Masterarbeit Tagebuch

29. Juni 2025

Inhaltsverzeichnis

---

1 Offene Probleme mit der Pipeline 27.06.2025

Nr. Problem Was passiert? **1** Rollen werden nicht normiert role leer, obwohl Kontext sie liefert **2** Titel („Herr“, „Fräulein“) fehlt Gender und Anrede-Infos bleiben ungenutzt **3** Doppelte Personen Gleiche Personen (Teilname, Vollname) werden nicht gemerged **4** Pseudo-Events Beliebiger Fließtext wird als Event gespeichert **5** Orte doppelt / keine Typisierung alternate$_{p}lace_{n}amehatDuplikate, typefehlt$**6**$AutorenEmpfängerleerSignat$

Problem A

Dein Context-Window gibt `recipient_score=100` an Stellen, wo es nicht sollte, weil es nicht prüft, ob der „Name“ ein valider Name ist.
Beispiel: ”der doch”darf nie ein Empfänger sein.

Problem B

Dein Tokenizer zerschneidet Grußformeln oder Fließtext in Pseudo-Namen.
Beispiel: Alfons! So geht es im Leben. wird zu Alfons! Vielleicht.

Du brauchst hier:

- stärkere Stoppwörter
- eine strictere Regex: Nur isolierte Namen/Zeilen + Custom-Tags, keine halben Sätze.

Problem C

Die Rolle „Herrn“ wird fälschlich als Person mit eigenem Slot gehalten, statt als `title`.
Das muss unbedingt in `postprocess_roles` in den `title` wandern.

---

2 Feststellungen und Aufsetzung von VM, MongoDB, Cantaloupe und Website 21.06.2025

Feststellung:
Der `place_matcher` und alle anderen nutzen noch keine Infos, die durch eigenes Tagging aus Transkribus kommen (z. B. Ort “H” wird annotiert als “Herrischried”).

---

3 Genderspezifischer Bias 02.06.2025

Feststellung: Rollen werden nicht genderunabhängig gegengeprüft.

🧠 **Analyse durch ChatGPT:**
Derzeitige Lage in deinem Code:

- Die Rolle-Regex `ROLE_BEFORE_NAME_RE` sowie die Funktion `normalize_and_match_role()` sind nicht geschlechtsspezifisch differenziert.
- Fokus liegt auf maskulinen Endungen: `s, n, en, ern, em, e, er`.
- Feminine Endungen wie `in", innen", ë"` werden nicht behandelt.
- Die `ROLE_MAPPINGS_DE` arbeiten meist mit Grundformen wie `"Vorsitzender"`, `"Führer"`, `"Leiter"`.

Beispiel:

Eingabe	Erwartung	Aktuell erkannt?
`Vorsitzende Maria Müller`	Vorsitzender	✗ Nein
`Führerin Anna Schmidt`	Führer	✗ Nein
`Schriftleiterin`	Schriftleiter	✗ Nein

✔ **Lösung: Gender-sensitives Suffix-Matching ergänzen**
Füge in `normalize_and_match_role()` folgenden Block ein:

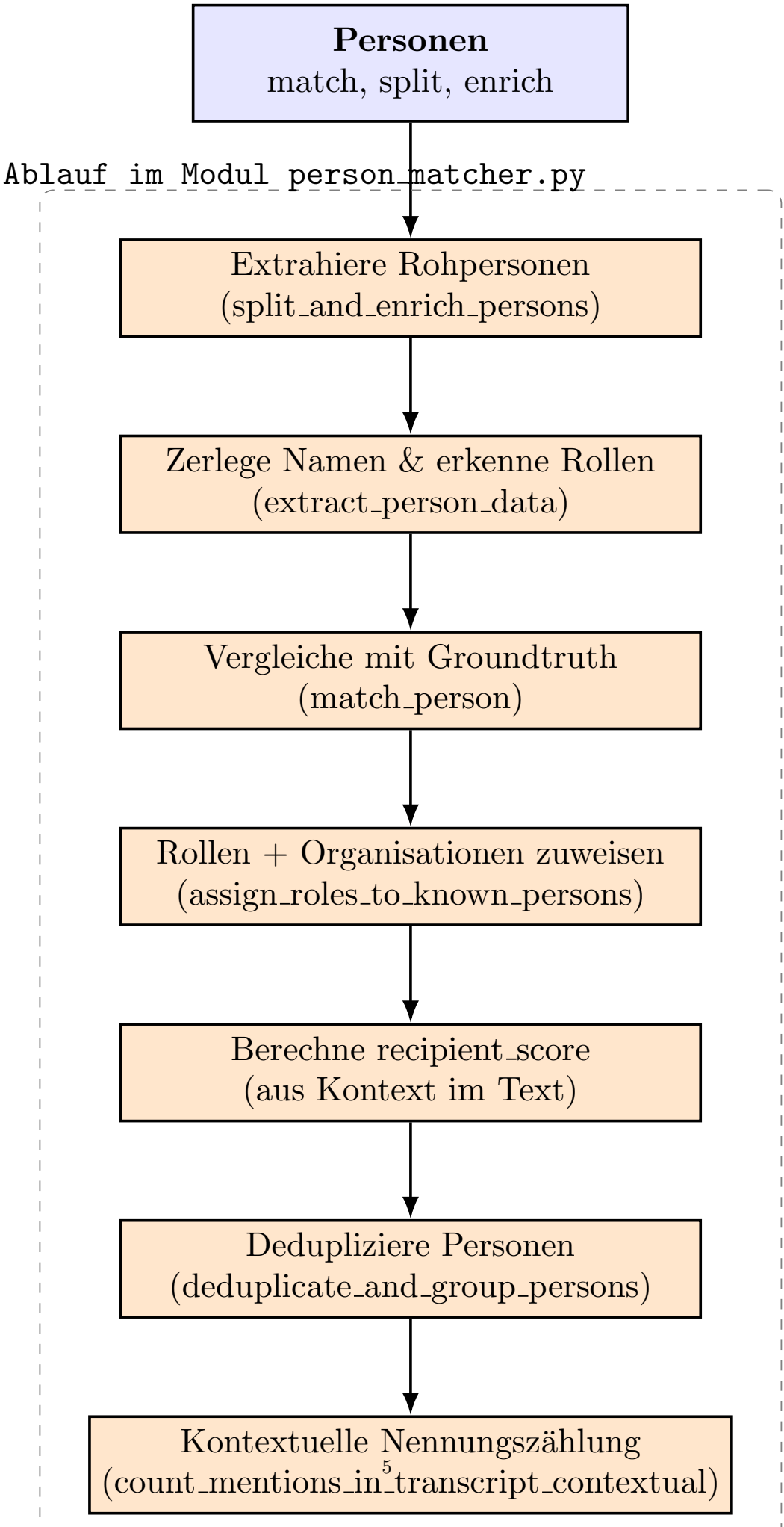
```python
# 2b. Versuch feminine Endungen zurückzuführen
feminine_suffixes = ["in", "innen", "e"]
for suffix in feminine_suffixes:
    if text_clean.endswith(suffix) and len(text_clean) > len(suffix) + 2:
        base = text_clean[: -len(suffix)]
        candidate = base + "er"
        if candidate in ROLE_MAPPINGS_DE:
            return ROLE_MAPPINGS_DE[candidate]
```

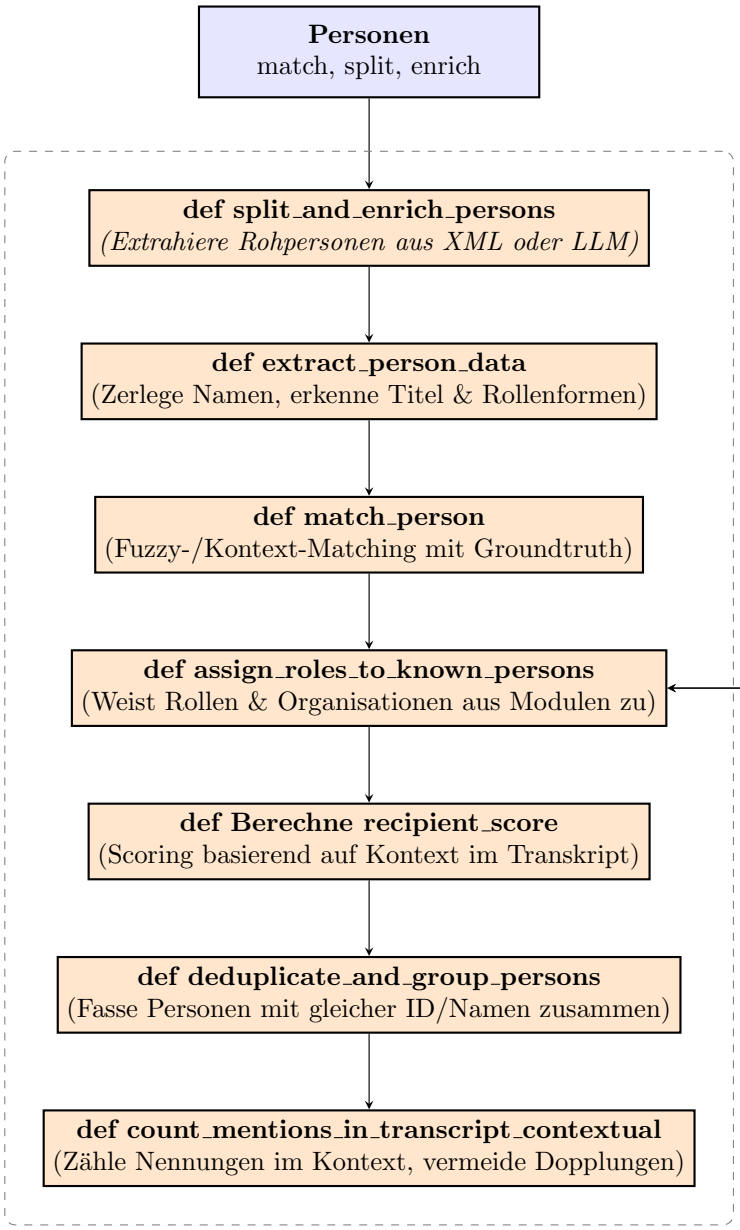
Wirkung:

- Führerin → Führer
- Lehrerin → Lehrer
- Vorsitzende → Vorsitzender

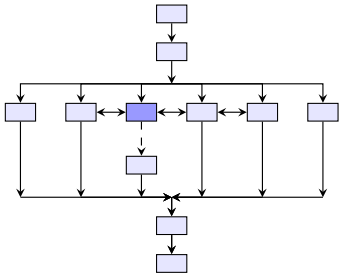
---

4 Prozessablauf Grafiken erstellen29.05.2025

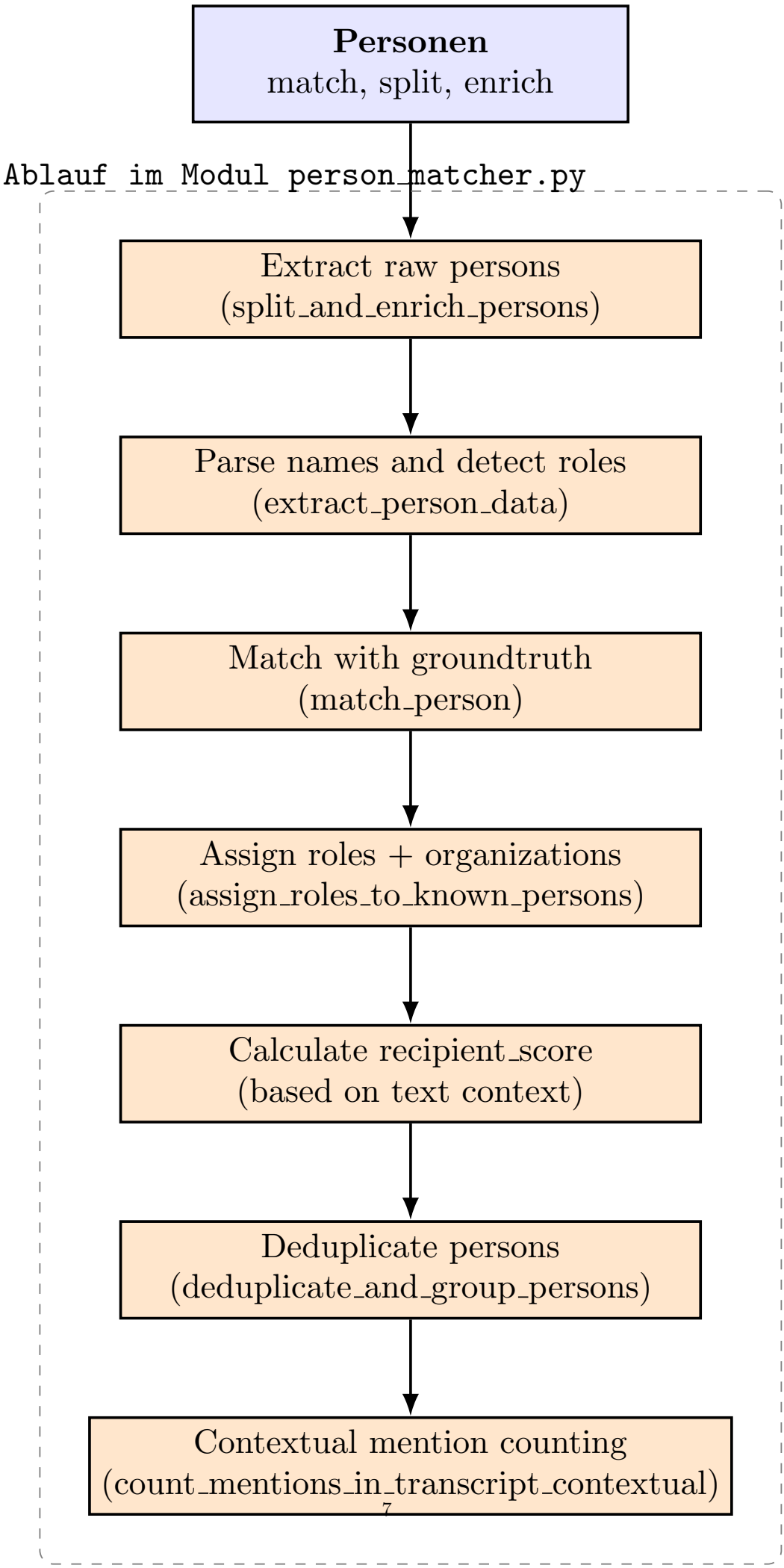




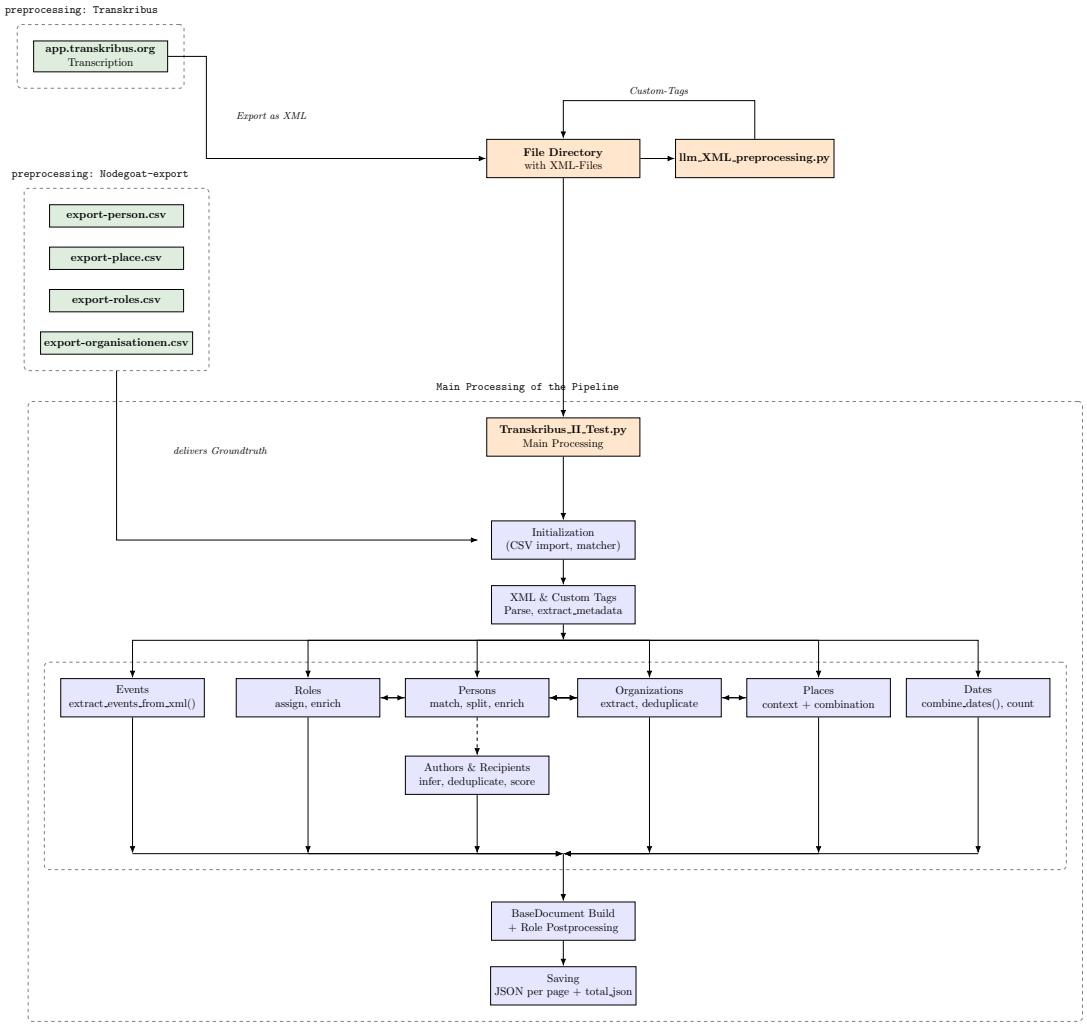
Oben: Prozess person_matcher.py
Rechts: Input aus role_matcher.py und organization_matcher.py

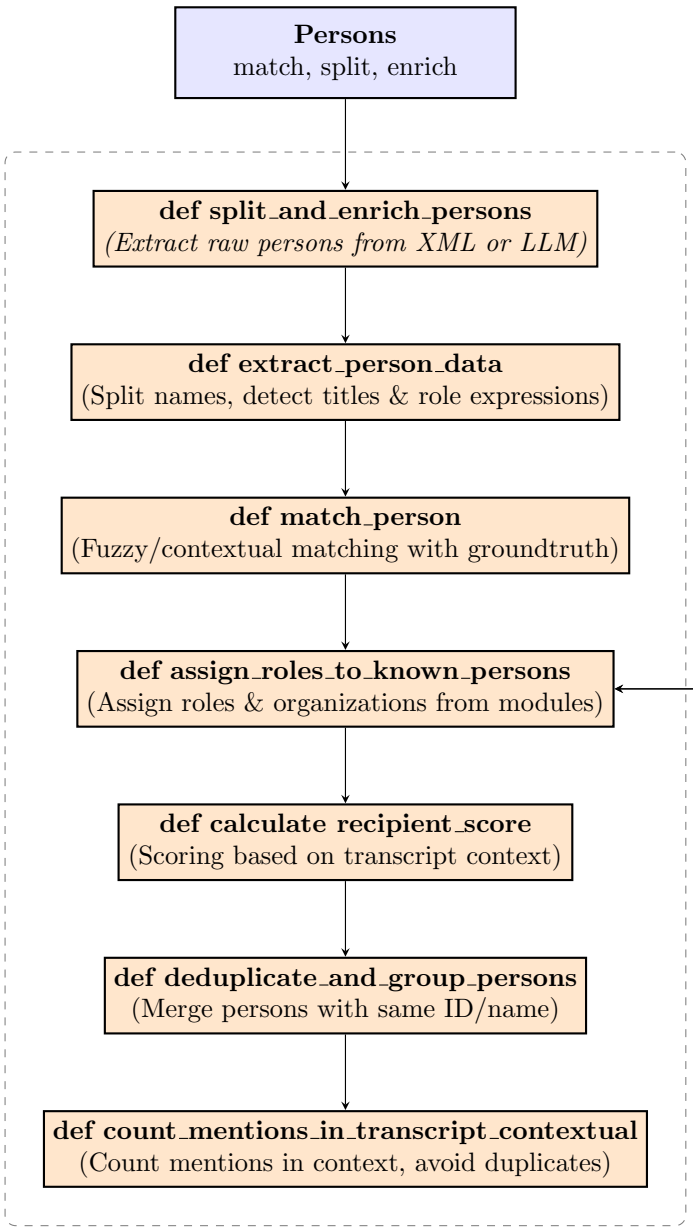


Oben: Gesamtübersicht-Schema

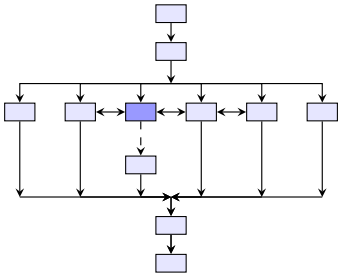

Personen
match, split, enrich
Ablauf im Modul person_matcher.py
Extract raw persons
(split_and_enrich_persons)
Parse names and detect roles
(extract_person_data)
Match with groundtruth
(match_person)
Assign roles + organizations
(assign_roles_to_known_persons)
Calculate recipient_score
(based on text context)
Deduplicate persons
(deduplicate_and_group_persons)
Contextual mention counting
(count_mentions_in_transcript_contextual)

4.1 Diagram english





Top: process in person_matcher.py
Right: input from role_matcher.py and organization_matcher.py



Above: Overall schema overview

---

5 Debugging place_names mit Malus und Bonus - GND-ID Überlegungen

23.05.2025

Noch offene Fehler:

- ⚠ **Wichtigster: Aufnahme von ungematchten Rollen, Personen etc.**
  im moment gibt es kein Fallback, wie ungematchte Personen, Orte, Rollen etc in die Groundtruth gelangen. Ggf. das postprocessing Modul LLM dafür benutzen?
- **Postprocessing**
  Personen mit GND abgleichen, Orte mit Nodegoat

- **Rollenmatching verbessern**
  Rollen brauchen associated Place und Org (Bürgermeister Murg → "Murg"). Bürgermeister wurde auch nicht mit Theodor Grass gematched (Akte 132); Wegen Zeilenumbruch. Hier Verfeinern.
  Auch `!\nR Weiss\nOrtsverbandsleiter des V.D.A."` wird nicht richtig dargestellt

- **`role_schema` bleibt häufig leer – trotz vorhandener Rolle**
  Beispiel: `Alfons Zimmermann`, Rolle: `Vereinsführer`, aber `role_schema: ''`
  Obwohl `Vereinsführer` ein bekannter Rollenwert ist, wird keine `role_schema`-Zuordnung vorgenommen.

- **asssociated Organisationen noch nicht dedupliziert**
  Beispiel Akte_098:


```
      "recipients": [
    {
      "forename": "Alfons",
      "alternate_name": "",
      "familyname": "Zimmermann",
      "title": "",
      "role": "",
      "associated_place": "",
      "nodegoat_id": "ngGN4K68rGNSrHtG1Gf48vNZdHgDwSNLtGK",
      "confidence": "header-2line",
      "needs_review": false,
      "review_reason": "",

      "associated_organisation":
      {
        "name": {
          "name": {
            "name": "Männerchor Murg",
            "nodegoat_id": "Männerchor Murg"
          },
          "nodegoat_id": {
            "name": "Männerchor Murg",
            "nodegoat_id": "Männerchor Murg"
          }
        },
        "nodegoat_id": {
          "name": {
            "name": "Männerchor Murg",
            "nodegoat_id": "Männerchor Murg"
          },
          "nodegoat_id": {
            "name": "Männerchor Murg",
            "nodegoat_id": "Männerchor Murg"
          }
        }
      }
    }
    ]

```


  Vermutung: Der Aufbau und das Befüllen des Associated_org Dicts ist beschädigt. Überprüfen, und die Org um Place ergänzen?

- **Geringster Fehler: Confidence Output**
  der ist im moment noch nicht aussagekräftig

Zusammenfassung:

- **Erkennung zusammengesetzter Ortsnamen:**
  Ortskombinationen wie `Laufenburg-Rhina` werden nun erkannt, wenn Teilorte (z. B. `Rhina`) in

benachbarten Zeilen gemeinsam mit einem Hauptort (z. B. `Laufenburg`) auftreten. Dafür wurde `extract_place_lines_from_xml()` angepasst und `matcher.surrounding_place_lines` korrekt befüllt. Ein kombinierter Treffer wird mit einem Bonus von +5 Punkten bewertet.

- **Kontextsensitives Orts-Matching:**
  Wird in der vorherigen Zeile ein Organisationsbegriff (z. B. `Männerchor`) erkannt, erhält ein potentieller Ort in der Folgezeile einen Malus (-5 Punkte) beim `recipient_place`-Scoring.
- **Fehlerbehebung im JSON-Output:**
  Inkorrekte Verschachtelungen von `associated_organisation` wurden identifiziert (z. B. `"name: {"name: ...}`) und bereinigt. Die Organisationseinträge werden nun flach und eindeutig gespeichert.
- **Modulpflege:**
  Diverse zentrale Module der Pipeline (`place_matcher.py`, `organization_matcher.py`, `document_schemas.py` u. a.) wurden aktualisiert und hochgeladen.
- **recipient_place-Debugging:**
  Die Funktion `assign_sender_and_recipient_place(...)` wurde erweitert und mit gezielten Debug-Ausgaben versehen, um das Matching-Verhalten besser nachvollziehen zu können.
- **Modularisierung der Orts-Extraktion:**
  Die Extraktion von Absende- und Empfangsorten wurde in eine Plug-and-Play-Funktion `extract_places_and_date(` ausgelagert.

Während der Analyse wurde festgestellt, dass zusammengesetzte Ortsnamen (z. B. `Laufenburg-Rhina`) bislang nicht korrekt erkannt wurden. Stattdessen wurde nur der erste Teilname (z. B. `Laufenburg`) extrahiert.
Dies wurde korrigiert: Im Umfeld von `+-1 Zeile` und `+-3 Wörtern` werden nun ergänzende Ortsbestandteile gesucht und mit `name` sowie `alternative_placenames` aus der Groundtruth-Liste abgeglichen.

✓ Ergänzend wurde ein kontextbasierter Malus eingeführt: Wenn in der Zeile vor einem Ortsnamen ein Organisationsbegriff (z. B. `Männerchor`) erscheint, wird der `recipient_place`-Score um 5 Punkte reduziert.

✓ Im Rahmen eines GND-Tests wurde festgestellt, dass einzelne Personen (z. B. `Alfred Joos`) in der GND vorhanden sind. Ein erstes Skript zur API-Abfrage wurde implementiert, liefert aktuell jedoch noch keine Treffer. Weitere Prüfung erforderlich.

✓ Die fehlerhafte Struktur in `associated_organisation`, bei der Organisationen ohne `nodegoat_id` mehrfach verschachtelt wurden, ist nun bereinigt. Die Ausgabe erfolgt in flacher Struktur mit `name` und `nodegoat_id`.

Erledigt:

- Matching von `creation_place` und `recipient_place` mittels LLM-Custom-Tags und Regex
- Erkennung und Zusammenführung zusammengesetzter Ortsnamen (`Laufenburg` + `Rhina` → `Laufenburg-Rhina`)
- Erweiterung der Orts-Kontextverarbeitung über `surrounding_place_lines`
- Korrektur der JSON-Struktur bei `associated_organisation`
- Erweiterung des Personenmodells um `gnd_id` und Referenzlinks nach Nodegoat

---

6 Debugging 22.05.2025

⏭ Nächste Schritte (Priorisiert)

- Filter mentioned_persons auf realistische Einträge (Name oder ID oder Rolle mit Kontext)
- role_schema sicher zuweisen (auch bei späteren Zuordnungen)
- Debug-Ausgabe bereinigen: viele Wiederholungen (z.B. JSON-ready recipients mehrfach)

Kritische Bewertung der Pipeline-Ausgabe

Schwächen und akute Baustellen

1. **Viel zu viele False Positives in `mentioned_persons`**
   `Herrn, Otto, Zimmermann` (ohne Kontext), `"Dirigenten"` → werden als Personen behandelt, obwohl sie ohne ID und ohne echten Namen sind.
   Diese Einträge müssen **immer** im finalen JSON landen, obwohl sie vorher nach `review_reason` als droppable klassifiziert wurden.
   Lösung: Die `droppable`-Klassifikation so überarbeiten, dass alles, was menschlich oder eine Rolle ist, im JSON bleibt – mit `needs_review = true` und Eintrag in `unmatched_persons.json`.

2. **`role_schema` bleibt häufig leer – trotz vorhandener Rolle**
   Beispiel: `Alfons Zimmermann`, Rolle: `Vereinsführer`, aber `role_schema: ''`
   Obwohl `Vereinsführer` ein bekannter Rollenwert ist, wird keine `role_schema`-Zuordnung vorgenommen.

3. **Mehrere identische Personen mit unterschiedlichen Schreibweisen**
   Beispiel: `Zimmermann` ohne Vorname (Score 30, ohne ID)
   vs. `Alfons Zimmermann` (Score 100, mit ID)
   Beide landen dedupliziert im Output, sollten aber zusammengeführt werden – idealerweise mit erklärendem Kommentar im JSON.

4. **Rollenextraktion aus 2-Zeilenblöcken funktioniert nur selektiv**
   Beispiel: `R Weiss` in Zeile 1, Folgezeile: „Ortsverbandsleiter des V.D.A."
   Zwar wird „Ortsverbandsleiterërkannt, aber nicht korrekt der Person in der Zeile davor zugewiesen, wenn dort nur Initialen stehen.

5. **Datumserkennung zählt doppelt**
   Für das Datum `01.01.1938` wird in manchen Fällen ein `count = 2` erzeugt, obwohl es nur einmal im Text vorkommt.

6. **Events über mehrere Zeilen werden nicht zusammengeführt**
   Beispiel: Zwei aufeinanderfolgende Zeilen mit Eventinformationen werden getrennt gespeichert. Es fehlt die Fusion zu einem vollständigen `event["name"]`.

7 Event- und Orts-Matching Debugging 18.05.2025

Kurzbeschreibung

⚠ Begriff für die MA: Co-Kreativität für die Verschmelzung von Mensch und Maschine.
Erkennung und Speicherung von Ortsangaben in Events funktioniert noch nicht zuverlässig. Einzelwörter wie „Stifter", „75jährigen"werden fälschlich als Orte erkannt. Orte aus Eventblöcken werden nicht korrekt übernommen oder dedupliziert.

Identifizierte Probleme

⚠ Debugging von `event_matcher.py`, fehlerhafte Zeilen-fusion identifiziert
⚠ Naives Token-Matching führt zu falsch-positiven Orten
⚠ Ortseinträge aus Event-texten bleiben leer, obwohl valide Orte im Text enthalten sind
⚠ Wikidata- und Geonames-Ergänzung funktioniert nur eingeschränkt

Nächste Schritte

❑ Token-Filter für ungeeignete Wörter in `event_matcher` einbauen
❑ Orts-Matching kontext-basiert statt token-basiert umsetzen
❑ Fehlende IDs automatisch ergänzen, falls Nodegoat-Eintrag vorhanden
❑ Orte aus Events mit globaler `mentioned_places`-Liste deduplizieren

Offene Fragen

❑ Können strukturierte Event-blöcke direkt mit Ortserkennung kombiniert werden?
❑ Wie sollen Einzel-tokens ohne Kontext (z.B. „am", „Stifter") behandelt werden?

---

8 Änderungen in Transkribus_II_Test.py und infer_authors_recipients
08.05.2025

Kernaussagen der Änderungen

Die heutigen Änderungen betreffen die Robustheit der JSON-Erstellung sowie die korrekte Verknüpfung von Autoren, Empfängern und erwähnten Personen. Die zentralen Anpassungen im Skript `Transkribus_II_Test.py` sowie in der Hilfsfunktion `infer_authors_recipients` umfassen:

- Die Autoren- und Empfängerfelder im `BaseDocument` werden zunächst leer gesetzt, um Redundanz zu vermeiden und spätere LLM-basierte Bereinigungen zu ermöglichen.
- Die Funktion `infer_authors_recipients` wurde erweitert: Nach der Erkennung von Autoren und Empfängern wird überprüft, ob diese auch in `mentioned_persons` enthalten sind. Falls nicht, werden sie ergänzt.
- Die Funktion `resolve_llm_custom_authors_recipients` wird konsequent erst nach dem Aufbau aller Entitäten aufgerufen.
- Die Ortserkennung wurde vereinheitlicht. Orte werden konsistent dedupliziert, mit Nodegoat-IDs angereichert und in das finale JSON übernommen.
- Die Speicherung des finalen Dokuments erfolgt erst nach Validierung und Fehlerausgabe, was die Stabilität des Workflows verbessert.

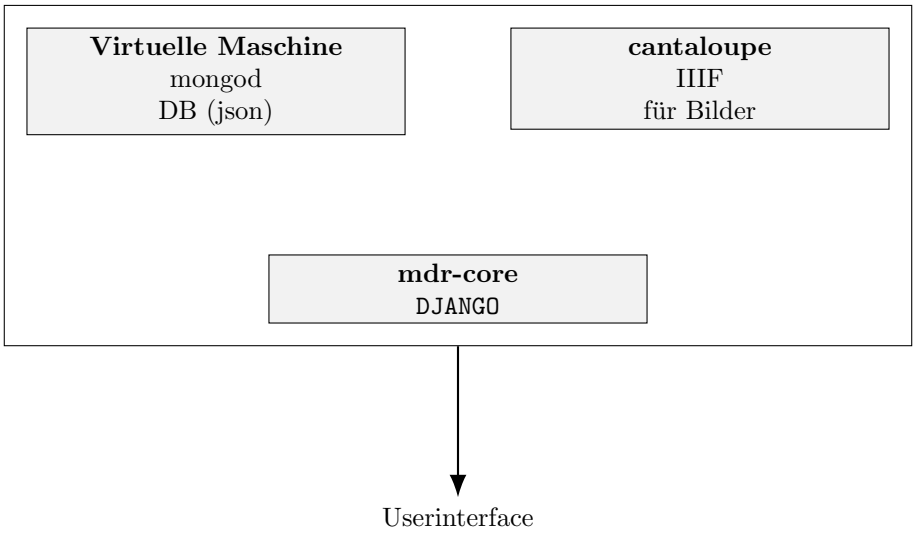
Nächste Schritte

❑ Sicherstellen, dass keine Duplikate durch mehrfaches Setzen von Autoren/Empfängern entstehen
❑ Validierungs-Fehler in Konflikt-Logs systematisch dokumentieren
❑ Einführung eines konsistenten Fallbacks für leere oder fehlerhafte Personen- und Ortsdaten

9 Systemübersicht Virtuelle Maschine 07.05.2025

Für die Darstellung und Endnutzung der für die Masterarbeit gewonnen Daten ist von Sorin und mir in einem Feedbackgespräch die Idee entstanden, eine Virtuelle Maschine der IT-Services der Uni Basel zu nitzen. dafür wurde Lukas am 07.10 per Mail angefragt. Ziel ist ein Userinterface zu baeuen, indem per Suchmaske die JSON Dateien durchsucht werden können, per IIIF-Server auf die Bilddateien der Akten zuzugreifen, und in einem weiteren Schritt ggf auch auf meine Nodegoat-Datenbank zuzugreifen.

`maennerchor_murg.philhist@unibas.ch`



9.1 Fehleranalyse und Verbesserungsvorschläge der JSON-Extraktion

Im aktuellen Stand der automatisierten Extraktion historischer Metadaten aus Transkribus-XML-Dokumenten zeigt sich, dass einige der generierten JSON-Dateien inhaltlich nur teilweise oder gar nicht gefüllt sind. Die Debug-Ausgaben des Skripts deuten auf folgende Hauptprobleme hin:

- **Leere JSON-Dateien:** In mehreren Fällen (z. B. bei `Akte_696`) wurden keinerlei Entitäten erkannt. Dies betrifft insbesondere Dokumente mit wenigen klaren semantischen Ankern oder fehlerhafter OCR-Struktur.
- **Rollen werden häufig nicht erkannt:** Obwohl die Normalisierung deklinierter Rollen (z. B. *Ehrenvorsitzenden* → *Ehrenvorsitzender*) implementiert wurde, erscheinen viele Rollen nicht im JSON. Vermutlich erfolgt keine korrekte Zuordnung zum Rollenschema.
- **Einzelne Namen wie "Herrn" oder "Ortsverbandsleiter" werden fälschlich als Person erkannt:** Die Blacklist filtert einige dieser Fälle, jedoch nicht konsistent oder umfassend genug.
- **Falsch-positive Personen:** In Kontexten wie "Dirigenten" oder "mich" werden Begriffe extrahiert, die keine validen Personennamen sind.
- **Mehrdeutigkeiten bei zusammengesetzten Ausdrücken:** Konstruktionen wie "Vereinsführer des Männerchor" führen zur fehlerhaften Aufspaltung zwischen Personen- und Organisationsentitäten.

9.2 Vorgeschlagene Verbesserungen

- **Erweiterung der Rollenextraktion:** Rollen sollten auch erkannt werden, wenn sie in Verbindung mit Personennamen stehen (z. B. "Ehrenvorsitzenden Burger"). Eine robustere Normalisierung mit Regex-Support für maskuline/feminine Deklinationen ist bereits in Arbeit.
- **Blacklist für generische Begriffe erweitern:** Begriffe wie "mich", "Herrn", "Ortsverbandsleiter" sollen systematisch ausgeschlossen oder nur in Kombination mit validen Namen akzeptiert werden.
- **Verbesserung des Kontextverständnisses bei Personenextraktion:** Eine Gewichtung nach Position im Satz (z. B. Grußformeln) oder in Verbindung mit Rollenwörtern könnte helfen, valide Personen zu priorisieren.
- **Evaluation der OCR-Qualität pro Datei:** Ein hoher Anteil fehlerhafter Tokens könnte vorab mit einem Heuristikwert ermittelt und markiert werden.
- **Fallback-Strategien für autor/recipient:** Falls kein `author` oder `recipient` erkannt wird, aber Custom-Tags mit solchen Informationen vorhanden sind, soll ein LLM-gestützter Vorschlag übernommen werden.

Die nächsten Schritte umfassen gezielte Tests mit exemplarischen Dokumenten, um die Wirkung der geplanten Anpassungen zu evaluieren. Besonders relevant ist dabei die korrekte Auflösung von Rollenzuweisungen und die zuverlässige Filterung nicht-personaler Begriffe.

10 Änderungen im assigned_roles_module und person_matcher06.05.2025

Kernaussagen der Änderungen

Die letzten Änderungen konzentrieren sich auf die Verbesserung der Entitäten-Erkennung und das Matching von Personen, Orten und Rollen. Es wurden gezielte Verbesserungen und Anpassungen in den folgenden Bereichen vorgenommen:

- Orte werden nun wieder korrekt mit der Nodegoat-ID gematcht. Dies stellt sicher, dass die geografischen Entitäten präzise zugeordnet und identifiziert werden.
- Das `assigned_roles_module`-Script wurde optimiert, um den Genus der erkannten Rollen zu normalisieren und diese im Nominativ Singular darzustellen.
- Es muss jedoch noch weiter optimiert werden, um sicherzustellen, dass auch Einzelrollen wie "Der Vereinsführer"korrekt als erwähnte Personen erkannt werden.

- Der `person_matcher` kann nun auch Personen matchen, wenn nur ein Vor- oder Nachname vorhanden ist. Dies erweitert die Flexibilität des Matchings erheblich und verbessert die Erkennungsrate bei unvollständigen Namen.
- Die Deduplizierung von Personen funktioniert weiterhin wie erwartet. Das System sorgt für eine effiziente Zusammenführung ähnlicher Entitäten, ohne Duplikate zu erzeugen.

Nächste Schritte

❑ Weiterführung der Optimierungen im Bereich der Rollenerkennung und der Personenzuordnung
❑ Verbesserung der Fehlerbehandlung bei besonderen Rollenkombinationen
❑ Automatisierte Tests für die neuen Matching-Strategien einführen

Strategie zur Verbesserung des Rollen-Matchings per ChatGPT

Idee: überarbeiteter Workflow, der die beiden Module `assigned_roles_module` und `person_matcher` systematisch einbindet. Ziel ist die zuverlässige Erkennung von Rollenangaben wie ≫*Ehrenvorsitzender Burger*≪ und deren saubere Trennung in Rolle und Person sowie die Ergänzung bereits erkannter Personen um zusätzliche Rollen.

- **Früherkennung von Rollen in Personenangaben:**
  Bereits beim Parsen der `custom_data["persons"]` wird jeder `raw_token` auf Rollen untersucht. Dazu wird `extract_role_in_name(token)` aufgerufen, um Kombinationen wie ≫*Ehrenvorsitzender Burger*≪ zuverlässig aufzuteilen.
- **Rollenerkennung im Fließtext:**
  Zusätzlich wird `extract_standalone_roles(text)` auf das Transkript angewendet, um unabhängig von XML-Metadaten Rollen wie ≫*Der Vereinsführer*≪ zu identifizieren.
- **Normalisierung und Abgleich:**
  Die erkannten Rollen werden mit `normalize_role_name()` in ihre Grundform gebracht (z. B. „Ehrenvorsitzenden" → „Ehrenvorsitzender"). Die zugehörigen Namen werden mit `normalize_name()` in Vor- und Nachname getrennt und mit `match_person()` gegen bekannte Personen abgeglichen.
- **Erweiterung vorhandener Personen:**
  Stimmen Namen oder IDs mit bereits erkannten Personen überein, wird die Rolle hinzugefügt, ohne ein Duplikat zu erzeugen. Der Abgleich erfolgt über `normalize_name_string()` und `deduplicate_persons()`.
- **Bewertung der Übereinstimmung:**
  Bei eindeutigem Match wird `match_score = 100` und `confidence = "llm-matched"` gesetzt. Bei unklarem Abgleich kann `confidence = "partial"` verwendet werden, um die Herkunft der Information nachvollziehbar zu machen.
- **Zusammenführung aller Kandidaten:**
  Die Ergebnisse aus XML, Fließtext und Rolle-im-Namen-Kombinationen werden zu einer finalen Personenliste kombiniert, die dedupliziert und mit Rollen angereichert als `mentioned_persons` im JSON-Dokument gespeichert wird.

---

11 Verbesserte Deduplizierung in Person- und Orts-Matchern
30.04.2025

Optimierung der Deduplizierungslogik für Personen und Orte in den JSON-Ausgabedateien

Kurzbeschreibung

Die Module zur Deduplizierung von Personen und Orten in den JSON-Ausgabedateien wurden grundlegend überarbeitet. Die neue Implementierung verhindert, dass Entitäten mehrfach in der Ausgabe erscheinen, und priorisiert dabei Einträge mit vorhandener Nodegoat-ID. Zudem werden alternative Namensformen zusammengeführt, um ein umfassenderes Verständnis der identifizierten Entitäten zu ermöglichen.

Erledigte Aufgaben

Überarbeitung des person_matcher.py Moduls

✓ Gruppierung von Personen nach nodegoat_id oder normalisiertem Namen implementiert
✓ Priorisierung von Einträgen mit Nodegoat-ID bei der Deduplizierung
✓ Kombination und Konsolidierung von Rolleninformationen aus mehreren Nennungen
✓ Optimiertes Scoring-System für die Auswahl der besten Einträge

Optimierung des place_matcher.py Moduls

✓ Neuimplementierung der deduplicate_places-Methode mit verbesserter Gruppierungslogik
✓ Zusammenführung alternativer Ortsnamen in einem deduplizierten Eintrag
✓ Priorisierung von Ortseinträgen nach Vorhandensein und Qualität von IDs
✓ Deduplizierung basierend auf normalisiertem Ortsnamen für Einträge ohne ID

Integration im Hauptskript

✓ Anpassung der Verarbeitung in transkribus_to_base.py für deduplizierte Personen
✓ Beseitigung von Doppeleinträgen durch vereinheitlichte Deduplizierung
✓ Konsistente Verwendung der verbesserten Matcher-Module
✓ Erhalt und Zusammenführung aller relevanten Informationen in deduplizierten Entitäten

Nächste Schritte

❑ Quantitative Evaluierung der Deduplizierungsqualität auf dem Gesamtdatensatz
❑ Dashboard zur Visualisierung der Entitätsbeziehungen entwickeln
❑ Integration mit Batch-Prozessierung für alle Dokumente
❑ Automatisierte Tests für die Matching-Qualität implementieren

Offene Fragen

❑ Wie kann die Konsolidierung von semantisch identischen Rollen verbessert werden?
❑ Könnte eine Clustering-Methode zur Identifikation weiterer Duplikate beitragen?
❑ Wie sollen Grenzfälle mit sehr ähnlichen, aber nicht identischen Entitäten behandelt werden?

12 Vereinheitlichung der Schnittstellen und Dataframes 14.04.2025

Integration der Datenvalidierung mit CSV-Export und Optimierung der Ortsanreicherung

Kurzbeschreibung

Heute die Module zur Datenverarbeitung erweitert mit einem vereinheitlichten Interface für Validierungsergebnisse. Die CSV-Export-Funktionalität wurde implementiert und die Ortsabgleichslogik deutlich optimiert. Zusätzlich wurden Dataframe-Manipulationen für strukturierte Analysedaten verbessert.

Erledigte Aufgaben

Validierung und CSV-Export

✓ CSV-Exportformat für Validierungsfehler implementiert
✓ Strukturierte Sammlung von Fehlertypen nach Dokumenten und Kategorien
✓ Aggregierte Statistiken zur Datenqualität hinzugefügt
✓ Pandas-Dataframes für konsistente Datenanalyse integriert

Optimierung des Ortsabgleichs

✓ ID-Generierung für neue Ortsentitäten systematisiert
✓ Fuzzy-Matching durch exakte Vorfilterung beschleunigt
✓ Kaskadierendes Matching (erst exakt, dann Teilstring, dann Fuzzy)
✓ Priorisierung von IDs nach Qualität (Nodegoat ¿ Geonames ¿ Wikidata)

Schnittstellen-Vereinheitlichung

✓ Konsistente Parameter und Rückgabewerte zwischen Modulen
✓ Verbesserte Fehlerbehandlung mit spezifischen Exceptions
✓ Logging-Framework für detaillierte Diagnose eingerichtet
✓ Type-Annotations für bessere Code-Dokumentation hinzugefügt

Nächste Schritte

❑ Dashboard zur Visualisierung von Entitätsbeziehungen entwickeln
❑ Automatisierte Tests für die Matching-Qualität erstellen
❑ Batch-Prozessierung für große Dokumentmengen optimieren

Offene Fragen

❑ Wie können wir die Matching-Parameter automatisch optimieren?
❑ Wann erhalten wir die vollständigen Nodegoat-IDs für alle Entitäten?

---

13 Validierung und Orts-Matching 13.04.2025

Erweiterung der Datenvalidierung und Fehlerausgabe im Transkriptions-Workflow

Kurzbeschreibung

Erweiterung des Validierungsmoduls für strukturierte Prüfung exportierter JSON-Dokumente. Verbesserung des Ortsabgleichs mit Groundtruth-Tabelle zur eindeutigen Zuordnung von Geonames- und Nodegoat-IDs.

Erledigte Aufgaben

Validierungsmodul (`validation_module.py`)

✓ Erweiterte Validierungslogik `validate_extended()` implementiert
✓ Prüfung auf Pflichtfelder: `recipient`, `creation_date`, `creation_place`
✓ Prüfung von `mentioned_places` auf Geonames- und Nodegoat-ID
✓ Ausgabe einer statistischen Fehlerübersicht (z.B. „Geonames-ID fehlt: 5×")

Transkribus-Hauptskript

✓ Einbindung des erweiterten Validierungsmoduls
✓ Speichern von Fehlern pro Datei in strukturierter Liste
✓ Ausgabe der häufigsten Fehlerarten am Ende der Verarbeitung
✓ Fix für `UnboundLocalError` bei fehlgeschlagenem Dokumentexport

Ortsabgleich (Place-Matching)- Funktioniert noch nicht

✓ Lowercase + Strip für Eingabe und Vergleichswerte vereinheitlicht
✓ Threshold bei Fuzzy-Matching auf 75 gesetzt
✓ Fehlerbehandlung bei fehlenden oder ungültigen Wikidata-IDs (z.B. float statt string)
✓ Entfernung des Feldes `type` aus allen Place-Objekten und JSON-Exports
✓ Debug-Ausgabe zur Match-Qualität hinzugefügt

Nächste Schritte

❑ Validierungsfehler zusätzlich in CSV speichern
❑ Orte ohne IDs visuell markieren oder zur Nachprüfung kennzeichnen
❑ Schutzmechanismus gegen ID-Veränderung im LLM-Enrichment einbauen

Offene Fragen

❑ Wie bekomme ich die Groundtruth-Daten? ❑ Prüfung auf fehlerhafte Personenentitäten (z.B. „des", „Vereinsführer")

---

14 LLM-Anreicherung und Metadaten-Extraktion 12.04.2025

Integration von ChatGPT-API und Erweiterung der Metadaten-Extraktion

Kurzbeschreibung

Heute wurden die LLM-Anreicherungsskripte optimiert und die Metadatenextraktion aus JSON-Dateien weiterentwickelt. Die Skripte ermöglichen jetzt eine präzisere Erkennung von Entitäten und eine strukturierte Anreicherung der Transkribus-Daten mit zusätzlichen Metadaten durch den Einsatz von Large Language Models.

Erledigte Aufgaben

Optimierung von ChatGPT-API-picture-to-JSON_2024.py

✓ Code-Kommentare für bessere Dokumentation ergänzt
✓ API-Kostenkalkulation präzisiert
✓ Prompt-Engineering für die Analyse historischer Dokumente verbessert
Ergebnis: Zuverlässigere Extraktion von Metadaten aus Bildmaterial mit detaillierter Kostenkontrolle

Weiterentwicklung der CSV-Merger-Skripte

✓ Metadaten_CSV_Merger_V2.py mit erweiterter Normalisierungsfunktion implementiert
✓ Verbesserte Zusammenführung mehrerer CSV-Quellen
✓ Robustere Fehlerbehandlung bei der CSV-Verarbeitung hinzugefügt
Ergebnis: Umfassendere Gesamtübersicht über alle Akten mit konsistenter Nummerierung

Nächste Schritte

❑ Automatisierte Validierung der extrahierten Metadaten implementieren
❑ Named Entity Recognition für Militäreinheiten und Dienstgrade integrieren
❑ Dashboard zur Visualisierung der Metadaten-Qualität entwickeln

Offene Fragen

❑ Wie kann die Qualität der LLM-Extraktion objektiv bewertet werden?
❑ Skalierungsmöglichkeiten für größere Dokumentenmengen?

15 Integration mit Transkribus XML Export 11.04.2025

Erweiterung der Datenpipeline zur Anreicherung von Transkribus-XML-Dateien

Kurzbeschreibung

Das neue Modul `Transkribus_XML_Data_Enricher_with_CSV.py` wurde entwickelt, um XML-Dateien aus Transkribus mit strukturierten Metadaten aus CSV-Dateien anzureichern. Dieses Skript ermöglicht die Zusammenführung von manuell erstellten Metadaten und automatisch extrahierten Informationen, um vollständigere Dokumentbeschreibungen zu erstellen.

Erledigte Aufgaben

Entwicklung des XML-Enricher-Skripts

✓ XML-Parser für Transkribus-Dateien implementiert
✓ Matching-Algorithmus zwischen XML- und CSV-Daten entwickelt
✓ Automatische Anreicherung der XML-Dateien mit CSV-Metadaten
Ergebnis: Funktionierendes Skript zur Anreicherung von XML-Dateien mit zusätzlichen Metadaten

Integration des Finder-Tag-Systems

✓ `Image_Tags_to_list.py` zur Extraktion von macOS Finder-Tags erstellt
✓ Automatische Konvertierung von Finder-Tags in strukturierte CSV-Daten
✓ Bereinigung und Normalisierung der extrahierten Tag-Informationen
Ergebnis: Erfolgreiche Integration des visuellen Tagging-Systems in die Metadatenpipeline

Technische Details zur XML-Anreicherung

```
# XML-Namespace festlegen (wie in den Transkribus-Dateien)
NS = {"ns": "http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15"}

# CSV einlesen und für Matching vorbereiten
df_csv = pd.read_csv(csv_file_path, sep=";", dtype=str, on_bad_lines='skip')
```

```
df_csv = df_csv[df_csv["Akte_Scan"].str.contains("Akte", na=False)]

# Extraktion der Seitenzahl für eindeutiges Matching
df_csv["csv_page_number"] = df_csv["Akte_Scan"].apply(
    lambda x: re.search(r"_S(\d+)", x).group(1) if re.search(r"_S(\d+)", x) else None
)

# XML-Dateien mit CSV-Informationen anreichern
for xml_file in xml_files:
    tree = ET.parse(xml_path)
    root = tree.getroot()

    # Metadaten aus Transkribus extrahieren
    transkribus_meta = root.find(".//ns:TranskribusMetadata", NS)
    doc_id = transkribus_meta.get("docId", "").strip()
    page_id = transkribus_meta.get("pageId", "").strip()

    # Matching mit CSV-Daten
    csv_match = find_matching_csv_entry(doc_id, xml_page_number)

    # CSV-Daten in XML integrieren
    csv_elem = ET.Element("CSVData")
    if csv_match:
        for key, value in csv_match.items():
            child = ET.SubElement(csv_elem, key)
            child.text = value

    # Angereicherte XML speichern
    root.append(csv_elem)
    tree.write(output_path, encoding="utf-8", xml_declaration=True)
```

Nächste Schritte

❑ Integration der XML-Anreicherung in die bestehende Metadatenextraktion
❑ Entwicklung eines Validierungssystems für die angereicherten XML-Dateien
❑ Ausweitung der Extraktionsmöglichkeiten auf weitere Metadatenfelder

---

16 Integration von person_matcher.py mit transkribus_to_base_schema.py

09.04.2025

Verbesserung der Personenerkennung durch Nutzung einer gemeinsamen CSV-Referenz

Kurzbeschreibung

Die beiden Module person_matcher.py und transkribus_to_base_schema.py wurden überarbeitet, um eine konsistente Personenerkennung zu gewährleisten. Die zentrale Verbesserung ist die Nutzung der CSV-Datei mit Personenmetadaten als gemeinsame Ground Truth. Dies ermöglicht eine zuverlässigere Erkennung und Matching von Personennamen mit Variationen in der Schreibweise.

Erledigte Aufgaben

Integration von person_matcher.py

✓ person_matcher.py mit Funktionen für CSV-Einlesen und Fuzzy-Matching erweitert
✓ Schnittstelle für den Zugriff auf bekannte Personen standardisiert
✓ match_person-Funktion für die Suche in der CSV-Datei optimiert
Ergebnis: Vereinheitlichter Personenabgleich über beide Module hinweg

Überarbeitung von transkribus_to_base_schema.py

✓ Funktionen aus person_matcher.py integriert
✓ Gemeinsamen CSV-Pfad für Personenreferenzen eingerichtet
✓ Personenerkennungslogik verbessert, um Fuzzy-Matching zu nutzen
Ergebnis: Zuverlässigere Erkennung von Personennamen mit konsistenter Referenzierung

17 Objektorientierte Umstrukturierung der Datenextraktions-Pipeline 08.04.2025

Verbesserung des Skripts auf Basis der Empfehlungen von Claude Code

Kurzbeschreibung

Die bestehende Datenextraktionspipeline wurde grundlegend überarbeitet und auf eine objektorientierte Struktur umgestellt. Statt der bisherigen Dictionary-basierten Implementierung nutzen wir nun die in ‘document_schemas.py‘ definierten Klassen wie ‘BaseDocument‘, ‘Person‘, ‘Organization‘ und ‘Place‘. Diese Umstellung bringt erhebliche Vorteile bei der Datenvalidierung und -konsistenz mit sich.

Erledigte Aufgaben

Objektorientierte Umstrukturierung von transkribus_to_base_schema.py

✓ Umstellung der Datenstrukturen auf Klassen statt Dictionaries.
✓ Integration der Validierungsfunktionen mit Fehlerausgabe.
✓ Anpassung der JSON-Serialisierung mittels to_json()-Methode.
Ergebnis: Robusteres Skript mit automatischer Datenvalidierung und besserer Wartbarkeit.

Details zur Umstrukturierung

```
# Alter Ansatz:
result = create_base_schema(metadata_info, transcript_text)

# Neuer objektorientierter Ansatz:
from document_schemas import BaseDocument, Person, Place, Event, Organization

doc = BaseDocument(
    attributes=metadata_info,
    content_transcription=transcript_text,
    mentioned_persons=[Person(**p) for p in custom_data["persons"]],
    mentioned_organizations=[Organization(**o) for o in custom_data["organizations"]],
    mentioned_places=[Place(**pl) for pl in custom_data["places"]],
    mentioned_dates=custom_data["dates"]
)

# Mit Validierung
if not doc.is_valid():
    print(f"Fehler in Dokument: {doc.validate()}")
```

Vorteile der neuen Implementierung

- **Datenvalidierung:** Automatische Überprüfung auf strukturelle Korrektheit
- **Typsicherheit:** Verbesserte Erkennung von Fehlern bereits zur Programmierzeit
- **Objektorientierung:** Zusammengehörige Daten und Funktionen in einer Klasse
- **Bessere Wartbarkeit:** Klare Schnittstellen und Kapselung von Implementierungsdetails
- **Erweiterbarkeit:** Einfachere Anpassung an neue Anforderungen durch Vererbung

Technische Details zur Integration

```
# In person_matcher.py: CSV-Einlese-Funktion
def load_known_persons_from_csv(csv_path: str) -> List[Dict[str, str]]:
    """
    Lädt die bekannten Personen aus der CSV-Datei.
    """
    if not os.path.exists(csv_path):
        print(f"Warnung: CSV-Datei nicht gefunden: {csv_path}")
        return []

    try:
        df = pd.read_csv(csv_path, sep=";")
        persons = []
```

```
        # Extrahiere relevante Spalten
        for _, row in df.iterrows():
            forename = row.get("schema:givenName", "").strip()
            familyname = row.get("schema:familyName", "").strip()

            if forename or familyname:  # Mindestens ein Namensteil vorhanden
                person = {
                    "forename": forename,
                    "familyname": familyname,
                    "id": row.get("Lfd. No.", ""),
                    # Weitere Metadaten
                }
                persons.append(person)

        return persons
    except Exception as e:
        print(f"Fehler beim Laden der CSV-Datei: {e}")
        return []

# Verbesserte person_matcher Funktion mit CSV-Standardwert
def match_person(
    person: Dict[str, str],
    candidates: List[Dict[str, str]] = None,
    threshold: int = 70
) -> Tuple[Optional[Dict[str, str]], int]:
    """
    Match a person against known persons from CSV (if no candidates provided)
    """
    if not person:
        return None, 0

    # Wenn keine Kandidaten angegeben, nutze CSV-Daten
    if candidates is None:
        candidates = KNOWN_PERSONS
```

Nächste Schritte

❑ Testen der verbesserten Personenerkennung mit einem größeren Datensatz
❑ Entwicklung von Methoden zur automatischen Zusammenführung von Personenduplikaten
❑ Integration fortgeschrittener NER-Methoden für noch bessere Personenerkennung
❑ Überprüfung aller generierten JSON-Dateien auf Validität

Offene Fragen

❑ Wie lässt sich die Konsistenz zwischen der CSV-Datei und extrahierten Personen automatisch überprüfen?
❑ Sollen Schwellwerte für das Fuzzy-Matching je nach Dokumenttyp angepasst werden?

18 Implementierung der Datenextraktions-Pipeline 03.04.2025

Diese Implementierung wurde mit Unterstützung von Claude Code und Sorin von RISE durchgeführt. Erster Einsatz von Claude Code

Kurzbeschreibung

Implementierung der Datenkonvertierungspipeline vom Transkribus XML-Format in das projektspezifische JSON-Basisschema. Es wurden zwei Python-Module entwickelt: ein Schema-Modul und ein Konvertierungsmodul, die zusammen die Grundlage für die strukturierte Datenhaltung bilden.

Erledigte Aufgaben

Definition des Basis-Schemas

✓ Erstellung des ***document_schemas.py*** Moduls mit Klassen für verschiedene Dokumenttypen.
Ergebnis: Implementierung der Basisklassen `Person`, `Organization`, `Place`, `Event` und `BaseDocument` sowie spezialisierte Klassen für `Brief`, `Postkarte` und `Protokoll` mit Validierungsfunktionen.

Konvertierungs-Pipeline

✓ Implementierung des ***transkribus_to_base_schema.py*** Skripts für die XML-Verarbeitung.
Ergebnis: Vollständige Pipeline für die Extraktion von Metadaten und Text aus Transkribus XML-Dateien sowie Konvertierung in das JSON-Basisschema mit automatischer Erkennung von benannten Entitäten (Personen, Organisationen, Orte, Daten).

Nächste Schritte

❑ Integration der Tag-Regeln aus der Transkribus-Anleitung in die Konvertierungspipeline.
❑ Erweitern der automatischen Metadatenextraktion für komplexere Inhaltsanalysen.
❑ Ausführen der Pipeline auf dem vollständigen Datensatz und Prüfung der Ergebnisse.

Offene Fragen

❑ Sollen zusätzliche spezialisierte Dokumenttypen implementiert werden?
❑ Wie können Entitätsreferenzen zwischen verschiedenen Dokumenten hergestellt werden?
Diese Implementierung wurde mit Unterstützung von Claude Code und Sorin von RISE durchgeführt.

---

19 Festlegung Tagging Regeln 11.03.2025

Kurzbeschreibung

Hier werden die Tagging Regeln in Transkribus beschrieben

Personen `person`

Mit dem Tag `person` sollen alle strings getaggt werden, die eine direkte Zuordnung eier Person ermöglichen.

☞ Beispiel 1: Vereinsführer, Alfons, Zimmermann, Alfons Zimmermann, Z. A.Zimmermann, Herr Zimmermann, Herr Alfons Zimmermann, etc
☞ Beispiel 2: Gauleitung, Gauleiter, Gauleiter Max Mustermann, Gauleiter M., Gauleiter Mustermann, etc
☞ Beispiel 3: Oberlehrer, Chorleiter, etc, wenn Ort, Name oder Organisation bekannt.

Signaturen `signature`

Mit dem Tag `signature` sollen alle Strings getaggt werden, die eine handschriftliche Unterschrift darstellen. Dieses Tag dient dazu, Unterschriften von klar erkennbaren Personennamen im Fließtext zu unterscheiden.

☞ Beispiel 1: **Eindeutig lesbare Signaturen** werden direkt getaggt: `<signature>R. Weiss</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>Unleserlich</unclear></sig`

☞ Beispiel 4: **Wenn eine Signatur einer bekannten Person zugeordnet werden kann, aber nicht vollständig lesbar ist, bleibt die Signatur erhalten und wird nicht als „Person"getaggt**: `<signature>A. Zimm<unclear>[...]</unclear></si`

Organisationen `organization`

Mit dem Tag `organization` sollen alle Strings getaggt werden, 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.

☞ Beispiel 3: Wenn der Organisationsname mit einer Positionsangabe kombiniert ist, wird nur der Organisationsname getaggt: „<person>Vereinsleiter <person>des **<organisation>Männerchor Murg</organisation>**".

Orte `place`

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

Hintergund ist Fehlervermeidung: Manche Organisationen haben mehrere Standorte (z. B. "Badischer Sängergau"kann in Karlsruhe oder Mannheim sein). Ein explizites Tagging vermeidet Verwechslungen. Flexibilität: Falls sich eine Organisation an mehreren Orten befindet oder in einem bestimmten Kontext mit einem anderen Ort assoziiert wird, bleibt die Information strukturiert erhalten.

☞ Beispiel 1: Murg (Baden), Freiburg, Berlin, Murgtal, Schwarzwald, etc.
☞ Beispiel 2: Orte mit näherer Bestimmung, z.B. „bei Berlin", „im Murgtal", "Mayerhof Nebenzimmer"werden getaggt, und die nähere Bestimmung muss innerhalb des Tags. Beispiel: „ **<place>im Murgtal</place>**".
☞ Beispiel 3: Adressen oder Ortsangaben in Verbindung mit Organisationen, z.B. „<organisation>Universität **<place>Basel</place>** </organisation>öder „Postfach 6, **<place>Murg </place>**".

Datum `date`

Mit dem Tag `date` werden alle expliziten Datumsangaben markiert und bereits im Tag um das format dd.mm.yyyy ergänzt.

☞ Beispiel 1: 9. Oktober 1940, 20.10.1940, den 3. Mai 1938, etc.
☞ Beispiel 2: Relative Datumsangaben („gestern", „letzten Freitag") werden getaggt.
☞ Beispiel 3: Falls ein Datum in Kombination mit einem Ort steht, wird nur das Datum getaggt: „**<place>Murg (Baden)</place>**, den **<date>9. Oktober 1940</date>**".

Abkürzungen `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 ausgeschriebene Variante im selben Dokument vorhanden ist, bleibt die Abkürzung getaggt: **<person><abbrev>Dr.</abbrev>Weiß</person>**

Unclear `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 `sic`

Mit dem Tag `sic` werden Wörter markiert, die absichtlich in einer falschen oder ungewöhnlichen Schreibweise beibehalten werden.

☞ Beispiel 1: Offensichtliche Tippfehler, wenn sie im Originaltext so vorkommen: „Er hatt **<sic>einen</sic>** große Freude."
☞ Beispiel 2: Veraltete oder falsche Schreibweisen: „**<sic>Feber</sic>**"für Februar.
☞ Beispiel 3: Falls eine Korrektur notwendig ist, kann sie als Kommentar ergänzt werden.

20 Überblick Hilfscript ChatGPT_Api_TranskribusXML_to_JSONv3.py

06.03.2025

Dieses Skript verarbeitet XML-Dateien aus einer definierten Ordnerstruktur und sendet deren Inhalt an die OpenAI-API zur automatischen Analyse und Extraktion relevanter Metadaten. Das generierte Ergebnis wird als JSON-Datei gespeichert.

Das Skript ist speziell auf historische Dokumente (z. B. aus dem Männerchor Murg Corpus, 1925-1945) zugeschnitten. Es analysiert und strukturiert die Daten, um relevante Informationen wie Autoren, Empfänger, Orte, Ereignisse und Zeitangaben zu extrahieren.

21 Voraussetzungen & Einrichtung

21.1 API Key setzen

Bevor das Skript ausgeführt wird, muss der OpenAI-API-Schlüssel als Umgebungsvariable gesetzt werden. Dies kann dauerhaft in der /.zshrc-Datei erfolgen:

```
export OPENAI\_API\_KEY=''sk-...''
source ~/.zshrc
```

Alternativ kann das Skript mit VS Code gestartet werden, indem es aus dem Terminal mit folgendem Befehl aufgerufen wird:

```
code .
```

21.2 Python-Bibliotheken installieren

Folgende Bibliotheken werden benötigt:

```
pip install openai
```

folgende Module werden gebraucht:

```
import json
import os
import xml.etree.ElementTree as ET
import openai
import re
import time
```

22 Funktionsweise

Das Skript ist in folgende Hauptbestandteile gegliedert:

22.1 Basis-Einstellungen

- Zähler zur Erfassung der verarbeiteten Dateien und Token-Kosten.
- API-Schlüssel wird aus der Umgebungsvariable geladen.
- Pfad-Definitionen für Input- und Output-Verzeichnisse.

22.2 Verzeichnisstruktur durchlaufen

Das Skript iteriert über alle 7-stelligen Ordner im Basisverzeichnis (base_input_directory). In diesen Ordnern sucht es nach Unterordnern mit Namen Akte_xxx_pdf oder Akte_xxx, die wiederum einen page-Ordner enthalten.

Falls kein page-Ordner gefunden wird, wird die Verarbeitung dieser Akte übersprungen.

22.3 Verarbeitung der XML-Dateien

Jede XML-Datei im page-Ordner wird eingelesen und analysiert:

- Seitennummer extrahieren: Die Dateinamen haben das Muster p001.xml, p002.xml usw.
- XML-Daten einlesen: Nutzung von xml.etree.ElementTree zur Extraktion von TranskribusMetadata und TextEquiv-Daten.
- Fehlermanagement: Falls eine Datei nicht geparst werden kann oder keinen verwertbaren Text enthält, wird sie übersprungen.

22.4 Strukturierung der Daten in JSON

Die extrahierten Informationen werden in einem JSON-Format gespeichert. Dazu gehören:

- Metadaten (Dokument-ID, Seiten-ID, Bild- und XML-URL)
- Autor und Empfänger mit Name, Rolle und zugehöriger Organisation
- Erwähnte Personen, Organisationen, Ereignisse und Orte
- Dokumentart (z. B. Brief, Protokoll, Rechnung)
- Dokumentformat (z. B. Handschrift, maschinell, mit Unterschrift, Bild)

22.5 API-Anfrage an OpenAI

Ein Prompt wird erstellt, um die Inhalte durch die OpenAI-API analysieren zu lassen. Dabei wird explizit vorgegeben:

- Die historische Relevanz der Dokumente (1925-1945).
- Die Aufgabenstellung (Identifikation von Dokumenttypen, Metadaten und Inhalten).
- Die genaue Formatierung der JSON-Antwort.

23 Fehlerbehandlung

Das Skript enthält mehrere Mechanismen zur Fehlerbehandlung:

- Fehlermeldungen beim XML-Einlesen (try-except beim Parsen)
- Fehlermeldungen bei API-Anfragen (try-except um den OpenAI-Aufruf)
- Fehlermeldungen bei JSON-Speicherung (try-except beim Schreiben der Datei)
- Logging von Problemen (print(f"Fehler beim Parsen der API-Antwort: e "))

Falls Fehler auftreten, werden sie ausgegeben und das Skript setzt die Verarbeitung der nächsten Datei fort, anstatt komplett abzubrechen.

24 Fazit

Dieses Skript automatisiert die Verarbeitung von XML-Dokumenten, extrahiert deren Inhalte und strukturiert die Daten in JSON-Format, das von OpenAI-API analysiert wird. Die Ergebnisse werden gespeichert und abschließend statistisch ausgewertet. Durch Fehlerbehandlung und Logging wird sichergestellt, dass auch bei Problemen das Skript robust bleibt. Dokumentation geschrieben mit ChatGPT.

---

25 Tagging der JPGEGs im AppleFinder 26.10.24

Kurzbeschreibung

Überlegung: JPEGs sollen bereits im Apple Finder mit Tags versehen werden, um eine effiziente, automatisierte Transkription der Chorunterlagen des Männerchors Murg zu ermöglichen. Geplant ist die Kombination von ChatGPT und Transkribus zur Erkennung unterschiedlicher Dokumententypen. Ein Tag-System, bestehend aus **„Maschinell"** für maschinengeschriebene und **„Handschrift"** für handschriftliche Dokumente, gewährleistet die gezielte Zuordnung zur jeweils geeigneten OCR-Software *(Maschinenschrift mit ChatGPT, Handschrift mit Transkribus "German Giant").*

Dokumente, die sowohl maschinell erstellten Text als auch handschriftliche Elemente enthalten, werden entsprechend ihrer Hauptinformationsgehalt getaggt. Zusätzlich erhalten alle Dokumente mit Unterschriften den Tag **„Unterschrift"**, um eine gezielte Verarbeitung dieser Elemente sicherzustellen.

25.1 AppleFinder Tags

- **Handschrift**
- **Maschinell**
- **mitUnterschrift**
- **Bild**

Erledigte Aufgaben

Handschriften tagging

✓ taggen ● **Handschrift** in AppleFinder.
✓ taggen ● **Maschinell** in AppleFinder.
✓ taggen ● **Bild**
✓ taggen ● **mitUnterschrift**

Ergebnis: Handschriften, Maschinell, Bilder und alle handschriftlichen Unterschriften getaggt

Nächste Schritte

❑ Skripte schreiben, um maschinelle Text zu extrahieren ❑ Transkribus für Handschriftliches anschmeißen.
❑ Nach Gemeinsamkeiten in den Texten suchen, um automatisierte Abfrage für ChatGPT zu erstellen.
❑ Ggf. Aufteilung in unterschiedliche Korpora (Briefe handschr. Briefe schreibmaschine, Zeitungsunterlagen.)
❑ Transkribus für Handschriften verwenden.

Offene Fragen

❑ Sollen die Bilder gelöscht werden?

---

26 JPG Datenbereinigung - leere Seiten löschen 25.10.2024

JPG Datenbereinigung

Alle JPGs ohne Inhalt, also beispielsweise Rückseiten, werden gelöscht. Regel: sobald etwas handschriftlich oder gedruckt auf einer Seite steht, bleibt es erhalten. Im Moment sind auch Bilder (Bsp. Postkarten inbegriffen.Bilder mit Taggs

26.1 Anmerkung

Geschichte/Chronik/Gründung des Männerchors in Akte 323

Erledigte Aufgaben

JPG Datenbereinigung

✓Alle JPGs ohne Inhalt, also beispielsweise Rückseiten, werden gelöscht.
Ergebnis: Reiner JPG Korpus mit Schriftgut, aber auch Bildern (bspw. Postkarten)

Nächste Schritte

❑ Nach Gemeinsamkeiten in den Texten suchen, um automatisierte Abfrage für ChatGPT zu erstellen.
❑Ggf. Aufteilung in unterschiedliche Korpora (Briefe handschr. Briefe schreibmaschine, Zeitungsunterlagen.)
❑ Transkribus für Handschriften verwenden.

Offene Fragen

❑ Sollen die Bilder gelöscht werden?
❑ Handschriftliche, maschinengeschriebene und gemischte Daten taggen? Ggf. erst später mit ChatGPT.
❑ Transkribus für Handschriften verwenden?

---

27 Datennormalisierung PDF zu JPEG 24.10.2024

Kurzbeschrieb

Heute zwei Python-Skripte zur Normalisierung der Akten geschrieben:

Erledigte Augaben

PDF zu JPG Konvertierung

✓ Skript ***JPEG-to-PDF.py*** geschrieben.
Ergebnis Alle PDF-Seiten in JPGs umgewandelt, Dateinamen mit Seitenzahlen formatiert.

Prüfung der Aktennummern

✓ Skript ***Check-if-all-files-complete.py*** geschrieben.
Ergebnis: Überprüft, ob Akten von 001 bis 425 vorhanden sind. Alle Akten sind vollständig in JPG umgewandelt.

Nächste Schritte

❑ Daten für OCR-Bereinigung vorbereiten, leere Seiten manuell entfernen.

Offene Fragen

❑ Handschriftliche, maschinengeschriebene und gemischte Daten taggen? Ggf. erst später mit ChatGPT.
