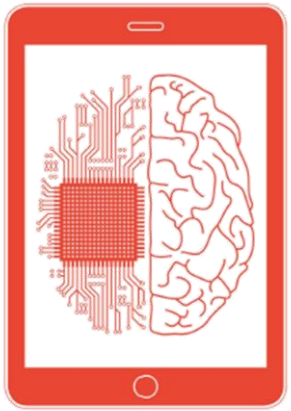
Fachhochschule Nordwestschweiz
Hochschule für Life Sciences



Anamnese – MI Projekt

Sivanajani Sivakumar

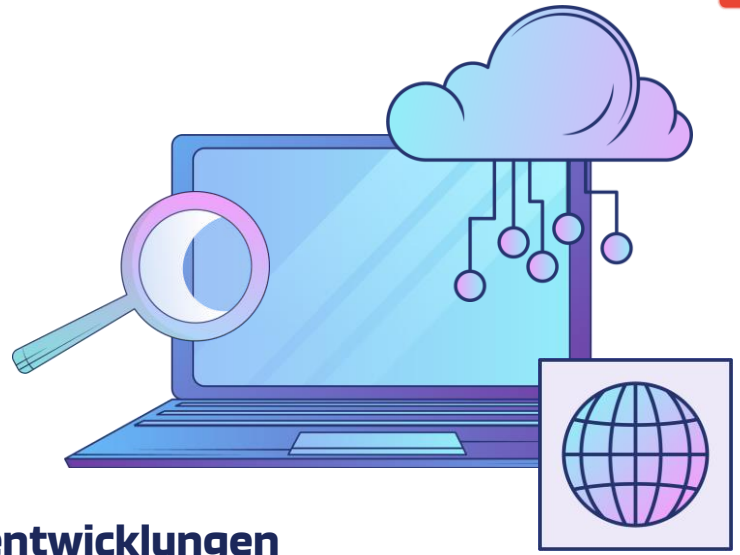
Inhaltsverzeichnis

01 Herausforderungen und Lösungen

02 Von Schlüsselwörtern zu Sätzen

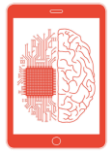
03 Ergebnisse

04 Mögliche Verbesserungen und Weiterentwicklungen



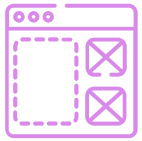
01

Herausforderungen und Lösungen



High-Performance-Computing

Herausforderung: Die Umgebung auf dem HPC war komplex einzurichten, und hat uns die meiste Zeit geraubt.



Import von Bibliotheken

Herausforderung: Unterschiedliche Bibliotheken benötigten verschiedene Versionen, was zu Konflikten führte, insbesondere mit conda.

Lösung: Mit einer virtuelle Umgebung, in der alle Bibliotheken und Abhängigkeiten sauber isoliert wurden, konnte das Problem gelöst werden.

```python
import cv2  # Für Bildverarbeitung
import pytesseract  # Für Texterkennung (OCR)
from pytesseract import Output
import numpy as np  # Für numerische Berechnungen
from pdf2image import convert_from_path  # Zum Konvertieren von PDF zu Bildern
import csv  # Zum Exportieren von Ergebnissen in CSV-Dateien
import Levenshtein  # Für Zeichenbasierte Ähnlichkeitsbewertung
import os  # Für Datei- und Verzeichnisoperationen
import re  # Für reguläre Ausdrücke zur PID/Datumsextraktion
```

02

Von Schlüsselwörtern zu Sätzen

get_sentence_positions

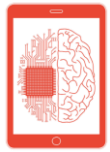
```python
def get_sentence_positions(df, similarity_threshold=50):
    sentence_positions = []
    for catalog_sentence in sentences_catalogue:
        best_match = None
        max_similarity = 0
        best_top = None
        best_bottom = None
        for _, row in df.iterrows():
            if 'text' in row and pd.notna(row['text']):
                ocr_text = row['text']
                top = row['top']
                bottom = row['top'] + row['height']
                similarity = calculate_similarity(catalog_sentence, ocr_text)
                if similarity > max_similarity:
                    max_similarity = similarity
                    best_match = ocr_text
                    best_top = top
                    best_bottom = bottom
        if max_similarity >= similarity_threshold:
            sentence_positions.append((catalog_sentence, best_match, max_similarity, best_top, best_bottom))
    return sentence_positions
```

Warum wichtig?

Hilft dabei, relevante Sätze im OCR-Text zu identifizieren und ihre Position zu bestimmen.

Hauptoperationen:

- Iteriert durch alle Sätze im sentences_catalogue.
- Vergleicht jeden Satz mit dem OCR-Output und berechnet die Ähnlichkeit.
- Speichert Sätze, die die Ähnlichkeitsschwelle überschreiten, zusammen mit Position und Ähnlichkeitswert.



find_checkboxes_for_sentences

```python
def find_checkboxes_for_sentences(sentence_positions, checkboxes):
    results = {}

    for sentence in sentence_positions:
        catalog_sentence, _, _, ystart, yend = sentence
        overlapping_checkboxes = []

        # Iteriere über alle Checkboxen
        for checkbox in checkboxes:
            cx, cy, cw, ch, is_marked, fill_amount, value = checkbox

            # Prüfe, ob die Checkbox mit dem Satz überlappt
            if ystart < cy + ch and yend > cy:
                # Berechne den Überlappungsgrad
                overlap = get_overlap(ystart, yend, cy, cy + ch)
                overlapping_checkboxes.append((overlap, checkbox))

        # Sortiere die Checkboxen nach ihrem Überlappungsgrad (absteigend)
        overlapping_checkboxes.sort(reverse=True, key=lambda x: x[0])

        # Füge die besten Checkboxen (maximal 2) zum Ergebnis hinzu
        results[catalog_sentence] = [cb[1] for cb in overlapping_checkboxes[:2]]

    return results
```

Warum wichtig?

Ordnet Checkboxen den relevanten Sätzen zu.

Hauptoperationen:

- Berechnet die Überlappung zwischen Sätzen und Checkboxen basierend auf ihren vertikalen Koordinaten.
- Sortiert Checkboxen nach dem Grad der Überlappung.
- Speichert die besten passenden Checkboxen für jeden Satz (maximal 2 Checkboxen pro Satz).

match_sentence_checkbox

```python
def match_sentence_checkbox(sentence_with_positions, checkboxes):
    sentence_checkbox_map = []

    # Iteriere durch Sätze und Checkboxen
    for sentence, y_start, y_end in sentence_with_positions:
        for x, y, w, h, yes_or_no in checkboxes:
            if y_start * 0.99 < y < y_end:
                sentence_checkbox_map.append((sentence, yes_or_no, y))

    return sentence_checkbox_map
```

Warum wichtig?

Ermöglicht die direkte Zuordnung von Checkbox-Werten ("Ja"/"Nein") zu den passenden Sätzen.

Hauptoperationen:

- Prüft, ob eine Checkbox innerhalb des vertikalen Bereichs eines Satzes liegt.
- Verknüpft den Satz mit dem Wert der Checkbox.

process_sentences_and_checkboxes

```python
def process_sentences_and_checkboxes(sentences_with_positions, checkboxes):
    sentences_values = []

    for sentence in sentences_with_positions:
        selected_checkboxes = find_checkboxes_in_range(sentence, checkboxes)

        value = determine_value_for_keyword(selected_checkboxes)

        if value is not None:
            sentences_values.append((sentence[0], value, sentence[1]))
    return sentences_value
```

Warum wichtig?

Fasst die Analyse von Checkboxen und Sätzen zusammen und liefert eine zusammenhängende Auswertung.

Hauptoperationen:

- Sucht Checkboxen, die mit den Sätzen überlappen.
- Bestimmt den Wert der Checkboxen für jeden Satz.
- Speichert die Ergebnisse in einer Liste.

sentence_truefalse

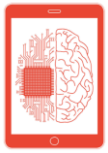
```python
def sentence_truefalse(filtered_positions, sentences_values):
    sentence_trueFalse = []

    for position in filtered_positions:
        if len(position) != 5:
            print(f"Unerwartete Struktur in filtered_positions: {position}")
            continue

        catalog_sentence, best_match, similarity, y_start, y_end = position

        for sentence, value, sentence_y_start in sentences_values:
            if catalog_sentence in sentence or sentence in catalog_sentence:
                sentence_trueFalse.append((catalog_sentence, sentence, value, similarity, y_start, y_end))
                print(f"Gefundener Satz: {sentence} matches with: {catalog_sentence}")

    sorted_sentence_trueFalse = sorted(sentence_trueFalse, key=lambda cb: cb[4])
    return sorted_sentence_trueFalse
```

Warum wichtig?

Erstellt die finale Verknüpfung zwischen den Katalogsätzen, extrahierten Sätzen und den Checkbox-Werten.

Hauptoperationen:

- Prüft, ob ein Satz aus dem Satzkatalog mit einem extrahierten Satz übereinstimmt.
- Verknüpft den Satz mit dem entsprechenden Wert ("Ja", "Nein").
- Sortiert die Ergebnisse nach der vertikalen Position der Sätze.



clean_text

```python
def clean_text(text):
    text = re.sub(r'[^a-zA-ZäöüÄÖÜß0-9\s(),.-]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

Warum wichtig?

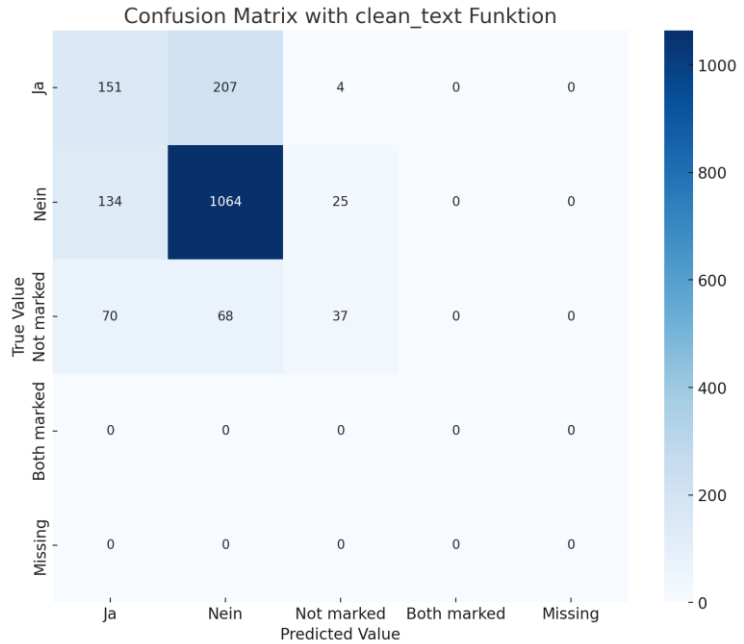
Sorgt dafür, dass der extrahierte Text aus Bildern oder OCR-Quellen standardisiert ist und keine störenden Elemente enthält.

Hauptoperationen:

- Entfernt nicht-alphanumerische Zeichen mit einer regulären Expression.
- Reduziert mehrere Leerzeichen auf eines.
- Entfernt führende und abschliessende Leerzeichen.

clean_text

```python
def clean_text(text):
    text = re.sub(r'[^a-zA-ZäöüÄÖÜß0-9\s(),.-]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```



Warum wichtig?

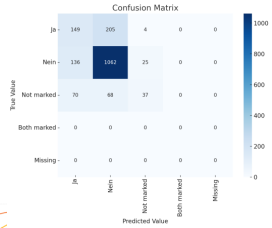
Sorgt dafür, dass der extrahierte Text aus Bildern oder OCR-Quellen standardisiert ist und keine störenden Elemente enthält.

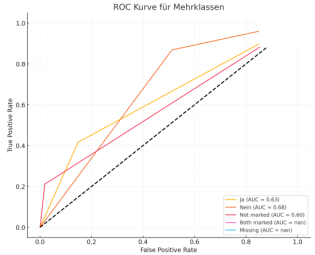
Hauptoperationen:

- Entfernt nicht-alphanumerische Zeichen mit einer regulären Expression.
- Reduziert mehrere Leerzeichen auf eines.
- Entfernt führende und abschließende Leerzeichen.

Nicht verwendet, weshalb:

- Die Resultate zeigten keine grosse Verbesserung.

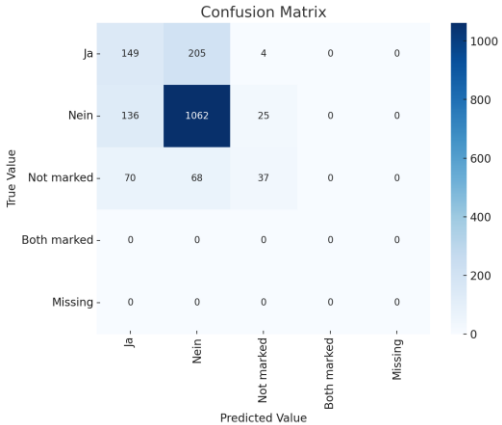




03

Ergebnisse

Ergebnisse bei Sätze-Funktion

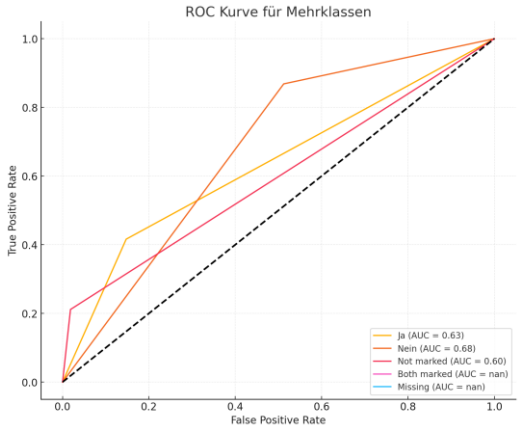


Konfusion Matrix

- Kategorie 'Nein' zeigt die beste Präzision (1062 korrekte Klassifikationen).
- Häufige Verwechslungen zwischen 'Ja' und 'Nein'.
- Schwächen bei der Erkennung von 'Not marked'

ROC-Kurve

- Beste AUC-Werte für 'Nein' (0.68).
- Verbesserungsbedarf bei 'Ja' (AUC: 0.63) und 'Not marked' (AUC: 0.60).
- Keine sinnvolle Bewertung für 'Both marked' und 'Missing' aufgrund fehlender Beispiele.



Ergebnisse bei Sätze-Funktion

Konfusion Matrix

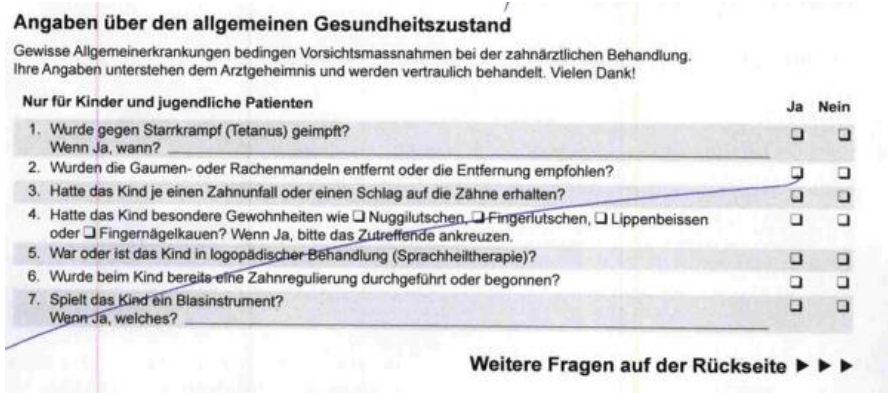
- Kategorie 'Nein' zeigt die beste Präzision (1062 korrekte Klassifikationen).
- Häufige Verwechslungen zwischen 'Ja' und 'Nein'.
- Schwächen bei der Erkennung von 'Not marked'

ROC-Kurve

- Beste AUC-Werte für 'Nein' (0.68).
- Verbesserungsbedarf bei 'Ja' (AUC: 0.63) und 'Not marked' (AUC: 0.60).
- Keine sinnvolle Bewertung für 'Both marked' und 'Missing' aufgrund fehlender Beispiele.

Fehlerquellen durch manuelle Eingriffe

- Einige Patienten haben Text durchgestrichen, um Optionen bewusst nicht zu markieren.
- Durchgestrichener Text überschneidet gelegentlich Checkboxen.
- Das System interpretiert diese Überlappung fälschlicherweise als markierte Checkbox.
- Lösung: vlt. Algorithmen wie Hough Line Transform



Angaben über den allgemeinen Gesundheitszustand

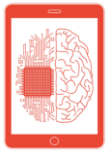
Gewisse Allgemeinerkrankungen bedingen Vorsichtsmassnahmen bei der zahnärztlichen Behandlung.
Ihre Angaben unterstehen dem Arztgeheimnis und werden vertraulich behandelt. Vielen Dank!

Nur für Kinder und jugendliche Patienten	Ja	Nein
1. Wurde gegen Starrkrampf (Tetanus) geimpft? Wenn Ja, wann?	❑	❑
2. Wurden die Gaumen- oder Rachenmandeln entfernt oder die Entfernung empfohlen?	❑	❑
3. Hatte das Kind je einen Zahnunfall oder einen Schlag auf die Zähne erhalten?	❑	❑
4. Hatte das Kind besondere Gewohnheiten wie ❑ Nuggilutschen, ❑ Fingerlutschen, ❑ Lippenbeissen oder ❑ Fingernägelkauen? Wenn Ja, bitte das Zutreffende ankreuzen.	❑	❑
5. War oder ist das Kind in logopädischer Behandlung (Sprachheiltherapie)?	❑	❑
6. Wurde beim Kind bereits eine Zahnregulierung durchgeführt oder begonnen?	❑	❑
7. Spielt das Kind ein Blasinstrument? Wenn Ja, welches?	❑	❑

Weitere Fragen auf der Rückseite ▶ ▶ ▶

04

Mögliche Verbesserungen und Weiterentwicklungen



Optimierungspotenzial

Deep Learning

- Einsatz von CNNs oder Vision Transformers für präzisere Checkbox- und Texterkennung.
- Verwendung von YOLO/Detectron2 zur spezifischen Checkbox-Differenzierung.

Optimierte Bildverarbeitung

- Morphologische Filter zur Entfernung durchgestrichener Texte.

Fuzzy Matching

- Einsatz von Fuzzy-Logik zur flexibleren Satz- und Keyword-Verarbeitung.
- Verbesserung der Vergleichsgenauigkeit durch Gewichtung unsicherer Erkennungen.

```python
from fuzzywuzzy import fuzz

def calculate_similarity_fuzzy(correct_sentence, ocr_sentence):
    return fuzz.ratio(correct_sentence, ocr_sentence)
```

Rechtschreibkorrektur

- Integration von PySpellChecker zur Verbesserung von OCR-Ergebnissen.

```python
from spellchecker import SpellChecker

spell = SpellChecker(language='de')

def correct_text(text):
    corrected_words = [spell.correction(word) for word in text.split()]
    return ' '.join(corrected_words)
```

Fachhochschule Nordwestschweiz
Hochschule für Life Sciences

Thanks!