




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. Ziel ist es, dieses Archiv digital zugänglich, die beteiligten Personen sowie deren Netzwerke und dessen geographische Ausdehnung sichtbar zu machen.

Inhaltsverzeichnis

1	Einleitung	2
1.1	Ziel und Relevanz der Arbeit	2
1.2	Formulierung der Forschungsfrage	2
1.3	Aufbau der Arbeit	2
1.4	Geografischer und historischer Kontext	2
2	Forschungsstand und Forschungslücke	3
3	Korpus	5
3.1	Quellen	6
3.1.1	Quellentradierung	6
3.1.2	Quellenbeschrieb	7
3.1.3	Sichtung & Kategorisierung in Akten	8
3.2	Digitalisierung der Quellen	9
3.3	Transkription	10
3.3.1	Tagging mit Transkribus und LLM	12
4	Methodisches Vorgehen	14
4.1	LOD – Linked Open Data	14
4.1.1	Protégé	15
4.1.2	GraphDB	16
4.1.3	LOD-Ontologie	16
4.2	Wikidata	17
4.3	GeoNames	18
4.4	Transkriptionen (Methodenvergleich)	19
4.4.1	Tesseract	19
4.4.2	LLM	19
4.4.3	Transkribus	20
4.5	Large Language Models	22
4.6	Msty	23
4.7	Alphabet – Gemini	23
4.8	Anthopic – Claude	23
4.9	OpenAI – ChatGPT	23

4.10	Nodegoat	23
4.11	Webtool	25
5	Pipeline	27
5.0.1	Übersichtsgrafik der Pipeline	27
5.1	Vorverarbeitung	28
5.2	Hauptmodul – Transkribus_to_base	30
	extract_metadata_from_xml (...)	32
	get_document_type (...)	32
	extract_text_from_xml (...)	33
	extract_custom_attributes (...)	33
5.3	Module im Detail	38
5.3.1	document_schemas.py	38
	Person	38
	Organization	39
	Place	39
	Event	39
	BaseDocument	40
	Documenttype	40
5.3.2	__init__.py	41
5.3.3	Person_matcher.py	42
5.3.4	Assigned_Roles_Module.py	43
5.3.5	place_matcher.py	44
5.3.6	organization_matcher.py	44
5.3.7	letter_metadata_matcher.py	44
5.3.8	type_matcher.py	44
5.3.9	unmatched_logger.py	48
5.4	KEINE AHNUNG WAS DIE HIER MACHEN	49
5.4.1	validation_module.py	49
5.4.2	validation_module.py	49
5.4.3	test_role_schema.py	49
5.4.4	llm_enricher.py	49
5.4.5	enrich_pipeline.py	49

6	Analyse & Diskussion der Ergebnisse	49
6.1	Visualisierung auf der VM	49
7	Fazit und Ausblick	49
7.1	Zusammenfassung der zentralen Erkenntnisse	49
7.2	Methodische Herausforderungen und Lösungen	49
7.3	Ausblick auf zukünftige Forschung und mögliche Erweiterungen der Daten- bank	49
7.4	Digitale Erfassung und Strukturierung der Quellen	50
7.4.1	Gliederung in Akten	50
7.4.2	Digitalisierung und Transkription	50
7.5	Tagging in Transkribus	50
7.6	Digitalisierungsprozess und Herausforderungen	50
A	Anhang	55
A.1	PDF_to_JPEG.py	55
A.2	Tagging in Transkribus	56
A.2.1	Strukturelle Tags	56
A.2.2	Inhaltliche Tags	57

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 1853¹ 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.

¹gemeinde_murg_geschichte_nodate.

2 Forschungsstand und Forschungslücke

Die vorliegende Arbeit knüpft an zwei Vorarbeiten an, die in den Jahren 2018 und 2022 am Departement Geschichte der Universität Basel durchgeführt werden. In zwei Transkribus-Seminaren werden erste Teilbestände der „Männerchor Akten 1925–1944“ erschlossen und in einem Korpus von 137 Einzeldokumenten zusammengeführt.² 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 Buchner³, Christian Hartmann⁴, Werner Haupt⁵, Christoph Rass⁶, Georg Tessin⁷ und Christian Zentner⁸ zu nennen.

Darüberhinaus werden eigene Recherchen in den Beständen des *Bundesarchivs – Militärarchiv Freiburg*⁹ durchgeführt. Ergänzende Recherchen stammen aus den Suchlisten des *Deutschen Roten Kreuzes (DRK)*¹⁰. Hinzu kommen philatelistische Übersichts-Websites¹¹, die bei der Entzifferung von Briefmarken und Stempeln helfen. Absolut essentiell für den Erfolg dieser Recherchen sind Citizen-Science-Foren¹². Sie ergänzen und validieren eigene Forschung.

Parallel zur inhaltlichen Erschliessungen entsteht 2022 eine erste digitale Storymap mit *ArcGIS*, die zentrale Ergebnisse des Projekts öffentlich zugänglich macht. Grundlage bil-

²burkhardt_feldpost_2022.

³buchner_handbuch_1989.

⁴hartmann_wehrmacht_2010.

⁵haupt_buch_1982.

⁶rass_deutsche_2009.

⁷tessin_verbande_1977.

⁸zentner_illustrierte_1983.

⁹hollmann_freiburg_2025.

¹⁰reuter_drk_2025.

¹¹noauthor_feldpost_nodate.

¹²vor Allem werden verwendet: *Forum der Wehrmacht* (hermans_forum_nodate) und das *Lexikon der Wehrmacht* (altenburger_lexikon_nodate).

det 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 Zeichen¹³ 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)¹⁴, 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-Partnern¹⁵.

¹³haupt_buch_1982.

¹⁴richard_linking_2022.

¹⁵richard_linking_2022.

Für eine Einordnung zu historischen Netzwerkanalysen sei auf Gamper&Reschke¹⁶ verwiesen. Der Sammelband *Knoten und Kanten III* verdeutlicht, dass die historische Netzwerkanalyse zwar von einem interdisziplinär etablierten Methodenkanon profitiert, jedoch nach wie 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 systematischen 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 dementsprechend 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 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.

¹⁶gamper_knoten_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 patentiert¹⁷. 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.

¹⁷noauthor_heftstreifen_2023.

3.1.2 Quellenbeschreibung

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.

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.

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,

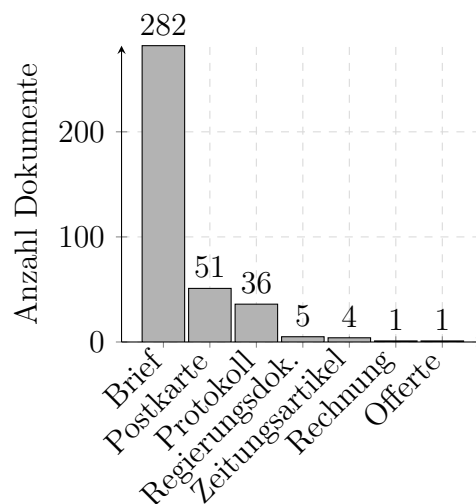


Abbildung 1: Verteilung der Dokumententypen im untersuchten Bestand (150 Akten – 381 Seiten).

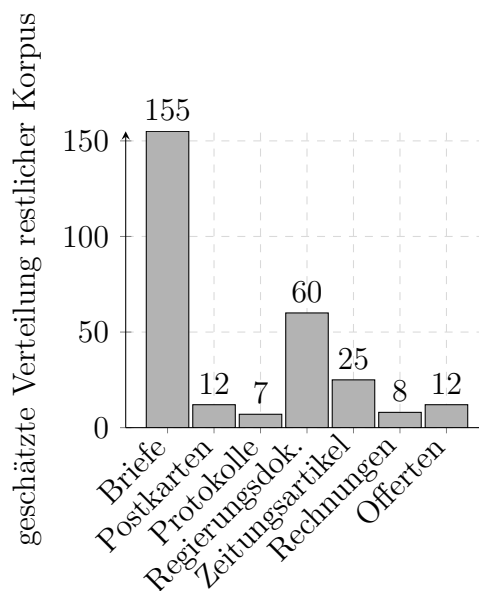


Abbildung 2: geschätzte Verteilung der Dokumententypen im restlichen Bestand.

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*“ zweimal gesichtet und gelesen. Beim ersten Durchgang werden explorativ nach zusammenhängenden Unterlagen gesucht, die im Anschluss zu einer Akte 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 Aktenübersicht¹⁸ im CSV-Format erstellt. Sie setzt sich zusammen aus der Aktennummer, die die Reihenfolge innerhalb 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 interne 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 Transkribus-Dokument-ID abgelöst. Beide IDs werden zu besserer 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

¹⁸genannt 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 analoge 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 Apple¹⁹ 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

¹⁹vgl. [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.²⁰ 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 Baselines²¹ 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.

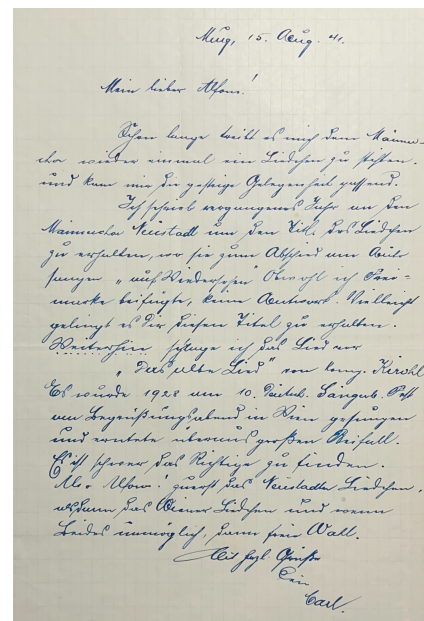


Abbildung 3: Beispiel für handschriftlichen Text in Akte_076

Ausgangspunkt für dieses Projekt ist das generischen Modell *The German Giant I*, das mit einer CER²² 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

²⁰[burkhardt_githubpdf_to_jpegpy_2025](#).

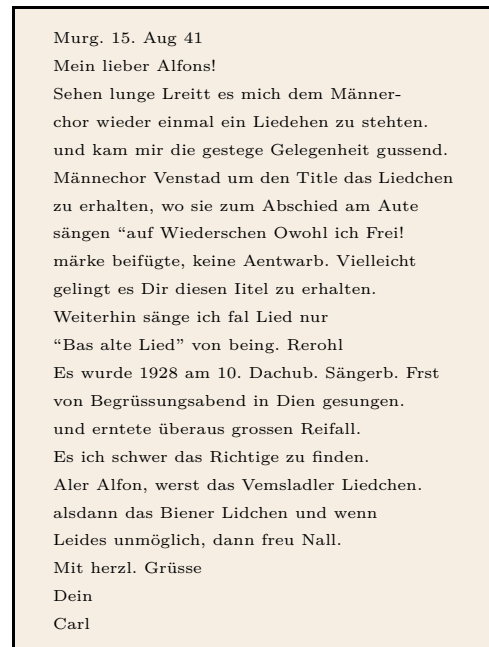
²¹Baselines = Schriftausrichtung

²²**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 Groundtruth-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.

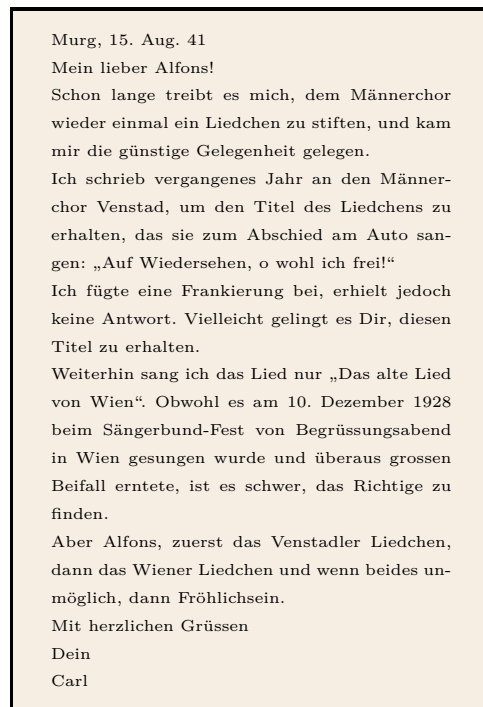
Die automatische Transkription wird in [Abbildung 4](#) dargestellt, die überarbeitete LLM-Version folgt in [Abbildung 5](#).

The image shows a rectangular box with a light beige background and a thin black border, containing a transcribed letter. The text is in German and appears to be a letter from Carl to Alfons. The transcription is as follows:

Murg. 15. Aug 41
Mein lieber Alfons!
Sehen lunge Leitt es mich dem Männer-
chor wieder einmal ein Liedchen zu stehen.
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 herz. Grösse
Dein
Carl

Abbildung 4: LLM-Version von [Abbildung 3](#)

Die so entstehende Groundtruth wird für das Training des Modells ([ModellID: 287793](#))²³ 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

²³ ([burkhardt_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 [4.4](#)).

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 verbinden²⁴. 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.²⁵ 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-

²⁴garoufallou_metadata_2020.

²⁵smith_owl_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 URIs²⁶ geführt werden, als dasselbe reale Objekt²⁷ und ermöglicht so eine präzise Zusammenführung von Informationen — ein Grundpfeiler für die Interoperabilität im Semantic Web. Die OWL-Spezifikation baut auf RDF²⁸ 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 vor²⁹, die sich im Grad ihrer Ausdruckstärke unterscheiden.³⁰ 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.³¹ 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.

²⁶Abk. **URI** Uniform Resource Identifier

²⁷`smith_owl_2004`.

²⁸Abk. **RDF** Resource Description Framework

²⁹OWL Lite, OWL DL und OWL Full

³⁰`smith_owl_2004`.

³¹`garoufallou_metadata_2020`.

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 grosse Grundlage des Global und damit aufwendig. Der Fokus der Arbeit verschiebt sich dementsprechend von der Ontologieentwicklung auf die Extraktion von Entitäten.

Der bestehende Datensatz ist zu klein, um eine umfangreiche Ontologie lohnend zu machen. Hinzu kommen externe Quellen, und deren Zugänglichkeit. Zuverlässige Quellen für Informationen über militärische Einheiten und deren Feldpostnummern sind das „Forum



Abbildung 6: Ausschnitt der TTL-Ontologie.

der Wehrmacht“³² und der „Suchdienst des DRK“³³. In beiden Fällen liegen die Daten jedoch nicht als LOD vor, sondern im Forum als einfache Strings und beim Deutschen Roten Kreuz als OCR-PDF³⁴ historischer Suchlisten aus der Nachkriegszeit. Ein manuelles Recherchieren dieser Daten scheint zu diesem Zeitpunkt den Rahmen der Arbeit zu sprengen. Die in diesem Schritt geleistete Vorarbeit beim Sortieren und Klassifizieren von Entitäten, besonders in Verknüpfung mit selbst erstellten Wikidata-Klassen wird in späteren Prozessschritten wieder aufgegriffen³⁵.

4.2 Wikidata

*Wikidata*³⁶ 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.“*³⁷

Wikidata entspricht, ebenso wie das nachfolgend beschriebene GeoNames, den FAIR-Prinzipien: Die Daten sind **F**indable und **A**ccessible, **I**nteroperable und **R**eusable³⁸.

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 Arbeitszeits 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

³²`altenburger_lexikon_nodate.`

³³`reuter_drk_2025.`

³⁴OCR = Optical Character Recognition

³⁵siehe Abschnitt [Nodegoat](#)

³⁶`noauthor_wikidata_nodate.`

³⁷`martinez_comparative_nodate.`

³⁸`wilkinson_fair_2016.`

klar kommuniziert. Mit regidem Löschen neuer Einträge wird die Verlässlichkeit und den 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.

4.3 GeoNames

Ebenso wie Wikidata bietet *GeoNames*³⁹ 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. Alle Einträge sind in neun Feature-Klassen und über 600 spezifische Feature-Codes kategorisiert. 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-System⁴⁰ 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.

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.

³⁹`noauthor_geonames_nodate.`

⁴⁰`noauthor_wgs84_nodate.`

4.4 Transkriptionen (Methodenvergleich)

4.4.1 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 ein Open-Source-Projekt zur Texterkennung in Bilddateien, die OCR-Engine Tesseract. Tesseract wird seit den 1980er-Jahren entwickelt, zunächst von Hewlett-Packard, später von Google weitergeführt, und ist über GitHub öffentlich zugänglich.⁴¹

Die Software basiert seit Version 4 auf einem LSTM-basierten⁴² neuronalen Netzwerk, das besonders bei der Erkennung von zusammenhängenden 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.⁴³

Vorbereitend für den Einsatz von Tesseract müssen alle gescannten PDF in das JPEG Format umgewandelt werden, wofür ein kurzes Python-Script verwendet wird⁴⁴. 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 (`-psm`) konnte keine zufriedenstellende Texterkennung erzielt werden.

Tesseract wurde daher nicht weiterverwendet.

4.4.2 LLM

Analog zum Einsatz des OCR-Systems Tesseract⁴⁵ stellt die Integration von Large Language Models (LLMs) von Beginn an einen zentralen Bestandteil der Projektkonzeption dar. Aus diesem Grund wird in einer frühen Phase auch der Einsatz von LLMs, spezifisch ChatGPT, bei der Transkription der Unterlagen erprobt.

Der Einsatz von LLMs wie ChatGPT für die Transkription historischer Quellen erweist

⁴¹weil_tesseract-ocrtesseract_2025.

⁴²beck_review_2020.

⁴³weil_tesseract-ocrtesseract_2025.

⁴⁴burkhardt_githubpdf_to_jpegpy_2025.

⁴⁵siehe Abschnitt ??

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 Stellen⁴⁶ 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 Inkonsistenzen in der Promptstrukturierung und dem Kontext des LLMs führen, was wiederum direkten Einfluss auf die Verarbeitung der Unterlagen hat.

Zum Zeitpunkt der explorativen Nutzung stellt das zentrale 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.

4.4.3 Transkribus

Transkribus ist eine webbasierte Plattform zur automatisierten Handschrifterkennung (HTR) und Texterkennung (OCR), die sich seit ihrer Entwicklung im EU-Projekt READ

⁴⁶Beispielsweise durch das Einfügen von „[...]“

(Recognition and Enrichment of Archival Documents)⁴⁷ als Standardwerkzeug in den digitalen Geschichtswissenschaften etabliert hat⁴⁸. 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 Abkürzungsaufösungen oder Listenstrukturen nicht zuverlässig im XML ausgegeben. Informationen, die manuell innerhalb der Transkriptionsumgebung gepflegt wurden, gehen im strukturierten Export unter Umständen verloren. 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.

⁴⁷noauthor_recognition_nodate.

⁴⁸muhlberger_transkribus_2019.

Diese Einschränkungen wurden auch in aktuellen Studien festgestellt. So verweisen Capurro et al.⁴⁹ 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.

4.5 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.

⁴⁹capurro_experimenting_2023.

4.6 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 Msty⁵⁰ eingesetzt. Zu den zentralen Funktionen gehören parallele Chatinterfaces („Parallel Multiverse Chats“), eine flexible Verwaltung lokaler Wissensbestände („Knowledge Stacks“)⁵¹, 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.

Wichtig 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 klares Bild, welches Modell sich am besten eignet. Beprobt werden die Folgenden Anbieter und Modelle:

4.7 Alphabet – Gemini

4.8 Anthropic – Claude

4.9 OpenAI – ChatGPT

4.10 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**⁵² eine praktikable und zugleich forschungsnahe Alternative eingeführt. Im Folgenden soll das Tool näher beschrieben, und ihre Anwendung für das Projekt erläutert werden.

Nodegoat ist eine webbasierte Plattform, die sich besonders in den Digital Humanities etabliert hat. Es unterstützt Forschende in der Modellierung, Verwaltung, Analyse und Visualisierung komplexer Datenbestände. Ein zentrales Merkmal von Nodegoat ist die

⁵⁰noauthor__msty__nodate.

⁵¹noauthor__msty__nodate.

⁵²kessels__nodegoat__2013.

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. Die Plattform folgt einem modularen Prinzip: Über das UI⁵³ 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:

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-Format⁵⁴. So lassen sich beispielsweise für **Persons** in den Sub-Objekten **Geburt** oder **Tod** das Datum und der Ort modellieren, sowie die Todesursache notieren.

⁵³Abk.: **UI** User-Interface

⁵⁴[thomson_geographic_2017](#).

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 Programmierschnittstelle (API) 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. Dies soll nachfolgend erläutert werden.

Kritisiert werden muss an Nodegoat die fehlende Dokumentation. Viele Informationen finden sich nur über Drittparteien⁵⁵ oder, wie unten geschildert, auf konkrete Nachfrage bei den Entwicklern. Der dann geleistete Support ist jedoch ausgesprochen zeitnah und umfassend.

4.11 **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.

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

Hierbei steht die Type-ID *11680* im Beispiel für den jeweiligen Modelltyp „Organisation“ und die Zeichenkette am Ende für die eindeutige Nodegoat-ID des Objekts⁵⁶.

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

⁵⁵[gubler_nodegoat_nodate](#).

⁵⁶im Beispiel eine gekürzte Nodegoat-ID

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 (Kessels und van Bree) 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, 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.

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, indem sie die logische Gliederung der Pipeline abbildet, wie sie im weiteren Verlauf des Kapitels analysiert wird. Sie dient als Referenz für nachfolgende Module, wo sie als schematische Zeichnung bei der Einordnung der verschiedenen Verarbeitungsschritte und Modulabhängigkeiten zum Einsatz kommt. 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 der Forschungsumgebung 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 ein 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](#).

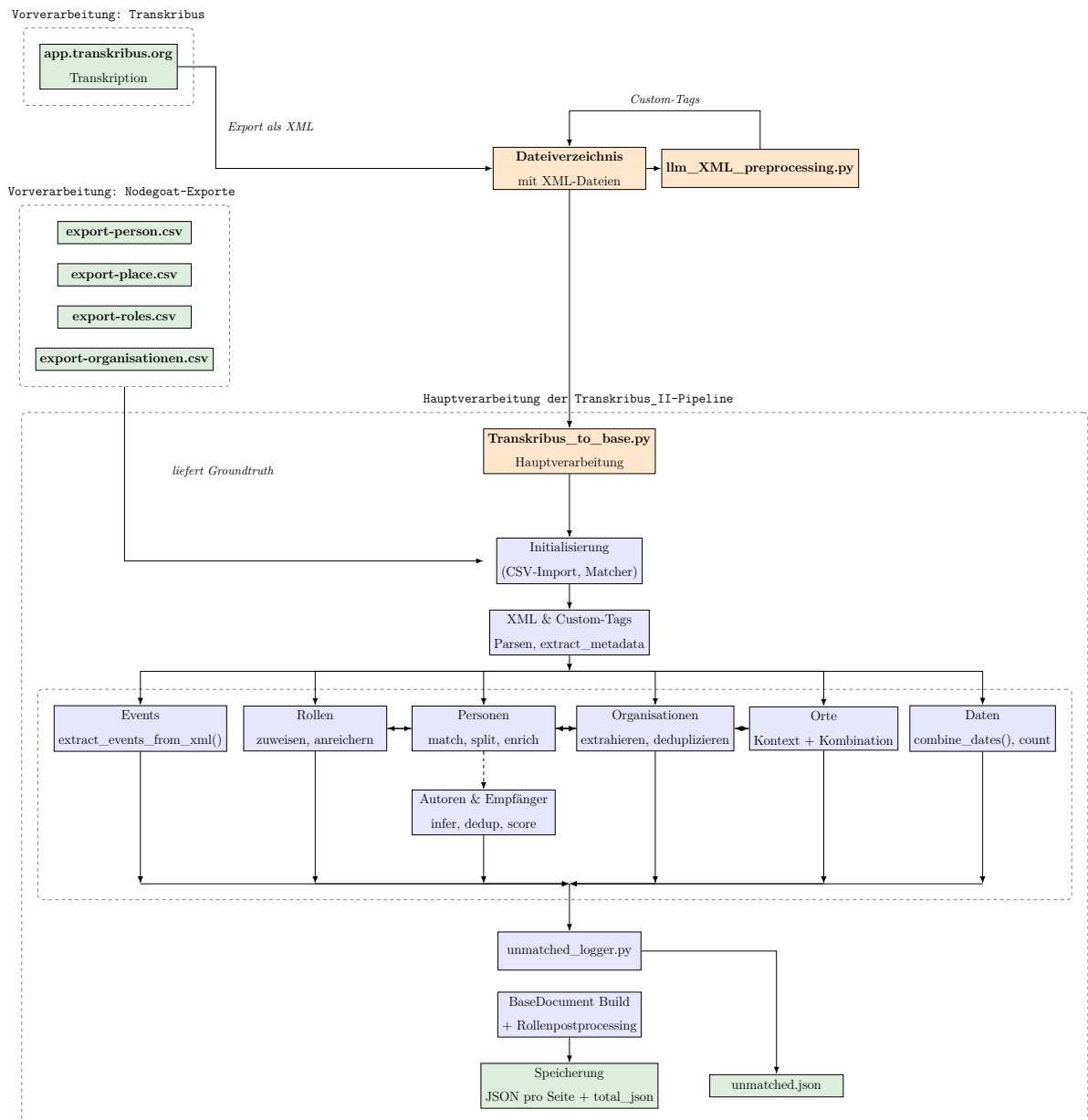


Abbildung 7: Übersicht der gesamten XML-to-JSON-Pipeline

5.1 Vorverarbeitung

Neben Transkribus, das bei der Transkription einen elementaren Schritt in der Vorverarbeitung aller Dokumente darstellt, müssen die Seiten noch weiter vorbereitet werden, um detaillierte Ergebnisse zu gewinnen. Die Kernherausforderung der vorliegenden Arbeit ist die Named Entity Recognition. Wie bereits im Abschnitt ?? ausgeführt, werden viele Inhalte bereits während des Transkriptionsprozesses manuell getaggt. Aufgrund der Menge an Unterlagen und des begrenzten zur Verfügung stehenden Zeitrahmens wird für das Projekt auch eine zweite Verarbeitungsform gewählt.

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.

Zunächst wird mit der Funktion `get_api_client()` eine Verbindung zur OpenAI-Programmierschnittstelle aufgebaut. 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, 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 Sprachmodells ist eine vollständige XML-Datei, die sämtliche bestehenden 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

1. Initialisierung und Pfadlogik

Die Initialisierungsphase des Skripts⁵⁷ 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` für den Vergleich von ähnlichen Strings. Zusätzlich wird das aktuelle Datum mit Zeitstempel generiert und als Konsolenoutput ausgegeben. Das unterstützt die Nachvollziehbarkeit von Verarbeitungsläufen und das Debugging in grösseren Verarbeitungsszenarien.

⁵⁷bis ca. Line 250

Das Verzeichnis der Skriptdatei wird mithilfe des `pathlib`-Moduls identifiziert. Anschließend wird von dort ausgehend das Projektwurzelverzeichnis nachvollzogen. Diese dynamisch bestimmte Wurzel fungiert als zentraler Referenzpunkt für die Auflösung aller nachfolgenden Verzeichnispfade, absolute Pfadangaben werden vollständig vermieden. Das Projekt ist so organisiert, dass sämtliche zentralen Datenverzeichnisse, Ressourcen und Modulstrukturen relativ zur Wurzel definiert und im Code programmatisch zugänglich gemacht sind. Dazu gehören beispielsweise die Unterverzeichnisse `Data` für CSV-basierte Groundtruth-Informationen, `Module` für modulare Verarbeitungsfunktionen sowie `Transkribus_test_In` und `Transkribus_test_Out` als standardisierte Ein- und Ausgabeverzeichnisse für XML-Dateien und strukturierte JSON-Outputs.

Zur Gewährleistung der Modulverfügbarkeit wird dem globalen Importpfad das Verzeichnis `Module` über `sys.path.insert` hinzugefügt. Dadurch lassen sich alle enthaltenen Funktions- und Klassendefinitionen zentral importieren. Die einzelnen Komponenten sind in modularisierten Unterdateien wie `Person_matcher.py`, `place_matcher.py` und `document_schemas.py` abgelegt. Diese Struktur fördert die klare Trennung thematischer Verantwortlichkeiten innerhalb des Codes.

Darüber hinaus erfolgt die Definition aller relevanten Ein- und Ausgabepfade. Der Pfad `TRANSKRIBUS_DIR` verweist auf das Eingabeverzeichnis für XML-Dateien, während `OUTPUT_DIR` die Ausgabe der JSON-Dateien sowie untergeordnete Strukturen für nicht gematchte Entitäten, Logdateien und CSV-Dumps aufnimmt.

Die Initialisierungslogik umfasst ferner das standardisierte Einlesen der Groundtruth-Dateien aus den jeweiligen Nodegoat-Exporten, die Zentral im Ordner `/Data/Nodegoat_Export` gespeichert sind. Dabei werden die dort hinterlegten Informationen aus den CSV-Dateien geladen, in `pandas.DataFrames` überführt und durch Fehlerbehandlung in Form von `tryexcept`-Blöcken abgesichert. Eine Konsolenausgabe informiert über Anzahl und Status der geladenen Einträge.

Im Anschluss daran wird überprüft, ob das deutsche Sprachmodell `de_core_news_sm` von `spaCy` verfügbar ist und gegebenenfalls auf ein Fehlen hingewiesen. Wichtig ist die Abfrage eines gültigen API-Schlüssels für das OpenAI-Modul. Ist dies nicht der Fall wird der nachgelagerte 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. Sie dient als Referenz für alle nachfolgenden Matching- und Deduplikationsvorgänge.

Abschliessend stehen zwei Hilfsfunktionen bereit, um neu erkannte Personen in die bestehende CSV-Struktur zu überführen. Eine Duplikatsprüfung auf Basis unscharfer Namensähnlichkeit wird durch `Rapidfuzz` realisiert und stellt sicher, dass nur bisher unbekannte Einträge ergänzt werden.

Die Initialisierungs- und Pfadlogik legt damit den Grundstein für eine skalierbare, reproduzierbare und systematisch strukturierte Weiterverarbeitung aller Eingabedaten.

2. Extraktion von Struktur und Fliesstext

Die Verarbeitung einer einzelnen Transkribus-Seite beginnt mit der systematischen Analyse der zugrundeliegenden XML-Datei. Ziel dieses Abschnitts ist die strukturierte Extraktion zentraler Informationen, die als Grundlage für die weitere Anreicherung und Validierung dienen. Dabei wird zwischen technischen Metadaten, transkribiertem Fliesstext und semantisch annotierten Entitäten (in sogenannten `custom`-Tags) unterschieden. Die nachfolgend beschriebenen Funktionen sind für die Extraktion dieser drei Bereiche zuständig.

`extract_metadata_from_xml (...)` Diese Funktion extrahiert maschinenlesbare Metadaten aus dem `< TranskribusMetadata>`-Block der XML-Datei. Dazu gehören die eindeutige Dokumenten-ID (`docId`), die Seiten-ID (`pageId`), die Transkriptions-ID (`tsid`), sowie die zugehörige Bildreferenz und die XML-Quelldatei. Diese Informationen werden zentral in einem Metadaten-Dictionary gespeichert, das später in das Attributfeld des JSON-Objekts übernommen wird. Sie bilden die technische Grundlage für die spätere Identifikation und Reproduzierbarkeit jeder Seite.

`get_document_type (...)` Anhand des Dateinamens und optionaler Muster im Text wird eine heuristische Klassifikation des Dokumenttyps vorgenommen. Typische Kategorien sind etwa „Brief“, „Postkarte“, „Protokoll“ oder „Zeitungsartikel“. Die Funktion verwendet reguläre Ausdrücke oder Schlüsselwörter zur Erkennung und überführt das Ergebnis in das Feld `document_type`. Diese Klassifikation ist insbesondere für die spätere Netzwerk- und Objektanalyse im Backend von Bedeutung, da sie das Dokument struktu-

rell verortet.

extract_text_from_xml (...) Der transkribierte Fliesstext wird aus den Elementen `<TextEquiv>` bzw. `<Unicode>` extrahiert. Dabei werden alle Textinhalte zeilenweise aus den `TextLine`-Knoten der XML-Datei gelesen und zu einem zusammenhängenden String zusammengefügt. Jede Zeile wird durch einen Zeilenumbruch (`\n`) getrennt, um den späteren kontextuellen Zugriff auf Zeilenpositionen (z.B. bei der Erwähnung von Personen oder Rollen) zu ermöglichen. Der extrahierte Text dient als Grundlage für regelbasierte Erkennungsverfahren sowie für LLM-gestützte Analysen in späteren Verarbeitungsschritten.

extract_custom_attributes (...) Die semantische Annotation in den Transkribus-Dokumenten erfolgt über `custom`-Attribute, die bestimmten Textbereichen zugeordnet sind (z.B. `person{offset:42; length:14}`). Die Funktion `extract_custom_attributes()` iteriert über alle `TextLine`-Elemente und prüft, ob ein solches Attribut vorhanden ist. Ist dies der Fall, wird der betroffene Textabschnitt anhand der Offset-Angaben extrahiert und weiterverarbeitet. Die Extraktion erfolgt kategoriebezogen, wobei für jede Entitätstyp eigene Subfunktionen aufgerufen werden:

- **extract_person_from_custom (...)**: Diese Funktion extrahiert Personennamen, gegebenenfalls Titel (z.B. „Herr“, „Frau“) und mögliche Rollenbezeichnungen. Sie bereitet die Einträge für ein späteres Matching gegen die bekannte Personenliste vor. Neben Vor- und Nachnamen werden auch unvollständige Angaben übernommen und mit dem Attribut `needs_review=true` markiert, wenn kein eindeutiger Abgleich möglich ist.
- **extract_place_from_custom (...)**: Analog zur Personenerkennung werden Ortsnamen anhand der Offset-Daten aus dem Text extrahiert. Die Funktion übergibt die Einträge an eine vorbereitete `PlaceMatcher`-Instanz, die auf Basis von Groundtruth-Daten sowie ggf. externer APIs (GeoNames, Wikidata) einen Abgleich durchführt. Unsichere Orte werden ebenfalls mit `needs_review` versehen.
- **extract_organization_from_custom (...)**: Diese Funktion erkennt Organisationen (z.B. „Männerchor Murg“) in den XML-Anmerkungen. Sie liefert zunächst einen Roh-String, der in späteren Verarbeitungsschritten mit bekannten Organisationen abgeglichen wird.
- **extract_date_from_custom (...)**: Die Funktion erkennt Datumsangaben (z.B.

„1. Januar 1943“) und normalisiert sie bei Bedarf. In einem späteren Schritt werden identische Daten zusammengeführt und gezählt (`combine_dates()`), um Mehrfachnennungen zu erfassen.

Alle extrahierten Custom-Tags werden in einem strukturierten Dictionary gesammelt, das unter anderem folgende Schlüssel enthält: `"persons"`, `"roles"`, `"places"`, `"organizations"`, `"dates"`. Diese Struktur dient als zentrale Datenquelle für die nachfolgende Entitätenanreicherung, Deduplikation und Netzwerkanalyse.

Zweck: Parsen der XML-Datei und strukturierte Extraktion von Text, Metadaten und Custom-Tags.

- `extract_metadata_from_xml (...)`
- `get_document_type (...)`
- `extract_text_from_xml (...)`
- `extract_custom_attributes (...)` mit:
 - `extract_person_from_custom (...)`
 - `extract_place_from_custom (...)`
 - `extract_organization_from_custom (...)`
 - `extract_date_from_custom (...)`

3. Named Entity Recognition

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 (...)`

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 (...)`

5. 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.

6. 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 vorhandene 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 Origineleintrag 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`.⁵⁸ Letzterer gibt an, wie oft die Person im Transkript erwähnt wurde, unter Berücksichtigung einer kontextbasierten

⁵⁸Detailliert wird auf diese Logiken in [Person_matcher.py](#) eingegangen, sie sollen hier nur umrissen werden.

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 separat 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.

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

Hier steht eine menge schöner
Text

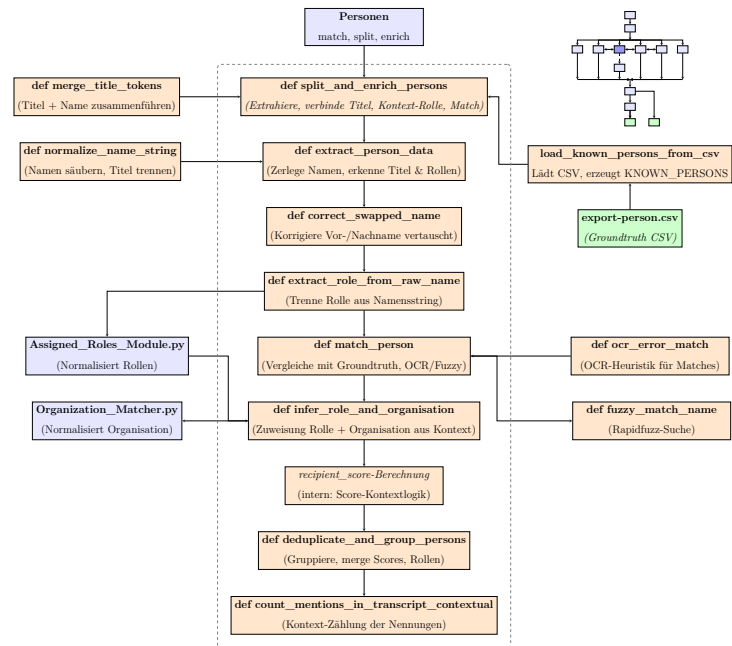


Abbildung 8:
Oben links: Prozessdiagramm für
Personen_matcher.py ,
Oben rechts: Pipelineübersicht

5.3.4 Assigned_Roles_Module.py

Hier steht eine menge schöner Text

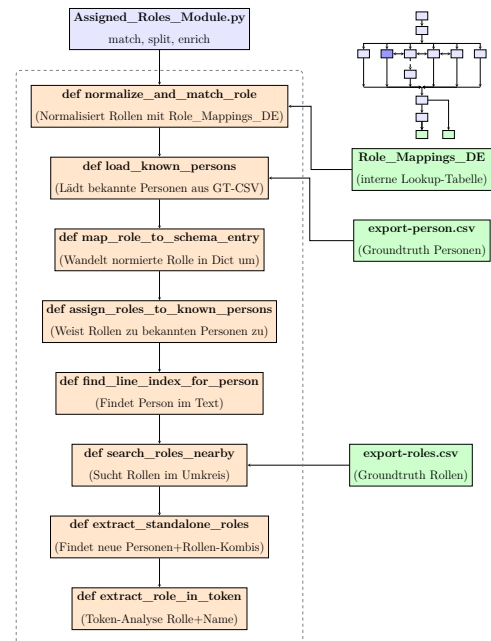


Abbildung 9:

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

5.3.5 place_matcher.py

Hier steht eine menge schöner Text

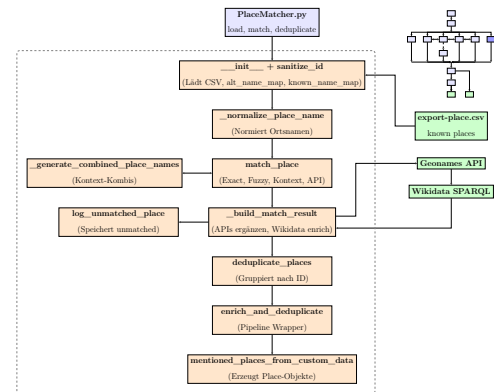


Abbildung 10:
Links: Prozessdiagramm für
`place_matcher.py`,
Rechts: Pipelineübersicht

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

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` kompati-

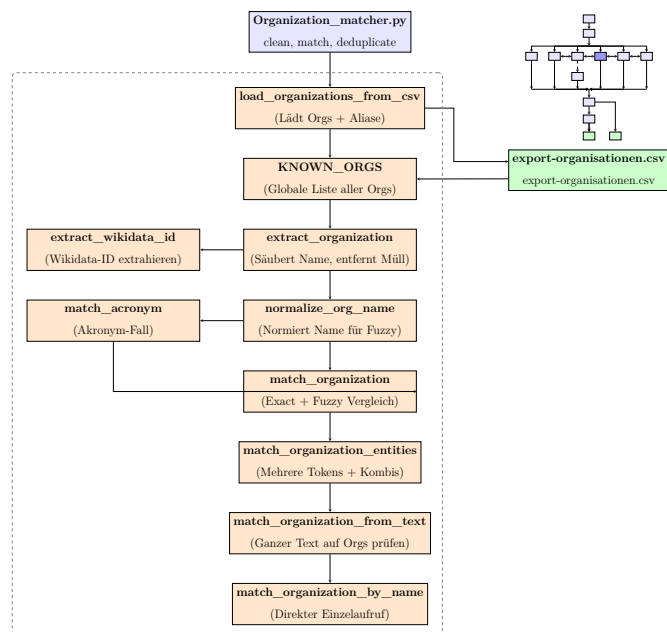


Abbildung 11: Links: Prozessdiagramm für
`Organization_matcher.py`,
Rechts: Pipelineübersicht

bel ist und Hauptnamen, alternative Bezeichnungen sowie Identifier enthält.

5.3.7 letter_metadata_matcher.py

5.3.8 type_matcher.py

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 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 zentrale 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 Events ein eher abstraktes Konstrukt sind, 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 7.5 erläutert. Lässt sich aus dem weiteren Kontext⁵⁹ ein vollständiges Datum erschliessen, kann dieses anschliessend in struk-

⁵⁹Zum Beispiel durch Hinweise im Seiteninhalt oder durch übergeordnete Informationen im Umfeld der Akte

turierter 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 regulärer Ausdrücke, 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 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.

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

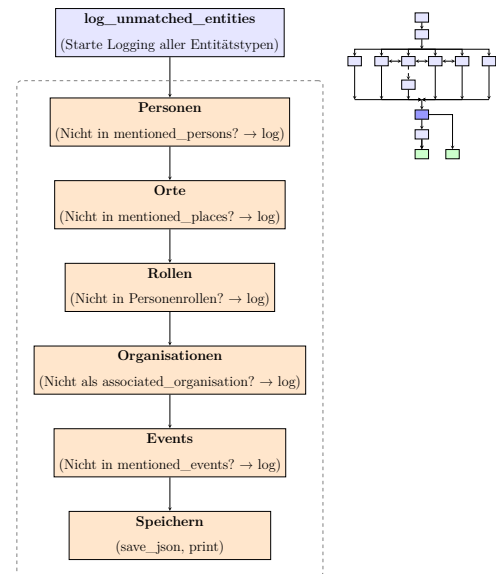


Abbildung 12:
Oben links: Prozessdiagramm für `unmatched_logger.py`,
Oben rechts: Pipelineübersicht

Gesamtüberblick zu erhalten.

5.4 KEINE AHNUNG WAS DIE HIER MACHEN

5.4.1 validation_module.py

5.4.2 validation_module.py

5.4.3 test_role_schema.py

5.4.4 llm_enricher.py

5.4.5 enrich_pipeline.py

6 Analyse & Diskussion der Ergebnisse

6.1 Visualisierung auf der VM

7 Fazit und Ausblick

7.1 Zusammenfassung der zentralen Erkenntnisse

7.2 Methodische Herausforderungen und Lösungen

7.3 Ausblick auf zukünftige Forschung und mögliche Erweiterungen der Datenbank

A Anhang

A.1 PDF_to_JPEG.py

```
1 import os
2 import fitz # PyMuPDF
3
4 def convert_pdf_to_jpg(src_folder, dest_folder):
5     # Überprüfen, ob der Zielordner existiert, und ihn ggf. erstellen
6     if not os.path.exists(dest_folder):
7         os.makedirs(dest_folder)
8
9     # Durchgehen durch alle Dateien im Quellordner
10    for root, dirs, files in os.walk(src_folder):
11        for file in files:
12            # Überprüfen, ob die Datei eine PDF-Datei ist
13            if file.lower().endswith(".pdf"):
14                # Vollständigen Pfad zur PDF-Datei erstellen
15                pdf_path = os.path.join(root, file)
16                # PDF-Datei öffnen
17                doc = fitz.open(pdf_path)
18                # Durch alle Seiten der PDF-Datei gehen
19                for page_num in range(len(doc)):
20                    page = doc[page_num]
21                    # Seite in ein PiMap-Objekt umwandeln (für die Konvertierung in
22                    ↪ JPG)
23                    pix = page.get_pixmap()
24                    # Dateinamen ohne Dateiendung extrahieren
25                    filename_without_extension = os.path.splitext(file)[0]
26                    # Ausgabedateinamen erstellen mit führenden Nullen für die
27                    # Seitennummer
28                    output_filename = f"{filename_without_extension}_S{page_num +
29                    ↪ 1:03d}.jpg"
30
31                    # Vollständigen Pfad zur Ausgabedatei erstellen
32                    output_path = os.path.join(dest_folder, output_filename)
33                    # Bild speichern
34                    pix.save(output_path)
35                    # PDF-Datei schliessen
36                    doc.close()
37
38                # Erfolgsmeldung ausgeben
39                print(f"{file} wurde erfolgreich umgewandelt und gespeichert
40                in {dest_folder}")
41
42    # Pfade zu den Ordnern mit den PDF-Dateien (Quelle) und den JPG-Dateien (Ziel)
43    src_folder = r"/Users/svenburkhardt/Documents/D_Murger_Männer_Chor_Forschung/Scan_Mä_
44    ↪ nnerchor/Männerchor_Akten_1925-1945/Scan_Männerchor_PDF"
```

```

43 dest_folder = r"/Users/svenburkhardt/Documents/D_Murger_Männer_Chor_Forschung/Master_J
   ↳ arbeit/JPEG_Akten_Scans"
44
45
46 # Funktion aufrufen, um die Konvertierung durchzuführen
47 convert_pdf_to_jpg(src_folder, dest_folder)
48

```

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.