblick auf LOD-Kompatibilität.

Für die Integration in die eigene Webanwendung wird daher ein hybrides Modell verfolgt: In den JSON-Daten der Quelltexte werden Nodegoat-IDs gespeichert, um langfristig eine offene Verknüpfbarkeit sicherzustellen. Gleichzeitig wird über die API der jeweilige Objekttyp (z.B.„Person“, „Organisation“) abgefragt, um die semantischen Eigenschaften der Entität zu laden. Diese werden mit dem Model-Endpunkt kombiniert (z.B. `https://api.nodegoat.dasch.swiss/model/type/11680`), um etwa Labelstrukturen oder benutzerdefinierte Felder korrekt abzubilden.

Auf diese Weise kann die Webanwendung den Nutzenden nicht nur eine reine Objektliste liefern,

[76] im Beispiel eine gekürzte Nodegoat-ID
[77] Kessels und van Bree

sondern auch kontextreiche Detailansichten generieren. Sie sind als iFrame eingebettet oder dynamisch gerendert und visualisieren alle relevanten Informationen direkt aus Nodegoat. Um die Serverlast zu minimieren, wird hierbei eine Caching-Strategie empfohlen, sodass wiederholte API-Abfragen effizient verarbeitet werden können.

### 4.7.1 NDRCore

https://ndrcore.org/ –> CMS und API-Wrapper

### 4.7.2 mongoDB

https://www.mongodb.com/de-de –> Datenbanksystem für Korpusdaten

### 4.7.3 Cantaloupe

https://cantaloupe-project.github.io/ –> IIIF Image API server

### 4.7.4 Nodegoat

Zusammengefasst fungiert Nodegoat somit als zentrales Bindeglied zwischen der internen Datenhaltung und der öffentlichen Präsentation: Es vereinfacht die Pflege konsistenter Groundtruth-Daten, unterstützt deren Ausspielung über standardisierte Schnittstellen und ermöglicht eine modular erweiterbare Verknüpfung mit Webportalen und Suchsystemen.

# 5 Pipeline

### 5.0.1 Übersichtsgrafik der Pipeline

Die untenstehende Grafik soll im Folgenden als visuelle Orientierung dienen. Sie bildet die logische Gliederung der Pipeline ab, wie sie im weiteren Verlauf des Kapitels analysiert wird. Zum Einsatz kommt sie als Referenz für nachfolgende Module in Form schematischer Zeichnungen bei der Einordnung der verschiedenen Verarbeitungsschritte und Modulabhängigkeiten. Als zentrale Übersicht gliedert sich diese Darstellung in drei farblich differenzierte Ebenen, die im Folgenden erläutert werden.

Grün markiert sind externe Ressourcen, die die Grundlage der Verarbeitung bilden. Dazu zählen zum einen XML-Dateien, die aus dem Transkriptionsprogramm Transkribus stammen, und zum anderen strukturierte Groundtruth-Daten im CSV-Format, die zuvor aus Nodegoat exportiert wurden. Dieser Prozess ist im Kapitel Transkriptionen (Methodenvergleich) und Nodegoat dargestellt.

Orange steht für die zentralen Verarbeitungsschritte der Pipeline. Dazu gehört insbesondere das Preprocessing durch das LLM, wie im Abschnitt Vorverarbeitung ausgeführt, sowie das Hauptmodul `transkribus_to_base.py`. Dieses Modul koordiniert den gesamten Verarbeitungsablauf und wird im nachfolgenden Kapitel Hauptmodul – Transkribus_to_base ausführlich behandelt.

Blau gekennzeichnet sind die einzelnen Funktionsmodule, in die sich `transkribus_to_base.py` unterteilt. Diese übernehmen spezialisierte Aufgaben, etwa die Erkennung und Anreicherung von Personen, Orten, Organisationen oder Ereignissen. Eine detaillierte Beschreibung der jeweiligen Module erfolgt im Abschnitt Module im Detail.

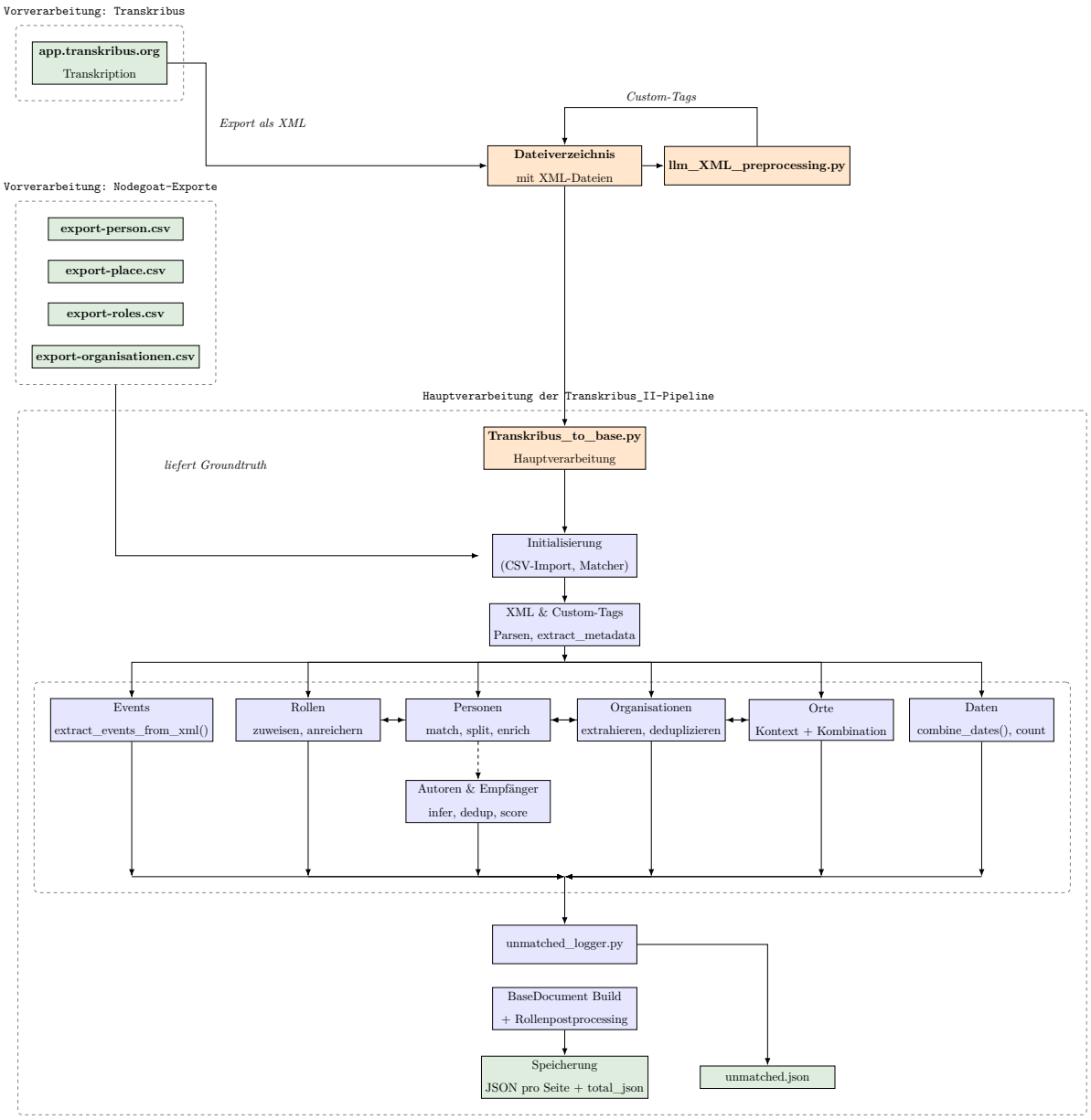



Abbildung 7: Übersicht der gesamten XML-to-JSON-Pipeline

.

## 5.1 Vorverarbeitung

Neben Transkribus, das bei der Transkription einen elementaren Schritt in der Vorverarbeitung aller Dokumente darstellt, müssen die Seiten für bessere Ergebnisse weiter vorbereitet werden. Die Kernherausforderung der vorliegenden Arbeit ist die Named Entity Recognition (NER). Wie bereits im Abschnitt Transkribus ausgeführt, werden viele Inhalte bereits während des Transkriptionsprozesses manuell auf Basis von im Anhang erläuterten Regeln getaggt[78]. Dieser Prozess ist zwar sehr genau, überschreitet aber auch den begrenzten zeitlichen Rahmen aufgrund der Menge an Unterlagen. Daher wird für das Projekt auch eine zweite Verarbeitungsform gewählt, eine **NER durch ein LLM**.

Im Zentrum des Vorverarbeitungsskripts steht eine strukturierte Anbindung an die OpenAI-API, um ausgewählte PAGE-XML-Dateien aus dem Transkribus-Export automatisiert mit Annotationen anzureichern. Das Skript ist modular aufgebaut und folgt einer klar definierten Abfolge von Verarbeitungsschritten, die im Folgenden näher erläutert werden.

Die Vorverarbeitung beginnt mit der Funktion `get_api_client()`, die eine Verbindung zur OpenAI-Programmierschnittstelle aufbaut. Dabei wird der API-Schlüssel über eine Umgebungsvariable geladen und für spätere Anfragen bereitgestellt. Die zentrale Annotation erfolgt in der Funktion `annotate_with_llm()`, die den vollständigen Unicode-Text der XML-Datei verarbeitet und an das Modell GPT-4o übergibt. Grundlage ist ein präzise formulierter Prompt [79], der die Struktur des zu erwartenden Outputs definiert. Der Prompt spezifiziert, dass ausschliesslich `<TextLine>`-Elemente bearbeitet und mit einem `custom`-Attribut versehen werden dürfen. Innerhalb dieses Attributs werden ausschliesslich tatsächlich erkannte Entitäten in standardisiertem Format codiert, darunter Personen (`person`), Rollen (`role`), Orte (`place`), Organisationen (`organization`), Daten (`date`) sowie autoren- und empfängerbezogene Markierungen (`author`, `recipient`, `creation_place`). Für jedes einzelne Tag gibt es genaue Anweisungen an das Modell. Auch ein Beispielresultat der Annotation wird jedes Mal mitgeliefert, um möglichst wenig Varianz in den Antworten zu erhalten. Gleichzeitig liefert das Beispielresultat auch Informationen über Abkürzungen, die nicht von dem Modell als Organisationen erkannt werden. Das „WhW - das Winterhilfswerk“ ist eine Abkürzung, die offenbar nicht in den Trainingsdaten des Modells vorkommt. Dieses Vorgehen stellt den Versuch dar, ein nicht domänenspezifisch trainiertes Modell auf eine historische Quellenlage zu adaptieren. Es ist jedoch davon auszugehen, dass ein speziell auf den historischen Korpus abgestimmtes Sprachmodell deutlich präzisere Ergebnisse liefern würde.

Die Antwort des Sprachmodels ist eine vollständige XML-Datei, die sämtliche bestehenden

[78] vgl. Anhang Tagging in Transkribus
[79] vgl. Anhang Prompt der LLM Vorverarbeitung

Strukturinformationen beibehalten soll. Der Rückgabewert wird zunächst in der Funktion `clean_llm_output()` auf mögliche Formatierungen überprüft. Diese Funktion extrahiert den tatsächlichen XML-Inhalt aus dem Rückgabestring, etwa wenn dieser durch Markdown-Wrapper wie ```` ```xml-content''' ```` eingefasst wurde.

Die konkrete Verarbeitung einzelner Dateien erfolgt über die Funktion `process_file()`, die eine originale Transkribus-XML-Datei einliest, an das Modell übergibt, das Ergebnis prüft und anschliessend unter verändertem Dateinamen (Suffix `_preprocessed`) abspeichert. Vor dem Schreiben erfolgt eine strukturelle Validierung mittels XML-Parser, um syntaktische Fehler oder unvollständige Rückgaben zu erkennen. Fehlermeldungen werden protokolliert, fehlerhafte Dateien übersprungen.

Die eigentliche Ausführung der Batch-Verarbeitung wird durch die `main()`-Funktion gesteuert. Diese durchläuft das in `TRANSKRIBUS_DIR` konfigurierte Arbeitsverzeichnis, wobei alle Unterordner der Form `<7-stelliger Ordner>/Akte_<Nummer>/page/` rekursiv analysiert werden. XML-Dateien, die bereits mit `_preprocessed` enden, werden übersprungen. Zusätzlich wird ein Verzeichnis ignoriert, wenn bereits mehr als fünfzig Prozent der Dateien annotiert wurden. Die verbleibenden Dateien werden schrittweise mit dem Modell verarbeitet. Hintergrund ist eine kosteneffiziente Verarbeitung, die verhindern soll, dass Dateien mehrfach durch die API bearbeitet werden. Die Funktion `annotate_with_llm()` berechnet darüber hinaus die Anzahl der vom Modell verarbeiteten Eingabe- und Ausgabetokens. Auf Basis dieser Werte wird für jede Datei eine Schätzung der anfallenden API-Kosten vorgenommen. Diese Informationen werden für jede Anfrage protokolliert, um eine transparente Kostenkontrolle sicherzustellen.

Am Ende dieses Prozesses entsteht pro annotierter Seite eine neue, syntaktisch validierte XML-Datei, die alle Annotationen als `custom`-Attribute enthält. Diese dienen in den nachfolgenden Modulen der Pipeline (insbesondere `transkribus_to_base.py`) als Grundlage für die strukturierte Extraktion und Validierung von Entitäten.

## 5.2 Hauptmodul – Transkribus_to_base

### 5.2.1 Initialisierung und Pfadlogik

Die Initialisierungsphase des Skripts[80] dient der Einrichtung sämtlicher Systempfade, Datenquellen, Modulabhängigkeiten und Ressourcen. Im Zentrum steht dabei die Festlegung der Projektstruktur sowie die dynamische Integration aller untergeordneten Module und Datenbestände.

Zu Beginn werden die Python-Standardbibliotheken sowie externe Abhängigkeiten geladen, darunter `pandas` für Tabellenverarbeitung, `spacy` für die linguistische Analyse und `rapidfuzz`

[80] bis ca. Line 250

für den Vergleich von ähnlichen Strings. Zur besseren Nachvollziehbarkeit von Verarbeitungsläufen beim Debugging wird das aktuelle Datum mit Zeitstempel generiert und als Konsolenoutput ausgegeben.
Das Projekt ist so organisiert, dass sämtliche zentralen Datenverzeichnisse, Ressourcen und Modulstrukturen relativ zur Wurzel definiert und im Code programmatisch zugänglich gemacht sind. Das Projektwurzelverzeichnis wird dynamisch erkannt, und fungiert als zentraler Referenzpunkt für alle nachfolgenden Verzeichnispfade. Dadurch lassen sich alle enthaltenen Funktions- und Klassendefinitionen zentral importieren. Absolute Pfadangaben werden vollständig vermieden.

Zur Initialisierungslogik gehört beispielsweise das standardisierte Einlesen der Unterverzeichnisse `Data` für CSV-basierte Groundtruth-Informationen sowie `Module` für Verarbeitungsfunktionen.[81]. Die jeweiligen Nodegoat-Exporte sind zentral im Ordner `/Data/Nodegoat_Export` gespeichert. Die dort hinterlegten Informationen werden aus den CSV-Dateien geladen, in `pandas.DataFrames` überführt und durch Fehlerbehandlung in Form von `try- & exept`-Blöcken abgesichert. Eine Konsolenausgabe informiert über Anzahl und Status der geladenen Einträge.
Es folgen Abfragen, ob das deutsche Sprachmodell `de_core_news_sm` von `spaCy` und ein gültiger API-Schlüssel für ChatGPT in der Umgebungsvariable geladen ist. Gegebenenfalls wird auf ein Fehlen hingewiesen. Sollte letztgenannter Schlüssel nicht vorhanden sein, wird der am Ende folgende Enrichment-Prozess automatisch deaktiviert.
Die standardisierte Datenstruktur für Personen wird unmittelbar nach dem Einlesen der Groundtruth-Dateien erzeugt. Hierzu werden die relevanten Felder aus der CSV-Datei `export-person.csv` extrahiert und als strukturierte Python-Dictionaries gespeichert. Diese beinhalten Attribute wie `forename`, `familyname`, `alternate_name`, `title` sowie die eindeutige `nodegoat_id`. Die Struktur orientiert sich an der in `document_schemas.py` definierten Klasse `Person`, bildet jedoch in dieser Phase noch keine Instanzen davon.[82] Sie dient als Referenz für alle nachfolgenden Matching- und Deduplikationsvorgänge.
Abschliessend stehen zwei Hilfsfunktionen[83] bereit, um neu erkannte Personen in die bestehende CSV-Struktur zu überführen. Auf Basis des „fuzzy stringmatchings“ mit der sogenannten Levensthein-Distanz[84] wird eine Duplikatsprüfung durch `Rapidfuzz` realisiert und stellt sicher,

[81] vgl. Module im Detail
[82] Ein Relikt des dynamisch entstandenen Codes, das aus Zeitgründen noch nicht aufgelöst wurde
[83] **Hilfsfunktionen:** `load_known_persons` und `person_exists_in_known_list`
[84] **Levensthein-Distanz:** Die minimale Anzahl von elementaren Editieroperationen (Einfügen, Löschen, Ersetzen), die notwendig sind, um ein Wort a in ein anderes Wort b zu überführen. vgl: Levenshtein 2025.

dass nur bisher unbekannte Einträge ergänzt werden.
Damit ist die Initialisierungs- und Pfadlogik abgeschlossen, der Grundstein für eine skalierbare, reproduzierbare und systematisch strukturierte Weiterverarbeitung in den folgenden Modulen gelegt.

### 5.2.2 Extraktion von Struktur und Fliesstext

Die Verarbeitung einer Transkribus-Seite beginnt mit der strukturierten Auswertung der zugehörigen PAGE-XML-Datei. Ziel ist die Gewinnung zentraler Informationen, die sowohl die technische Identifikation als auch die inhaltliche Analyse des Dokuments ermöglichen. Dieser Extraktionsprozess lässt sich in drei Segmente gliedern:

1. Extraktion technischer Metadaten
2. Extraktion des transkribierten Fließtexts
3. Extraktion semantisch annotierter Entitäten aus Custom-Tags

Die folgenden Funktionen bilden jeweils die Kernoperationen dieser drei Schritte und bereiten die Daten für die nachgelagerte Entitätenanreicherung, Validierung und den Export in ein standardisiertes JSON-Format vor.

#### *1. Technische Metadaten*

*extract_metadata_from_xml ( )*

Diese Funktion liest die im `<TranskribusMetadata>`-Block gespeicherten Informationen aus, die für die eindeutige Identifikation jeder Seite erforderlich sind. Dazu zählen insbesondere die Dokumenten-ID (`docId`), die Seiten-ID (`pageId`), die Transkriptions-ID (`tsid`) sowie die Pfade zu Bild und XML-Datei. Diese Informationen werden in einem Dictionary gespeichert und später im Attributblock des JSON-Dokuments hinterlegt. Im Falle fehlender Metadaten wird ein leeres Dictionary zurückgegeben, um die Robustheit der Verarbeitung zu gewährleisten.
*get_document_type ( )*

ist eine in type_matcher.py genauer ausgeführte Funktion. Sie liefert Informationen, ob ein Dokument beispielsweise als Brief, Postkarte oder Protokoll kategorisiert werden kann.

#### *2. Transkribierter Fließtext*

*extract_text_from_xml ( )*

Der aus dem PAGE-XML extrahierte Fließtext bildet die Grundlage für eine Vielzahl regelbasierter und LLM-gestützter Analyseschritte. Die Funktion iteriert über alle `TextLine`-Elemente und extrahiert jeweils den Inhalt der Elemente `<TextEquiv>` bzw. `<Unicode>`-Unterelements. Die resultierenden Zeilen werden in einzeilige Strings überführt und mit Zeilenumbrüchen (`\n`)

getrennt. Dadurch bleibt der Zeilenkontext auch in der späteren Analyse (z.B. bei der Zuordnung von Rollen oder Orten) erhalten. Zusätzlich wird dieser Text verwendet, um die in den Custom-Tags angegebenen Offset-Positionen auf den tatsächlichen Textinhalt zu projizieren.

### *3. Semantische Annotation über Custom-Tags*

*extract_custom_attributes ( )*

Diese zentrale Funktion wertet die semantischen Annotationen aus, die über das `custom`-Attribut in den `TextLine`-Elementen gespeichert sind. Jede Zeile wird einzeln geprüft, ob sie eines oder mehrere Custom-Tags enthält. Ist dies der Fall, werden die in diesen Tags enthaltenen Entitätsangaben mittels spezialisierter Subfunktionen analysiert und strukturiert erfasst. Für jede Entität wird der zugehörige Textausschnitt mithilfe der im Tag angegebenen `offset`- und `length`-Angaben aus dem Fließtext extrahiert. Die Ergebnisse werden in einem Dictionary gesammelt, das nach Entitätskategorien gegliedert ist. Die relevanten Subfunktionen sind im Folgenden beschrieben:

- **extract_person_from_custom ( )** erkennt Personen anhand expliziter Namensangaben im XML-Tag (`firstname`, `lastname`) oder, falls diese fehlen, heuristisch über Offset-basierte Textsegmente. Die Namen werden mit `split_name_string()` zerlegt und ggf. durch `correct_swapped_name()` korrigiert. Zusätzlich werden Titel [85] sowie Rollen[86] identifiziert und in strukturierter Form gespeichert. Der Abgleich mit bekannten Personen erfolgt über die Funktion `match_person()`, die Fuzzy-Matching und ID-Zuweisung auf Basis einer CSV-basierten Groundtruth durchführt. Bei unvollständigen Matches wird das Feld `needs_review=true` gesetzt. Als ergänzender Mechanismus dient `fallback_match_by_familyname_and_gender ( )`, um die Nodegoat-ID einer weiblichen Personen bei fehlendem Vornamen dennoch zu identifizieren[87].

- **extract_organization_from_custom ( )** verarbeitet Custom-Tags der Form `organization` oder `org`. Die Funktion extrahiert über Offset-Angaben den relevanten Textabschnitt und identifiziert zusätzlich eine potenzielle `wikidata_id` mit Hilfe der Hilfsfunktion `extract_wikidata_id ( )`. Die resultierenden Objekte enthalten strukturierte Felder für Name, Position, Wikidata-ID sowie Ort und Organisationstyp. Die Einträge werden dann mit `organization_matcher.py` gegen eine Groundtruth-Organisationstabelle validiert.

- **extract_place_from_custom ( )** ist zuständig für die Extraktion von Ortsnamen aus

[85] z.B. „`Herr`“, „`Fräulein`“
[86] z.B. „`Führer`“, „`Schneiderin`“
[87] z.B. um `„Frau Zimmermann"` nicht mit `„Alfons Zimmermann" zu matchen`

Custom-Tags der Form `place`. Basierend auf den Offset-Positionen wird der Ortstext extrahiert und über eine Instanz des `place_matcher.py`-Moduls mit bekannten Orten abgeglichen. Dabei werden GeoNames- und Wikidata-Daten einbezogen. Treffer werden anhand eines Score-Werts bewertet und mit Feldern wie `matched_name`, `geonames_id` oder `confidence` angereichert. Auch fehlerhafte oder nicht eindeutig zuordenbare Orte werden zurückgegeben, um eine nachträgliche manuelle Validierung zu ermöglichen.

- **extract_date_from_custom ( )** erkennt Datumsangaben anhand typischer Custom-Tag-Strukturen. Dabei wird das Datum aus dem Text extrahiert und in eine standardisierte Zeichenkette überführt. Die extrahierten Daten werden später zur Zählung und semantischen Kontextualisierung genutzt.

- **parse_custom_attributes ( )** Diese Hilfsfunktion übernimmt die Umwandlung der Custom-Attributinhalte (z.B. `offset:10; length:12`) in strukturierte Dictionaries. Sie bildet die Grundlage für alle weiteren Extraktionsfunktionen und sorgt für eine einheitliche Zugriffsebene auf die annotierten Positionsdaten.

Die in `extract_custom_attributes()` erkannten Entitäten werden in einem standardisierten Dictionary gespeichert, das fünf zentrale Schlüssel enthält: `persons`, `roles`, `organizations`, `places` und `dates`. Dieses Dictionary bildet die Basis für nachgelagerte Validierungsschritte, die Deduplikation der Entitäten sowie die finale Konvertierung in ein BaseDocument-konformes JSON-Schema.

### 5.2.3 Orchestrierungsfunktion

`process_transkribus_file()` ist der Kern der Dokumentenverarbeitung. Hier werden alle Module gebündelt aufgerufen, die Informationen der Verarbeitungsschritte abgefragt, und in das BaseDokument final zusammengefügt. Diese Orchestrierung erfolgt in insgesamt 18 Schritten, die nachfolgend erläutert werden sollen.

#### *1. XML-Verarbeitung und Dokument-Identifikation*

Das XML-Dokument wird mit `ET.parse()` geladen und in ein Elementbaumobjekt `root` überführt. Danach ermittelt `type_matcher.py` den Dokumenttyp, während `extract_metadata_from_xml ( )` die extrahierten Metadaten liefert.

#### *2. Transkriptextraktion*

Der mit extract_text_from_xml ( ) aus dem XML extrahierte Transkriptionstext wird zu einem fortlaufenden Fließtext zusammengesetzt. Dokumente mit zu kurzem oder fehlendem Text werden an dieser Stelle ausgeschlossen (`return None`).

### *3. Raw_author und Raw_recipient*

Die Funktionen `match_authors()` und `match_recipients()` aus dem Modul `letter_metadata_matcher.py` liefern potenzielle Verfasser:innen und Adressat:innen. Die Ausgabe dieser Funktionen kann in unterschiedlichen Formaten erfolgen – als `Person`-Instanz, als `dict` mit Namensfeldern oder als Liste solcher Elemente.
Zur Vereinheitlichung und robusten Weiterverarbeitung werden alle Rückgaben in einheitliche `List[Person]`-Objekte überführt. Dabei wird für jedes Element geprüft, ob es sich bereits um eine gültige Instanz der Klasse `Person` handelt, oder ob es über `code.from_dict()` aus einem Dictionary erzeugt werden muss. Diese Zwischenspeicherung in `temp_authors` und `temp_recipients` dient als Grundlage für spätere Validierungs-, Deduplikations- und Anreicherungsprozesse innerhalb der Pipeline.

### *4. LLM-gestützte Verfeinerung*

`infer_authors_recipients()` verwendet ein Sprachmodell zur Klassifikation von Personenrollen auf Basis des Textkontexts. `ensure_author_recipient_in_mentions()` stellt sicher, dass alle erkannten Personen auch in `mentioned_persons` auftauchen.

### *5. Rollenanreicherung für Autor:innen*

`assign_roles_to_known_persons()` weist bekannten Personen basierend auf Textkontext ihre Rollen zu. Diese werden direkt in die Objekte der `authors`-Liste eingetragen.

### *6. Extraktion aus Custom-Tags*

Die Funktion `extract_custom_attributes()` durchsucht XML-Custom-Tags nach annotierten Entitäten. Ergänzend erkennt `extract_standalone_roles()` kontextuell verwendete Rollenbezeichnungen ohne direkte Personenbindung.

### *7. Organisationen im Fließtext*

Zusätzlich zu Custom-Tags werden Organisationen direkt im Fließtext mittels `match_organization_from_text()` identifiziert und in die Datenstruktur übernommen.

### *8. Personenanreicherung und Deduplikation*

Rollenbezeichnungen innerhalb von Tokens (z. B. „Dirigent Maier“) werden durch `extract_role_in_token()` erkannt. `deduplicate_persons()` entfernt doppelte Einträge, und `get_best_match_info()` gleicht mit bekannten Personenprofilen ab.

### *9. Autor-Rückübertragung*

Zur Verbesserung der Datenqualität werden fehlende Felder bei `authors` aus den angereicherten Personendaten (`enriched_persons`) ergänzt.

### *10. Logging neuer Personen*

Nicht in der Groundtruth enthaltene Personen werden für spätere Kontrolle im Logfile vermerkt.

### *11. Konvertierung zu `Person`-Objekten*

Sämtliche angereicherten und nicht verworfenen Personeninformationen werden in valide Instanzen der Klasse `Person` überführt und der Liste `mentioned_persons` hinzugefügt.

### *12. Organisationen verarbeiten*

Die Liste der Organisationen wird in Instanzen der Klasse `Organization` übertragen und entsprechend normalisiert.

### *13. Orte deduplizieren und zuordnen*

Ortsangaben werden durch `place_m.deduplicate_places()` bereinigt und in Instanzen der Klasse `Place` überführt.

### *14. Ereignisse extrahieren*

Die Funktion `extract_events_from_xml()` identifiziert strukturierte Ereignisse aus den XML-Tags.

### *15. Konstruktion des `BaseDocument`*

Alle extrahierten und angereicherten Informationen werden in einem strukturierten `BaseDocument`-Objekt gebündelt.

### *16. Zweite Rollenanreicherung (Autoren und Empfänger)*

Rollen und Rollenschemata werden erneut über `assign_roles_to_known_persons()` ermittelt. `flatten_organisation_entry()` sorgt für saubere Strukturen bei verknüpften Organisationen.

### *17. Finaler Deduplikationsschritt*

`deduplicate_and_group_persons()` konsolidiert die Entitätenlisten endgültig. `infer_gender_for_person()` füllt ggf. fehlende Geschlechtsangaben. Auf Basis von ID oder Score werden die finalen `authors` und `recipients` bestimmt.

### *18. Validierung und Speicherung*

Das erstellte Dokument wird durch `validate_extended()` validiert. Bei Erfolg erfolgt die Ausgabe als JSON-Datei. Die Funktion `update_total_json()` aktualisiert das zentrale Gesamtverzeichnis `total_json.json`.

Zweck: Erkennung, Anreicherung und Zuordnung von Personen, Rollen, Orten, Organisationen und Ereignissen.

- `mentioned_places_from_custom_data ( )`

- `extract_and_prepare_persons ( )`
- `assign_roles_to_known_persons ( )`
- `match_organization_entities ( )`
- `extract_events_from_xml ( )`
- `combine_dates ( )`
- `assign_sender_and_recipient_place ( )`

### 5.2.4 Deduplikation und Validierung

Zweck: Zusammenführung mehrfach erkannter Entitäten und finale Konsistenzprüfung.

- `deduplicate_and_group_persons ( )`
- `ensure_author_recipient_in_mentions ( )`
- `count_mentions_in_transcript_contextual ( )`
- `postprocess_roles ( )`
- `mark_unmatched_persons ( )`
- `validate_extended ( )`

#### JSON-Export und Logging

Zweck: Erstellung der finalen JSON-Dateien im gewünschten Basisschema und Protokollierung von problematischen Einträgen.

- Erstellung von `BaseDocument ( )`
- `doc.to_json ( )`
- `update_total_json ( )`
- `log_unmatched_entities ( )`
- Terminalausgabe bei Validierungsfehlern

JSon Export weil menschenlesbar und leichte Abwandelbarkeit in andere Formate.

### 5.2.5 Review-Prozess

Zweck: Markierung und Protokollierung unsicherer, unvollständiger oder nicht eindeutig gematchter Entitäten für eine spätere manuelle Überprüfung.

- `mark_unmatched_persons ( )` – Kennzeichnung von Personen ohne ID, mit niedrigem Score, unklarem Namen
- `needs_review = true` bei allen problematischen Einträgen
- `review_reason` zur Beschreibung der Ursache (z.B. „nur Vorname“, „nicht in Groundtruth“)
- `log_unmatched_entities ( )` – Protokollierung in den Dateien:
  - `unmatched_persons.json`
  - `unmatched_places.json`
  - `unmatched_roles.json`
  - `unmatched_events.json`
- Kontextbasierte Filterung durch Zeilenumfeld (z.B. keine Dopplung bei Rolle+Name in direkter Nachbarschaft)

**DAS HIER IST EIN ALTER ABSCHNITT, DER FÜR DIE OBEN ZU ERGÄNZENDEN KAPITEL VERWENDET WERDEN SOLL**

---

Die wesentliche Verarbeitung der durch ChatGPT verarbeiteten XML-Files für jede einzelne Seite wird im Hauptmodul `Transkribus_to_base.py` gesteuert. Es ist das umfangreichste Modul für dieses Projekt, dessen Funktionsweise im Folgenden beschrieben werden soll.
Nach Abschluss der Vorverarbeitung und der Anreicherung mit Annotationen werden die XML-Dateien mithilfe des Moduls `transkribus_to_base.py` in eine strukturierte JSON-Repräsentation überführt. Diese stellt das im Projekt definierte Basisschema dar und dient als Grundlage für die nachfolgenden Analyseschritte. Ziel ist es, aus dem strukturierten und annotierten XML-Dokument ein validiertes JSON-Objekt zu erzeugen, das alle im Dokument erkannten Entitäten eindeutig, formal konsistent und datenmodellkonform beschreibt.
Das Modul `transkribus_to_base.py` ist als zentraler Verarbeitungsknoten konzipiert. Es verarbeitet die Inhalte der zuvor erzeugten XML-Dateien schrittweise, prüft und transformiert sie und strukturiert sie in einer einheitlichen Objektklasse ( `BaseDocument` ). Die Verarbeitung beginnt mit dem Einlesen der XML-Datei. In einem ersten Schritt werden aus dem XML-Header Informationen wie `docId` , `pageId` , `tsid` sowie Referenzen zu Bild- und Quelldateien extrahiert. Diese Metadaten bilden die Basis für die eindeutige Identifikation jeder Seite. Zusätzlich wird aus dem Dateinamen das Dokumentformat (z.B. Brief, Postkarte, Protokoll) abgeleitet.

Dieses wird später im Feld `document_type` gespeichert und dient der Klassifikation innerhalb der Datenstruktur.

Parallel dazu wird der vollständige Transkriptionstext aus den `<TextEquiv>` bzw. `<Unicode>`-Blöcken extrahiert. Dieser Text bildet die Grundlage für alle heuristischen, regelbasierten und modellgestützten Erkennungsverfahren. Die im vorherigen Schritt von ChatGPT annotierten `custom`-Attribute werden nun systematisch ausgelesen und auf ihre Struktur analysiert. Dabei werden Personen, Rollen, Orte, Organisationen und Daten extrahiert. Jede dieser Kategorien wird durch eine eigene Funktionsgruppe behandelt, die intern auf vorab geladene Groundtruth-Daten zurückgreift. Die Groundtruth-Dateien stammen aus *Nodegoat* und werden projektweit als CSV-Dateien verwaltet.

Die Extraktion von Personen erfolgt über die Funktion `extract_person_from_custom()`, die für jeden in der XML-Datei annotierten `person`-Tag eine initiale Zerlegung vornimmt. In einem mehrstufigen Matchingverfahren wird versucht, die extrahierten Namen mit bekannten Personen zu verknüpfen. Dabei kommen Fuzzy-Matching-Techniken zum Einsatz, die über die Funktion `match_person()` gesteuert werden. Zusätzlich werden Titel wie „Herr“, „Frau“, „Sängerbruder“ oder „Witwe“ als Geschlechtsindikatoren erkannt und gespeichert. Für jede identifizierte Person wird ein Eintrag erzeugt, der sowohl die extrahierten als auch die gematchten Informationen enthält. Bei fehlender Übereinstimmung wird der Eintrag mit dem Vermerk `needs_review` gekennzeichnet.

Die Ortsverarbeitung basiert auf einem spezialisierten `PlaceMatcher`-Objekt, das die extrahierten Ortsnamen mit bekannten Ortsbezeichnungen aus der Groundtruth sowie mit externen Ressourcen wie Geonames oder Wikidata abgleicht. Bei unklaren oder mehrdeutigen Ortsangaben kann der Matcher mehrere Kandidaten zurückgeben. In diesem Fall erfolgt eine Gewichtung anhand von Konfidenz- und Ähnlichkeitswerten. Die Funktion `extract_place_from_custom()` ist dabei für die Initialextraktion zuständig, während die Funktion `deduplicate_places()` eine Zusammenführung ähnlicher Ortsangaben durchführt.

Zusätzlich zu den durch das Sprachmodell erzeugten Custom-Tags werden weitere Entitäten heuristisch aus dem Fliesstext erkannt. Besonders betrifft dies Rollenbezeichnungen, die in unmittelbarer Nähe zu Personennamen vorkommen. Eine regelbasierte Extraktion dieser Kontexte wird durch die Funktion `assign_roles_to_known_persons()` realisiert. Auch hier wird das Ergebnis validiert und – sofern die Rolle einer standardisierten Ontologie entspricht – in das Feld `role_schema` überführt.

Die Kombination der verschiedenen Erkennungsmethoden kann zu Duplikaten führen. Um konsistente und eindeutige Entitäten zu erzeugen, erfolgt ein deduplizierender Abgleich über die Funktion `deduplicate_and_group_persons()`. Diese vergleicht alle Personen aus den Kategorien `authors`, `recipients` und `mentioned_persons` untereinander. Dabei werden vorhan-

dene Scores (wie `match_score`, `recipient_score` und `confidence`) zusammengeführt und priorisiert.

Das so angereicherte Dokument wird in ein Objekt der Klasse `BaseDocument` überführt. Dieses enthält strukturierte Felder für Metadaten, Volltext, Autoren, Empfänger, erwähnte Personen, Orte, Organisationen, Datumsangaben und Ereignisse. Jede Entität wird gemäss den Typdefinitionen in `document_schemas.py` validiert. Ein abschliessender Validierungsschritt erfolgt über `validate_extended()`, das auf Fehler in der Struktur, Inkonsistenzen oder fehlende Pflichtfelder prüft.

Abschliessend wird das Dokument im JSON-Format gespeichert. Neben der Einzelseite wird auch eine aggregierte Datei `total_json.json` fortgeschrieben, in der alle Seiten einer Akte gesammelt werden. Zusätzlich wird für jede Seite eine Prüfung auf nicht zuordenbare Entitäten durchgeführt. Diese werden in der Datei `unmatched.json` gespeichert, um eine spätere manuelle Nachbearbeitung zu ermöglichen. Das Ergebnis dieser Konvertierung bildet die Grundlage für die weitere Verarbeitung in `Nodegoat` sowie für die explorative Analyse der Akteursnetzwerke.

---

## 5.3 Module im Detail

### 5.3.1 document_schemas.py

Das Modul `document_schemas.py` definiert die zentrale Schema- und Datenstruktur zur Modellierung aller extrahierten Inhalte aus den Transkribus-Dokumenten des Projekts. Es gewährleistet die einheitliche Repräsentation, Serialisierung und Validierung der im Projekt verarbeiteten Entitäten und ihrer Relationen. Die definierten Klassen bilden die Grundlage für die JSON-Ausgabe der angereicherten Dokumente und dienen zugleich der Nachvollziehbarkeit und strukturierten Weiterverarbeitung in externen Anwendungen wie Nodegoat.

#### *Person*

Die Klasse `Person` bildet Einzelpersonen ab, wie sie in den Briefen, Postkarten oder Protokollen erscheinen. Neben klassischen Namensfeldern (`forename`, `familyname`, `title`) unterstützt die Klasse auch alternative Namen (`alternate_name`) und Rolleninhalte. Die Rollenverarbeitung erfolgt in einem zweistufigen Verfahren: Zum einen wird die Freitextrolle (`role`) als Originaleintrag gespeichert, zum anderen erfolgt eine normierte Zuordnung über `role_schema`, basierend auf dem internen Mapping in `Assigned_Roles_Module.py`.

Titel wie `"Herr"` oder `"Frau"` dienen in `person_matcher.py` zugleich als Grundlage zur Ableitung des Geschlechts der Person. Das Feld `gender` erlaubt in `document_schemas.py` eine geschlechtsspezifische Zuordnung, basierend auf Titelbezeichnungen oder Groundtruth-Matching.

Ergänzt wird die Person um Kontexte wie `associated_place` und `associated_organisation`, sowie um Bewertungsparameter wie `match_score`, `recipient_score`, `confidence` und `mentioned_count`.[88] Letzterer gibt an, wie oft die Person im Transkript erwähnt wurde, unter Berücksichtigung einer kontextbasierten Zähllogik. `match_score` beschreibt einen Wert der Wahrscheinlichkeit, dass es sich bei der im Dokument beschriebenen Person auch um die Person handelt, die schlussendlich gematcht wurde. Vor- und Nachnamen erhöhen beispielsweise den Score, während das Fehlen solch einzelner Informationen den Wert senken.

Das Feld `confidence` beschreibt die Modell-Sicherheit der Zuordnung, z.B.beim LLM-basierten Matching. `recipient_score` bewertet die Wahrscheinlichkeit, dass es sich bei der Person um den tatsächlichen Empfänger des Dokuments handelt.

Ein optionales `needs_review`-Flag sowie ein `review_reason` ermöglichen die gezielte Markierung unsicherer oder maschinell nur teilweise auflösbarer Fälle.

### *Organization*

Die Klasse `Organization` dient der Abbildung aller im Text genannten Vereine, Gruppen oder Institutionen, darunter z.B. Gesangsvereine, NS-Organisationen. Neben dem Namen, Typ und eventuellen Alternativnamen (`alternate_names`) wird insbesondere die eindeutige `nodegoat_id` gespeichert. Für den Spezialfall militärischer Einheiten enthält die Klasse ein separates Feld `feldpostnummer`. Da sie in der Pipeline jedoch nicht seperat getaggt und verarbeitet sind, werden diese aktuell ausschliesslich über die Groundtruth abgerufen. Über `match_score` und `confidence` werden auch in der Klasse `Organization` Qualität und Unsicherheiten der Zuordnung nachvollziehbar gemacht. Organisationen können zusätzlich über das Feld `associated_place` mit einem geographischen Ort verknüpft werden, z.B. „Männerchor Murg“ mit „Murg“.

### *Place*

Die Klasse `Place` strukturiert geographische Orte, wie sie z.B. als Absender-, Empfänger- oder Veranstaltungsorte im Dokument auftreten. Unterstützt werden neben der Hauptbezeichnung (`name`) auch alternative Formen (`alternate_place_name`), sowie standardisierte Identifikatoren aus Geonames, Wikidata und Nodegoat. Diese Orte sind über eigene Felder sowohl im Metadatenblock (`creation_place`, `recipient_place`) als auch in der Liste `mentioned_places` verortet, falls erstere nicht genau zugeordnet werden können. Wie bei Personen, kann auch bei Orten über das Flag `needs_review` eine manuelle Überprüfung unsicherer Matches angestossen werden.

[88] Detailliert wird auf diese Logiken in Person_matcher.py eingegangen, sie sollen hier nur umrissen werden.

### *Event*

Ereignisse werden über die Klasse `Event` abgebildet. Diese enthält einen Namen, eine optionale Beschreibung, Datumsangaben und Referenzen auf beteiligte Personen, Orte und Organisationen. Über das Feld `inferred` wird angegeben, ob das Ereignis direkt im Text genannt oder aus dem Kontext erschlossen wurde. Die genaue Event-Extraktion findet sich in event_matcher.py.

### *BaseDocument*

Alle genannten Entitäten werden im zentralen Objekttyp `BaseDocument` zusammengeführt. Diese Klasse bildet die Grundlage für die strukturierte Speicherung jedes einzelnen Dokuments und enthält unter anderem:

- einen Attributblock mit `document_type`, `object_type`, `creation_date`, `creation_place`, `recipient_place`
- Listen der `authors`, `recipients` und `mentioned_persons`
- Listen von `mentioned_organizations`, `mentioned_places`, `mentioned_events` und `mentioned_dates`
- die Originaltranskription (`content_transcription`) sowie inhaltliche Tagging-Kategorien (`content_tags_in_german`)
- optionale Zusatzinformationen im Feld `custom_data`, z.B.Zwischenoutputs oder Debug-Daten

Zur automatisierten Konvertierung vom oder ins JSON-Format stehen die Methoden `to_dict()`, `from_dict()`, `to_json()` und `from_json()` zur Verfügung. Darüber hinaus erlaubt die Methode `validate()` eine strukturierte Prüfung der Einträge auf Konsistenz und Vollständigkeit, etwa im Hinblick auf unvollständige Daten oder ungültige Formate.

### *Documenttype*

Für die wichtigsten Dokumenttypen existieren spezialisierte Unterklassen wie `Brief`, `Postkarte` oder `Protokoll`, die zusätzliche Felder (z.B. `greeting`, `postmark`, `meeting_type`) aufnehmen. Diese können über die zentrale sogenannte „factory function“ `create_document()` automatisch erzeugt werden, wenn ein Dokumenttyp erkannt wurde. Diese spezialisierten Erkennungstypen stehen aktuell lediglich für eine spätere Verwendung bereit, und kommen in der aktuellen Anwendung nur in der Benennung des jeweiligen Typs zum Einsatz.

Alle Ergebnisse werden anschliessend in strukturierter Form in eine JSON-Datei geschrieben, wobei jeder Eintrag dem oben definierten Schema entspricht.

### 5.3.2 ___init___.py

Das Modul `__init__.py` fungiert als zentrale Importschnittstelle für alle Funktionalitäten der Projektpipeline. Es aggregiert sämtliche zentralen Komponenten aus den verschiedenen Teilmodulen (z.B. `person_matcher.py`, `document_schemas.py`, `place_matcher.py`) und stellt sie über das `__all__`-Array einheitlich zur Verfügung. Dadurch wird eine übersichtliche, modulübergreifende Nutzung der wichtigsten Funktionen und Klassen in anderen Programmteilen (z.B. in `transkribus_to_base.py`) ermöglicht. Auf diese Weise kann das Hauptprogramm `transkribus_to_base.py` auf alle notwendigen Komponenten mit einem einzigen Importbefehl zugreifen, ohne die internen Modulpfade kennen zu müssen. Die Datei übernimmt damit die Funktion einer projektinternen API und gewährleistet eine saubere Trennung zwischen interner Modulstruktur und externer Nutzung.

### 5.3.3 Person_matcher.py

Die Erkennung und das Matching von Personen stellt bei weitem die grösste Herausforderung in diesem Projekt dar. Wie vorausgehend bereits angedeutet, gibt es eine Vielzahl von Möglichkeiten, eine Person in einem Text zu benennen oder nur Anzudeuten, was ein zuverlässiges Matching erschwert. Oft ist der Kontext relevant um Personen eindeutig zu identifizieren. Hinzu kommen ganz praktische Probleme. Ein Stringabgleich ist beispielsweise kasus- und genussensitiv. Bei einer Match-Prüfung müssen die

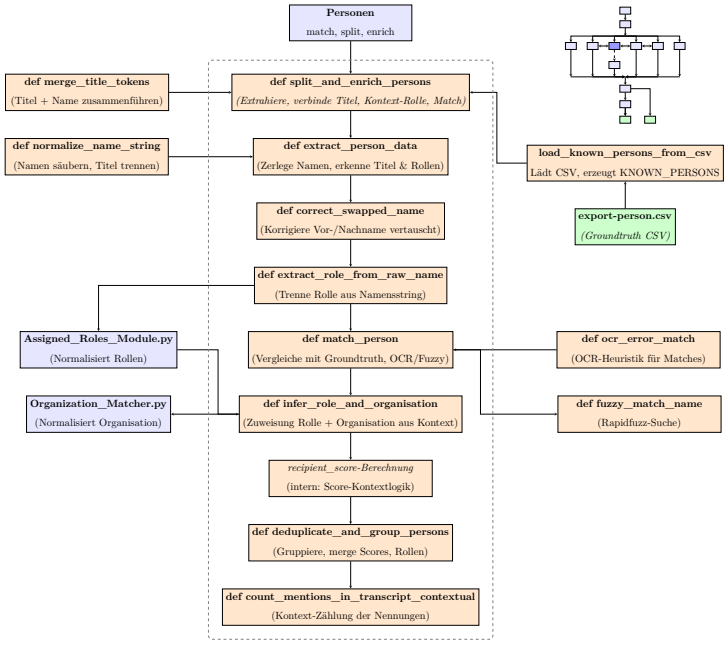



Abbildung 8:
Oben links: Prozessdiagramm für `Personen_matcher.py`,
Oben rechts: Pipelineübersicht

Algorithmen daher ein Wort im Nominativ Maskulinum genau so erkennen wie im Akkusativ Femininum. Daraus ergibt sich für den Person_Matcher allein auf heurisitscher Ebene eine grosse Komplexität, die im Folgenden lediglich andeutungsweise dargelegt werden kann.

Im Laufe dieser Arbeit sind hierfür vier Parameter entwickelt worden, die diese Zuordnung ermöglichen sollen: **Vorname, Nachname, Rolle, Geschlecht**
Der Aufbau des Moduls gliedert sich wie folgt:

1. Blacklist & Configuration
2. Initialisierung
3. Namen- und Titelerkennung
4. Levenshtein-Fallback
5. Fuzzy-Matching
6. Matching
7. *Extract Person Data* mit Rolleninfos
8. Split und Enrichment
9. Deduplication
10. Detail-Info zum besten Match

Die nachfolgende Erläuterung des Moduls orientiert sich eng an dessen Aufbau im Quellcode, um die interne Struktur und Funktionalität möglichst transparent nachzuvollziehen.

### *Blacklist & Configuration*

In der Initialisierung des Moduls `person_matcher.py` werden neben der Groundtruth-Tabelle `export-person.csv` eine Vielzahl domänenspezifischer Konfigurationslisten geladen, um die Erkennung und Validierung von Personennamen zu unterstützen. `NICKNAME_MAP` liefert beispielsweise ein Dictionary mit üblichen deutschen Spitznamen.[89] Diverse Blacklists statuieren Worte, die beispielsweise nicht menschlich sind (bsp. `"freundliche Grüsse"`). Zu den Blacklists zählt auch `Unmatchable_Single_Names`, die auf nicht eindeutig verknüpfbare Namen hinweist.[90] `Pronoun` und nach Geschlecht zusammengefasste `Title_Tokens`[91] markieren Tokens zur gezielten Weiterverarbeitung in nachgelagerten Matching- und Anreicherungsschritten. Alle Ausschlusslisten werden in `NON_PERSON_TOKENS` zusammengeführt und dienen der Negativselektion im gesamten Matchingprozess.

### *Thresholds, CSV-Laden und Groundtruth-Erstellung*

FÜr das Matching werden zwei essentielle Thresholds definiert, die erreicht werden müssen. für `forename` muss eine 80% Ähnlichkeit, für `familyname` eine 85% Ähnlichkeit erreicht werden. Diese Zahlen ergeben sich durch unterschiedliche Tests. Sie bieten einen Kompromiss aus Toleranz gegenüber beispielsweise OCR-Fehlern heraus, und liefern dennoch präzise Trefferquoten. Alle Personendaten werden als Liste von Dictionaries[92] aus der Groundtruth geladen. Sie wer-

[89] Durch ChatGPT generiertes Mapping geläufiger deutscher Vor- und Kosenamen auf ihre kanonische Form, einige Fälle sind händisch ergänzt

[90] `"Otto"` oder `"Döbele"` sind so häufig vorkommende Namen, dass sie alleine nicht für ein Matching ausreichen

[91] `TITLE_TOKENS = MALE_TITLE_TOKENS | FEMALE_TITLE_TOKENS | NEUTRAL_TITLE_TOKENS`

[92] `List[Dict[str, str]]`

den in die Variablen `KNOWN_PERSONS`, `GROUNDTRUTH_SURNAMES` und `GROUNDTRUTH_FORENAMES` übergeben, um später gegen die genannten Thresholds zu testen.

### *Namensnormalisierung und Titelerkennung*

Für eine konsistente Weiterverarbeitung werden alle erkannten Namenstrings standardisiert, und auf mögliche Rollenstrings überprüft, weil Rollen oft durch das LLM als Person mit markiert werden. Daher durchlaufen die Namestrings zwei Normalisierungen (normalize_name_string und normalize_name). Erstere bereinigt den Text grob und trennt Titel vom Rest. Da in den Quellen Initialen oft vorkommen[93], werden zwischen den beiden Funktionen ebenfalls auf eine Initiale geprüft. Die Funktion `normalize_name()` extrahiert dann eventuell vorhandene Titel mithilfe eines dynamisch aus `TITLE_TOKENS` generierten regulären Ausdrucks. Anschliessend werden Klammern entfernt, Unicode-Zeichen vereinheitlicht (NFKD-Normalisierung), Diakritika[94] gelöscht und Sonderzeichen bereinigt. Die verbleibenden Namensbestandteile werden in Vor- und Nachname aufgeteilt. Dabei berücksichtigt die Funktion auch Nickname-Mappings, indem erkannte Kurzformen wie `Hansi` automatisch in ihre kanonische Form `Johannes` überführt werden. Enthält der Name nur ein Token, wird dieses als Vorname gewertet; andernfalls erfolgt eine Trennung in `forename` und `familyname`. Das Ergebnis ist ein Dictionary mit den Feldern `title`, `forename` und `familyname`, die in späteren Matching- und Review-Schritten verwendet werden. So wird mit `normalize_name_string` aus `"Dr. Hansi Müller"` → `forename:johannes, lastname: müller, title: "Dr"`

### *Levenshtein-Fallback*

Nach der Namensnormalisierung folgen zwei zentrale Funktionen zur Fehlerkorrektur und Heuristik-basierter Umkehrung von Namensbestandteilen.

Die Funktion `ocr_error_match()` verwendet die Levenshtein-Distanz zur Berechnung der Ähnlichkeit zwischen einem gegebenen Namen und einer Liste möglicher Kandidaten, um einzelne Tokens gegen bekannte Groundtruth-Namen zu prüfen. Dabei wird der Eingabename `name` zunächst in Kleinschreibung normalisiert (`name_lower`), ebenso wie jeder Kandidat aus der Vergleichsliste (`cand_lower`). Obwohl beide Seiten formal gleich behandelt werden, liegt der Unterschied in ihrer Herkunft: `name_lower` entstammt der unkorrigierten Texterkennung und kann daher auch Schreibfehler beinhalten, während `cand_lower` aus kuratierten Groundtruth-Listen stammt und damit als Referenz dient. Die Distanz zwischen beiden wird per `Levenshtein.distance()` berechnet; der beste Treffer mit der geringsten Distanz wird als Ergebnis zurückgegeben, ergänzt um einen prozentualen Score.

Darauf aufbauend prüft `correct_swapped_name()`, ob Vor- und Nachname im gegebenen

[93] Bsp.:`"A. Zimmermann"`

[94] Sonderbuchstaben, Bsp: `ä, ê, š, ç`

String möglicherweise vertauscht wurden. Dazu wird jeweils ermittelt, ob der Vorname einem bekannten Nachnamen entspricht oder umgekehrt. Neben direktem Lookup in `KNOWN_FORENAMES` und `KNOWN_SURNAMES` kommt hier ebenfalls ein Levenshtein-Fallback zum Einsatz, um kleine Abweichungen zu tolerieren. Wird auf diese Weise erkannt, dass eine Umkehrung wahrscheinlicher ist, wird das Tupel entsprechend getauscht zurückgegeben.

### *Fuzzy-Matching*

Um Namensähnlichkeiten robust zu erkennen, implementiert `fuzzy_match_name()` ein Fuzzy-Matching-Verfahren auf Basis des `fuzz.ratio()`-Scores. Für jedes Vergleichspaar (`name` / `candidate`) wird anschliessend ein Ähnlichkeitswert berechnet, und der beste Treffer bestimmt. Liegt der höchste Score über dem anfangs definierten Threshold, wird der entsprechende Kandidat samt Score zurückgegeben; andernfalls wird `None` geliefert. Dieses Verfahren ergänzt die Levenshtein-basierte Fehlerbehandlung um eine robustere, Token-basierte Ähnlichkeitsmessung, die auch bei Namensvarianten und OCR-Fehlern zuverlässige Resultate liefert.

### *Matching*

Der `Person_Matcher` verwendet nun die vorverarbeiteten Namensstrings als `person`, und die wahrscheinlichen Personen der Groundtruth als `candidate`. Gleich zu Beginn wird der

*Sonderfall*

*"nur Rolle, keine Namen"* geprüft. Wenn die Person nur eine Rolle, aber keinen Namen hat, wird ein Minimal-Objekt mit niedrigem Score zurückgegeben. Zudem erhält das Objekt dann den `Review-Reason`:`"Nur Rolle ohne vollständigen Namen erkannt"`.

*Inhalts-Filter*

Es folgt zur Absicherung eine Normalisierung der Roh-(Vor-)Namen, und ein Check, ob die Strings in den Blacklists (`Pronoun_Tokens`, `Role_Tokens`, `Non_Person_Tokens` vorkommen, um sie frühzeitig auszuschliessen. Diese Fälle liefern für `Match:None, 0`

*Unique-Name*

Wenn nur der Vor- oder Nachname vorhanden ist, wird geprüft, ob der Name eindeutig in der Groundtruth vorkommt. Falls ja, wird sofort ein Match ausgegeben.

*Blacklist-Checks*

Hier werden problematische Tokens vor dem eigentlichen Matching identifiziert und blockiert. Getestet werden `Non_Person_Tokens` und `Role_Tokens` in zwei if-Blöcken und deren Vorkommen in Vor- oder Nachnamen. Im zweiten Fall (`if not ln and fn.lower() in ROLE_TOKENS`) wird erkannt, ob ein Rollenbegriff fälschlich als Vorname (`fn`) identifiziert wurde. In beiden Fällen wird das Matching abgebrochen, die Person mit `match_score = 0` und `confidence = "blacklist"` markiert sowie zur manuellen Nachprüfung

( `needs_review = True` ) ausgegeben.

*Initialen-Check*

Falls der Vorname `fn` lediglich eine Initiale darstellt, wird mit `is_initial(fn)` versucht, einen passenden Eintrag in `candidates` zu finden. Dazu wird die Initiale ( `init = fn[0].upper()` ) mit dem ersten Buchstaben des Vornamens jedes Kandidaten verglichen und zusätzlich geprüft, ob der normalisierte Nachname ( `ln` ) mit demjenigen des Kandidaten übereinstimmt. Wird ein solcher Eintrag gefunden, wird er mit hoher Übereinstimmung ( `match_score = 95` ) zurückgegeben.

*Fuzzy Matching über Vornamen und Nachnamen*

Liegt ein Vorname `fn` mit mindestens drei Zeichen sowie ein Nachname `ln` vor, wird zunächst versucht, anhand eines Teilabgleichs zu matchen: Der Anfang des gegebenen Vornamens ( `fn` ) muss mit dem Kandidatennamen übereinstimmen, während der Nachname exakt übereinstimmt ( `normalize_name_string(ln)` ). In diesem Fall wird der Treffer mit einem Score von `89` zurückgegeben.

Zusätzlich erfolgt eine *Reverse-Prüfung*, um mögliche Vertauschungen von Vor- und Nachname zu erkennen. Dabei wird geprüft, ob die normalisierten ersten vier Zeichen des eingegebenen Nachnamens ( `ln` ) mit dem Vornamen eines Kandidaten übereinstimmen und gleichzeitig die normalisierten ersten vier Zeichen des eingegebenen Vornamens ( `fn` ) mit dessen Nachnamen. Voraussetzung ist ausserdem, dass nicht bereits eine Übereinstimmung von `fn` und `ln` mit den Feldern `forename` und `familyname` des Kandidaten vorliegt. Wird diese Bedingung erfüllt, werden die Namensfelder im Rückgabeobjekt vertauscht und der Match mit einem reduzierten Score von `88` zurückgegeben.

*Levenshtein-Fallback bei Namensvertauschung*

Zunächst wird geprüft, ob der normalisierte Vorname `fn` exakt mit dem eines Kandidaten übereinstimmt und der Nachname eine Levenshtein-Distanz von höchstens 1 zum Nachnamen des Kandidaten aufweist. In diesem Fall wird ein Match mit Score `90` zurückgegeben.

Anschliessend erfolgt eine Reverse-Prüfung. Dabei wird getestet, ob der Nachname `ln` mit dem Vornamen eines Kandidaten übereinstimmt und der Vorname `fn` dem Nachnamen des Kandidaten bis auf eine maximale Levenshtein-Distanz von 1 entspricht. Das soll absichern, dass vertauschte Namensbestandteile auch mit geringen OCR-Fehlern korrekt gematched werden.

*Matching über Einzelkomponenten*

Dieser Abschnitt deckt Fälle ab, in denen nur unvollständige Namensinformationen vorliegen. Ist lediglich ein Nachname vorhanden, wird ein fuzzy-Matching gegen alle Kandidaten-Nachnamen mittels Fuzzy-Matching durchgeführt; erreicht ein Kandidat den konfigurierten Schwellenwert ( `thr["familyname"]` ), wird dieser mit dem berechneten Score übernommen. Liegen weder Vor-

noch Nachname vor, aber eine Rollenbezeichnung (`role_raw`), erfolgt ein Abgleich mit `role` und `alternate_name` der Kandidaten; bei Übereinstimmung wird ein Treffer mit Score `80` zurückgegeben.

Zusätzlich werden Namensbestandteile, die als generisch oder nicht personenspezifisch klassifiziert sind (`NON_PERSON_TOKENS`, speziell behandelt. Ist der Vorname ein solcher unerwünschter Token, aber ein Nachname vorhanden, wird dieser gegen alle `familyname`-Felder geprüft (Score `85`); umgekehrt wird bei generischem Nachnamen der Vorname geprüft. Als letzte Rückfallebene kommt `ocr_error_match()`[95] zum Einsatz: Ist nur ein Nachname vorhanden, wird dieser mit einer Toleranz von bis zu zwei Zeichen mit den Nachnamen aller Kandidaten verglichen; bei Übereinstimmung wird der entsprechende Treffer mit Score `85` übernommen. Alle Varianten markieren strukturunscharfe, aber potenziell gültige Entitäten.

*Matching über Nachname und Gender*

Dieser Abschnitt implementiert eine gender-basierte Disambiguierung, die dann greift, wenn der Nachname mit mehreren Personen übereinstimmt, aber das Geschlecht bekannt oder ableitbar ist. Zunächst wird versucht, dieses anhand von Titeln wie `"Frau"` oder `"Herr"` zu bestimmen. Ist anschliessend sowohl ein Nachname als auch ein valides Geschlecht verfügbar, werden alle Kandidaten mit identischem Nachnamen gesammelt und auf Übereinstimmung des Geschlechts gefiltert. Existiert genau ein solcher Treffer, wird dieser mit hoher Konfidenz (`confidence = "gender_ln_match"`) und einem Score von `95` zurückgegeben, ohne dass ein manueller Review erforderlich ist. Das Matching über Geschlecht und Nachname ist besonders wichtig, da Frauen in den Unterlagen bis auf sehr wenige Ausnahmen nie mit Vornamen genannt werden. Ohne den Geschlechtsabgleich würden sie mit (Ehe-)Männern gematched werden. Diese Funktion versucht also aktiv, gegen den Geschlechterbias in den historischen Unterlagen vorzugehen[96]

*Last Fallback - Aufnahme mit Review*

Falls mindestens eines der Felder `fn` (Vorname), `ln` (Nachname) oder `role_raw` (Rolle) vorhanden ist, aber kein reguläres Matching möglich war, wird ein Fallback-Objekt erzeugt und zur manuellen Nachprüfung markiert (`needs_review = True`). Die Funktion prüft, welche Felder fehlen, und protokolliert dies als `review_reason` (z. B. „missing_forename; missing_role“).

Eine Aufnahme erfolgt nur, wenn entweder der Vor- oder Nachname in einer Groundtruth-Liste vorkommt (`appears_in_groundtruth()`) oder eine Rollenbezeichnung vorhanden ist. In diesem Fall wird ein Personeneintrag mit `confidence = "partial-no-id"` und

[95] aus Levenshtein-Fallback

[96] Geschlechterbias tritt bei dieser Arbeit in diversen Formen auf, *Source Bias, Archival Bias, Model Bias, und Algorythic Bias*. Letzterer soll mit dieser Funktion abgeschwächt werden. Vgl.: Burkhardt 2025a.

`match_score = None` zurückgegeben. Andernfalls wird die Person vollständig verworfen.

### *Extract Person Data mit Rolleninfos*

Die Funktion `extract_person_data(row)` dient der strukturierten Aufbereitung einzelner Personeneinträge. Dafür holt sich die Funktion Informationen über die jeweilige Datenzeile ( `row` ) aus der `export-person.csv` . Ziel ist es, aus den Namensangaben und Metadaten ein standardisiertes Personenobjekt zu erzeugen, wie in document_schemas.py beschrieben.

Der übergebene Namensstring wird zu Beginn nach typischen Titeln durchsucht. Wird ein solcher Titel erkannt, wird er im Feld `title` gespeichert und aus dem Namensstring entfernt. Der verbleibende Name wird auf mögliche Rolleninhalte analysiert, die über `extract_role_from_raw_name()` aus dem Assigned_Roles_Module.py ermittelt werden. Hierbei wird beispielsweise der String „Schriftführer Ganter“ in die Bestandteile `role=Schriftführer` und `name=Ganter` aufgetrennt.

Anschliessend wird der bereinigte Namensstring in Einzelteile zerlegt. Das erste Token wird als Vorname, das letzte als Nachname interpretiert. Um typische OCR-Fehler oder Namensinversionen zu korrigieren, wird nochmal die Funktion `correct_swapped_name()` aufgerufen, welche Vor- und Nachnamen bei Bedarf vertauscht.

Für den Rollenteil des Eintrags wird zunächst das Feld `role` aus der ursprünglichen CSV-Zeile übernommen. Falls bereits beim Parsing eine Rolle erkannt wurde, wird diese priorisiert. Der erkannte Rollentext wird dann durch `normalize_and_match_role()` in eine standardisierte Form überführt. Über `map_role_to_schema_entry()` erfolgt anschliessend die semantische Abbildung auf ein normiertes Rollenschema[97].

Auch das Geschlecht der Person wird in diesem Schritt bestimmt. Dazu wird zunächst das übergebene Geschlechtsfeld bereinigt und vereinheitlicht. Sollte dieses keine gültige Angabe enthalten, wird als Fallback `infer_gender_for_person()` aufgerufen. Diese Funktion versucht, das Geschlecht über den Titel oder durch Abgleich mit der Groundtruth-Liste zu ermitteln.

Neben Namen, Titel, Rolle und Geschlecht werden auch zusätzliche Kontextinformationen aus der Groundtruth extrahiert, sofern vorhanden. Dazu zählen `alternate_name` , der Wohnort und weitere. So kann für jedes Dokument garantiert werden, dass alle verfügbaren Daten für eine gegebenenfalls nötige manuelle Prüfung vorhanden sind. Plausibilitätsprüfungen funktionieren so unmittelbar, ohne Zwischenschritte.

Die Funktion erzeugt abschliessend ein vollständiges Personenobjekt mit Attributen wie `confidence` , `match_score` und `needs_review` , die für spätere Matching- und Validierungsschritte relevant sind. Personen ohne Nodegoat-ID werden standardmässig zur manuellen Nachprüfung markiert.

[97] `role` ist der detektierte String, `role_schema` die normalisierte Form

Zur systematischen Dokumentation unvollständiger Einträge wird zusätzlich die Funktion `get_review_reason_for_person()` bereitgestellt. Sie prüft, ob für die betreffende Person kritische Felder wie Vorname, Nachname oder Rolle fehlen und erzeugt eine entsprechende Review-Kennzeichnung.

Für Sonderfälle, bei denen keine Namen, sondern lediglich Rollentitel wie `„Vereinsführer"` erkannt wurden, dient die Funktion `detect_and_convert_role_only_entries()`. Sie prüft, ob das einzige übergebene Token einem bekannten Rollenschema entspricht. Ist dies der Fall, wird eine strukturierte Rollenangabe ohne Namensinformation erzeugt und ebenfalls zur Überprüfung markiert.

Schliesslich wird in Zusammenarbeit mit dem `Assigned_Roles_Module` mit `extract_metadata_names()` eine Funktion bereitgestellt, um auch aus dem Fliesstext zusätzliche Personenhinweise zu extrahieren. Hierbei kommen wieder RegEx zum Einsatz, die typische Anredeformen oder Signaturen am Briefende erkennen und zur weiteren Analyse aufbereiten.

#### *Split und Enrichment*

#### *Deduplication*

#### *Detail-Info zum besten Match*

### 5.3.4 Assigned_Roles_Module.py

Hier steht eine Menge schöner Text

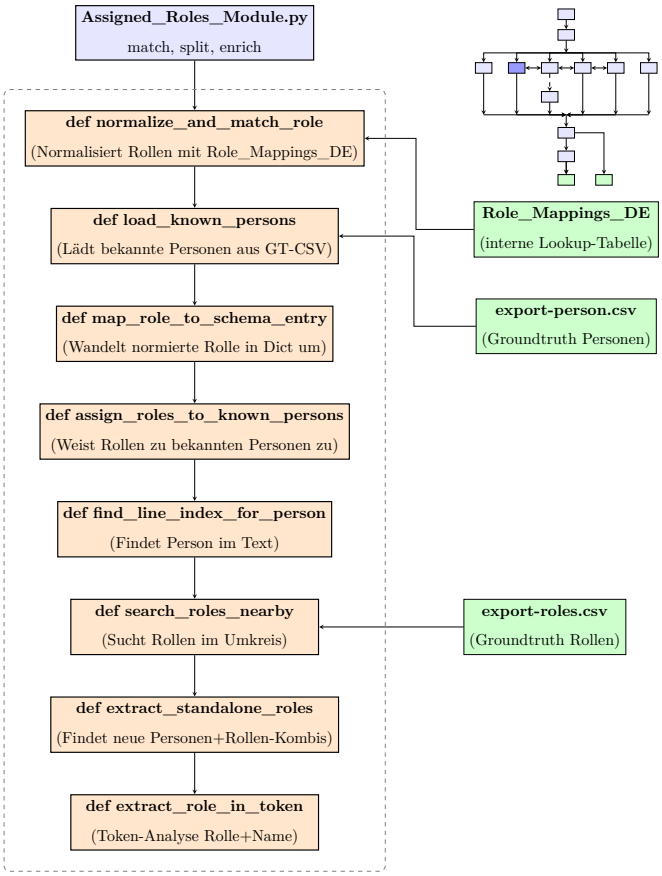



Abbildung 9:
Links: Prozessdiagramm für `Assigned_Roles_Module.py`,
Rechts: Pipelineübersicht

### 5.3.5 place_matcher.py

Hier steht eine Menge schöner Text Ortsnamenserkennung basiert stark auf GT, beispielsweise Murg 26 unterschiedliche Alternativnamen, daher Code vergleichsweise kurz

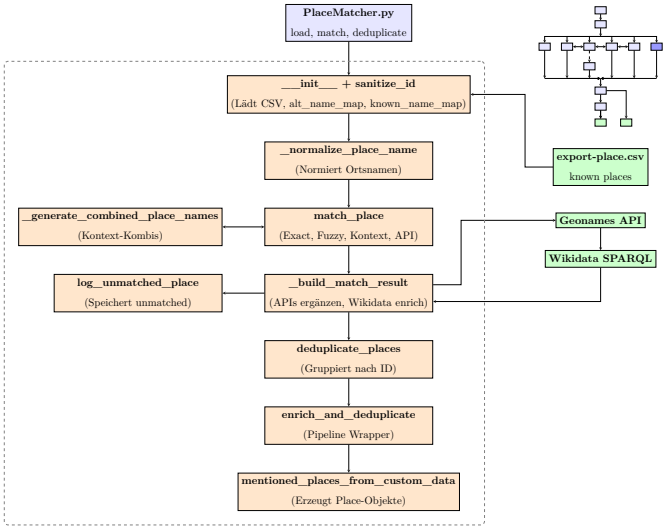



Abbildung 10:
Links: Prozessdiagramm für `place_matcher.py`,
Rechts: Pipelineübersicht

### 5.3.6 organization_matcher.py

`organization_matcher.py` dient der Erkennung und Normalisierung von Organisationen in historischen Transkripten. Es kombiniert reguläre Ausdrücke mit unscharfem Vergleich (fuzzy matching), um eingegebene Strings mit bekannten Organisationen aus der Groundtruth-Datei `export-organisationen.csv` abzugleichen.

`extract_organization({})` bereinigt im nächsten Schritt die Eingabestrings zunächst, indem

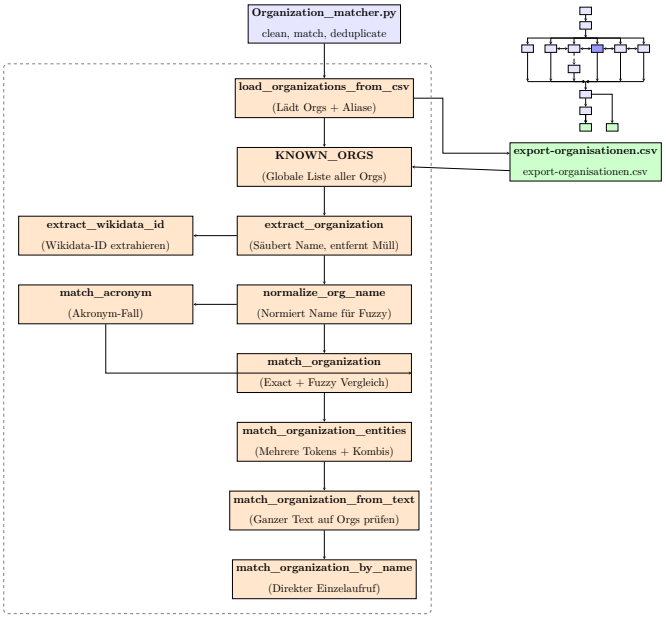



Abbildung 11: Links: Prozessdiagramm für `Organization_matcher.py`,
Rechts: Pipelineübersicht

sie umschliessende Klammern entfernt, innere Satzzeichen säubert und führende bzw. abschliessende Interpunktion streicht. Darüber hinaus wird eine Blacklist abgeglichen, um etwa Einträge wie „Verein“ als alleinstehende Organisation auszuschliessen - ein klares Match kann hier algorithmisch vorerst nicht gewährleistet werden.

Die bekannten Organisationen werden über die Funktion `load_organizations_from_csv()` eingelesen. Dabei entsteht eine strukturierte Liste von Dictionaries, die mit den Anforderungen aus `document_schemas.py` kompatibel ist und Hauptnamen, alternative Bezeichnungen sowie

Identifier enthält.

Um die Organisationen vergleichbar zu machen wird in `def normalize_org_name ( )` in drei schritten normalisiert.

Zunächst werden mithilfe einer RegEx[98] sämtliche Zeichen entfernt, die nicht alphanumerisch sind. Erlaubt bleiben dabei Gross- und Kleinbuchstaben (einschliesslich deutscher Umlaute), Ziffern sowie Leerzeichen. So wird beispielsweise aus dem Ausdruck „Männerchor Murg e.V.“ der bereinigte String „Männerchor Murg eV“.

Im zweiten Schritt wird der bereinigte String durch `.lower()` vollständig in Kleinbuchstaben transformiert. Abschliessend werden mit `.strip()` etwaige führende oder nachfolgende Leerzeichen entfernt, sodass der Rückgabewert für weitere Vergleichsoperationen normiert vorliegt. Verdeutlichend wird aus:

```
Input:  "  (Männerchor Murg e.V.)  "
Output: "männerchor murg ev"
```

Um nicht nur vollständige Namen zu prüfen, sondern auch Akronyme[99], wird gezielt nach solchen in den `alternate_names` bekannter Organisationen gesucht.

Das Matching der Organisationen erfolgt im Anschluss in den drei zentralen Funktionen `match_organization()`, `match_organization_from_text()` und `match_organization_entities()`.

Die Funktion `match_organization()` bildet dabei den Kern des Vergleichsprozesses. Sie prüft zunächst, ob es sich bei der Eingabe um ein kurzes Akronym handelt und nutzt hierfür die Funktion `match_acronym()`. Falls kein solcher Sonderfall vorliegt, wird der Eingabewert mithilfe von Fuzzy Matching mit `fuzz.ratio` gegen alle bekannten Organisationsnamen sowie deren alternative Bezeichnungen geprüft. Das Ergebnis ist das bestbewertete Match mit einem Score oberhalb eines standardmässig auf vordefinierten Schwellenwerts .

Die Funktion `match_organization_from_text()` erweitert diesen Vergleich auf vollständige Transkriptionstexte. Dabei wird zunächst überprüft, ob ein bekannter Organisationsname direkt im Text vorkommt. Falls dies nicht der Fall ist, wird ein fuzzy-Teilvergleich durchgeführt (`fuzz.partial_ratio`), um auch unvollständige oder abweichende Schreibweisen zu erkennen. `match_organization_entities()` verarbeitet eine Liste roher Organisationseinträge aus der Extraktion und bereinigt sie mit `extract_organization()`. Zudem werden aufeinanderfolgende Namen wie „Männerchor“ und „Murg“ zu einem kombinierten Eintrag zusammengeführt. Anschliessend wird für jeden dieser bereinigten Namen ein fuzzy-Matching gegen die Groundtruth durchgeführt, wobei Treffer mit Score und `confidence = "fuzzy"` zurückgegeben werden.

[98] **Abk.:** Regular Expression

[99] Abkürzungen

### 5.3.7 letter_metadata_matcher.py

Der letter_metadata_matcher widmet sich auf der Extraktion inhaltlicher Metadaten aus Briefen und Postkarten. Vornehmlich hier Informationen über die Autorenschaft, den Empfängern und Empfängerinnen sowie deren Rollen. Darüber hinaus sollen auch Daten und Orte nachvollzogen werden, mit einem Fokus auf den Absende- und Empfangsort. Da diese Briefe und Postkarten sehr ähnlich strukturiert sind, liegt der Fokus dieses Moduls auf der Extraktion ihrer Daten.

#### *Pattern Matching in Briefen und Postkarten*

Die zentrale Annahme ist, dass diese Metadaten meist im Textkörper selbst enthalten sind und anhand wiederkehrender sprachlicher Muster bzw. Indikatoren erkannt werden können. Ein Autor oder eine Autorin beendet einen Brief beispielsweise in der Regel mit einer Grussformel und einer Unterschrift. Die Pipeline macht sich das zunutze, indem diese Patterns als Marker in einem String dienen, und zusätzlich zu den Custom-Tags im XML File auf eine Autorenschaft hinweisen können. Diese Marker sind in fünf Listen auf RegEx-Basis notiert.

- `GREETING_PATTERNS` als Schlussformel in einem Brief zeigt die Autorenschaft an.
  Beispiel: `mit freundlichen Grüßen` oder `Heil Hitler`.
- `RECIPIENT_PATTERNS` Erkennt direkte Anreden von Empfänger:innen im Text.
  Beispiel: `An Herrn/Frau` oder `Lieber Fritz`
- `direct_patterns` zeigen die unmittelbaren Empfänger eines Briefes an.
- `INDIRECT_RECIPIENT_PATTERNS` zeigen an, dass ein Brief oder Postkarte an einen Zwischenempfänger geht.
  Beispiel: `zu Händen von`
- `SIGNATURE_PATTERNS` ist eine auf das nachfolgende ROLE_PATTERNS stützende RegEx, die reine Rollenzeilen am Ende eines Briefes erkennt. Beispiel:
  `der Vereinsführer`
  `Alfons Zimmermann`.
- `ROLE_PATTERNS` ist eine dynamisch generierte RegEx, die auf der Rolle basierende Signaturen überprüft. Grundlage ist eine aus `export-roles.csv` geladene Liste bekannter Rollenbegriffe.

#### *Extraktion von Autorinnen und Autoren*

Die konkrete Anwendung dieser Patterns erfolgt in den jeweiligen Funktionen, deren Grundlage eine Verarbeitung der raw strings ist. Die Funktion `extract_authors_raw` analysiert hierfür

zu Beginn alle Textzeilen innerhalb des Transkription, ohne dabei auf strukturierte XML-Tags angewiesen zu sein. Stattdessen orientiert sich die Funktion an den oben erläuterten typischen linguistischen Mustern. Somit kann die Funktion auch dann Ergebnisse liefern, wenn strukturierte XML-Tags wie `<author>` fehlen oder unvollständig sind, was als Backup-Lösung die LLM-Annotation komplettiert. Ziel ist es, möglichst frühzeitig Hinweise auf Autor:innen zu erkennen und diese mit einem einfachen Scoring-System zu bewerten. Das erkannte Scoring wird in einem separaten Feld `author_score` festgehalten und dient als Bewertungsgrundlage für die spätere Auswahl der wahrscheinlichsten Autor:innen.

Da Autoren in der Regel am Briefende unterschreiben, werden die Zeilen des Transkripts von unten nach oben analysiert. Dabei wird zunächst geprüft, ob eine typische Rollensignatur wie `Der Vereinsführer` alleine am Dokumentende steht. Wird eine solche erkannt, wird unmittelbar eine entsprechende Rolle extrahiert, ohne dass ein Name erforderlich ist.

Findet sich keine Rollenzeile, durchsucht die Funktion den Text nach einer Schlussformel basierend auf den in `GREETING_PATTERNS` definierten Mustern. Der Textabschnitt, der auf diese Grussformel folgt, wird im nächsten Schritt nach Signaturen durchsucht. Die Auswertung erfolgt dabei in mehreren aufeinanderfolgenden Prüfungen.

Zunächst wird versucht, eine Kombination aus Rolle und Nachname zu identifizieren[100]. Alternativ wird geprüft, ob Rolle und Name in zwei aufeinanderfolgenden Zeilen vorkommen. In Fällen, in denen sowohl Rolle und Name in benachbarten Zeilen auftreten – etwa `"Der Vereinsführer"` gefolgt von `"Alfons Zimmermann"` – wird diese kontextuelle Nähe berücksichtigt. Die Funktion vergibt in solchen Fällen einen erhöhten `match_score`, da die unmittelbare Abfolge als zuverlässiger Hinweis auf eine zusammengehörige Signatur gewertet wird. Technisch erfolgt die Prüfung auf $\pm 1$ Zeile Abstand im Transkripttext.

Weitere Varianten umfassen typische Signaturformate wie `Max Ganter, Schriftführer` oder Namensinitialen wie `M. Ganter`. Auch vollständige Namen (Vorname + Nachname) oder einzelne Namen (z.B. nur `Fritz`) werden erkannt und mit Platzhaltern übernommen. In Ausnahmefällen werden auch unvollständige Namensformen[101] übernommen, sofern sie in typischen Signaturkontexten auftreten. Dabei wird der fehlende Vor- oder Nachname mit einem Platzhalter ( `None` ) belegt und der Eintrag entsprechend mit `needs_review = true` markiert. Die Übernahme solcher Platzhalter erfolgt jedoch selektiv und nur bei klarer Signaturstruktur.

Wird kein valides Format erkannt, liefert die Funktion ein leeres Dictionary mit einem Eintrag für `closing`, der die gefundene Grussformel dokumentiert.

Neben der heuristischen Pattern-Suche durch `extract_authors_raw()` stützt sich

[100] z.B. `Der Vereinsführer Zimmermann`

[101] etwa Initialen oder Einzelnamen

die Pipeline auch auf strukturierte Annotationen aus dem XML. Die Funktion `extract_author_from_custom_tag()` sucht gezielt nach `<author>`-Tags und übernimmt diese, sofern sie vollständig sind. Im Anschluss wird jeder Kandidateneintrag durch `resolve_llm_custom_authors_recipients()` validiert und bewertet. Insbesondere dann, wenn er von einem LLM vorgeschlagen oder aus unstrukturiertem Text extrahiert wurde kommt die Bewertung zum tragen. Abschliessend übernimmt `letter_match_and_enrich()` die Anreicherung durch bekannte Personen aus der Groundtruth-Liste. Hierbei werden unvollständige Einträge (z.B. nur ein Vorname) durch vollständigere bereits in `mentioned_persons` aufgenommene Personen ersetzt, sofern möglich. Auch in Fällen, in denen keine strukturierte LLM-Annotation vorliegt, erfolgt eine nachgelagerte Prüfung gegen die Groundtruth. Die Funktion `resolve_llm_custom_authors_recipients()` gleicht unvollständige oder generische Einträge – etwa lediglich `"Otto"` oder `"Herrn"` – mit den zuvor erkannten `mentioned_persons` ab. So können auch partielle oder schwach formatierte Namensangaben aufgelöst und angereichert werden.

Die Pipeline deckt jedoch nicht nur einzelne Autoren ab. `match_authors ( )` kombiniert die Roh-Extraktion der Autor:in ( `extract_authors_raw` ) mit der Anreicherung durch `letter_match_and_enrich` . Ist das Ergebnis leer, prüft sie, ob zumindest eine Rolle ohne Namensangabe vorliegt. In diesem Sonderfall wird ein Eintrag mit leerem Namen, aber übernommener Rolle erstellt. Die Rolle wird normalisiert ( `normalize_and_match_role` ) und in ein `role_schema` überführt. Der Rückgabewert enthält `needs_review = True` sowie eine niedrige `match_score` (10), um auf die Unsicherheit der reinen Rollenangabe hinzuweisen.

### *Extraktion von Empfängerinnen und Empfängern*

Die Extraktion potenzieller Empfänger:innen erfolgt analog zur Autorenerkennung über eine Kombination aus regulären Ausdrücken, strukturierten XML-Tags und nachgelagerter Anreicherung. Im Gegensatz zur Autorenschaft, die typischerweise am Ende eines Dokuments steht, werden Empfänger:innen meist im Briefkopf genannt. Die Funktion `extract_recipients_raw()` konzentriert sich daher auf die ersten fünf Zeilen des Transkripts. Dieser Abschnitt enthält häufig Adressierungen wie `„An Herrn Otto Bollinger"` oder zeilenweise strukturierte Formate, die auf eine Empfängeradresse hindeuten. Die Funktion arbeitet unabhängig von expliziten XML-Tags wie `<recipient>`, wodurch sie auch bei unvollständig annotierten Dokumenten Ergebnisse liefern kann.

Die Erkennung erfolgt in drei heuristischen Stufen, abhängig von der Länge und Struktur der jeweiligen Adressierung. Einzeilige Formulierungen wie `„An Herrn Otto Bollinger"` werden als besonders zuverlässig bewertet ( `recipient_score = 90` , `confidence "header-inline"` ). Dreizeilige Strukturen wie

An

Herrn

Otto Bollinger

werden als semistrukturiert interpretiert und mit einem Score von 80 markiert (`confidence = "header-3line"`). Bei zweizeiligen Konstruktionen – etwa `Herrn` in einer Zeile und ein Name in der nächsten – wird aus methodischer Vorsicht ein niedrigerer Score (70) vergeben, da diese auch als Fliesstextnennung interpretiert werden könnten (`confidence = "header-2line"`). Diese Liste von Empfängerkandidaten wird anschliessend in mehreren Schritten angereichert und validiert.

In einem weiteren Schritt werden `<recipient>`-Tags aus dem XML durch `extract_recipient_from_custom_tag()` übernommen, sofern vorhanden. Wie bei den Authoren durchlaufen Rezipienten danach die Funktion `resolve_llm_custom_authors_recipients()`. So werden die Rohdaten zusammengeführt, und anschliessen eine Validierung vorgenommen. Insbesondere unvollständige oder generische Einträge wie `„Herrn"` oder häufig vorkommende Namen wie `„Otto"`[102] werden entweder mit dem Vermerk `needs_review = true` versehen oder durch bereits `mentioned_persons` ersetzt. Diese Ersetzung erfolgt abschliessend in `letter_match_and_enrich()`, wo wie bei den Autor:innen unvollständige Einträge – etwa nur der Vorname – durch vollständigere Einträge aus `mentioned_persons` ersetzt werden, sofern diese zuvor im Text erkannt wurden. Mit `extract_multiple_recipients_raw()` können darüber hinaus potenzielle Empfänger:innen aus dem Transkripttext anhand direkter und indirekter Anredeformen aus den jeweiligen Patternlisten extrahiert werden. Im Gegensatz zu `extract_recipients_raw()`, das sich auf die ersten Zeilen des Dokuments konzentriert, analysiert die Funktion `extract_multiple_recipients_raw()` den gesamten Transkripttext. Sie erkennt direkte und indirekte Anredeformen auch im Fliesstext[103] und erweitert damit die Empfängerliste über die Briefkopfregion hinaus. Direkte Formulierungen wie `Lieber Otto` werden mit `recipient_score = 100` als besonders zuverlässig bewertet. Indirekte Anreden wie `zu Händen des Herrn Alfons Zimmermann` werden mit Score 70 erfasst. Die Ergebnisse enthalten Vorname, Nachname (falls vorhanden), Rolle (leer), Score und Confidence-Label. Durch die unterschiedlichen Ratings im Recipient-Scores können so Staffelungen dargestellt werden.

[102] die Groundtruth kennt 11 verschiedene Personen namens Otto

[103] etwa im Kontext von Zwischenempfängern oder weitergeleiteten Briefen

### *Extraktion des Absendeort und Empfangsort*

Die Extraktion des Absendeorts (`creation_place`) und des Empfangsorts (`recipient_place`) erfolgt in einem mehrstufigen Verfahren, das strukturierte XML-Informationen, heuristische Pattern-Erkennung und ein darauf aufbauendes Groundtruth-Matching kombiniert. Ziel ist es, auch in unvollständig annotierten oder uneinheitlich formulierten Briefen möglichst präzise Ortsinformationen zu identifizieren und anzureichern. Die Verarbeitung ist dabei modular aufgebaut und erfolgt durch vier aufeinander abgestimmte Funktionen.

Die Funktion `extract_places_and_date()` bildet den Einstiegspunkt für die Ortserkennung. Sie durchsucht alle Zeilen im XML-Dokument (`<TextLine>`) nach Attributen vom Typ `custom`, in denen strukturierte Angaben zu `creation_place`, `recipient_place` und `creation_date` hinterlegt sein können. Wird ein solches Attribut gefunden, wird der zugehörige Ort mittels RegEx aus dem Text extrahiert. Grundlage hierfür sind die LLM-ergänzten, im Transkribus-Export enthaltenen Offset- und Length-Parameter, die den Ortstext innerhalb der Zeile markieren. Die Funktion liefert drei `raw-strings` zurück: den vermuteten Absendeort (`raw_creation_place`), den Empfangsort (`raw_recipient_place`) sowie das Datum der Entstehung (`creation_date`)[104]. Sind keine Custom-Tags vorhanden, aktiviert die Funktion ein Fallback-Verfahren, das den Briefkopf anhand typischer Formate mit regulären Ausdrücken analysiert. Erkannt werden insbesondere Muster wie `Ort, den dd.mm.yyyy` oder `An ... in Ort`. Die so ermittelten Raw-Strings werden in `assign_sender_and_recipient_place(...)` weiterverarbeitet. Diese Hauptfunktion ruft zunächst `extract_places_and_date()` auf, um die Rohdaten zu extrahieren, und analysiert dann alle Zeilen des Transkripts, um auch Kontexte für zusammengesetzte Ortsbezeichnungen[105] zu erfassen. Anschliessend wird das Orts-Matching über die Hilfsfunktion `enrich_place_candidate()` durchgeführt. Diese Funktion gleicht den erkannten Komposit-Ortsnamen mit einer Groundtruth-Liste ab, in der alle bekannten Orte und ihre Varianten hinterlegt sind. Neben exakten Treffern erlaubt das Matching auch Fuzzy Matching über die Levenshtein-Distanz. Ergibt sich dabei ein Treffer, wird ein Score berechnet und mit weiteren Metadaten [106] angereichert. Im Kontext von Empfängeradressen wird zusätzlich geprüft, ob im unmittelbaren Umfeld ein Vereins- oder Organisationsname vorkommt. In diesem Fall wird ein Score-Malus vergeben, da das ein Hinweis auf eine mögliche Verwechslung zwischen Ort und Organisation sein kann.

Eine besondere Stärke der Pipeline liegt in der Erkennung zusammengesetzter Ortsnamen. Die Funktion `find_combined_place()` durchsucht benachbarte Zeilen nach Kombinationen wie

[104] dazu mehr im nächsten Abschnitt
[105] z.B. Laufenburg-Rhina
[106] z.B. `nodegoat_id`

`Laufenburg` und `Rhina`, die zusammen in einem Eintrag der Groundtruth-Liste vorkommen (`Laufenburg-Rhina`). Wird eine solche Kombination erkannt, wird sie bevorzugt, erhält einen Bonus im Matching-Score und ersetzt die Einzeltreffer. Weil so auch Teilorte oder Stadtteile abgebildet werden können, wird die Ortserkennung deutlich präziser.

Abschliessend wird durch `finalize_recipient_places()` der am besten bewertete Empfängerort ausgewählt. Die Funktion sortiert alle erkannten Ortskandidaten nach Score und übernimmt den besten Eintrag als `recipient_place`. Weitere valide, aber weniger eindeutige Orte werden ebenfalls gespeichert, jedoch mit dem Attribut `needs_review = true` gekennzeichnet, um eine manuelle Prüfung zu ermöglichen.

Die finale Funktion `match_place()` kann mehrere potenzielle Ortstreffer zurückgeben, insbesondere wenn es konkurrierende Varianten mit ähnlichem Namen gibt. In diesem Fall wählt die Funktion `finalize_recipient_places()` den Ort mit dem höchsten Score als `recipient_place` aus, während weitere plausible, aber unsicherere Treffer mit dem Vermerk `needs_review = true` gespeichert werden. Durch das modulare Zusammenspiel der vier Funktionen können auch in schwierig strukturierten oder unvollständig annotierten Dokumenten präzise Ortsangaben extrahiert werden.

### *Extraktion des Erstellungsdatum*

Die Erkennung von Entstehungsdaten (`creation_date`) erfolgt im Modul `letter_metadata_matcher.py` primär über die von oben bereits bekannte Funktion `extract_places_and_date( )`. Im Zentrum steht die Auswertung strukturierter Custom-Tags im PAGE-XML sowie ein regulärer Fallback-Mechanismus zur heuristischen Analyse des Briefkopfs.

Im ersten Schritt durchsucht die Funktion alle `<TextLine>`-Elemente nach dem Attribut `custom`, das von Transkribus für semantische Annotationen verwendet wird. Wird ein Eintrag vom Typ `creation_date` oder `date` erkannt, wird mittels RegEx nach einem `when="YYYY-MM-DD"`-Feld innerhalb des custom-Attributs gesucht. Dieser Wert wird direkt übernommen und als `creation_date` gespeichert. Da bei dem manuellen Tagging besonders auf die korrekten Daten geachtet, und diese händisch mit `when¯` immer ergänzt wurden, sind diese Angaben in jedem Fall vorhanden.

Wird jedoch kein Custom-Tag erkannt, greift ein Fallback-Verfahren: Die Funktion analysiert hierfür die ersten fünf Zeilen des Dokuments – typischerweise der Briefkopf – auf typische Orts-Datumskombinationen. Dazu zählt insbesondere das Muster

`Ort, den dd.mm.yyyy`

Die RegEx erfasst dabei sowohl den Ort (als möglichen Absendeort) als auch das Datum. Sie sind daher auch gemeinsam in dieser Funktion aufgenommen. Wird ein solches Datum gefunden,

wird es als `creation_date` übernommen, selbst wenn kein zugehöriger Ort erkannt werden konnte. Die Erkennung erfolgt dabei tolerant gegenüber dem optionalem Wort „den“ und erlaubt sowohl zwei- als auch vierstellige Jahresangaben[107]. Ist kein solcher Eintrag vorhanden, bleibt `creation_date = None`.

In Kombination mit `assign_sender_and_recipient_place()` wird das erkannte Datum anschliessend gemeinsam mit den Ortseinträgen an die Hauptverarbeitung zurückgegeben und als `attributes["creation_date"]` in den Metadaten des Dokuments gespeichert. Im Gegensatz zu `mentioned_dates`[108], die beliebige Datumsangaben im Fliesstext zählen, bezieht sich `creation_date` ausschliesslich auf das Erstellungsdatum des Dokuments

### 5.3.8 type_matcher.py

Dieses Modul besteht aus nur einer einzigen Funktion: `def get_document_type ( )`. Diese zerlegt den Namen der zu verarbeitenden Datei und extrahiert daraus die siebenstellige Transkribus-ID und die Seitennummer. Es folgt ein Vergleich in der Akten_Gesamtübersich.csv, die als Groundtruth alle Dateien im Korpus listet.

Da während der Transkription in dieser Tabelle auch die Dokumententypen festgehalten wurden, kann über einen einfachen Lookup für jede Seite der zugehörige Typ extrahiert werden. Mögliche Kategorien sind unter anderem „Brief“, „Postkarte“, „Protokoll“.[109]

Falls kein passender Eintrag in der Groundtruth gefunden wird, bietet die Funktion ein optionales Fallback: Wird zusätzlich ein `xml_path` übergeben, so wird versucht, den Dokumenttyp direkt aus einem `<Dokumententyp>`-Tag innerhalb der zugehörigen XML-Datei auszulesen. Auf diese Weise wird sichergestellt, dass jede Seite im JSON-Output einen Typ zugewiesen erhält – entweder verlässlich aus der CSV oder über das XML-Fallback.

Die extrahierte Typinformation wird im Attributblock des finalen JSON unter `attributes["document_type"]` gespeichert. So kann später seitengenau nach Dokumententypen gefiltert und eine differenzierte Verteilung der Typen innerhalb einer Akte nachvollzogen werden.

### event_matcher.py

Das Modul `event_matcher.py` dient der Erkennung, Strukturierung und Anreicherung von Ereignistags aus den Transkribus-Dokumenten. Die zugrundeliegende Funktion `extract_events_from_xml()` verarbeitet die gesamte XML-Datei zeilenweise und extrahiert jene Textstellen, die als Ereignis markiert wurden oder im unmittelbaren Zusammenhang mit

[107] erkannt werden also sowohl beispielsweise 1939, als auch nur 39
[108] mehr in date_matcher.py
[109] siehe Kapitel Quellenbeschrieb

einem Ereignisblock stehen. Grundlage dafür ist die Auswertung des `custom`-Attributs einzelner `<TextLine>`-Elemente, insbesondere wenn dieses explizit mit „event“ gekennzeichnet ist.
Die Funktion arbeitet blockbasiert: Ereignisblöcke bestehen in der Regel aus mehreren aufeinanderfolgenden Zeilen, die durch inhaltliche Merkmale wie Bindestriche, kleingeschriebene Fortsetzungszeilen oder fehlende Datumsmarkierungen zusammenhängend interpretiert werden. Die Funktion `is_continuation()` prüft dabei, ob eine Zeile an die vorhergehende angeschlossen werden kann, etwa durch typische Anschlusswörter oder formale Merkmale. Sobald ein abgeschlossener Block erkannt ist, wird dieser mit der Hilfsfunktion `build_event()` zu einem vollständigen Ereignisobjekt zusammengeführt.
In `build_event()` wird der aus mehreren Zeilen bestehende Textblock zunächst als Fliesstext zusammengesetzt und inhaltlich analysiert. Es werden mehrere Entitäten erkannt und dem Ereignis zugeordnet:

- **Orte:** Über die Funktion `match_place()` aus dem Modul `place_matcher.py` werden potenzielle Ortsnamen im Text identifiziert und mit der Groundtruth abgeglichen. Dabei wird jeder erkannte Ort zusätzlich durch eine Plausibilitätsprüfung überprüft, bevor er als `Place`-Objekt übernommen wird.

- **Daten:** Die Funktion `extract_custom_date()` durchsucht die XML-Zeile nach Datumsangaben in den XML-Tags. Wenn kein strukturiertes Datum vorhanden ist, aber einfache numerische Formate wie „15.03“ im Fliesstext erkannt werden, werden diese als Datum übernommen.

- **Organisationen:** Über `match_organization_from_text()` wird der Textblock mit bekannten Organisationseinträgen abgeglichen. Bei Übereinstimmung werden entsprechende Organisationen als strukturierte Objekte ergänzt.

- **Personen:** Mögliche Namen werden durch reguläre Ausdrücke identifiziert und mit Hilfe der Funktion `extract_name_with_spacy()` in Vor- und Nachnamen getrennt. Anschliessend erfolgt ein Abgleich mit der Personen-Groundtruth über `match_person()`. Positive Treffer werden inklusive Match-Score und Herkunftskennzeichnung als `Person`-Objekte dem Ereignis zugeordnet.

Die fertigen Ereignisse bestehen jeweils aus einer Kurzbeschreibung (in der Regel der ersten Zeile), einer ausführlichen Beschreibung (bestehend aus allen zugehörigen Textzeilen), einem Datumsfeld, einer Ortsangabe und einer strukturierten Liste aller beteiligten Orte, Organisationen und Personen. Der vollständige Satz aller so erkannten Ereignisse wird am Ende als Liste von `Event`-Objekten zurückgegeben.
Das Modul arbeitet vollständig dateibasiert und benötigt als einzige Eingabe den Pfad zur Transkribus-XML-Datei sowie eine initialisierte Instanz des `PlaceMatcher`. Es greift auf zen-

trale Komponenten der Projektarchitektur zurück, darunter die Groundtruth-Listen für Orte, Personen und Organisationen. Die extrahierten Ereignisse werden im finalen JSON unter dem Attribut `events` gespeichert. Da ein Event ein eher abstraktes Konstrukt ist, liegt der Fokus der Pipeline weniger auf diesem Modul, das zu einem späteren Zeitpunkt beispielsweise durch präziseres Prompten für das Eventtagging optimiert werden soll.

### date_matcher.py

Das Modul `date_matcher.py` dient der systematischen Extraktion, Normalisierung und Zählung von Datumsangaben in den Transkribus-Dokumenten. Es basiert auf der Auswertung strukturierter Angaben, die im Rahmen der Transkription über XML-Custom-Tags im `custom`-Attribut einzelner `<TextLine>`-Elemente eingebettet wurden. Diese Daten gelten innerhalb des Korpus als zuverlässig, da sie während der manuellen Korrekturprozesse in Transkribus einheitlich normiert und im Format `dd.mm.yyyy` ergänzt wurden.

Treten im historischen Text verkürzte Datumsangaben wie *„1. d. Mts“* auf, so handelt es sich um Abkürzungen, die bei der Transkription mit einem entsprechenden `abbreviation`-Tag markiert werden. Die Funktionsweise dieser Markierungen sowie die heuristische Auflösung solcher verkürzter Angaben wird im Kapitel **??** erläutert. Lässt sich aus dem weiteren Kontext[110] ein vollständiges Datum erschliessen, kann dieses anschliessend in strukturierter Form übernommen und im JSON als normiertes Datum gespeichert werden.

Innerhalb der Verarbeitungspipeline wird das Modul über die Funktionen `extract_custom_date()` und `combine_dates()` aufgerufen. Zunächst durchläuft `extract_custom_date()` das XML-Dokument und extrahiert alle `custom`-Attribute, die ein `date{...}`-Muster enthalten. Die Inhalte dieser Attribute werden bereinigt und zur weiteren Analyse an die Funktion `extract_date_from_custom()` übergeben.

Diese Funktion überprüft mithilfe RegEx, ob der String tatsächlich eine gültige Datumsangabe enthält. Dabei wird insbesondere nach einem `when`-Feld gesucht, das im Inneren des `date`-Blocks enthalten ist. Die in diesem Feld hinterlegten Daten werden anschliessend mit der Funktion `parse_custom_attributes()` als Key-Value-Paare interpretiert. Liegt ein gültiges Datum vor, wird dessen Format mit `normalize_to_ddmmyyyy()` überprüft und gegebenenfalls vereinheitlicht.

Unterstützt werden mehrere Eingabeformate, darunter standardisierte Formen wie `dd.mm.yyyy`, ISO-Formate wie `yyyy-mm-dd` oder zweistellige Jahresangaben, die automatisch in vierstellige Jahre des 20. Jahrhunderts umgewandelt werden. Zusätzlich erkennt die Funktion auch

[110] Zum Beispiel durch Hinweise im Seiteninhalt oder durch übergeordnete Informationen im Umfeld der Akte

Intervallangaben wie `01/03.04.1944`, bei denen ein Datumsbereich über einen Schrägstrich kodiert ist. Solche Intervalle werden in strukturierter Form mit einem `from`- und `to`-Wert als `date_range` gespeichert.

Die Funktion `combine_dates()` führt schliesslich alle erkannten Einzel- und Intervallangaben zusammen, zählt deren Häufigkeit im Dokument und erstellt eine deduplizierte, sortierte Liste für den späteren Export. Dabei wird jede identifizierte Angabe – ob Einzeldatum oder Zeitspanne – um eine Zählung der Nennungen ergänzt. Bei Intervallen wird zusätzlich der Originalstring dokumentiert, aus dem die Angabe hervorging.

Das Ergebnis der Verarbeitung wird im Feld `mentioned_dates` gespeichert. Jeder Eintrag enthält entweder ein einzelnes Datum oder einen Datumsbereich, ergänzt um die Häufigkeit und gegebenenfalls den ursprünglichen Wortlaut aus dem `custom`-Attribut.

Das Modul arbeitet unabhängig von externen Ressourcen und benötigt lediglich das XML-Baumobjekt des jeweiligen Dokuments. Die so gewonnenen Zeitangaben bilden die Grundlage für die chronologische Einordnung, Kontextualisierung und Auswertung der digitalen Quellenbasis.

### 5.3.9 unmatched_logger.py

Das Modul `unmatched_logger.py` dient der systematischen Protokollierung von Entitäten, die in der aktuellen Version der Groundtruth noch nicht enthalten sind. Diese Protokolle bilden die Grundlage für weiterführende Recherchen, durch die die Groundtruth schrittweise ergänzt und verbessert werden kann.

Innerhalb der Verarbeitungs-Pipeline wird das Modul `unmatched_logger.py` über die Funktion `process_single_xml()` im Hauptprogramm aufgerufen.

Bereits in der Testphase kam das Modul mehrfach zum Einsatz, um die Erkennung und Zuordnung bislang nicht erfasster Entitäten zu überprüfen.

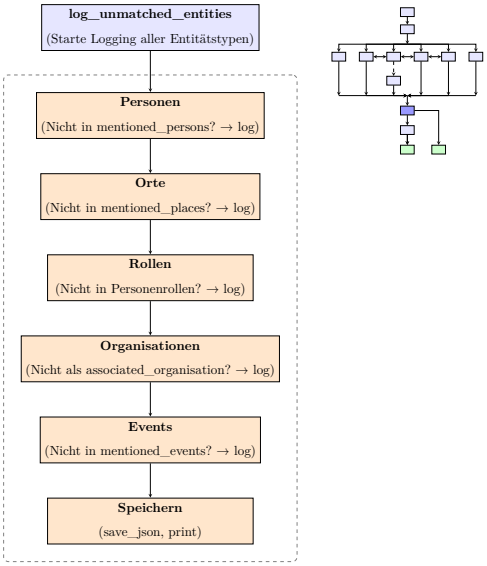



Abbildung 12:
Oben links: Prozessdiagramm für `unmatched_logger.py`,
Oben rechts: Pipelineübersicht

Im Kern stellt das Modul die Funktion `log_unmatched_entities` bereit. Diese übernimmt die von den zuvor beschriebenen Matcher-Funktionen ermittelten Entitäten und prüft, ob sie in den entsprechenden Groundtruth-CSV-Dateien vorhanden sind.

Die Suche erfolgt iterativ innerhalb der Listenstrukturen für Personen, Orte, Rollen, Organisationen und Ereignisse. Wird eine Entität über ein XML-Custom-Tag einer dieser Kategorien zugewiesen, ohne dass sie in der Groundtruth verzeichnet ist, wird sie in einer spezifischen JSON-

Datei protokolliert.

Die folgenden Dateien werden dabei erzeugt:

- unmatched_persons.json
- unmatched_places.json
- unmatched_roles.json
- unmatched_events.json
- unmatched_organisations.json

Zusätzlich werden alle Einträge in einer zusammengeführten Datei `unmatched.json` gespeichert, um einen vollständigen Überblick nicht zugeordneter Entitäten zu gewährleisten.

Alle Ergebnisse werden zudem in einer Datei `unmatched.json` gespeichert, um einen Gesamtüberblick zu erhalten.

### 5.3.10 validation_module.py

Das Modul `validation_module.py` übernimmt die strukturierte Qualitätskontrolle der erzeugten JSON-Dokumente in der finalen Verarbeitungsphase. Es wird in der zentralen Verarbeitungsfunktion `process_single_xml()` unmittelbar vor dem Speichern der Ausgabedateien aufgerufen:

```
validation_result = validate_extended(doc)
    if validation_result:
        print(f"[WARN] Validierungsfehler im Dokument {full_doc_id}:")
        for err_type, errors in validation_result.items():
            print(f"  - {err_type}: {', '.join(errors)}")

```

Die Validierung erfolgt anhand des erzeugten `BaseDocument`-Objekts und basiert auf festen Regeln zur Überprüfung von Pflichtfeldern, Formatvorgaben sowie typischen Erkennungsfehlern bei Personen- oder Ortsangaben.

Die Hauptfunktion `validate_extended(doc)` analysiert das Dokument auf Inkonsistenzen und gibt ein Dictionary mit Fehlermeldungen pro Feld zurück. Neben formalen Anforderungen (z. B. korrektes Datumsformat) werden auch häufige Named-Entity-Fehlklassifikationen erkannt, etwa wenn Titelwörter als Vorname interpretiert werden.

Beispielhafte Prüfungen:

- **Datumsformat**: Ein Eintrag wie `1941.5.28` wird zurückgewiesen, da das erwartete Format `YYYY.MM.DD` lautet.

  `[creation_date] Fehlend oder ungültiges Format (YYYY.MM.DD)`

- **Leere Empfängerliste**: Bei Dokumenten vom Typ `Brief` oder `Postkarte` wird überprüft, ob mindestens ein valider Empfänger mit Name oder Rolle vorhanden ist. Ist dies nicht der Fall, erfolgt eine Fehlermeldung.
  `[recipients] Empfänger fehlt oder ist ungültig`
- **Fehlerhafte Personennamen**: Wird ein Vorname wie `des` oder `Herrn` erkannt, erfolgt eine heuristische Warnung auf möglichen OCR- oder Parsingfehler.
  `[mentioned_persons[0]] Möglicher Fehlname: des`

Die Hinweise des Validierungsmoduls werden insbesondere während der Erprobungs- und Optimierungsphasen intensiv genutzt, etwa zur Feinjustierung der Methoden der Named-Entity-Recognition bei Personen. Sie dienen nicht nur der strukturellen Sicherung eines gültigen JSON-Outputs, sondern ermöglichen auch gezielte Nachkorrekturen im Rahmen der qualitativen Auswertung und visuellen Darstellung der Daten.

## 5.4 KEINE AHNUNG WAS DIE HIER MACHEN

### 5.4.1 llm_enricher.py

Das Modul `llm_enricher.py` dient der nachträglichen semantischen Anreicherung von JSON-Dokumenten mithilfe von GPT-4. Es wird gegen Ende des Verarbeitungsprozesses innerhalb von `transkribus_to_base.py` aufgerufen, nachdem die initiale Konvertierung der Transkribus-XML-Dateien abgeschlossen und das Entitätenmatching bereits erfolgt ist. Das Ziel besteht darin, fehlende Felder wie `author`, `recipient`, `creation_date` oder `content_tags_in_german` auf abzufangen und ggf. zu ergänzen.
Die Funktion `enrich_document_with_llm` übergibt das vollständige JSON-Dokument an das Sprachmodell. Dieses erhält einen Prompt mit präzisen Anweisungen zur Vervollständigung strukturierter Felder, inklusive Regeln z.B. Schutz von IDs wie der `nodegoat_id`, Kombination von Ortsnamen, Erkennung von Vereinsnamen als Organisationen). Der Rückgabewert wird mit dem Originaldokument zusammengeführt und unter `_enriched.json` gespeichert. Zusätzlich wird die LLM-Nutzung dokumentiert (`input_tokens`, `output_tokens`, `cost_usd`).

#### *Abgrenzung zu `llm_preprocessing.py`*

Trotz des gemeinsamen Einsatzes eines Sprachmodells unterscheiden sich die beiden Module grundlegend in Funktion, Datenbasis und Zielsetzung. Während `llm_preprocessing.py` im Frühstadium der Verarbeitung eingesetzt wird, um Textabschnitte vorzubereiten, Prompts zu formulieren oder Entitäten lokal zu erkennen, operiert `llm_enricher.py` am Ende des Workflows auf dokumentweiter Ebene des neu generierten JSON. Die folgende Tabelle fasst die Unterschiede zusammen:

| | `llm_preprocessing.py` | `llm_enricher.py` |
|---|---|---|
| **Ziel** | Vorbereitung des LLM-Einsatzes: Segmentierung, Prompt-Generierung und lokale Entitätenerkennung | Nachträgliche Anreicherung strukturierter JSONs mit fehlenden Metadaten auf Dokumentebene |
| **Fokus** | Lokale NER in isolierten Transkriptzeilen oder Abschnitten | Globale Dokumentinterpretation, Rollen- und Feldzuordnung, Dublettenfilterung |
| **Input** | Textsegmente oder XML-Zeilen aus Transkribus | Strukturiertes JSON-Dokument nach algorithmischer Vorverarbeitung |
| **Output** | Markierte Entitäten in Form von XML-Tags oder Listen (z.B. Personen, Orte) | Vervollständigtes JSON mit algorithmisch unausgefüllten Feldern |

***Notwendigkeit beider Module trotz LLM-Einsatz***

Auch wenn beide Module aktuell das selbe Sprachmodell verwenden, besteht keine redundante Dopplung. Sie erfüllen komplementäre Aufgaben in einer zweistufigen Pipeline: Zuerst erfolgt mit `llm_preprocessing.py` die Erkennung und Markierung von Entitäten in isolierten Textblöcken zur Named-Entity-Recognition. Anschliessend übernimmt `llm_enricher.py` die Integration, Interpretation und strukturelle Vervollständigung dieser Einheiten im Gesamtkontext des Dokuments. Es handelt sich somit um ein hybrides Verfahren nach dem Prinzip: *Erkennen → Zuordnen → Strukturieren.*

Das Modul `llm_enricher.py` wird in `transkribus_to_base.py` optional aufgerufen, nachdem die regulären Verarbeitungsschritte abgeschlossen sind. Nur wenn bestimmte Kernfelder im JSON-Dokument fehlen, wird das Enrichment aktiviert. Dies verhindert unnötige LLM-Aufrufe und erhöht die Effizienz. Die Rückgabe wird mit dem Originaldokument vereinigt, bestehende IDs und Daten werden geschützt, neue Felder ergänzt. Damit wird eine nachträgliche Qualitätssteigerung des Outputs erzielt – bei vollständiger Nachvollziehbarkeit.

# Bibliographie

## Artikel

Beck, Maximilian u. a. (2020). „A review on the long short-term memory model. Artificial intelligence review“. In: *Artificial intelligence review* 53.8, S. ff. 5929–5955 (zit. auf S. 21).

Capurro, Carlotta, Vera Provatorova und Evangelos Kanoulas (29. Nov. 2023). „Experimenting with Training a Neural Network in Transkribus to Recognise Text in a Multilingual and Multi-Authored Manuscript Collection“. In: *Heritage* 6.12, S. ff. 7482–7494. ISSN: 2571-9408. DOI: 10.3390/heritage6120392. URL: https://www.mdpi.com/2571-9408/6/12/392 (zit. auf S. 24).

Levenshtein, Vladimir I. (2025). „Binary Codes Capable of Correcting Deletions, Insertions and Reversals“. Übers. von Doklady Akademii Nauk SSSR, Akademie der Wissenschaften UdSSR. In: *Soviet Physics Doklady* (), S. ff. 707–710. URL: https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf (zit. auf S. 35).

Martinez, Roxana und Gonzalo Pereyra Metnik (o. D.). „Comparative Study of Tools for the Integration of Linked Open Data: Case study with Wikidata Proposal“. In: () (zit. auf S. 17).

Wilkinson, Mark D. u. a. (15. März 2016). „The FAIR Guiding Principles for scientific data management and stewardship“. In: *Scientific Data* 3.1. Publisher: Nature Publishing Group, S. 160018. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18. URL: https://www.nature.com/articles/sdata201618 (zit. auf S. 17).

## Monografien

Buchner, Alex (1989). *Das Handbuch der Deutschen Infanterie 1939 – 1945*. 2. Aufl. Friedberg: Podzun-Pallas, 1989. ISBN: 3-7909-0301-9 (zit. auf S. 3).

Gamper, Markus und Linda Reschke (27. Apr. 2015). *Knoten und Kanten III: Soziale Netzwerkanalyse in Geschichts- und Politikforschung*. Hrsg. von Martin Düring. transcript Verlag, 27. Apr. 2015. ISBN: 978-3-8394-2742-2. DOI: 10.1515/9783839427422. URL: https://www.degruyter.com/document/doi/10.1515/9783839427422/html (zit. auf S. 5).

Hartmann, Christian (2010). *Wehrmacht im Ostkrieg - Front und militärisches Hinterland 1941/42*. 2. Auflage. Bd. 75. Quellen und Darstellungen zur Zeitgeschichte Herausgegeben vom Institut für Zeitgeschichte. München: R. Oldenbourg Verlag, 2010 (zit. auf S. 3).

Haupt, Werner (1982). *Das Buch der Infanterie*. 1. Aufl. Friedberg, Hanau: Podzun-Pallas, 1982. ISBN: 3-7909-0176-8 (zit. auf S. 3 f.).

Rass, Christoph und René Rohrkamp (2009). *Deutsche Soldaten 1939-1945 Handbuch einer biographischen Datenbank zu Mannschaften und Unteroffizieren von Heer, Luftwaffe und Waffen-SS*. Aachen, 2009 (zit. auf S. 3).

Richard, Smiraglia und Scharnhorst Andrea (3. Mai 2022). *Linking Knowledge. Linked Open Data for Knowledge Organization and Visualization*. Version Number: editorsversion, prior to publication. Zenodo, 3. Mai 2022. DOI: 10.5771/9783956506611. URL: https://zenodo.org/records/6513663 (zit. auf S. 4).

Tessin, Georg (1977). *Verbände und Truppen der deutschen Wehrmacht und Waffen-SS im Zweiten Weltkrieg 1939-1945*. Bd. Band 1 - Die Waffengattungen — Gesamtübersicht. Osnabrück: HIBLIO Verlag, 1977 (zit. auf S. 3).

Zentner, Christian (1983). *Illustrierte GEschichte des Zweiten Weltkriegs*. München: Südwest Verlag GMbH, 1983 (zit. auf S. 3).

## Onlinequellen

Altenburger, Andreas (2023). *Lexikon der Wehrmacht*. (Zugriff am 15. 01. 2023). URL: https://www.lexikon-der-wehrmacht.de/Gliederungen/Infanteriedivisionen/205ID.htm (zit. auf S. 3, 16).

Burkhardt, Sven (12. Dez. 2022). *Feldpost an den Männerchor Murg - Storymaps*. ArcGIS StoryMaps. (Zugriff am 12. 03. 2025). URL: https://storymaps.arcgis.com (zit. auf S. 3).

*Claude Code* (2025). Anthropic. (Zugriff am 22. 07. 2025). URL: https://docs.anthropic.com/en/release-notes/claude-code (zit. auf S. 26).

*DRK Suchdienst | Suche per Feldpostnummer* (2025). DRK Suchdienst. Unter Mitarb. von Christian Reuter. (Zugriff am 12. 03. 2025). URL: https://vbl.drk-suchdienst.online/Feldpostnummer/FPN.aspx (zit. auf S. 3, 16).

*Feldpost Number Database | GermanStamps.net* (2025). (Zugriff am 09. 07. 2025). URL: https://www.germanstamps.net/feldpost-number-database/ (zit. auf S. 3).

*Forum Geschichte der Wehrmacht* (2025). Forum Geschichte der Wehrmacht. Unter Mitarb. von Dieter Hermans. (Zugriff am 12. 03. 2025). URL: https://www.forum-der-wehrmacht.de/ (zit. auf S. 3).

*Gemini – unser größtes und leistungsfähigstes KI-Modell* (6. Dez. 2023). Google. (Zugriff am 22. 07. 2025). URL: https://blog.google/intl/de-de/unternehmen/technologie/gemini/ (zit. auf S. 25).

*GeoNames* (2025). (Zugriff am 05. 07. 2025). URL: https://www.geonames.org/ (zit. auf S. 18).

*Geschichte Gemeinde Murg* (2025). Unter Mitarb. von Gemeinde Murg. (Zugriff am 29. 06. 2025). URL: https://www.murg.de/seite/33378/geschichte.html (zit. auf S. 2).

Gubler, Kaspar (2025). *nodegoat Tutorials*. HistData. (Zugriff am 11. 07. 2025). URL: https://histdata.hypotheses.org/nodegoat-tutorials-deutsch/nodegoat-tutorials (zit. auf S. 19).

Hollmann, Prof. Dr. Michael (2025). *Freiburg*. Bundesarchiv Freiburg im Breisgau (Abteilung Militärarchiv). (Zugriff am 12. 03. 2025). URL: https://www.bundesarchiv.de/das-bundesarchiv/standorte/freiburg/ (zit. auf S. 3).

Kessels, Geert und Pim van Bree (2013). *nodegoat: a web-based data management, network analysis & visualisation environment*. nodegoat. (Zugriff am 11. 07. 2025). URL: https://nodegoat.net/about (zit. auf S. 19).

*Model Release Notes* (2025). OpenAI Help Center. (Zugriff am 22. 07. 2025). URL: https://help.openai.com/en/articles/9624314-model-release-notes (zit. auf S. 27).

*Msty - Using AI Models made Simple and Easy* (2025). (Zugriff am 06. 07. 2025). URL: https://msty.app/ (zit. auf S. 25).

*OWL Guide* (10. Feb. 2004). *OWL Web Ontology Language Guide*. Unter Mitarb. von Michael K. Smith, Chris Welty und Deborah L. McGuiness. (Zugriff am 05. 07. 2025). URL: https://www.w3.org/TR/owl-guide/ (zit. auf S. 14 f.).

*Recognition and Enrichment of Archival Documents | READ | Projekt | Fact Sheet | H2020* (2025). CORDIS | European Commission. (Zugriff am 06. 07. 2025). URL: https://cordis.europa.eu/project/id/674943 (zit. auf S. 23).

Thomson, Martin, Erik Wilde und Adam Roach (7. Juni 2017). *Geographic JSON (geojson)*. (Zugriff am 11. 07. 2025). URL: https://datatracker.ietf.org/wg/geojson/charter/ (zit. auf S. 20).

*Transkribus* (Nov. 2024). Transkribus_Model_mmma. Unter Mitarb. von Sven Burkhardt. (Zugriff am 25. 06. 2025). URL: https://app.transkribus.eu (zit. auf S. 12).

*WGS84 | Landesamt für Geoinformation und Landesvermessung Niedersachsen* (2025). Landesamt für Geoinformation und Landesvermessung Niedersachsen. (Zugriff am 05. 07. 2025). URL: https://www.lgln.niedersachsen.de/startseite/wir_uber_uns/hilfe_support/lgln_lexikon/w/wgs84-190576.html (zit. auf S. 18).

*Wikidata* (2025). (Zugriff am 05. 07. 2025). URL: https://www.wikidata.org/wiki/Wikidata:Main_Page (zit. auf S. 17).

## Software

Brown, Tom B. u. a. (2020). *Language Models are Few-Shot Learners*. _eprint: 2005.14165. 2020. URL: https://arxiv.org/abs/2005.14165 (zit. auf S. 29).

Burkhardt, Sven (23. Apr. 2025b). *github/PDF_to_JPEG.py*. Version 1.0. Basel, 23. Apr. 2025. URL: https://github.com/Sveburk/masterarbeit/blob/main/3_MA_Project/Hilfs_Scripte/JPEG_to_PDF.py (zit. auf S. 10, 21).

## Vorträge und Manuskripte

Burkhardt, Sven (17. Juni 2025a). „Bias is not a Bug: Towards Methodological Awareness in AI-Assisted Humanities“. DH-CH Conference 2025. Rom, 17. Juni 2025. URL: https://dh-ch.ch/dhch-isr/25/presentations.html (zit. auf S. 53).

Durst, Emil (14. Feb. 1948). „Sklaven des zwanzigsten Jahrhunders - Ein Tatsachenbericht.“ historische Quelle, Manuskript, Einzelexemplar, historische Quelle, Manuskript, Einzelexemplar, Murg, 14. Feb. 1948 (zit. auf S. 27).

Hodel, Tobias (2019). „Machine Learning in Digital History: Texterkennung und Dokumentenanalyse mit Transkribus“. Universitäre Übung. Universitäre Übung. Universität Basel, 2019. URL: https://vorlesungsverzeichnis.unibas.ch/de/vorlesungsverzeichnis?id=243354 (zit. auf S. 3).

Mühlberger, Günter (25. Apr. 2019). „Transkribus Eine Forschungsplattform für die automatisierte Digitalisierung, Erkennung und Suche in historischen Dokumenten“. Kolloqium der ETH-Bibliothek. Zürich, 25. Apr. 2019. URL: https://ethz.ch/content/dam/ethz/associates/ethlibrary-dam/documents/Aktuell/Veranstaltungen/17-15-Kolloquium/2019-04-29_17-15-Kolloquium_transkribus.pdf (zit. auf S. 23).

Serif, Ina (2019). „Digital History: computergestützte Anwendungs- und Forschungsmöglichkeiten in den Geschichtswissenschaften“. Universitäre Übung. Universitäre Übung. Universität Basel, 2019. URL: https://vorlesungsverzeichnis.unibas.ch/de/vorlesungsverzeichnis?id=243345 (zit. auf S. 3).

– (2022). „Von Bücherschätzen zu Datensätzen. Digitalisierung, Aufbereitung und Auswertung historischer Quellen“. Universitäre Übung. Universitäre Übung. Universität Basel, 2022. URL: https://vorlesungsverzeichnis.unibas.ch/de/vorlesungsverzeichnis?id=269454 (zit. auf S. 3).

# A Anhang

## A.1 PDF_to_JPEG.py

```python
import os
import fitz  # PyMuPDF

def convert_pdf_to_jpg(src_folder, dest_folder):
    # Überprüfen, ob der Zielordner existiert, und ihn ggf. erstellen
    if not os.path.exists(dest_folder):
        os.makedirs(dest_folder)

    # Durchgehen durch alle Dateien im Quellordner
    for root, dirs, files in os.walk(src_folder):
        for file in files:
            # Überprüfen, ob die Datei eine PDF-Datei ist
            if file.lower().endswith(".pdf"):
                # Vollständigen Pfad zur PDF-Datei erstellen
                pdf_path = os.path.join(root, file)
                # PDF-Datei öffnen
                doc = fitz.open(pdf_path)
                # Durch alle Seiten der PDF-Datei gehen
                for page_num in range(len(doc)):
                    page = doc[page_num]
                    # Seite in ein PixMap-Objekt umwandeln (für die Konvertierung in
                    ↪ JPG)
                    pix = page.get_pixmap()
                    # Dateinamen ohne Dateiendung extrahieren
                    filename_without_extension = os.path.splitext(file)[0]
                    # Ausgabedateinamen erstellen mit führenden Nullen für die
                    # Seitennummer
                    output_filename = f"{filename_without_extension}_S{page_num +
                    ↪ 1:03d}.jpg"


                    # Vollständigen Pfad zur Ausgabedatei erstellen
                    output_path = os.path.join(dest_folder, output_filename)
                    # Bild speichern
                    pix.save(output_path)
                # PDF-Datei schliessen
                doc.close()

                # Erfolgsmeldung ausgeben
                print(f"{file} wurde erfolgreich umgewandelt und gespeichert
                in {dest_folder}")

# Pfade zu den Ordnern mit den PDF-Dateien (Quelle) und den JPG-Dateien (Ziel)
src_folder = r"/Users/svenburkhardt/Documents/D_Murger_Männer_Chor_Forschung/Scan_Mä
↪ nnerchor/Männerchor_Akten_1925-1945/Scan_Männerchor_PDF"
```

```
dest_folder = r"/Users/svenburkhardt/Documents/D_Murger_Männer_Chor_Forschung/Master
↪  arbeit/JPEG_Akten_Scans"


# Funktion aufrufen, um die Konvertierung durchzuführen
convert_pdf_to_jpg(src_folder, dest_folder)

```

## A.2 Tagging in Transkribus

Transkribus und seine Modelle unterstützen nicht nur beim Transkribieren der Texte, sondern erlauben auch das Taggen von *Named Entities*. Für die vorliegende Arbeit sind dabei besonders Personen, Orte, Organisationen und Daten relevant. Um hierfür ein stringentes Verfahren zu entwickeln, wurden die Tags wie folgt definiert:

### A.2.1 Strukturelle Tags

**abbrev**

Mit dem Tag **abbrev** werden alle Abkürzungen getaggt, die für eine eindeutige Entität stehen.

☞ **Beispiel 1**: Dr., Prof., St., Hr., Frl., Dipl.-Ing., etc.

☞ **Beispiel 2**: Organisationskürzel, wenn sie eindeutig sind:
`<abbrev> V.D.A. </abbrev>`.

☞ **Beispiel 3**: Falls eine dazugehörige Entität vorhanden ist, wird die Abkürzung getaggt und wird gleichzeitig als zugehörige Entität getaggt:

`<person> <abbrev> Dr. </abbrev> Weiss </person>`

**unclear**

Mit dem Tag **unclear** werden unleserliche oder schwer entzifferbare Textstellen markiert.

☞ **Beispiel 1**: Unklare Zeichen oder fehlende Buchstaben:

`Er wohnte in <unclear> [...] <unclear>`.

☞ **Beispiel 2**: Teilweise lesbare Wörter:

`<place> Frei <unclear> [...] <unclear> <place>`.

**sic**

Mit dem Tag `sic` werden Wörter markiert, die im Originaltext in einer falschen oder ungewöhnlichen Schreibweise geschrieben wurden.

☞ Beispiel 1: Veraltete oder falsche Schreibweisen:

`<sic> daß </sic>` für dass mit tz.

☞ Beispiel 2: Offensichtliche Tippfehler, wenn sie im Originaltext so vorkommen:

`Wir haben <sic> einen </sic> grosse Freude.`

☞ Beispiel 3: Falls eine Korrektur notwendig ist, kann sie als Kommentar ergänzt werden.

### A.2.2 Inhaltliche Tags

`person`

Mit dem Tag `person` sollen alle Strings getaggt, die eine direkte Zuordnung einer Person ermöglichen.

☞ **Beispiel 1**: Vereinsführer, Alfons, Zimmermann, Alfons Zimmermann, Z. A. Zimmermann, Herr Zimmermann, Herr Alfons Zimmermann, etc.

☞ **Beispiel 2**: Funktionen wie Oberlehrer, Chorleiter, etc. Wenn Ort, Name oder Organisation bekannt sind. Eine Person kann sowohl mit ihrem Namen als auch ihrer Funktion (wie Dirigent) getaggt werden. Aus der Korrespondenz ist in der Regel eine zugehörige Organisation ersichtlich, mit deren Verknüpfung eine namentlich nicht genannte Person identifiziert werden könnte.

`signature`

Mit dem Tag `signature` werden alle Strings getaggt, die eine handschriftliche Unterschrift darstellen. Der Tag `signature` ist nahezu deckungsgleich mit dem Tag `person`. Er dient zur **graduellen Unterscheidung**, ob ein Name im Fliesstext als gesichert leserlich oder handschriftlich als Signatur vorliegt.

☞ **Beispiel 1**: Eindeutig lesbare Signaturen werden direkt getaggt:

`<signature> A. Zimmermann </signature>`.

☞ **Beispiel 2**: Teilweise unleserliche Signaturen werden mit dem Tag `unclear` innerhalb von `signature` markiert:

`<signature> R. We <unclear> [...] </unclear> </signature>`.

☞ **Beispiel 3**: Wenn nur ein Teil des Namens lesbar ist, aber eine Identifikation unsicher bleibt, sollte die Unterschrift vollständig im Tag `unclear` innerhalb von `signature` stehen:

`<signature> <unclear> etwas unleserliches </unclear> </signature>`.

☞ **Beispiel 4**: Wenn eine Signatur einer bekannten Person zugeordnet werden kann, aber nicht vollständig lesbar ist, bleibt die Signatur erhalten und wird **ohne** den Tag `person` zu verwenden:

`<signature> A. Zimm <unclear> [...] </unclear> </signature>`.

☞ **Beispiel 5**: Wenn eine Unterschrift vollständig transkribiert wurde und die Person bekannt ist, wird sie nur mit `signature` getaggt, **ohne** den Tag `person` zu verwenden:

`<signature> Alfons Zimmermann </signature>`.

`organization`

Mit dem Tag `organization` werden alle Strings getaggt, die eine direkte Zuordnung einer Organisation ermöglichen.

☞ **Beispiel 1**: Männerchor Murg, Verein Deutscher Arbeiter (V.D.A.), Murgtalschule, etc.

☞ **Beispiel 2**: Abkürzungen, wenn sie eine Organisation eindeutig bezeichnen, z.B. V.D.A., NSDAP, STAGMA, etc.

`place`

Mit dem Tag `place` werden alle Strings getaggt, die sich auf einen geografischen Ort beziehen.

☞ **Beispiel 1**: Murg (Baden), Freiburg, Berlin, Murgtal, Schwarzwald, etc.

☞ **Beispiel 2**: Orte mit näherer Bestimmung, z.B. „bei Berlin“, „im Murgtal“ werden getaggt:

`<place> im Murgtal</place>`.

`date`

Mit dem Tag `date` werden alle expliziten und implizierten Datumsangaben markiert.

☞ **Beispiel 1**: 29.05.1936

☞ **Beispiel 2**: 29. Mai 1936

☞ **Beispiel 3**: den 29. d. Mts.:

```
<date when=„29.05.1936"> den 2. <abbrev>  d. Mts. </abbrev> </date>
```

`event`

Mit dem Tag `event` werden expliziten und implizierten Ereignisse markiert. Diese Ereignisse haben einen zeitlichen oder räumlichen Bezug, und können benannt werden. Dazu zählen:

☞ **Beispiel 1**: „Jubiläumskonzert“

☞ **Beispiel 2** „Gründung des Vereins“

☞ **Beispiel 2**„Kriegsausbruch“ oder „Kriegsende“

Konzepte, die nicht klar in den Texten benannt werden, wie beispielsweise die Suche nach einem Dirigenten, können nicht immer Ereignis getaggt werden. Sie sollen später aber in der Datenbank implementiert werden.

## A.3 Prompt der LLM Vorverarbeitung

```
prompt = f"""
Systemrolle:
Du bist ein spezialisiertes XML-Annotationstool für historische Transkribus-Dokumente.

Aufgabe:
Analysiere das gesamte PAGE-XML-Dokument. Extrahiere Entitäten aus dem Unicode-Text aller <TextLine>-Elemente und füge strukturierte `custom="..."`-Attribute hinzu.

Strikte Regeln:

1. Dokumentanalyse:
- Verarbeite ausschliesslich <TextLine>-Elemente.
- Verwende nur <Unicode>-Inhalte als Eingabetext.

2. Globale Personenerkennung:
- Erkenne Personen (inkl. Titel, Vorname, Nachname).
- Speichere `offset` und `length` für jede erkannte Person pro TextLine.
- Verwende **immer dieselben Offsets** bei wiederholten Nennungen im Dokument.

3. Empfängerkennung (`recipient`):
- Der Kopfbereich endet an der ersten komplett leeren TextLine.
- Erkenne dort Anreden (z. B. „Herr", „Frau", „Sehr geehrter Herr ...").
- Verknüpfe Anrede mit passender Person und annotiere mit:
`recipient {{offset:X; length:Y;}}`

4. Autorenkennung (`author`):
- Der Fussbereich beginnt nach der letzten Grussformel (z. B. „Mit freundlichen Grüssen").
- Namen → `author {{offset:X; length:Y;}}`.
- Funktion (z. B. „Chorleiter") → `role {{offset:X; length:Y;}}`.

 5. Ort- und Datumsannotation:
- **Absendeort** (creation_place) und **Erstellungsdatum** (creation_date):
    zusätzlich zu den Tags place und date hinzu:
    creation_place {{offset:X; length:Y;}} und creation_date {{offset:X; length:Y; when:TT.MM.JJJJ;}}.
- **Empfangsort** (recipient_place):
    Füge im Empfänger-Block die passende Zeile mit:
    place {{offset:X; length:Y;}}.

6. Entitäten pro Zeile (in dieser Reihenfolge):
```

```
Füge **ein** Attribut `custom="..."` ein mit nur den tatsächlich erkannten Entitäten:


person {{offset:X; length:Y;}}
recipient {{offset:X; length:Y;}}
author {{offset:X; length:Y;}}
organization {{offset:X; length:Y;}}
place {{offset:X; length:Y;}}
date {{offset:X; length:Y; when:TT.MM.JJJJ;}}
role {{offset:X; length:Y;}}
event {{offset:X; length:Y;}} → optional mit when:TT.MM.JJJJ;

Hinweise:
- Füge **nur tatsächlich vorhandene Entitäten** ein.
- Keine Platzhalter.
- Format für `date` und `event` (falls Datum erkennbar): `when:TT.MM.JJJJ;`
- Mehrzeilige Events (z. B. bei Bindestrich am Ende oder fortgeführtem Satz) erhalten dieselbe `event`-Annotation in allen betroffenen Zeilen.

6. XML-Regeln:
- **Verändere nur** `custom`-Attribute innerhalb von `<TextLine>`.
- Belasse alle anderen XML-Strukturen vollständig unverändert.

7. Ausgabe:
- Gib ausschliesslich ein vollständiges, wohlgeformtes XML zurück.
- Kein Freitext, kein Kommentar, kein Markdown.

Beispielausgabe:
<?xml version="1.0" encoding="UTF-8"?>
<PcGts xmlns="http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15">
<Page imageFilename="dummy.jpg" imageWidth="1000" imageHeight="1000">
    <TextRegion id="r1">
    <TextLine id="tl1" custom="place {{offset:0; length:7;}} creation_place {{offset:0; length:7;}} date {{offset:8; length:9; when:28.05.1942;}} creation_date {{offset:8; length:9; when:28.05.1942;}}">
        <Coords points="0,0 100,0 100,10 0,10"/>
        <TextEquiv><Unicode>München 28.V.1942</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl2" custom="recipient {{offset:7; length:5;}} person {{offset:7; length:5;}} place {{offset:15; length:6;}} recipient_place {{offset:15; length:6;}}">
        <Coords points="0,20 100,20 100,30 0,30"/>
        <TextEquiv><Unicode>Lieber Otto, Berlin</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl3" custom="event {{offset:24; length:38; when:28.05.1942;}} place {{offset:42; length:7;}}">
        <Coords points="0,40 100,40 100,50 0,50"/>
        <TextEquiv><Unicode>Heute abend fand ein Konzert im Opernhaus in München statt, und ich</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl4" custom="organization {{offset:43; length:28;}} place {{offset:66; length:16;}}">
        <Coords points="0,60 100,60 100,70 0,70"/>
        <TextEquiv><Unicode>lauschte den himmlischen Stimmen des Männerchors Hintertuüpfingen eV.</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl5" custom="organization {{offset:34; length:3;}} organization {{offset:40; length:18;}}">
        <Coords points="0,80 100,80 100,90 0,90"/>
        <TextEquiv><Unicode>Das alles fand im Rahmen des WhW - des Winterhilfswerk statt.</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl6" custom="organization  {{offset:50; length:17;}} place {{offset:72; length:4;}} place {{offset:83; length:6;}}">
        <Coords points="0,100 100,100 100,110 0,110"/>
        <TextEquiv><Unicode>Ich hoffe wir sehen uns bald bei einem Auftritt des Männerchors Murg wieder, oder in Hänner?</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl7" custom="role {{offset:14; length:14;}} person {{offset:29; length:4;}}">
        <Coords points="0,120 100,120 100,130 0,130"/>
        <TextEquiv><Unicode>Grüss mir den Vereinsführer Asal,</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl8">
        <Coords points="0,140 100,140 100,150 0,150"/>
        <TextEquiv><Unicode>Alles Liebe,</Unicode></TextEquiv>
    </TextLine>
    <TextLine id="tl9" custom="author {{offset:6; length:17;}} person {{offset:6; length:17;}}">
        <Coords points="0,160 100,160 100,170 0,170"/>
        <TextEquiv><Unicode>Deine Lina Fingerdick</Unicode></TextEquiv>
    </TextLine>
    <!-- Neue Zeile für den Empfangsort -->
    <TextLine id="tl10" custom="salutation {{offset:0; length:2;}} recipient {{offset:3; length:13;}} address {{offset:18; length:21;}} place {{offset:41; length:4;}}">
        <Coords points="0,180 100,180 100,190 0,190"/>
        <TextEquiv>
        <Unicode>An Otto Bolliger, Adolf-Hitler Platz 1, Murg</Unicode>
        </TextEquiv>
    </TextLine>
    </TextRegion>
</Page>
</PcGts>

Hier ist das zu annotierende XML:
```

```
```
{xml_content}
```
"""
```