Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Bachelorarbeit Wirtschaftsingenieurwesen

Enhancing the Speech-to-Text Explorer: Development of a Comprehensive Backend Pipeline for Word Alignment and Error Analysis

Autoren	Gabriel Hunziker
Hauptbetreuung	Prof. Dr. Mark Cieliebak Katsiaryna Mlynchyk
Datum	07.06.2024

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum: Winterthur, 07.06.2024

Name Studierende: Gabriel Hunziker G. Hunziker

Zusammenfassung

In dieser Bachelorarbeit wird der Einfluss verschiedener Fehlerarten auf die Word Error Rate (WER) untersucht, einem wichtigen Mass für die Genauigkeit von Speech-to-Text (STT)-Systemen. Es werden sieben spezifische Fehlerarten betrachtet: Gross-/Kleinschreibung, Sonderzeichen, Umwandlung von Zahlen, Segmentierungsfehler, Homophone, Datumsformate und Kontraktionsfehler. Zur Durchführung dieser Analyse wurde das bestehende Tool „STT Explorer“ erweitert, um diese Fehlerarten aus der Berechnung der WER ausschliessen zu können. Nach diesen Anpassungen wurde eine detaillierte Analyse unter Verwendung des CEASR Korpus durchgeführt, um die Unterschiede in der WER vor und nach der Fehlerbereinigung zu vergleichen. Die Ergebnisse zeigen, dass durch die Einführung der Custom Word Error Rate (CWER) eine präzisere Bewertung und Verbesserung der STT-Systeme möglich ist.

Vorwort

Die Anwendung von Text-to-Speech Algorithmen wird in der heutigen Zeit immer wichtiger. Insbesondere meine Faszination wie Computer menschliche Stimmen verstehen können hat mich überzeugt mich diesem Projekt zu widmen. Erstaunlich ist unter anderem, wie gut beispielsweise Siri mittlerweile Schweizerdeutsch versteht. Die Weiterentwicklung einer Webseite in Typscript war für einen Wirtschaftsingenieur herausfordernd. Da ich jedoch sehr am Programmieren interessiert bin war dies eine willkommene Herausforderung.

Ich möchte mich herzlich bei meinen Betreuern Prof. Dr. Mark Cieliebak und Katsiaryna Mlynchyk bedanken, welche sich stets Zeit für meine Fragen genommen haben und mir mit ihren Inputs und Vorschlägen immer hilfreich bei Seite standen.

Disclaimer:
Natürlich wurde für diese Arbeit teilweise moderne KI-Basierte Tools wie DeepL Write oder Chat-GPT verwendet. Diese dienten jedoch nur als Unterstützung bei Formulierungen oder als Denkanstoss. Alle Texte und Inhalte wurden durch mich selbst erfasst und recherchiert.

Inhaltsverzeichnis

1. Einleitung

1.1. Ausgangslage

In der heutigen Zeit spielen KI-basierte Anwendungen eine immer wichtigere Rolle. Als Schnittstelle zwischen Mensch und Maschine dient vermehrt die Sprache. Diese wird nicht mehr nur in Textform, sondern zunehmend direkt als Audiosignal an den Computer weitergegeben, welcher dieses wiederum in einen Text umwandelt. Eine zuverlässige Spracherkennung ist von entscheidender Bedeutung, um diese Interaktion zu verbessern. In der Anwendung bei Firmen sind beispielsweise präzise Besprechungsprotokolle unabdingbar. Ein weiteres Beispiel ist der Einsatz eines digitalen Sprachassistenten, welcher durch verbesserte KI-Modelle immer beliebter wird.

Für diese und viele weitere Zwecke sind zuverlässige Speech-to-Text-(STT) Algorithmen unerlässlich. Um diese Algorithmen weiter zu verbessern, sind sowohl die Wissenschaft als auch die Unternehmen auf Werkzeuge zur Fehleridentifizierung in den verarbeiteten Texten angewiesen. Ein beliebtes Mittel ist das Testen von STT-Systemen anhand von Referenztexten und den von den Systemen transkribierten Texten. Um innerhalb dieser Fehler zu differenzieren und damit die Zuverlässigkeit der Spracherkennung zu verbessern, ist eine genaue Analyse essenziell.
Da hierbei grosse Datenmengen verarbeitet werden müssen, ist ein manueller Vergleich ineffizient und kaum durchführbar. Automatisierte Tools sind daher in diesem Aufgabenbereich unerlässlich. Bestehende Lösungen sind häufig nicht flexibel und kompliziert in der Anwendung. Hier setzt das Alignment Tool (STT-Explorer) der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW) an. Dieses webbasierte Tool soll die Aufgabe des Textvergleichs für STT-Systeme auf einfache und zuverlässige Weise ermöglichen und Abweichungen zwischen Referenz- und Hypothesentexten präzise erkennen, um die Genauigkeit und Zuverlässigkeit von Spracherkennungsanwendungen zu erhöhen.

Die aktuelle Anwendung ist für den Benutzer derzeit jedoch nicht anpassbar und versucht gleich zu Beginn automatisch den Text zu optimieren, damit das Alignment möglichst genau übereinstimmt. Zusätzlich ist die Berechnung des Ergebnisses sehr zeitaufwändig.

Bei der Recherche wurde keine Anwendung mit einem ähnlichen Funktionsumfang entdeckt. Es gibt jedoch diverse Python Bibliotheken, wie beispielsweise JiWER oder Evaluate.

1.2. Zielsetzung

Das Ziel dieser Arbeit besteht darin, das Backend der Applikation zu überarbeiten um die Flexibilität und die Geschwindigkeit zu erhöhen. Dazu soll die Berechnung des Alignments und der Word Error Rate (WER) optimiert werden. Weiter soll es dem Nutzer ermöglicht werden gewisse Textfehler zu Ignorieren. Hierfür soll neben der WER eine sogenannte Custom Word Error Rate (CWER) implementiert werden. Durch die CWER soll eine fehlerbasierte Analyse von Referenz- und Hypothesentexten ermöglicht werden.
Ein weiteres Ziel besteht darin, ein Backendframework einzuführen, welches eine API-Dokumentation nach Open API Standard zur Verfügung stellt.

Der erste Teil der Arbeit umfasst eine Erklärung der Word Error Rate, eine Zusammenfassung möglicher Fehlerarten von STT-Systemen sowie die Vorstellung der Idee der Custom Word Error Rate (CWER). Anschliessend wird die Funktionsweise der aktuellen Applikation vorgestellt, bevor auf die Anpassungen im eingegangen wird. Im nächsten Kapitel wird der Einfluss der verschiedenen Textfehlern, auf die WER mithilfe der CWER überprüft. Abschliessend findet eine Diskussion der Ergebnisse statt.

2. Grundlagen

2.1. Word Error Rate

Die Word Error Rate (WER) ist eine bekannte Metrik zur Bewertung der Genauigkeit von Speech-to-Text (STT) Systemen. Sie misst die Anzahl der Fehler in einem Hypothesentext im Vergleich zur korrekten Referenz.

2.1.1. Berechnung der Word Error Rate

Die Berechnung der WER erfolgt durch die Bestimmung des minimalen Editierabstand zwischen dem Referenztext und dem Hypothesentext. Dieser Editierabstand umfasst Einfügungen, Löschungen und Ersetzungen, welche erforderlich sind, um den Hypothesentext in den Referenztext zu übertragen. Die Formel für die Word Error Rate lautet:

$$WER = \frac{S + D + I}{N}$$

wobei:

- **S:** Anzahl der Ersetzungen (Substitution)
- **D:** Anzahl der Löschungen (Deletion)
- **I:** Anzahl der Einfügungen (Insertion)
- **N:** Gesamtanzahl der Wörter im Referenztext

2.1.2. Kritik an der Word Error Rate

Obwohl die WER eine weit verbreitete Metrik zur Bewertung der Genauigkeit von STT-Systemen ist gibt es einige Kritikpunkte, die ihre Aussagekraft und Anwendbarkeit einschränken.
Ein wesentlicher Kritikpunkt ist, dass die WER alle Fehlerarten gleichbehandelt, unabhängig davon, ob es sich um kleine Rechtschreibfehler handelt oder um Fehler, die die Bedeutung eines Satzes erheblich verändern. Beispielsweise wird ein fehlendes Komma genauso gewichtet wie das Ersetzen eines Wortes, wodurch sich die Bedeutung des Satzes vollständig verändern kann.
Des Weiteren berücksichtigt die WER die semantische Bedeutung der Wörter nicht. Wörter, die unterschiedlich geschrieben werden dürfen werden als Fehler gezählt. Auch unterschiedliche Schreibweisen von Dezimalzahlen oder Datumsangaben führen zu Fehlern, obwohl sie in ihrer Bedeutung identisch bleiben.
Ein weiterer Nachteil der WER ist, dass die Länge eines Satzes eine grosse Rolle spielt. In einem kurzen Satz aus drei Wörtern ergibt der Ersatz eines einzigen Wortes eine WER von über 30%, während in einem langen Satz eine einzelne Wortänderung einen viel geringeren Einfluss hat. Dies führt zu einer verzerrten Bewertung, insbesondere bei der Analyse von Texten unterschiedlicher Länge.
Diese Kritikpunkte verdeutlichen, dass die WER als Metrik zur Bewertung der Genauigkeit von STT-Systemen ihre Grenzen hat und ergänzende oder alternative Metriken in Betracht gezogen werden sollten, um eine umfassendere Bewertung zu ermöglichen.
[1], [2]

2.2. Verschiedene Textfehler

Bei der Analyse von Texten können verschieden Arten von Textfehlern auftreten, welche je nach Kontext und Anwendung unterschiedliche Auswirkungen auf die Bedeutung des Textes und die WER haben können. Zu den häufigsten Fehlerarten bei STT-Algorithmen gehören:

Singular / Plural:
Fehler, bei denen der Algorithmus das Singular transkribiert, obwohl das Plural gemeint ist, und umgekehrt.

Nummern als Zahlen:
Einige Systeme schreiben Zahlen als Worte aus, während andere Zahlen als Ziffern transkribieren. Dies führt zu Inkonsistenzen in den Texten.

Datumsangaben:
Datumsangaben können in den Referenztexten oder Hypothesen unterschiedliche Formate haben. Manche Systeme erkennen ein gesprochenes Datum und formatieren es im Text einheitlich. Des Weiteren gibt es für Datumsangaben auch international verschiedene Formate, was zu zusätzlichen Fehlern führen kann.

Sonderzeichen:
Sonderzeichen führen oft zu Fehlern beim Vergleichen der Texte, da diese von den STT-Systemen unterschiedlich gut erkannt und eingesetzt werden. Bei der WER-Analyse werden diese deshalb oft weggelassen. Ein fehlendes oder falsches Satzzeichen kann jedoch die Bedeutung eines Textes verändern.

Rechtschreibfehler:
Fehler bei der Schreibweise von Wörtern, die die Lesbarkeit beeinflussen, jedoch nicht unbedingt die Bedeutung des Textes verändern.

Kontraktionsfehler:
Vor allem im englischen Sprachgebrauch sind Textkontraktionen wie beispielsweise „gonna“ und „going to“ oder „we’ve“ und „we have“ eine häufige Fehlerquelle. Wenn die Texte bei Kontraktionen nicht korrekt getrennt werden, entsteht ein Fehler, da z.B. „we’ve“ als ein Wort und „we have“ als zwei Wörter gezählt werden.

Homophone:
Wörter, die gleich klingen, aber unterschiedliche Bedeutungen und Schreibweisen haben (z.B. „there“, „their“ und „they’re“ im Englischen), können zu Fehlern führen, wenn der Kontext nicht korrekt erkannt wird.

Formatierungsfehler:
Fehler, die durch falsche oder fehlende Formatierungselemente wie Gross- und Kleinschreibung oder Leerzeichen entstehen.

2.3. Custom Word Error Rate

Die Custom Word Error Rate (CWER) ist eine anpassbare Version der WER zur Bewertung der Genauigkeit von STT-Systemen und wurde im Rahmen dieser Bachelorarbeit ausgearbeitet. Sie soll als Ergänzung zur WER angewendet werden.

2.3.1. Definition
Die CWER soll es dem Benutzer ermöglichen, bestimmte Fehlerarten bei der Berechnung der WER zu ignorieren oder anzupassen, um eine genauere Bewertung zu ermöglichen. Das Ziel ist, dass der Benutzer individuell bestimmen kann, welche Fehlerarten er vor der Berechnung der WER ausschliessen möchte und welche nicht. Dadurch können grössere Mengen von Texten einfacher auf die verschiedenen Fehlerquellen analysiert werden.

2.3.2. Fehlerarten
Im Rahmen dieser Arbeit wurden die folgenden fünf verschiedenen Fehlerarten zur Korrektur in die CWER implementiert:

Ignorieren von Gross - / Kleinschreibung:
Diese Funktion ermöglicht es, Unterschiede in der Gross- und Kleinschreibung zu ignorieren Hierfür schreibt das CWER alle Wörter in den Texten klein.

Ignorieren von Kontraktionsfehlern
Diese Funktion erlaubt es dem Nutzer, Kontraktionen in Texten zu bereinigen. Dabei schreibt das CWER alle Kontraktionen aus

Ignorieren von Sonderzeichen
Diese Funktion entfernt alle Sonderzeichen aus den Texten. Dadurch werden Fehler vermieden, die nur durch fehlende oder unterschiedliche Satzzeichen verursacht würden.

Ignorieren von verschiedenen Datumsformaten
Diese Funktion formatiert Uhrzeiten und Datumsangaben in den Texten einheitlich. Diese werden ohne Trennung durch Sonderzeichen formatiert, sodass sie als ein einzelnes Wort betrachtet werden. Ein Fehler wird dadurch nur einmal gezählt.

Zahlen als Ziffern
Diese Funktion wandelt ausgeschriebene Zahlen in den Texten in Ziffern um.

Diese Anpassungsmöglichkeiten bieten die Möglichkeit, Texte flexibler zu analysieren. Dadurch können Schwachstellen der STT-Systeme gezielt identifiziert und untereinander verglichen werden.

2.4. Python Bibliotheken zur Berechnung der Word Error Rate

In der aktuellen Version des ZHAW STT-Explorer wird das Alignment der Wörter und die Berechnung der WER direkt im Backend ausgeführt. Dies ist jedoch vor allem bei grossen Textmengen nicht sehr effizient. Es gibt aber einige Python Bibliotheken, welche diese Arbeit effizienter ausführen. Im Rahmen dieser Bachelorarbeit wurde deshalb die Geschwindigkeit mehrerer Python Bibliotheken verglichen, um die effizienteste Lösung für den STT-Explorer zu evaluieren. Gleichzeitig sollen die Resultate der WER von den verschiedenen Bibliotheken verglichen werden, um sicherzustellen, dass die Fehlererkennung möglichst präzise ist.

2.4.1. Übersicht der verglichenen Bibliotheken

Im Vergleich wurden folgende Python-Bibliotheken analysiert:

JiWER:
JiWER ist eine Python-Bibliothek, die neben der WER auch andere Metriken wie die Match Error Rate (MER) oder die Character Error Rate (CER) berechnen kann. Sie basiert auf der minimalen Bearbeitungsdistanz, die durch RapidFuzz berechnet wird. RapidFuzz führt die Berechnungen in C++ durch, was besonders performant sein soll. Zudem bietet JiWER in Python eine grafische Darstellung des Alignments an [3], [4].
Werpy:
Werpy ist auf Geschwindigkeit optimiert und berechnet ebenfalls die WER zwischen zwei Sätzen. Es ist mehrheitlich flexibel und kann mit verschiedenen Input-Daten wie Listen oder NumPy-Arrays umgehen. Die Unterschiede der Texte werden mithilfe von Pandas und NumPy verarbeitet. Für die Berechnung kommt Cython zum Einsatz [5].
FastWER:
FastWER berechnet die WER und die CER von Texten. Die Berechnung geschieht ebenfalls in C++, was die Geschwindigkeit optimieren soll [6].
Evaluate:
Evaluate, entwickelt von der Hugging Face Community, bietet diverse Werkzeuge für das Natural Language Processing. Für die Berechnungen greift Evaluate auf diverse Python Bibliotheken, unteranderem JiWER, zu [7].

2.4.2. Vergleich

Für den Vergleich wurde der Korpus CEASR verwendet. CEASR ist ein Datensatz, der eine Sammlung öffentlich zugänglicher Sprachkorpora umfasst. Er enthält transkribierte Texte in verschiedenen Sprachstilen und von einer Vielzahl kommerzieller und Open-Source-STT-Systeme. CEASR beinhaltet sowohl englische als auch deutsche Texte. Die Analyse wurde direkt

in Python durchgeführt. Dabei wurde für alle Texte des Korpus die WER berechnet und die Zeit für die Berechnung aufgezeichnet. Die Einstellungen wurden dabei möglichst auf dem Standard belasse [8].

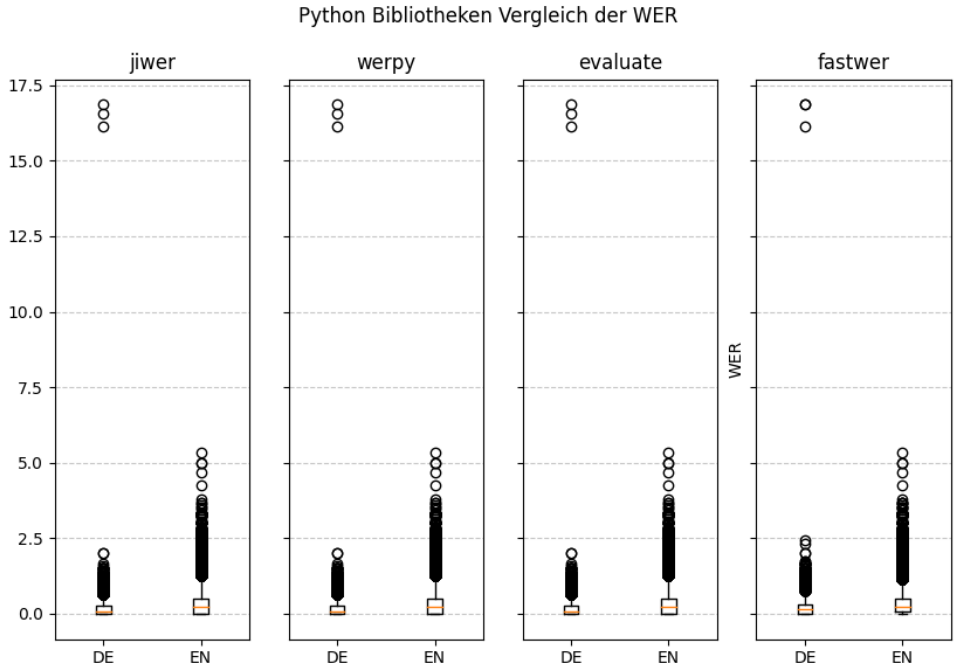


Abbildung 1: Vergleich der WER aus verschiedenen Python Bibliotheken

In den Boxplots in welchen die WER dargestellt sind, ist sichtbar, dass die verschiedenen Bibliotheken ähnliche Resultate berechnen. Auffällig ist, dass die Werte bei den englischen Texten höher sind als bei den deutschen. Es gibt sehr viele Ausreisser in beiden Versionen. Diese stammen wahrscheinlich von Texten, in denen der Referenz – oder Hypothesentext viel kürzer oder länger ist.

Tabelle 1: Mittelwert und Median WER verschiedener Python Bibliotheken

	Bibliothek	**Median WER**	**Mittelwert WER**
EN	JiWER	0.2069	0.3299
	Werpy	0.2069	0.3299
	Evaluate	0.2069	0.3299
	fastwer	0.2308	0.3452
DE	JiWER	0.0833	0.1567
	Werpy	0.0833	0.1556
	Evaluate	0.0833	0.1557
	fastwer	0.1428	0.2002

Da der Mittelwert durch kurze Sätze beeinflusst werden kann, ist der Median für diesen Vergleich das aussagekräftigere Mass. Erfreulich ist, dass sich die Resultate über alle Pakete hinweg sehr ähnlich sind. Nur das Paket fastwer bietet ein stark abweichendes Resultat, was möglicherweise daran liegt, dass die Texte anders vorverarbeitet werden. Im Boxplot ist zudem sichtbar, dass es viele Ausreisser gibt. Da alle Bibliotheken jedoch auch die Ausreisser gleich berechnen, kann ein Fehler der Tools ausgeschlossen werden. Wahrscheinlich handelt es sich hierbei um Referenzen und Hypothesen, deren Längen sich stark unterscheiden.

Tabelle 2: Mittlere Berechnungszeiten der WER für verschiedene Python Bibliotheken

Bibliothek	**Mittlere Bearbeitungszeit pro Berechnung**
JiWER	$4.466 * 10^{-5} s$
Werpy	$8.679 * 10^{-5} s$
Evaluate	$277.298 * 10^{-5} s$
fastwer	$1.107 * 10^{-5} s$

FastWER ist eindeutig die schnellste Bibliothek. JiWER erzielte das zweitbeste Resultat. Werpy ist etwas langsamer als JiWER, liegt jedoch in einem ähnlichen Bereich. In der Bearbeitungszeit hat Evaluate am schlechtesten abgeschnitten.

Analyse der Ergebnisse
Die Analyse zeigt, dass JiWER und Evaluate exakt die gleichen WERs berechnen, wobei Evaluate aufgrund der Verwendung von JiWER langsamer ist. Werpy liefert sehr ähnliche Ergebnisse wie JiWER und Evaluate, ist jedoch ebenfalls langsamer. Fastwer verwendet einen anderen Berechnungsansatz und ist schneller, liefert aber leicht unterschiedliche WER-Werte.
Die Untersuchung ergab, dass JiWER anscheinend innerhalb von Evaluate als Paket genutzt wird. Aufgrund der abweichenden Ergebnisse bei fastwer wird diese Bibliothek in der weiteren Analyse nicht mehr berücksichtigt. Ein Vorteil von JiWER ist die gute Dokumentation, die eine flexible Anpassung und Erweiterung der Funktionen ermöglicht. Daher könnte der STT-Explorer auf ähnliche Methoden wie JiWER zurückgreifen.

Aufgrund seiner Flexibilität, Genauigkeit und er guten Dokumentation wurde JiWer für die Berechnung der WER im Rahmen dieser Arbeit verwendet.

3. Übersicht der bestehenden Applikation

3.1. Überblick

Der STT-Explorer ist eine Webapplikation, welche zur Analyse und Bewertung von STT-Systemen entwickelt wurde. Sie ermöglicht es, die Genauigkeit verschiedener STT-Systeme durch den Vergleich von Referenz- und Hypothesentexten zu bewerten. Dazu berechnet das Tool die Word Error Rate (WER) und stellt das Ergebnis des Textalignments grafisch dar. Ziel ist, eine Plattform zu bieten, die diese Werkzeuge zur Bewertung von transkribierten Texten auch ohne aufwändige Programmierung zur Verfügung stellt. Das aktuelle Tool besteht aus einem Backend zur Berechnung der Daten und einem Frontend für die Darstellung. Beides läuft als Docker Container auf einem Server der ZHAW.
Dieses Kapitel stellt den aktuellen Stand und Aufbau des Tools vor.

3.2. Frontend

Das Frontend des STT-Explorers ist die Schnittstelle zwischen dem Anwender und dem Backend. Innerhalb des Frontends werden die Daten zur Analyse hochgeladen, selektiert und an das Backend zur Auswertung gesendet. Anschliessend werden aus dem Backend empfangenen Daten zur Auswertung grafisch dargestellt.
Das Frontend ist in Typescript geschrieben und verwendet React als Frontend-Framework. Im Rahmen dieser Arbeit soll das bestehende Frontend mehrheitlich beibehalten werden. Ziel ist, dass die neuen Funktionen möglichst nahtlos in das bestehende Design integrieren werden und alle nötigen nicht sichtbaren Anpassungen vorgenommen werden.

3.2.1. Verwendete Bibliotheken und Frameworks

React
React ist ein Frontend-Framework, welches ursprünglich von Facebook entwickelt wurde. Heute wird es durch die Meta Open Source Initiative weiterentwickelt. Der Fokus von React liegt auf der Erstellung effizienter und flexibler Benutzeroberflächen mithilfe modularer Codeeinheiten, den sogenannten Komponenten. Durch diese Komponenten lässt sich eine Applikation besser in separate, kürzere und wiederverwendbare Codeeinheiten unterteilen.

Vite
Vite ist ein Frontend-Entwicklungsserver, welcher durch das Hot Module Replacement (HMR) den Entwicklungsprozess beschleunigen kann. Dadurch kann der Code live angepasst werden und die Änderungen werden nach jedem Speichervorgang sofort sichtbar.

MUI
MUI ist eine der bekanntesten React-Komponentenbibliotheken, welche eine umfangreiche Sammlung an vorgefertigten und anpassbaren Komponenten bietet. Dadurch wird das Erstellen einheitlicher Benutzeroberflächen vereinfacht. Die MUI-Komponenten orientieren sich am Material Design Konzept von Google. Die Komponentenbibliothek ist für Mobileanwendungen und für den Desktop optimiert.

3.2.2. Bestehende Funktionen und Erscheinungsbild

Startseite und Dateiupload

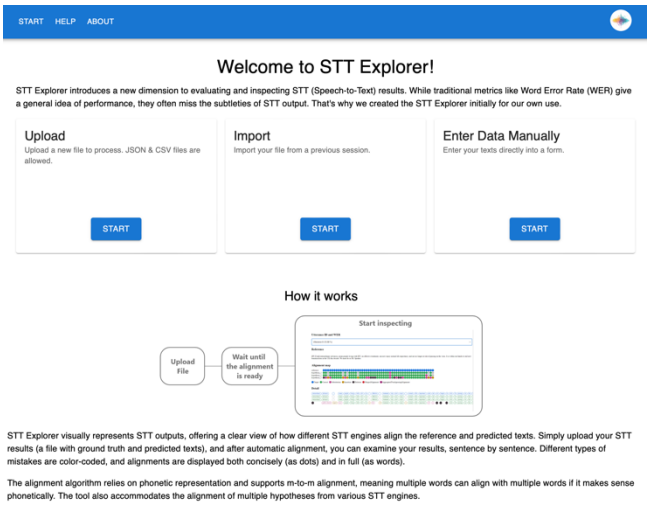


Abbildung 2: Aktuelle Startseite des STT-Explorers

Die Startseite des STT-Explorers beinhaltet eine kurze Erklärung des Tools und drei Startoptionen:

- **Upload**
  Mithilfe des Uploads kann eine neue Datei, welche Referenzen und Hypothesen enthält importiert werden.
- **Import**
  Beim Import wird ein exportiertes Projekt erneut geladen.
- **Enter Data Manually**
  Diese Option wird gewählt, um Texte manuell oder per Copy & Paste zu erfassen.

Textselektion

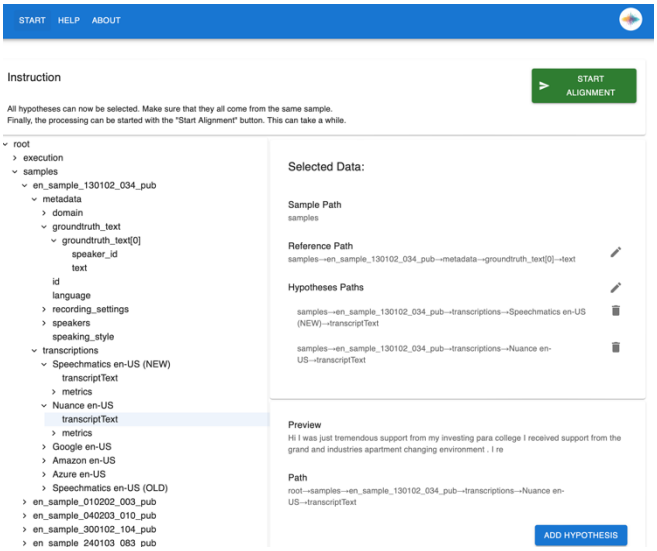


Abbildung 3: Textselektion des aktuellen STT-Explorers

Nachdem eine neue Datei selektiert wurde, muss der Benutzer die Pfade zu den Texten auswählen. Dieser Prozess umfasst mehrere Schritte:

1. **Auswahl des Pfades zu den Samples**
   Der Punkt, bevor sich die Pfade zu den verschiedenen Text- und Referenzkombinationen aufteilt.
2. **Auswahl des Referenztextes**

Aus einem der Unterpfade muss ein Referenztext ausgewählt werden.

3. **Auswahl der Hypothesen**
   Auswahl aller auszuwertenden Hypothesentexten zur markierten Referenz.
4. **Start Alignment**
   Nach der Auswahl des «Start Alignment» Buttons werden die Daten an das Backend zur Bearbeitung gesendet.

Auswertung der Ergebnisse

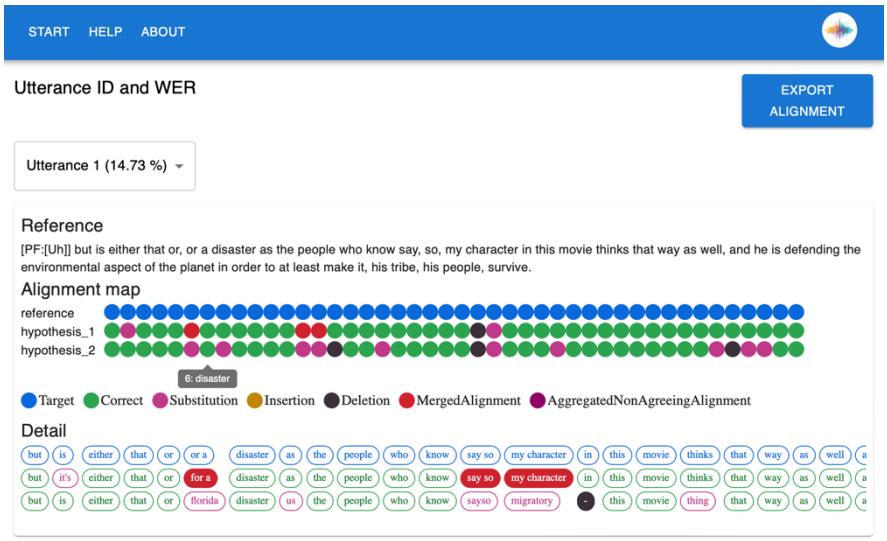


Abbildung 4: Auswertung der Ergebnisse des aktuellen STT-Explorers

Sobald die Berechnungen im Backend abgeschlossen sind, stehen die Ergebnisse zur Beurteilung im Frontend zur Verfügung. Dabei ist die Ansicht in verschiedene Bereiche unterteilt:

- **Auswahl des Textes**
  Im Dropdown-Menü kann ein verarbeiteter Text ausgewählt werden. Die Texte sind als Utterance mit einer Nummer beschriftet. Diese Nummer beschreibt die Reihenfolgen der Texte innerhalb des Datensatzes. Daneben ist die WER in Prozent ersichtlich.
- **Alignment Map**
  In der Alignment Map wird jedes Wort als Punkt dargestellt und farblich hervorgehoben. Je nach Farbe wird dadurch angezeigt, ob es korrekt ist, ersetzt, gelöscht oder eingefügt wurde. In der ersten Zeile befindet sich immer der Referenztext.
- **Detailbereich**
  Der Detailbereich ist grundsätzlich gleich aufgebaut, wie die Alignment Map. Anstelle der Punkte sind jedoch direkt die einzelnen Wörter sichtbar.
- **Export Alignment**
  Mithilfe dieses Buttons kann das Resultat exportiert werden. Dadurch kann es zu einem späteren Zeitpunkt erneut dargestellt werden, ohne dass das Backend die Berechnungen erneut durchführen möchte.

3.3. Backend

Das Backend spielt eine zentrale Rolle bei der Datenverarbeitung des STT-Explorers. Darin werden alle Berechnungen und Datenmanipulationen durchgeführt.

3.3.1. Verwendete Bibliotheken und Frameworks

Das Backend ist komplett in Python geschrieben. Python ist eine weitverbreitete Programmiersprache, welche unteranderem oft im Natural Language Processing (NLP)-Bereich zur Anwendung kommt.

Flask
Als Webframework innerhalb von Python wird Flask eingesetzt. Flask ist ein äusserst schlankes Framework, welches nur die notwendigsten Bibliotheken enthält. Dadurch eignet es sich sehr gut für kleinere Anwendungen im Backend.
Gunicorn
Als Webserver in der produktiven Umgebung kommt Gunicorn zum Einsatz, welcher für seine Effizienz und Zuverlässigkeit bekannt ist.

Des Weiteren werden für die Berechnungen des Alignments Pandas und Numpy verwendet. Hierbei handelt es sich um Bibliotheken, welche die Handhabung grösserer Datenmengen innerhalb von Phyton vereinfachen.

3.3.2. Bestehende Funktionen

Das Backend des STT-Explorers empfängt die Daten zur Auswertung vom Frontend über eine REST-Schnittstelle. Für diesen Datenaustausch sind im Backend zwei verschiedene Endpunkte vorgesehen:

- **/api/upload_file:**
  Dieser Endpunkt wird für die Auswertung von Dateien verwendet. Die Pfade zu den Samples, Referenz- und Hypothesentexten werden in derselben Anfrage übertragen.
- **/api/send_utterances:**
  Dieser Endpunkt dient dem Empfang der manuell eingegebenen Texte aus dem Frontend. Hierfür wird ein JSON-Objekt mit allen Referenzen und Hypothesen an diesen Endpunkt gesendet.

Nach dem Empfang der Daten aus dem Frontend und der Extraktion der Texte, werden diese aufbereitet. Dies beinhaltet das Entfernen von Sonderzeichen und Abkürzungen, sowie die Konvertierung von Gross- zu Kleinbuchstaben.
Anschliessend werden die Wörter der Hypothesentexte mit den Worten der Referenz verglichen und zu einem Alignment verarbeitet.
Das Resultat wird schlussendlich als Antwort in Form eines gesamten Datenpakets an das Frontend zurückgesendet.

4. Erweiterung des Backends

Da das Backend kontinuierlich erweitert wurde, ist der Aufbau mittlerweile etwas unübersichtlich geworden. Momentan bietet die aktuelle Version keine Option zur direkten Berechnung der WER im Backend. Diese Berechnung erfolgt derzeit im Frontend auf Basis der Fehler, die im Alignment markiert sind.
Ein weiterer Punkt, der verbessert werden muss, ist die Struktur der Endpunkte. Da das Backend nur zwei Endpunkte umfasst, müssen sämtliche Daten innerhalb einer einzigen Anfrage berechnet und anschliessend zurück an das Frontend gesendet werden. Dies führt insbesondere bei grossen Textdateien zu sehr langen Wartezeiten im Frontend. Zudem werden die Daten vor dem Alignment bereits stark vorverarbeitet, sodass die WER auch auf diese bearbeiteten Daten angewendet wird.

4.1. Ziele der Erweiterung

Ein Ziel der Erweiterung des Backends besteht darin, ein Framework für die REST-Schnittstelle zu integrieren, welches gleichzeitig eine Swagger API-Schnittstellendokumentation

bereitstellt. Durch diese Dokumentation des Backends soll in Zukunft die unabhängige Weiterentwicklung des Front- und Backends erleichtert werden.
Zudem soll zu Beginn eine Standard-WER ohne umfangreiche Textaufbereitung direkt im Backend berechnet werden können. Durch diesen Schritt kann die Leistung stark verbessert werden.

Ein weiteres Ziel ist, dem Nutzer die Möglichkeit zu geben, die berechnete WER im Rahmen der definierten CWER anzupassen. Dies ermöglicht einen flexibleren Einsatz des STT-Explorers.

Zusätzlich sollen zusätzliche Endpunkte eingeführt werden, um die Aufgaben besser zu verteilen und die Effizienz zu steigern. Dies umfasst separate Endpunkte für das Hochladen von Daten, die Berechnung der CWER und das Abrufen der Ergebnisse.

Diese Massnahmen erhöhen die Effizienz und Flexibilität des STT-Explorers.

4.2. Konzeption

4.2.1. API-Endpunkte

Aktuell müssen bei jeder Anfrage an das Backend sämtliche Daten übertragen werden. Da vor allem das Alignment sehr gross werden kann, ist diese Lösung nicht sehr effizient. Durch die neue API sollen die Daten im Backendbleiben und nur eine Upload-ID an das Frontendgesendtet werden. Mithilfe dieser Upload-ID kann das Frontend die benötigten Daten abrufen. Um diese Funktion bereitzustellen, sollen diverse Endpunkte für folgende Funktionen hinzugefügt werden:

- Upload der Daten
- Export eines Projektes
- Import eines exportierten Projektes
- Abfragen der CWER-Einstellungen
- Anpassung der CWER-Einstellungen
- Abfragen der Samples
- Abfragen der Referenzen
- Abfrage der Hypothesen
- Abfragen des Alignments

4.2.2. Datenstruktur

Zu Beginn des Projektes wurde überlegt, ob eine Datenbank zur Speicherung der Daten sinnvoll wäre. Grundsätzlich wäre eine Datenbankanbindung die beste Lösung, sie würde jedoch den Umfang des Backends enorm vergrössern und den Rahmen dieser Bachelorarbeit sprengen, da der Fokus auf der Implementierung der CWER liegen soll.

Deshalb sollen die Daten nun in CSV-Dateien gespeichert werden, welche die nötige Flexibilität und Lesbarkeit bieten. Es wurde darauf geachtet, dass die Struktur so gestaltet ist, dass eine spätere Datenbankerweiterung ohne grosse strukturelle Anpassungen im Code möglich ist.

Die Daten werden in vier verschiedenen .csv Dateien gespeichert:

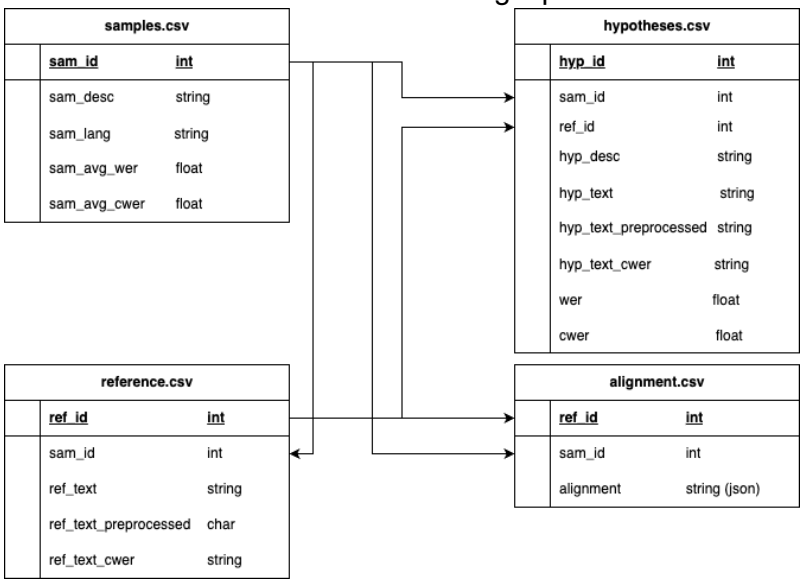


Abbildung 5: Konzeption der neuen Datenstruktur für das Backend

samples.csv:
Diese Datei enthält eine eindeutige Sample-ID, eine Beschreibung, die Sprache der Samples sowie die berechnete durchschnittliche WER und CWER über das gesamte Sample.

reference.csv:
Diese Datei enthält die Sample-ID, eine Referenz-ID, den Referenztext vor der Aufbereitung, den aufbereiteten Text für die WER und den bearbeiteten Text für die CWER. Die Kombination der Sample-ID und Referenz-ID ist eindeutig.

hypotheses.csv:
Diese Datei enthält eine Hypothesen-ID, eine passende Sample-ID und eine Referenz-ID. Weiter beinhaltet sie eine Beschreibung der Hypothese (z.B. den Namen des STT-Systems), den originalen Text der Hypothese, den aufbereiteten Text für die WER und den Hypothesentext für die CWER. Ausserdem werden zu jeder Hypothese die WER und die CWER gespeichert.
Die Kombination aus Sample-ID, Referenz-ID und Hypothesen-ID ist in dieser Datei eindeutig.

alignment.csv:
Diese Datei enthält die Referenz-ID und die dazugehörige Sample-ID, sowie das Alignment als String im JSON-Format.

Zusätzlich wird im selben Ordner eine **settings.json**-Datei abgelegt. Diese enthält die Einstellungen für die Vorverarbeitung und für die CWER.

Diese Dateien sollen auf dem Server in einem Ordner gespeichert werden, der mit der Upload-ID beschriftet ist. Dadurch werden die richtigen Dateien leicht identifizierbar.

4.3. Rest API

Da durch die Erweiterung des Backends viele neue Endpunkte hinzugefügt werden, wird die Bibliothek Flask-RESTx implementiert. Diese Bibliothek bietet Werkzeuge, um strukturierte und erweiterbare REST-APIs zu erstellen. Zuvor wurde keine spezielle Bibliothek für die REST-API verwendet, was bei einem Backend mit wenigen Endpunkten nicht zwingend notwendig ist. Mit der zunehmenden Anzahl von Endpunkten wird die API jedoch unübersichtlich, was die Verwendung einer spezialisierten Bibliothek notwendig macht.

Mit Flask-RESTx können die API-Endpunkte klar und strukturiert definiert werden. Diese erleichtert die Wartung und Erweiterung der API erheblich. Zudem integriert Flask-RESTx Swagger, eine automatische API-Dokumentation, die über die URL der API abrufbar ist. Diese erleichtert es zukünftigen Projektteilnehmern, die Funktionen der API zu verstehen und zu nutzen.
Ein weiterer Vorteil von Flask-RESTx ist die Verwendung von Namespaces. Durch sie können verwandte API-Endpunkte gruppiert und organisiert werden, was die Übersichtlichkeit des Codes erhöht und der Dokumentation erhöht.

Die neu Defnierten API Endpunkte, sind im Anhang angefügt.

4.4. Erstellung des Alignments

4.4.1. Aufbereitung der Texte

Bevor das Alignment berechnet wird, werden die Texte aufbereitet. Dies erfolgt durch das Aufteilen des Textes mithilfe von regulären Ausdrücken (Regex). Das Ziel dieser Aufbereitung ist, den Text in einzelne, analysierbare Komponenten zu zerlegen.

Das Regex nimmt folgende Änderungen am Text vor:

1. **Trennung bei Sonderzeichen**
   Der Text wird bei Sonderzeichen aufgesplittet. Mehrere aufeinanderfolgende Sonderzeichen werden jedoch nicht aufgetrennt.
2. **Trennung von Zahlen und Ziffern**
   Zahlen welche als Ziffern im Text stehen, werden von den Wörtern und Sonderzeichen getrennt.
3. **Behandlung von Leerzeichen**
   Leerzeichen am Anfang und am Ende des Textes, sowie doppelte Leerzeichen innerhalb des Textes werden entfernt.

4.4.2. Berechnung des Alignements

Nach der Textaufbereitung wird im nächsten Schritt das Alignment berechnet.

Zunächst wird der Datensatz in einzelne Gruppen aufgeteilt, die jeweils aus einem Referenztext und den zugehörigen Hypothesen bestehen. Dieser Schritt ermöglicht die Parallelisierung des Alignment-Prozesses und erhöht somit die Geschwindigkeit der Berechnungen im Backend.

Für jede Kombination aus Referenztext und Hypothese wird ein separates Alignment berechnet, welches im letzten Schritt zu einem einzigen Alignment zusammengefügt wird.

Da JiWER als die effizienteste Python-Bibliothek evaluiert wurde, wird die Berechnung ähnlich wie in JiWER durchgeführt.
Mit RapidFuzz werden die Editieroperationen (Editops) berechnet. Diese beschreiben die Schritte, die notwendig sind, um die Hypothese an die Referenz anzupassen, und beinhalten folgende Aktionen: Ersetzen, Einfügen oder Löschen.
Die Editops werden anschliessend in Opcodes umgewandelt. Opcodes sind strukturierte Anweisungen, die den Start- und Endindex der Operation sowie die Art der Aktion (Ersetzen, Einfügen oder Löschen) für den Referenz- und Hypothesentext zurückgeben.

Für jedes Alignment wird ein Pandas DataFrame erstellt, welches alle Wörter der Referenz und der Hypothese beinhaltet. Zudem umfasst dieses Alignment den Index des Referenzwortes, den Index des Hypothesenwortes, die Fehlerart und bei Einfügungen einen Index, welcher die aufeinanderfolgenden Einfügungen zählt. Dieser Index wird später benötigt, um die Alignments zusammenzufassen.

In der nachfolgenden Tabelle ist ein Beispiel für ein einzelnes Alignment aufgeführt:

Tabelle 3: Beispiel eines berechneten Alignments als Dataframe

ref_id	ref_text	hyp_id	hyp_I_count	hyp_text	type	cwer_ignored
-1	<NA>	0	0	hallo	I	False
0	Hallo	1	<NA>	hallo	S	False
1	das	2	<NA>	das	C	False
2	hier	3	<NA>	hier	C	False
3	ist	4	<NA>	ist	C	False
4	ein	5	<NA>	ein	C	False
4	<NA>	6	0	schoner	I	False
5	kurzer	7	<NA>	kurzer	C	False
6	Text	8	<NA>	Text	C	False
7	.	9	<NA>	diser	S	False

Spalten des Alignments:

- **ref_id**
  Index des Wortes im Referenztext
- **ref_text**
  Wort im Referenztext
- **hyp_id**
- Index des Wortes im Hypothesentext
- **hyp_I_count**
  Zähler für aufeinanderfolgende Einfügungen
- **hyp_text**
  Wort im Hypothesentext
- **type**
  Art der Operation (C = Korrekt, S = Ersetzen, I = Einfügen, D = Löschen)
- **cwer_ignored**
  Gibt an, ob die Operation bei der Berechnung der Custom Word Error Rate ignoriert wurde. Diese Spalte wird nur für das Alignement der CWER benötigt.

Da für jede Hypothese ein separates Alignment berechnet wird, müssen diese Alignments im nächsten Schritt zusammengefasst werden. Deshalb wird jedes Alignment in einer Schleife mithilfe der Pandas-Merge-Funktion zusammengeführt. Dabei wird ein outer Join auf die Spalten «ref_id», «ref_text» und «hyp_I_count» angewendet, um alle Informationen der verschiedenen Hypothesen in einer gemeinsamen Struktur zu vereinen, ohne Wörter zu verlieren.

4.4.3. Berechnung der Word Error Rate

Die WER wird direkt nach dem Erstellen jedes Alignments berechnet. Dazu wird die Anzahl der Fehler (Ersetzungen, Einfügungen und Löschungen) durch die Gesamtanzahl der Wörter im Referenztext dividiert.

4.5. Implementierung der Custom Word Error Rate

Um die CWER zu berechnen, müssen die Fehler im Text vorab bereinigt werden. Hierfür wurde für jede Fehlerart eine separate Funktion implementiert. Die Reihenfolge, in der die Fehler bereinigt werden, spielt dabei eine wichtige Rolle. Beispielsweise ist es notwendig, Sonderzeichen erst im letzten Schritt zu entfernen, da sonst die Erkennung anderer Optionen beeinträchtigt werden könnte.

Die Reihenfolge der Fehlerbereinigung wurde deshalb wie folgt festgelegt:

1. Ersetzen der Grossbuchstaben
2. Auflösung von Kontraktionen
3. Normalisierung der Datumsangaben
4. Umwandlung von Zahlen in Textform
5. Entfernung von Sonderzeichen

Die für die CWER angepassten Texte werden ebenfalls in den CSV-Dateien gespeichert, sodass sie im Frontend angezeigt und analysiert werden können.

Nach der Aufbereitung der Texte werden das Alignment und die Berechnung der WER (in diesem Fall die CWER) auf demselben Weg durchgeführt, wie für die normalen Texte.

In den nachfolgenden Unterkapitel wird beschrieben, wie die einzelnen Fehlerarten erkannt und bereinigt wurden.

Ersetzen von Grossbuchstaben

Das Ersetzen von Sonderzeichen wird mittels der Python-Funktion .lower() durchgeführt.

Auflösung von Kontraktionen

```
def remove_contractions(text):
    """
    Returns text string with writeen out contractions
    Example: "They're working." -> "They are working."
    Issues:
    Contextual contractions might be incorrect. Like He's -> (He is or He has)
    """
    def replace(word):
        for pattern, repl in {**simple_contractions, **contextual_contractions}.items():
            if pattern.fullmatch(word):
                replacement = repl if isinstance(repl, str) else repl[0]
                if word.istitle():
                    return replacement.capitalize()
                elif word.isupper():
                    return replacement.upper()
                elif word[0].isupper():
                    return replacement[0].upper() + replacement[1:]
                else:
                    return replacement.lower()
        return word
    words = text.split()
    expanded_words = []
    i = 0
    while i < len(words):
        # Check for triplets --> "I" "'" "d"
        if i + 2 < len(words):
            triplet = words[i]  + words[i+1]  + words[i+2]
            replaced_triplet = replace(triplet)
            if replaced_triplet != triplet:
                expanded_words.append(replaced_triplet)
                i += 3
                continue
        # Check for quintuplets --> "hadn" "'" "t" "'" "ve" "had not have"
        if i + 4 < len(words):
            quintuplets = words[i] + words[i+1] + words[i+2] + words[i+3] + words[i+4]
            replaced_quintuplets = replace(quintuplets)
            if replaced_quintuplets != quintuplets:
                expanded_words.append(replaced_quintuplets)
                i += 2
                continue
        # check for single words --> "gonna" -> "going to"
        single_word = words[i]
        replaced_word = replace(single_word)
        expanded_words.append(replaced_word)
        i += 1

    return " ".join(expanded_words)
```

Zur Bereinigung der englischen Kontraktionen werden diese durch das Ausgeschriebene Wort korrigiert. Die Herausforderung dabei ist, dass der Text zu diesem Zeitpunkt bereits bei den Sonderzeichen getrennt ist. Deshalb werden die Wörter jeweils in Dreier- und Fünfergruppen geprüft. Sollte dies nicht zu einem Match mit einer Kontraktion führen, wird dieses das Wort zusätzlich separat getestet um auch Kontraktionen, wie «gonna» zu ersetzen.
Die Überprüfung der Textteile oder des Wortes wird mit Hilfe einer Sammlung von Kontraktionen durchgeführt. Schlussendlich wird die Gross- und Kleinschreibung des Ausgangstextes wieder hergestellt.
Einfache Kontraktionen werden gut erkannt. Es gibt jedoch einige Probleme bei den Kontext bezogenen Textkontraktionen wie zum Beispiel «'s» oder «'d».

Normalisierung von Datumsangaben

Bei der Verarbeitung von Datums- oder Zeitangaben sollen Fehler durch ein einheitliches Muster korrigiert werden. Die Erkennung im Text erfolgt mittels eines regulären Ausdrucks. Das Parsen der Datums- und Zeitangaben wird mithilfe der Python-Bibliothek dateutil durchgeführt. Wird ein Datum erkannt, wird es durch das Format «%Y%m%d» ersetzt. Eine Datumsangabe mit Uhrzeit wird im Format «%Y%m%d%H%M» ausgegeben, und eine reine Zeitangabe wird durch das Format «%H%M» ersetzt. Die Idee ist, dass ein Datum in der CWER als ein einzelnes Wort betrachtet wird, wobei mithelfe dieser Lösung keine Informationen verloren werden.

Ein Trennzeichen wurde absichtlich nicht hinzugefügt, damit die Datums- und Zeitangaben später nicht wieder aufgetrennt werden.

Umwandlung von Zahlen in Textform

```
def cleanup_word2numbers(text: str):
    """
    Returns text string with numbers as digits.

    Issues:
    Doesn't recognise numbers with a 'and' correct.
    Because one hundred and ten could mean 110 or 100 10
       Run this function before removing special characters like "," because otherwise
       "two two" will become 22 instead of 2 2
    """
    fill_words = ['-', 'point']
    words = text.split()
    result = []
    current_number_words = []
    w2n = Word2Num()

    for word in words:
        if en_num_pattern.fullmatch(word) or (current_number_words and word in
fill_words):
            current_number_words.append(word)
        else:
            if current_number_words:
                try:
                    number = w2n.parse(" ".join(current_number_words))
                    if number is not None and int(number) == number:
                        result.append(str(int(number)))
                    else:
                        result.append(str(number))
                except ValueError:
                    result.extend(current_number_words)
                current_number_words = []
            result.append(word)

    if current_number_words:
        try:
            number = w2n.parse(" ".join(current_number_words))
            if number is not None and int(number) == number:
                result.append(str(int(number)))
            else:
                result.append(str(number))
        except ValueError:
            result.extend(current_number_words)

    return " ".join(result)
```

Als Wort ausgeschriebene Zahlen in den Texten sollen durch Ziffern ersetzt werden. Zur Umwandlung kann die Python Bibliothek Word2Num verwendet werden. Die Erkennung einer ausgeschriebenen Zahl erfolgt ebenfalls mit Regex. Danach wird überprüft, ob es sich beim darauffolgenden Wort ebenfalls um eine Zahl handelt. Dadurch lässt sich sicherstellen, dass auch grössere Zahlen erkannt werden. Sind die Zahlen durch einen Bindestrich oder durch das Wort «point» getrennt wird es ebenfalls als Teil der Zahl betrachtet. Das Wort and wurde absichtlich in der Suche weggelassen, da dieses oft Kontextabhängig ist.

Entfernung von Sonderzeichen

Die Funktion der Sonderzeichen entfernt mittels Regex alle Zeichen ausgenommen von Zahlen sowie Klein- und Grossbuchstaben.

5. Anpassungen am Frontend

Durch die Erweiterung der Endpunkte, der Änderung der Datenstruktur und der Implementierung der CWER sind im Frontend umfangreiche Änderungen nötig. Durch die Anpassungen empfängt das Frontend nun nicht mehr alle Texte und Alignments auf einmal, sondern ruft jeweils die Daten beim Backend ab, sobald ein Sample ausgewählt wird.

Textselektion

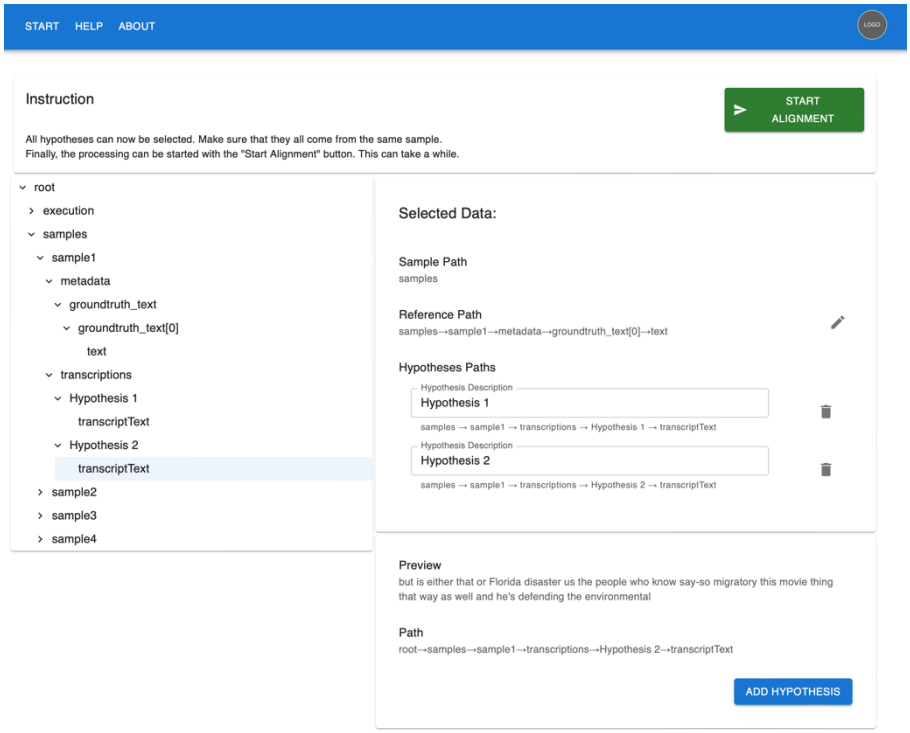


Abbildung 6: Textselektion inkl. der Option zur Beschriftung der Hypothesen

Innerhalb der Textselektion wurde die Option zur Beschriftung der Hypothesen hinzugefügt. Dadurch können die Texte während der Auswertung besser identifiziert werden.

Auswertung des Ergebnisses

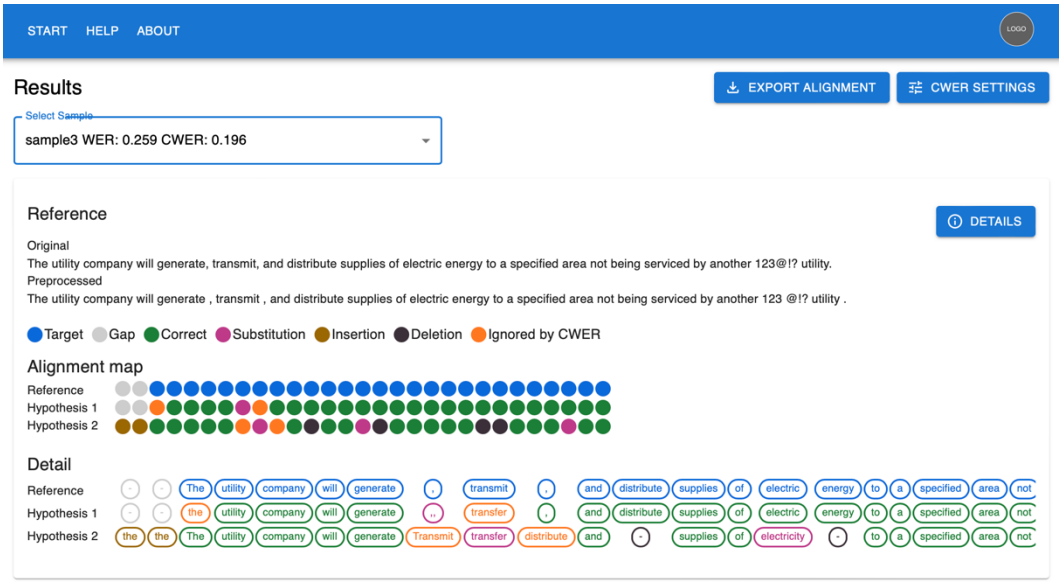


Abbildung 7: überarbeitete Version der Auswertungsseite

Beim Ergebnis wurden zwei weitere Buttons hinzugefügt. Der CWER Settings Button dient zur Konfiguration der CWER. Diese wird jeweils nach dem Speichern neu berechnet. Anschliessend wird das Dropdownmenü um die durchschnittliche CWER des Samples erweitert.

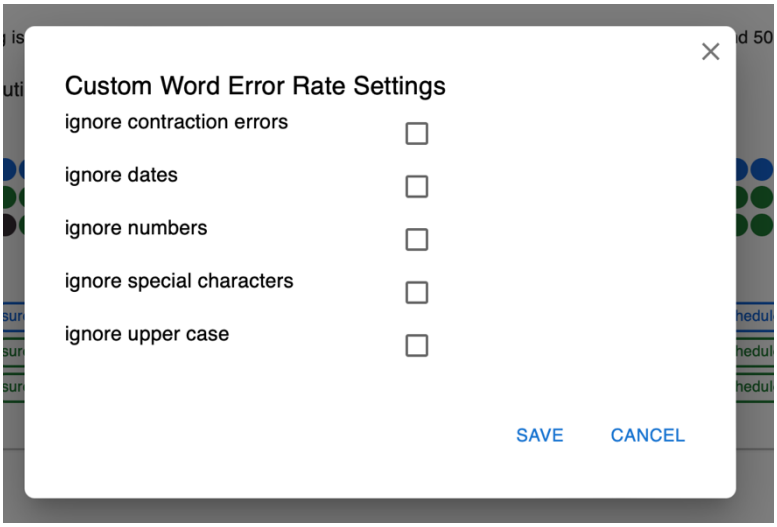


Abbildung 8: Auswahl der Optionen für die CWER

Hinter dem Details Button sind die Originaltexte und die bearbeiteten Texte der WER und CWER abrufbar. Ausserdem werden die einzelnen Werte für die WER und CWER pro Hypothese aufgelistet.

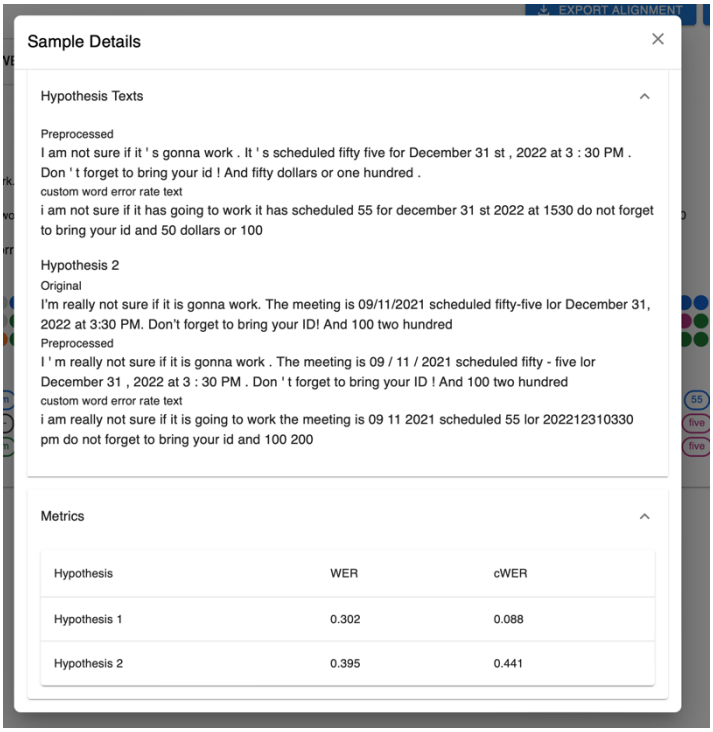


Abbildung 9: Detailansicht der Daten eines Samples

6. Vergleich der WER mit der CWER

Anhand der Daten des CEASR-Korpus soll der Einfluss des Unterschieds zwischen der Custom Word Error Rate (CWER) und der Word Error Rate (WER) evaluiert werden. Dazu wurden im ersten Schritt alle Referenz- und Hypothesentexte des englischsprachigen Teils extrahiert. Um die Dateigrösse zu reduzieren, wurde die Datei in zwei Hälften aufgeteilt.
Anschliessend wurden beide Teile für alle Fehlerkorrekturen mit dem Backend verarbeitet. Da sowohl die CWER als auch die WER viele Ausreisser enthalten, wurde jeweils pro Fehlerart und STT-System der Median für die WER und die CWER berechnet.

Tabelle 4: CEASR CWER Median und Mean pro System mit allen Korrekturen

Rang	System	WER Median	CWER Median
1	B5	0.076923	0.071429
2	B7	0.076923	0.071429
3	D1	0.083333	0.076923
4	D2	0.083333	0.076923
5	C1	0.214286	0.166667
6	C2	0.214286	0.166667
7	B3	0.187500	0.173913
8	B4	0.190476	0.179487
9	kaldi_librispeech	1.000000	0.206897
10	mozilla_deepspeech	0.250000	0.238095
11	kaldi_aspire	0.285714	0.272727
12	B8	0.304348	0.285714
13	sphinx4	0.500000	0.500000

Die obere Tabelle zeigt für jedes System die WER und CWER, welche vom STT-Backend berechnet wurde. Die Differenz, zwischen der WER und der CWER erscheint im Vergleich mit anderen Testdateien, welche während der Entwicklung verwendet wurden als sehr gering.

Die Korrektur wird vor allem durch das Entfernen von Grossbuchstaben verursacht. Das Entfernen der Kontraktionen, Sonderzeichen und das Ersetzen von Nummern hat einen extrem geringen Einfluss auf das Resultat.

7. Diskussion und Ausblick

Das Ziel dieser Arbeit war es, das Backend des STT-Explorers zu überarbeiten und zu erweitern, um eine vielseitigere Nutzung zu ermöglichen. Hierzu wurde ein neues Framework im Backend implementiert, das automatisch eine API-Dokumentation erstellt. Zusätzlich wurde die Datenstruktur überarbeitet, eine Standardberechnung der Word Error Rate (WER) definiert und eine performante Python Bibliothek für das Erstellen des Alignments integriert. Um die Flexibilität für die Benutzer zu erhöhen, wurde die Custom Word Error Rate (CWER) eingeführt, mit der aktuell fünf verschiedene Fehlerarten unabhängig voneinander ignoriert werden können. Ein weiterer Vorteil ist die direkte Verfügbarkeit aller Texte und berechneten Metriken innerhalb des Frontend.

Durch diese Erweiterung wurde die Performance der Anwendung stark verbessert. Dank der Anpassung der Datenstruktur ist die Applikation für eine allfällige zukünftige Datenbank vorbereitet.

Die Analyse der Custom Word Error Rate mithilfe der CEASR Korpus hat nicht zum erwünschten Erfolg geführt. Das Problem wurde jedoch wahrscheinlich durch eine falsche Handhabung der Daten verursacht. Bei mehreren Tests mit Benchmark Daten war die Erkennung der Fehler deutlich besser. Um dieses Problem einzugrenzen wäre eine präzisere Analyse der Daten erforderlich.

In der Alignment Map und im Detail des Frontend, werden die Korrekturen der Custom Word Error Rate nicht immer richtig hervorgehoben. Der Ansatz für das Matchen der Wörter auf das Alignment der Standard Word Error Rate führte nicht zum gewünschten Erfolg. Um dieses Problem zu beheben, müsste ein zuverlässigerer Algorithmus entwickelt werden.

In einer Zukünftigen Arbeit wäre eine Anbindung an eine Datenbank denkbar. Ausserdem sollte die Custom Word Error Rate weiter verbessert und zusätzliche Fehlererkennungen hinzugefügt werden.

Die vorliegende Bachelorarbeit hat gezeigt, dass die Erweiterung des STT-Explorers einen deutlichen Mehrwert für die Analyse und Bewertung von Speech-to-Text- Systemen bietet. Die Einführung der benutzerdefinierten Fehlerbewertung ermöglicht es, einzelne Systeme gezielt auf spezielle Fehler zu untersuchen.

Durch die Arbeit an diesem Projekt habe ich einen tiefen Einblick in die Herausforderungen des Natural Language Processing Bereiches erhalten. Vor allem die Erkennung der Textfehler habe ich mir zu Beginn weniger kompliziert vorgestellt.

Abschliessend hoffe ich, dass die Ergebnisse dieser Arbeit einen Beitrag zur weiteren Verbesserung von STT Systemen leisten kann oder zumindest als Grundlage für zukünftige Entwicklungen dienen kann.

8. Verzeichnisse

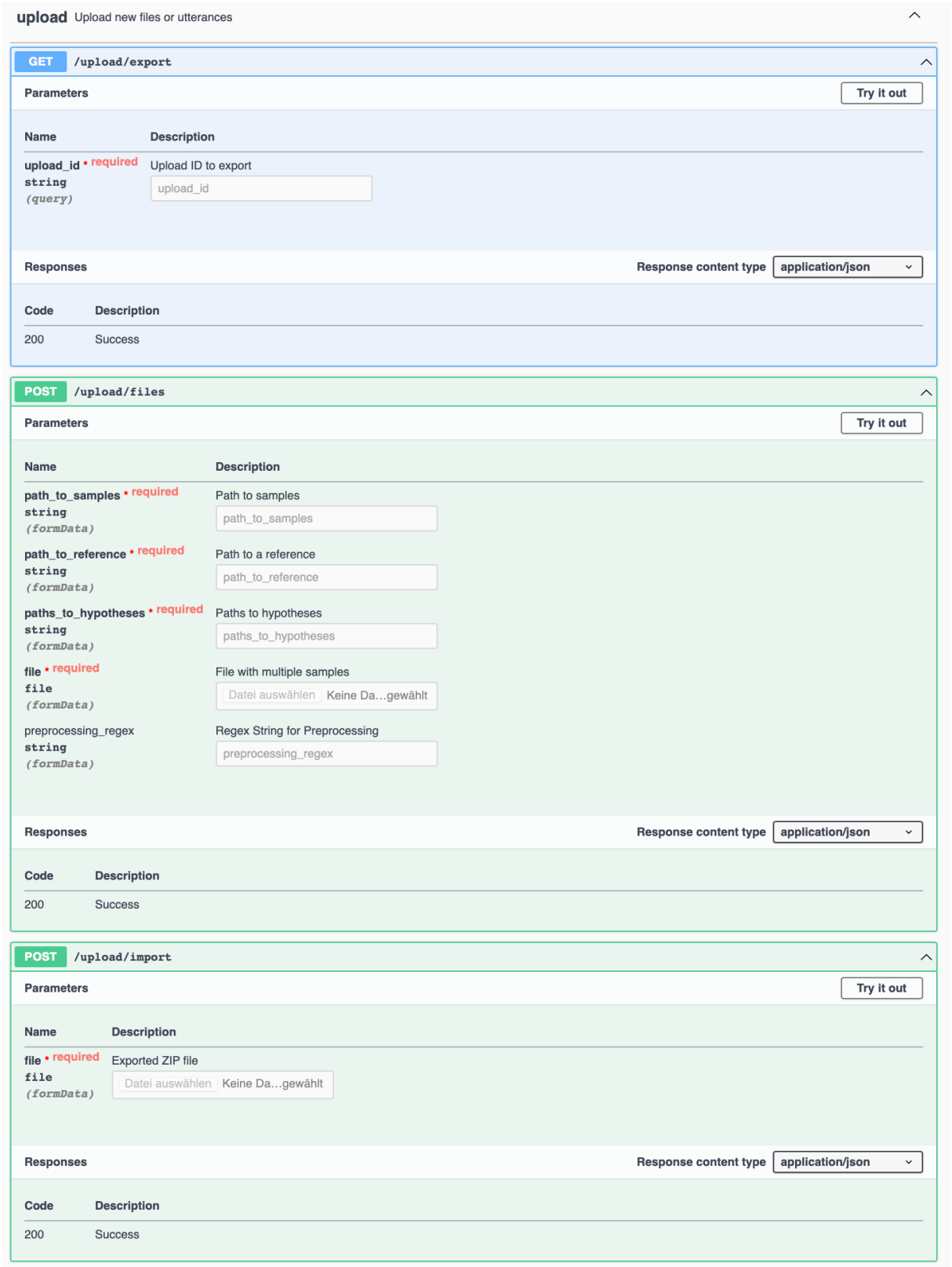
8.1. Literaturverzeichnis

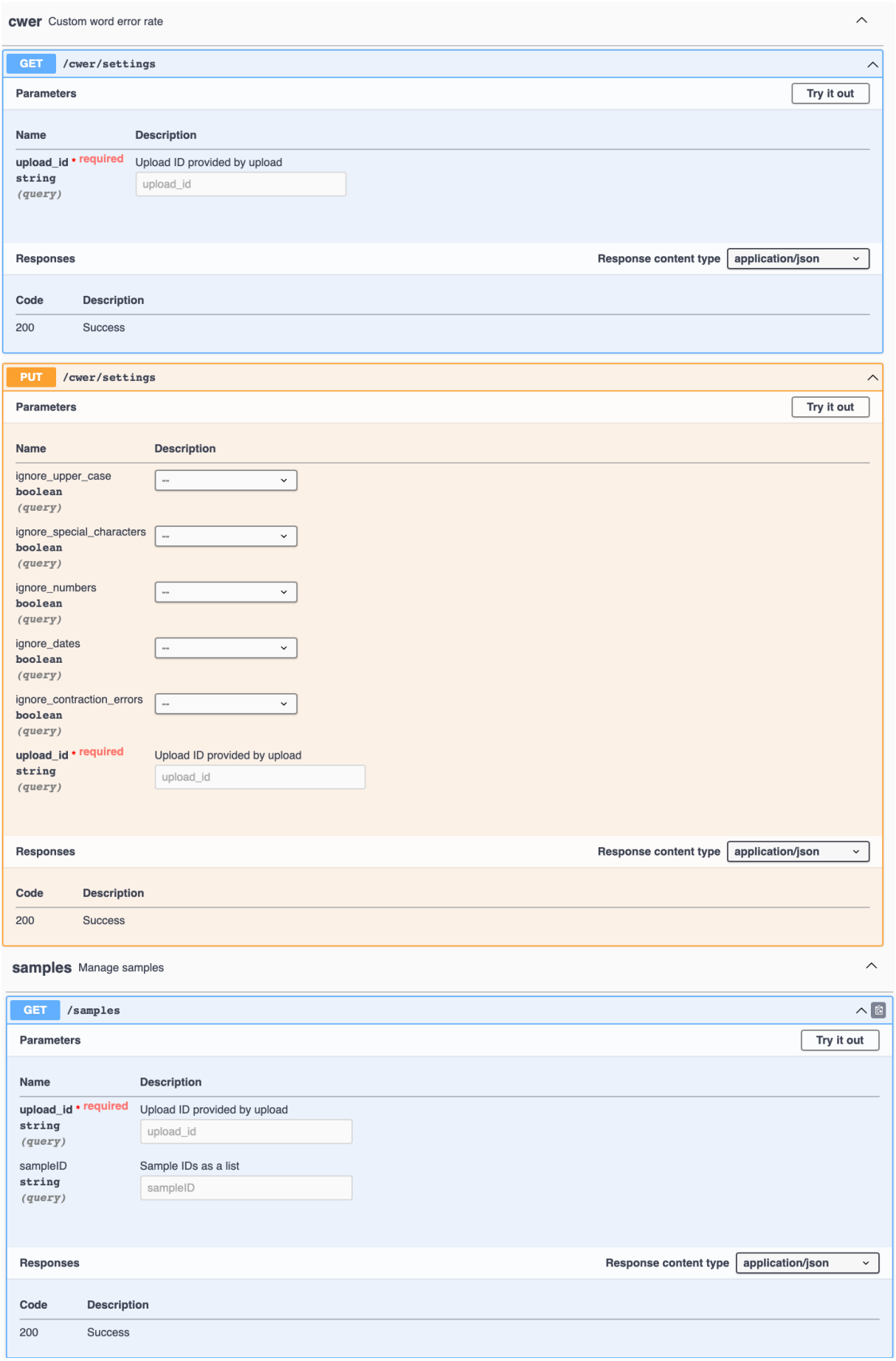
[1] «What is the Word Error Rate (WER) Score? — Klu». Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://klu.ai/glossary/wer-score
[2] J. Hughes, «The Problem with Word Error Rate (WER)», Speechmatics. Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://www.speechmatics.com/company/articles-and-news/the-problem-with-word-error-rate-wer
[3] «Usage - jiwer». Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://jitsi.github.io/jiwer/usage/
[4] «jitsi/jiwer: Evaluate your speech-to-text system with similarity measures such as word error rate (WER)». Zugegriffen: 22. Dezember 2023. [Online]. Verfügbar unter: https://github.com/jitsi/jiwer
[5] R. Armstrong, «werpy - Word Error Rate for Python». 8. Mai 2024. Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://github.com/analyticsinmotion/werpy
[6] «kahne/fastwer: A PyPI package for fast word/character error rate (WER/CER) calculation». Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://github.com/kahne/fastwer
[7] «huggingface/evaluate». Hugging Face, 6. Juni 2024. Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://github.com/huggingface/evaluate
[8] «Audio Recordings Download», CEASR. Zugegriffen: 7. Juni 2024. [Online]. Verfügbar unter: https://ceasr-corpus.github.io/audio-download/

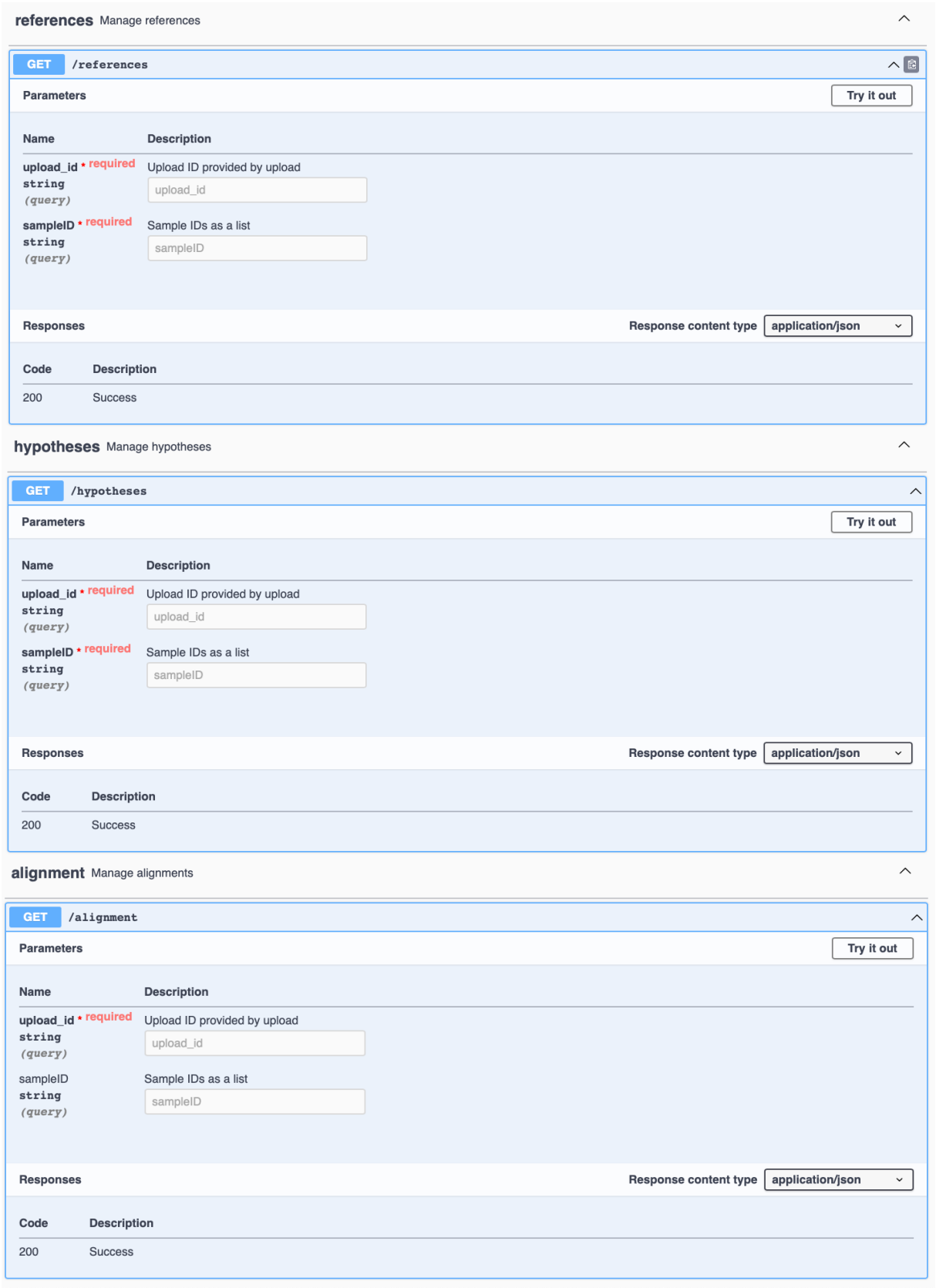
8.2. Abbildungsverzeichnis

8.3. Tabellenverzeichnis

9. Anhang


upload Upload new files or utterances
GET /upload/export
Parameters
Try it out
Name
Description
upload_id • required
string
(query)
Upload ID to export
upload_id
Responses
Response content type
application/json
Code
Description
200
Success
POST /upload/files
Parameters
Try it out
Name
Description
path_to_samples • required
string
(formData)
Path to samples
path_to_samples
path_to_reference • required
string
(formData)
Path to a reference
path_to_reference
paths_to_hypotheses • required
string
(formData)
Paths to hypotheses
paths_to_hypotheses
file • required
file
(formData)
File with multiple samples
Datei auswählen
Keine Da…gewählt
preprocessing_regex
string
(formData)
Regex String for Preprocessing
preprocessing_regex
Responses
Response content type
application/json
Code
Description
200
Success
POST /upload/import
Parameters
Try it out
Name
Description
file • required
file
(formData)
Exported ZIP file
Datei auswählen
Keine Da…gewählt
Responses
Response content type
application/json
Code
Description
200
Success


cwer Custom word error rate
GET /cwer/settings
Parameters
Try it out
Name
Description
upload_id * required
string
(query)
Upload ID provided by upload
upload_id
Responses
Response content type
application/json
Code
Description
200
Success
PUT /cwer/settings
Parameters
Try it out
Name
Description
ignore_upper_case
boolean
(query)
ignore_special_characters
boolean
(query)
ignore_numbers
boolean
(query)
ignore_dates
boolean
(query)
ignore_contraction_errors
boolean
(query)
upload_id * required
string
(query)
Upload ID provided by upload
upload_id
Responses
Response content type
application/json
Code
Description
200
Success
samples Manage samples
GET /samples
Parameters
Try it out
Name
Description
upload_id * required
string
(query)
Upload ID provided by upload
upload_id
sampleID
string
(query)
Sample IDs as a list
sampleID
Responses
Response content type
application/json
Code
Description
200
Success


references Manage references
GET /references
Parameters
Try it out
Name
Description
upload_id • required
string
(query)
Upload ID provided by upload
upload_id
sampleID • required
string
(query)
Sample IDs as a list
sampleID
Responses
Response content type
application/json
Code
Description
200
Success
hypotheses Manage hypotheses
GET /hypotheses
Parameters
Try it out
Name
Description
upload_id • required
string
(query)
Upload ID provided by upload
upload_id
sampleID • required
string
(query)
Sample IDs as a list
sampleID
Responses
Response content type
application/json
Code
Description
200
Success
alignment Manage alignments
GET /alignment
Parameters
Try it out
Name
Description
upload_id • required
string
(query)
Upload ID provided by upload
upload_id
sampleID
string
(query)
Sample IDs as a list
sampleID
Responses
Response content type
application/json
Code
Description
200
Success

10. Begriffe

Begriff	Abkürzung	Erklärung
Speech-to-Text	STT	Technologie, welche gesprochene Sprache in geschriebenen Text umwandelt.
Referenztext		Der korrekte Text, welcher als Grundlage für den Vergleich mit dem Hypothesentext verglichen wird.
Hypothesentext		Der von einem STT-System erzeugte Text, welcher mit dem Referenztext verglichen wird.
Word Error Rate	WER	Metrik zur Bewertung der Genauigkeit von Texten, in welcher die Anzahl der Fehler in einem Hypothesentext im Vergleich zum Referenztext misst.
Custom Word Error Rate	CWER	Eine individuell anpassbare Version der WER.
Alignment		Die Ausrichtung des Referenztextes und des Hypothesentextes, so dass möglichst viele Wörter übereinstimmen, ohne dessen Reihenfolge zu ändern.
Utterance		Äusserung (Referenz- und Hypothesenkombination)
Natural Language Processing	NLP	Forschungsbereich, welcher sich mit der Analyse und Verarbeitung von natürlicher Sprache in Form von Text befasst, um dessen Bedeutung zu verstehen.
Representational State Transfer	REST	Weitverbreitete Art einer API, welche vor allem zur Kommunikation zwischen Frontend und Backend Eingesetzt wird.
Swagger		Ein Open-Source Werkzeug um HTTP-Webservices zu Entwerfen und nach dem Open API Standard zu Dokumentieren.
Regex		Regulärer Ausdruck, welcher für das Erkennen von bestimmten Textstellen verwendet werden kann.