

# **Erkennung der auf Dialekt gesprochenen Befehle**

## Ein Vergleich zwischen bestehenden Spracherkennungsmodellen

Bachelorarbeit  
zur Erlangung des akademischen Grades  
**Bachelor of Science in Engineering**

Fachhochschule Vorarlberg  
Elektrotechnik Dual

Betreut von  
Wolfgang Auer

Vorgelegt von  
Bogdan Khamelyuk

Dornbirn, 08. Jun. 2022

## Kurzreferat

### Erkennung der auf Dialekt gesprochenen Befehle

Diese Arbeit erforscht die Möglichkeit der Erkennung des vorarlbergeren Dialektes mithilfe von den modernen Spracherkennungsmodellen. Als Modelle wurden von Facebook entwickelte einsprachiges Wav2Vec und sein mehrsprachiges analog XLSR genommen. Beide Modelle wurden mit den gesammelten Sprachbeispielen für den Dialekt adaptiert (eng. „finetuned“). Es wurde auch das auf deutsche Sprache zusätzlich vortrainierte XLSR (XLSRDe) verwendet, das später ebenso auf den Dialekt finetuned wurde. Die Qualität der Ergebnisse wurde mit der word-error-rate (WER) gemessen.

Sprachbeispiele im Dialekt wurden von den verschiedenen Quellen manuell gesammelt. Die gesamte Dauer von Trainingsdaten ist ca. 34 Minuten. Die Komplexität des Audiodatasets wurde durch die Anzahl der Wörter eingeschätzt, bzw. durch die Größe des allgemeinen Vokabulars des Datasets. Das Training hat sich stufenweise erfolgt. Es wurde versucht die Abhängigkeiten in der Erkennungsqualität zwischen verschiedenen Parametern festzustellen, indem unterschiedliche Datasetkombinationen erschafft wurden.

Die Ergebnisse dieser Arbeit zeigen, dass das XLSR die niedrigste durchschnitts-WER und die kleinste Differenz zwischen dem maximalen und minimalen WER hat. Darüber hinaus ist ersichtlich, dass die stimmebezogene Information keine Auswirkung auf die Erkennung hat, wenn die Daten komplett beschriftet sind. Die Arbeit soll dem Leser einen allgemeinen Überblick über die Erkennungsperformanz der drei Modelle geben, um dann ein passendes Modell für die weitere Entwicklung wählen zu können.

Schlagwörter:

Automatische Spracherkennung, ressourcenarme Sprachen, einsprachiges Modell, mehrsprachiges Modell, Fine-Tuning.

## **Abstract**

Recognition of commands spoken in dialect

This paper explores the possibility of recognising the Vorarlberg dialect of German using modern speech recognition models. The models used were the monolingual Wav2Vec developed by Facebook and its multilingual analogue XLSR. Both models were finetuned with the collected speech samples of the dialect. The XLSR additionally pre-trained for German (XLSRDe) was also used, which was later finetuned to the dialect as well. The quality of the results was measured with the word error rate (WER).

Speech samples in the dialect were collected manually from the different sources. The total duration of training data is approximately 34 minutes. The complexity of the audio dataset was estimated by the number of words, or the size of the general vocabulary of the dataset. The training was done in stages. An attempt was made to determine the dependencies in recognition quality between different parameters by creating different dataset combinations.

The results of this work show that the XLSR has the lowest average WER and the smallest difference between the maximum and minimum WER. Furthermore, it is evident that voice-related information has no effect on recognition when the data is fully labelled. The paper aims to give the reader a general overview of the recognition performance of the three models in order to be able to choose a suitable model for further development.

**Keywords:**

Automatic speech recognition, low-resource languages, monolingual model, multilingual model, fine-tuning.

# Inhaltsverzeichnis

|  |           |
|--|-----------|
| <b>Abkürzungsverzeichnis.....</b>                                | <b>5</b>  |
| <b>Formelverzeichnis .....</b>                                   | <b>6</b>  |
| <b>Abbildungsverzeichnis.....</b>                                | <b>7</b>  |
| <b>1 Einleitung .....</b>  | <b>8</b>  |
| <b>2 Theorie .....</b>   | <b>10</b> |
| 2.1 <i>HMM</i> .....   | 10        |
| 2.2 <i>RNN</i> .....   | 10        |
| 2.3 <i>Transformers</i> .....                                    | 12        |
| 2.3.1 Positional Encodings .....                                 | 13        |
| 2.3.2 Encoder .....  | 13        |
| 2.3.3 Decoder.....   | 15        |
| 2.4 <i>Dropout</i> .....   | 16        |
| 2.5 <i>Lernmethoden</i> .....                                    | 16        |
| <b>3 Modelle .....</b>   | <b>18</b> |
| 3.1 <i>Wav2Vec</i> .....   | 18        |
| 3.2 <i>XLSR</i> .....  | 19        |
| 3.3 <i>XLSRDe</i> .....  | 19        |
| <b>4 Umsetzung .....</b>   | <b>20</b> |
| 4.1 <i>Ziele</i> .....   | 20        |
| 4.2 <i>Datensammlung</i> .....                                   | 20        |
| 4.2.1 Client-Side .....  | 20        |
| 4.2.2 Server-Side .....  | 20        |
| 4.3 <i>Datenvorbereitung</i> .....                               | 21        |
| 4.3.1 Beschriftung .....   | 21        |
| 4.3.2 Datasets Erstellen .....                                   | 21        |
| 4.3.3 Audio pro Wort .....                                       | 22        |
| 4.4 <i>Fine-Tuning</i> .....                                     | 22        |
| <b>5 Ergebnisse .....</b>  | <b>24</b> |
| <b>6 Zusammenfassung und Ausblick .....</b>                      | <b>26</b> |
| <b>Literaturverzeichnis .....</b>                                | <b>27</b> |
| <b>Anhang.....</b>   | <b>28</b> |
| <i>Fragebogen aus der Webseite</i> .....                         | 28        |
| <i>Aufnahmeteil der Webseite</i> .....                           | 28        |
| <i>Frequenz von Modell-WER in einem bestimmten Bereich</i> ..... | 28        |
| <b>Eidesstattliche Erklärung .....</b>                           | <b>29</b> |

## **Abkürzungsverzeichnis**

ANN – Artificial Neural Network  
ASR – Automatic Speech Recognition  
BPTT - Backpropagation through time  
CNN – Convolutional Neural Network  
DNN – Deep Neural Network  
CV – Computer Vision  
DL – Deep Learning  
HMM – Hidden Markov Model  
KI – künstliche Intelligenz  
ML – Machine Learning  
NLP – Natural Language Processing  
NN – Neural Network  
PER – Phoneme Error Rate  
RNN – Recurrent Neural Network  
WER – Word Error Rate

## **Formelverzeichnis**

|  |    |
|--|----|
| Formel 1: Berechnung der Wahrscheinlichkeit des Wortes in HMM .....                            | 10 |
| Formel 2: Berechnung der Wahrscheinlichkeit des Wortes in HMM mit der bayessischen Regel ..... | 10 |
| Formel 3: Berechnung eines internen Zustandes von RNN zum Zeitpunkt t.....                     | 11 |
| Formel 4: Ausgabe der versteckten Schichten in RNN.....  | 11 |
| Formel 5: Gradient eines RNN-Netzwerkes mit der Kettenregel für eine Schicht .....             | 12 |
| Formel 6: Positions-kodierung .....  | 13 |
| Formel 7: neue Einbettungen.....   | 13 |
| Formel 8: Berechnung von Key-, Value- und Query-Werten .....                                   | 14 |
| Formel 9: Berechnung von Attention-Score .....   | 14 |
| Formel 10: Berechnung von Multi-Head Attention .....   | 14 |
| Formel 11: Ausgabe aus der Feed-Forward-Schicht .....  | 15 |

## **Abbildungsverzeichnis**

|   |    |
|---|----|
| Abbildung 1: Architektur eines HMM-Dekoders (Quelle: Autor).....  | 10 |
| Abbildung 2: Archtitektur eines RNN-Netzwerkes (Quelle: Autor) .....  | 11 |
| Abbildung 3: Verlauf der sigmoid-Funktion und ihrer Ableitung (Quelle: towardsdatascience.com) ...          | 12 |
| Abbildung 4: Transformer-Architektur (Quelle: Natural Language Processing With Transformers. O'Reilly)..... | 12 |
| Abbildung 5: Encoder-Block (Quelle: Autor) .....  | 15 |
| Abbildung 6: Decoder-Block (Quelle: Autor) .....  | 16 |
| Abbildung 7: Netzwerk ohne Dropout (links) und mit Dropout (rechts) (Quelle: Srivastava et al., 2014) ..... | 16 |
| Abbildung 8: Wav2Vec-Architektur (Quelle: Baevski et. al 2020) .....  | 18 |
| Abbildung 9: XLSR-Architektur (Quelle: Baevski et al., 2020) .....  | 19 |
| Abbildung 10: Screenshot aus Audacity (Quelle: Autor).....  | 21 |
| Abbildung 11: APW nach Dataset (Quelle: Autor) .....  | 22 |
| Abbildung 12: Das Herunterladen von wav2vec in GoogleColab (Quelle: Autor).....                             | 23 |
| Abbildung 13: Berechnung von WER (Quelle: Autor) .....  | 23 |
| Abbildung 14: WER nach Dataset (Quelle: Autor) .....  | 24 |
| Abbildung 15:Anzahl der Audiodateien und die Größe des Vokabulars (Quelle: Autor).....                      | 24 |
| Abbildung 16: Verteilung der WER-Werte nach Dataset (Quelle: Autor).....                                    | 25 |

# 1 Einleitung

Die Gründung der künstlichen Intelligenz (KI) als ein separates akademisches Fachgebiet geht zurück in das Jahr 1956, als bei einem sechswöchigen Workshop mit dem Titel *Dartmouth Summer Research Project on Artificial Intelligence* am Dartmouth College der Begriff „Artificial Intelligence“ erstmals erwähnt wurde (1). Zu den Aufgaben von KI gehören: Wissensrepräsentation, Lernen, Merkmalswahrnehmung, Natural Language Processing, Computer Vision usw. Diese Ziele werden hauptsächlich mithilfe von zwei KI-Technologien erreicht: Machine Learning (ML) und Deep Learning (DL).

Die Aufgaben die mithilfe von ML gelöst werden können, brauchen weniger Rechenaufwand im Vergleich zu DL. So zählen sich beispielsweise „Regression“, „Klassifizierung“ und „Auffälligkeitserkennung“ zu den typischen ML-Problemen. DL bedeutet implizit eine Implementierung von mehreren Schichten bestehend aus den Knoten, die einkommende Information mit den Parametern („weights“) und Vorurteil („bias“) auswerten. Das kann mit einem menschlichen Neuron verglichen werden. Der Zusammenhang von mehreren Neuronen nennt sich „Neuronales Netzwerk, was oft auch im Bezug zu DL mit dem Präfix „Deep“ (DNN) verwendet wird. DL-Aufgaben brauchen mehr Hardware-Leistung, da die Lösungsart, bzw. Entscheidungsart maximal menschnah sein muss. Aufgaben im NLP-Bereich, wie z.B. Q&A Bots, Textgenerierung, ASR zählen zu den klassischen Beispielen der DL-Anwendung.

In dieser Arbeit wird das ASR im Zusammenhang mit der Vorarlberger Mundart erforscht. Da die Vorarlberger Mundart im Internet im Form von Audio- und Textdateien fast kaum repräsentiert wird, kann sie als ressourcenarme Sprache bezeichnet werden. Moderne ASR-Systeme, wie Wav2Vec (1) oder XLSR (2), ermöglichen eine Spracherkennung dieser Art von Sprachen, indem das unüberwachte Lernen (eng. „unsupervised Learning“) im Paar mit dem repräsentativen Lernen (eng. „representation Learning“) verwendet wird. Im Vergleich zum Ansatz mit den log-mel Filterbanks, reduziert das Wav2Vec mit nur 8 Stunden Trainingsaudiodateien die WER bis 36% (1). XLSR ist ein mehrsprachiger Analog von Wav2Vec. Die mehrsprachige Interferenz im Vergleich zum einsprachigen Trainingsansatz lässt XLSR am „Common Voice“-Dataset eine PER-Reduktion auf 49% erreichen. Die Studie „Using Large Self-Supervised Models for Low-Resource Speech Recognition“ von Krishna et. al (3) hat in Indien einen Vergleich zwischen XLSR und Wav2Vec mit ressourcenarmen Sprachen gemacht. Die Ergebnisse bestätigen auch den Fakt, dass das mehrsprachige XLSR bessere WER-Resultate gegenüber dem einsprachigen Wav2Vec hat. So schreiben die Autoren in der Arbeit:

*“It can be seen that the multilingual pre-trained model outperforms all the other pre-trained models across all languages. This is due to the fact that the XLSR model has rich cross-lingual representations. ... We can see that the multilingual pre-trained model XLSR+CTC outperforms state-of-the-art supervised system Transformer-Multitask for Gujarati and Tamil.”*

Das wissenschaftliche Interesse dieser Arbeit wird durch den Vergleich zwischen nicht-vortrainiertem XLSR, auf Hochdeutsch vortrainiertem XLSR (XLSRDe) und nicht-vortrainiertem Wav2Vec an 34 Minuten längeren Dataset ausgedrückt. Das allgemeine Dataset-21DL („21“ für „21 Befehle“, „D“ für „Dornbirn“ und „L“ für „Lustenau“) wurde manuell mithilfe der selbsterstellten Webseite und Aufnahmen der Dornbirner Mundart<sup>1</sup> und Lustenauer Redewendungen<sup>2</sup> erstellt. Die Audiodateien, die durch die Webseite gesammelt wurden, werden als Dataset-21 bezeichnet. Das Dataset-21 besteht aus 567 Audiodateien, von denen jedes Audio nicht länger als 2 Sekunden dauert. In diesem Dataset befinden sich die Aufnahmen von 21 auf

<sup>1</sup> <https://mundartlexikon.dornbirn.at/>

<sup>2</sup> <https://www.lustenauer-mundart.at/redewendungen/>

Hochdeutsch geschriebenen Befehle, die in der Mundart des Lesers ausgesprochen wurden. Für das Dataset-Lustenau wurden 218 Redewendungen der Lustenauer Mundart aufgenommen. Die durchschnittliche Länge der Audios lag bei 3 Sekunden und hat zwischen 2 und 6 Sekunden fluktuiert. Das Dornbirner Dataset besteht aus 351 Audiodateien, wo einzelne Substantive und Verben (auch als Partizip II) ausgesprochen wurden. Die gesammelten Dialektbeispiele wurden auf Hochdeutsch beschriftet, da die Dialektschreibweise von Ort zu Ort unterschiedlich ist.

Es wurden unterschiedliche Kombination aus diesen Datasets erstellt, um die Korrelation zwischen der Größe des Vokabulars, der Anzahl der Trainingsaudiodateien und dem resultierenden WER zu erkennen. So ist ersichtlich, dass bei den komplexen Datasets, bei denen die Größe des Vokabulars bei über 700 Wörtern (Dataset-21DL und Dataset-DL) liegt, XLSRDe die bessere WERs als XLSR und Wav2Vec erreicht. Bei den Datasets, bei denen die Audio-pro-Wort Rate größer als 2 ist, erkennt die Wörter das monolinguistische Wav2Vec präziser als die anderen. Bei allen Modellen ist eine indirekte Proportionalität zwischen der Audio-pro-Wort Rate und der WER zu erkennen: umso umfangreicher die Trainingsdaten für ein Wort des Vokabulars vorgelegt wurden, desto besser ist die WER des Modells.

Außer auf wissenschaftlicher, beansprucht die Arbeit auch auf unternehmerischer Relevanz. Laut der Statistik (4) werden in den nächsten drei bis fünf Jahren weltweit 56% der Unternehmen im Banking-, Elektronik-, Bildungs- und Versicherungsbereichen ihre Nutzung der Sprachtechnologie erhöhen. In Japan beispielsweise werden die Umsatzwerte des Spracherkennungsmarktes von €85.158 Mio. im Jahr 2020 bis auf €182.286 Mio. im Jahr 2025 steigen (5). 32% der Befragten 90 Unternehmen aus der DACH-Region haben zum Zeitpunkt der Umfrage bereits KI-Technologien im Einsatz gehabt, während weitere 36% planten, in den nächsten 3 Jahren KI-Lösungen zu nutzen anzufangen (6).

Bevor es zur Umsetzung der gesetzten Ziele kommt, wird zuerst die Entwicklungsgeschichte des heutigen Standpunktes erwähnt. So werden die Merkmale von HMM, RNNs, CNNs und Transformers mit ihrer eigenen Architektur im Theorieteil beschrieben. Nachdem der Leser einen Eindruck bekommen hat, wie das Wav2Vec und sein analog XLSR funktionieren, werden die Modelle selbst kurz beschrieben. Als nächstes werden die Ausgangslage, Datensammlung und die gelieferten Ergebnisse beschrieben. Im Anhang werden die Codestücke angehängt, die für die Mikrofonaufnahmen, HTTPS-Übertragung auf den Server und die Kalkulation der Größe des Dataset-Vokabulars verwendet wurden.

## 2 Theorie

### 2.1 HMM

Das auf den Markowketten basierende Hidden Markov Model ist ein stochastisches Modell, wessen Ziel es ist, unbekannte Parameter durch beobachtete zu enträtseln. Die Hauptanwendungen für HMMs liegen in den Bereichen Spracherkennung, Schreiben, Bewegungserkennung und Bioinformatik.

Die Funktionsweise von HMM in der Spracherkennung basiert auf der Anpassung der von einem Decoder generierten Wortvektoren mit den Vektoren der Audioeingabe. Mathematisch wird dafür der Operator  $\arg \max$  verwendet:

$$\hat{\omega} = \underbrace{\arg \max}_{\omega} \{P(\omega|Y)\} \quad (1)$$

Formel 1: Berechnung der Wahrscheinlichkeit des Wortes in HMM

Da die Wahrscheinlichkeit des Wortes einer Audioeingabe schwierig zu modellieren sein kann, wird die bayessische Regel verwendet, um eine äquivalente Aussage zu bekommen:

$$\hat{\omega} = \underbrace{\arg \max}_{\omega} \{p(Y|\omega)P(\omega)\} \quad (2)$$

Formel 2: Berechnung der Wahrscheinlichkeit des Wortes in HMM mit der bayessischen Regel

Auf die Berechnung der Wahrscheinlichkeit des möglichen Wortes werden auch die voreingestellten akustischen Modelle, Aussprachevokabular und die Sprachmodelle, die spezifisch mit diesem Decoder verbunden sind berücksichtigt. Schlussendlich schaut das HMM-Spracherkennungsmodell wie folgt aus:

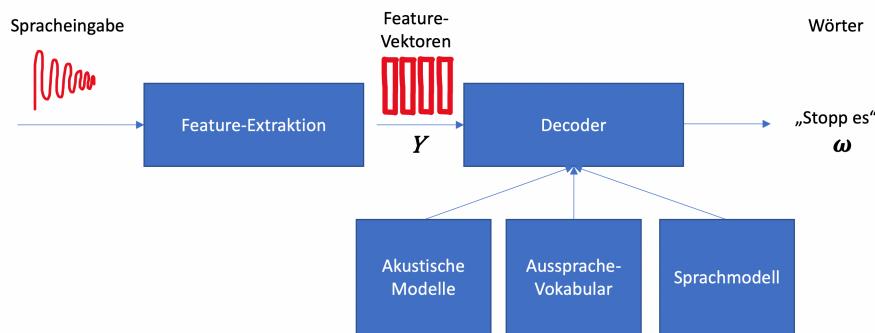


Abbildung 1: Architektur eines HMM-Dekoders (Quelle: Autor)

### 2.2 RNN

Wie aus Formel 2 ersichtlich ist, hängt das nächststehende Wort ausschließlich von dem vorherigen Wort ab. Dies führt dazu, dass die Eingabevektoren in HMM ohne Kontext dekodiert werden. Eine von den Lösungen, die der Ausgabe eine Art von Kontext geben kann, ist RNN.

RNNs eignen sich effektiv für den Umgang mit den sequenziellen Daten, da diese Netze über einen internen Speicher verfügen. Der Speicher beeinflusst das Resultat der Eingabeauswertung und wird aufrechterhalten. Bei der Verarbeitung einer neuen Sequenz wird der interne Speicher neu gesetzt.

Die RNNs werden auch bei der Spracherkennung verwendet. Nachdem die Merkmale der Audioeingabe durch die Mel-Frequenz-Cepstrum-Koeffizienten extrahiert worden sind, werden sie in Form eines Vektors als eine Eingangseinheit  $x$  zum Zeitpunkt  $t$  zum Netzwerk angegeben. Die Multiplikation mit dem Input-weight  $U$  führt die Eingangseinheit zu einer versteckten Schicht des Netzwerkes, wo der Zustand  $h$  des internen Speichers gespeichert wird. Parallel dazu wird der alte Zustand  $h^{(t-1)}$  mit einem entsprechenden Hidden-Weight  $W$  multipliziert. Die Ausgabe der Berechnung wird durch die sigmoid-Funktion begrenzt.

$$h^{(t)} = \sigma(Ux^{(t)} + Wh^{(t-1)}) \quad (3)$$

Formel 3: Berechnung eines internen Zustandes von RNN zum Zeitpunkt t

Nach einer Anzahl von Wiederholungen, bzw. nach dem die ganze Eingabesequenz ausgewertet wurde, wird der Zustand  $h$  mit dem output-weight  $V$  multipliziert und ebenso durch die softmax-Funktion bearbeitet. Diese Funktion findet einen Maximalwert der Eingabesequenz heraus.

$$Y^{(t)} = \text{softmax}(Vh^{(t)}) \quad (4)$$

Formel 4: Ausgabe der versteckten Schichten in RNN

Das Bild unten zeigt eine einfache Architektur eines RNN-Netzwerks. „Forward Pass“ bedeutet eine Vorwärtsübergabe der Parameter, bis die Verlustfunktion einen Vergleich mit dem Zielwert macht.

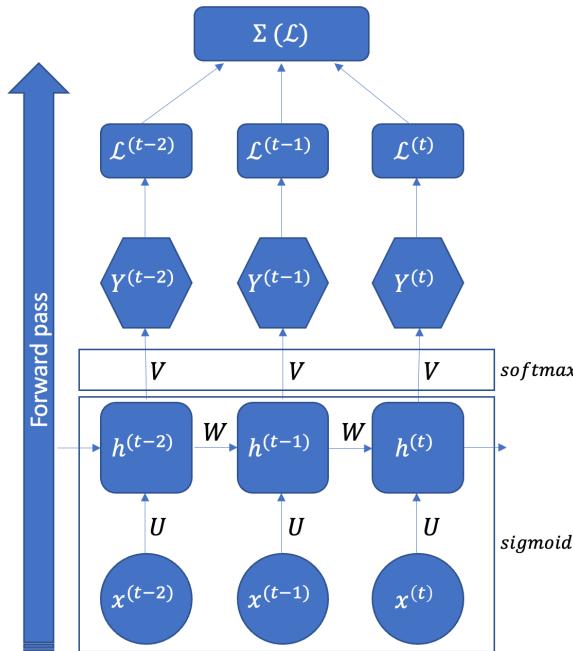


Abbildung 2: Architektur eines RNN-Netzwerkes (Quelle: Autor)

Der Lernvorgang von RNN basiert auf dem BPTT-Algorithmus, in welchem die weights durch das Gradientverfahren aktualisiert werden. Mit diesem Verfahren wird die Änderung des Verlustes in Bezug auf die Änderung von jedem einzelnen Parameter gemessen. Dafür wird die Kettenregel verwendet.

$$\frac{\partial \mathcal{L}}{\partial U} = \frac{\partial \mathcal{L}}{\partial Y} \times \frac{\partial Y}{\partial V} \times \frac{\partial V}{\partial h} \times \frac{\partial h}{\partial U} \quad (5)$$

Formel 5: Gradient eines RNN-Netzwerkes mit der Kettenregel für eine Schicht

Die RNNs sind nur mit kürzeren Datensequenzen effektiv. Wenn das Netzwerk aus mehr als 10 Schichten besteht, bzw. wenn es mehrere Sequenzelementen gibt, wird der resultierende Multiplikationsprodukt sehr niedrig sein. Dies führt dazu, dass der semantische Zusammenhang zwischen den Wörtern in längeren Sätzen verloren gehen kann. Darüber hinaus sind die Ableitungswerte der sigmoid-Funktion zwischen 0 und 0,25 begrenzt. Die resultierende Multiplikation von mehreren Ableitungen (Formel 5), deren Werte klein sind, führt zu einem entsprechend kleinen Gradient. Dieses Problem nennt sich „Vanishing Gradient“. Die Lösung dieses Problems, sowie die strukturellen Eigenschaften dieser Lösung werden im nächsten Kapitel beschrieben.

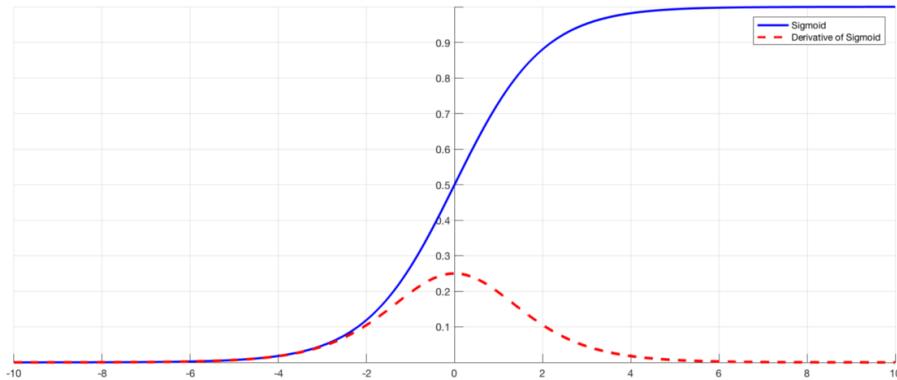


Abbildung 3: Verlauf der sigmoid-Funktion und ihrer Ableitung (Quelle: [towardsdatascience.com](https://towardsdatascience.com/the-sigmoid-function-and-its-derivative-101-10f3a2a2a2))

## 2.3 Transformers

Erstmals präsentiert von Vaswani et al. (7) sind Transformers ein Durchbruch im Bereich von NLP geworden. Der implementierte Attention-Mechanismus erlaubt es, die ganze Eingabesequenz als eine zu nehmen. Attention erlaubt es auch, auf bestimmte Sequenzelemente einen semantischen Akzent zu machen, indem die weights von diesen Elementen entsprechend erhöht oder verkleinert werden.

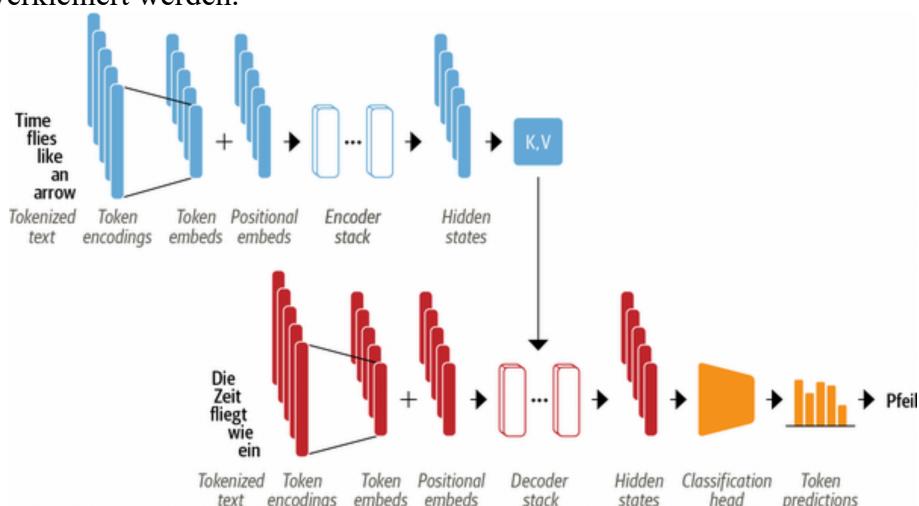


Abbildung 4: Transformer-Architektur (Quelle: Natural Language Processing With Transformers. O'Reilly)

### 2.3.1 Positional Encodings

Aus dem Grund, dass die Eingabesequenz als eine ganze eingelesen wird, fällt eine implizite Aufzählung von Sequenzelementen aus. Um das Problem zu vermeiden, wird eine auf die Position bezogene Enkodierung verwendet, die von den Transformers-Autoren vorgeschlagen wurde.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \quad (6)$$

Formel 6: Positonskodierung

, wo  $pos$  – ist die Position in der Sequenz,  $i$  – ist die Dimension dieses Vektors,  $d_{model}$  ist die Dimension des Modells.

Die Positions-kodierungen haben die gleiche Dimension  $d_{model}$  wie die Einbettungen, sodass die beiden zusammen als Matrizen addiert werden können. Die Wellenlängen bilden eine geometrische Progression von  $2\pi$  bis  $10000 * 2\pi$ .

### 2.3.2 Encoder

Encoder ist eine von den beiden Hauptschichten der Transformer-Architektur, der die tokenisierte Eingabesequenz in eine Sequenz von Einbettungsvektoren umwandelt. Diese Vektoren werden auch „hidden state“ oder „context“ genannt. Außer einer inhaltspezifischen, ist auch eine positionsspezifische Information in einem Einbettungsvektor enthalten.

Sobald Encoder die Einbettungen erhält, werden diese durch die Multi-Head self-attention Schicht und eine vollständig vernetzte feed-forward Schicht durchgeführt. Bei einem mehrschichtigen Encoder (siehe Abbildung 4) werden von den zugeführten Einbettungen Repräsentationen erstellt, die eine kodierte kontextuelle Information enthalten.

#### 2.3.2.1 Self-Attention

Die Hauptidee von Self-Attention besteht darin, anstatt eine feste Einbettung für jedes Token zu verwenden, eine ganze Sequenz aufzunehmen, um einen gewichteten Durchschnitt der einzelnen Einbettungen zu berechnen. Das Präfix „self“ bedeutet, dass die weights für alle Einbettungen berechnet werden. Die Aussage kann folgenderweise demonstriert werden:

$$x'_i = \sum_{j=1}^n w_{ji} x_j \quad (7)$$

Formel 7: neue Einbettungen

Die Koeffizienten  $w_{ji}$  sind die „attention weights“, die in ihrer Summe gleich 1 sind. Der Vorteil von einem gewichteten Durchschnitt besteht darin, dass das Wort im Bezug auf Kontext dekodiert wird. So kann zum Beispiel das Wort „fliegen“ ohne Berücksichtigung des ganzen Satzes bezüglich der Insekten oder als eine Bewegungsbeschreibung interpretiert werden. Wenn es jedoch eine kombinierte Repräsentation im Zusammenhang mit den weights von den anderen Wörtern gäbe, dann würde das Modell den kompletten Satz richtig interpretieren, bzw. dekodieren.

Ein mögliche Variante, um die self-attention Schicht zu implementieren, besteht aus 4 Schritten:

1. Jede Token-Einbettung wird auf drei Vektoren projiziert: Query, Key und Value;

2. Berechnung von Attention-Scores. Hier wird die Ähnlichkeit von query und key mithilfe des Skalarprodukts dieser Parameter bestimmt. Die Ausgabe dieses Schritts ist das „Attention Score“, dessen Wert die Ähnlichkeit zwischen query und key darstellt: je größer der Wert, desto größer ist der Zusammenhang von query und key;
3. Berechnung von Attention-Weights. Da die Skalarprodukte willkürlich große Zahlen ergeben können, müssen sie mit einem Skalierungsfaktor multipliziert und durch eine softmax-Funktion begrenzt werden;
4. Token-Einbettungen aktualisieren. Sobald die attention-weights berechnet wurden, werden sie mit dem Vektor  $v$  multipliziert, um eine Repräsentation der Einbettung  $x'_i$  zu bekommen:  $x'_i = \sum_{j=1}^n w_{ji} v_j$

In der Praxis werden mehrere Self-Attention Schichten parallel aufgebaut (Multi-head Self-attention). Somit wird eine größere Anzahl von semantischen Zusammenhängen aufgenommen, was eine präzisere Erkennung zulässt. Die allgemeine mathematische Repräsentation der Attention-Berechnung kann wie folgt dargestellt werden:

$$Q = XW_Q \quad K = XW_K \quad V = XW_V \quad (8)$$

Formel 8: Berechnung von Key-, Value- und Query-Werten

$$\text{Attention}(Q, K, W) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (9)$$

Formel 9: Berechnung von Attention-Score

$$\text{MultiHead}(X, X, X) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_o, \quad (10)$$

mit  $\text{head}_i = \text{Attention}(XW_Q, XW_K, XW_V)$

Formel 10: Berechnung von Multi-Head Attention

wo  $W_q \in \mathbb{R}^{d_{model} \times d_k}$ ,  $W_K \in \mathbb{R}^{d_{model} \times d_k}$ ,  $W_V \in \mathbb{R}^{d_{model} \times d_v}$  die Projektionsmatrizen für die Query, Key und Value sind;  $X \in \mathbb{R}^{l \times d_{model}}$  ist die Eingabematrice für die Sequenz mit der Länge  $l$ ;  $h$  ist die Anzahl von Attention-Heads und  $W_o \in \mathbb{R}^{hd_v \times d_{model}}$  ist die lineare Projektionsmatrice.

### 2.3.2.2 Feed-Forward

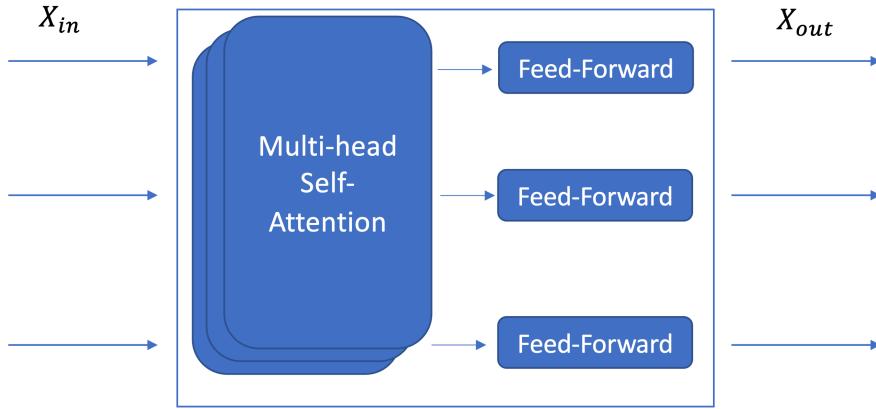


Abbildung 5: Encoder-Block (Quelle: Autor)

Nach dem die repräsentierenden Einbettungen für jedes Wort (hinsichtlich auch der anderen) anhand der self-attention Schicht erstellt wurden, werden diese durch die feed-forward Unterschicht geführt. Das Ziel dieser Schicht ist es, die Einbettung für die nächststehenden Decoder-Schicht vorzubereiten und separat weiterzuleiten. Das bedeutet auch, dass die Vektordimensionen nach der Attention-Schicht zu den Decoder-Dimensionen eingepasst werden.

$$FFN(x) = \text{ReLU}(XW_1 + b_1)W_2 + b_2 \quad (11)$$

Formel 11: Ausgabe aus der Feed-Forward-Schicht

wo  $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ ,  $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ ,  $b_1 \in \mathbb{R}^{1 \times d_{ff}}$ ,  $b_2 \in \mathbb{R}^{1 \times d_{model}}$  und  $d_{ff}$  ist die Dimension der ersten Schicht.

### 2.3.3 Decoder

Obwohl der Decoder im Allgemeinen dem Encoder sehr ähnlich ist, besteht ein Unterschied in der Anzahl der Attention-Unterschichten. So hat der Decoder außer einer Multi-Head-Attention-Schicht auch die Encoder-Decoder-Attention-Schicht. Diese Schicht betrachtet die aus dem Encoder ausgehende Keys- und Values-Vektoren durch die eigene Multi-Head-Attention-Schicht. Auf diese Weise lernt die Encoder-Decoder-Attention-Schicht, wie die Tokens aus zwei verschiedenen Sequenzen, z.B. Audiorepräsentationen und die Sprachrepräsentationen, in Beziehung gesetzt werden müssen. Der Decoder hat Zugang zu den Keys und Values des Encoders in jedem Block.

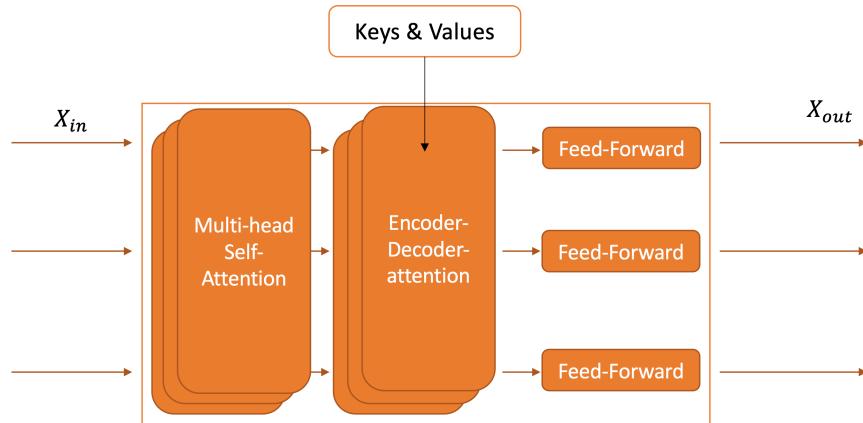


Abbildung 6: Decoder-Block (Quelle: Autor)

## 2.4 Dropout

Wenn große neuronale Netze, die auf relativ kleinen Datasets trainiert wurden, mit anderen Daten, z.B. einem Test-Dataset, evaluiert werden, dann werden schlechte Ergebnisse dargestellt. Der Grund dafür ist, dass große Netzwerke über eine größere Anzahl von Lernparametern verfügen, die die Gesetzmäßigkeiten in den Trainingsdaten herausfinden. Dafür wird das Dropout verwendet. Dropout ist eine Regularisierungstechnik, die die Netzeinheiten (also Neuronen) zusammen mit ihren Verbindungen zu einer Trainingszeit mit einer bestimmten Wahrscheinlichkeit fallen lässt.

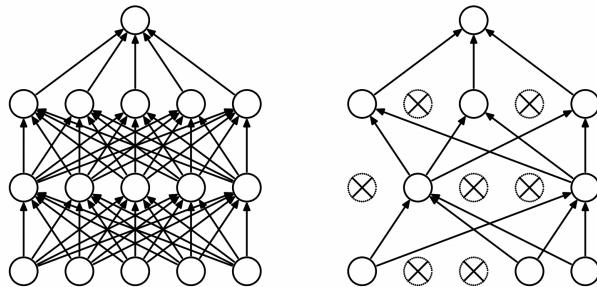


Abbildung 7: Netzwerk ohne Dropout (links) und mit Dropout (rechts) (Quelle: Srivastava et al., 2014)

Attention-Dropout ist eine andere Art von Dropout. Es wird in den Attention-basierten Architekturen verwendet. Beim Attention-Dropout fallen die Vektorelemente aus der Attention-Gleichung (Formel 9: Berechnung von Attention-Score) zufällig aus. Beim Skalarprodukt würde das erste Glied weggelassen werden.

## 2.5 Lernmethoden

Bevor ein Übergang zur Betrachtung der Spracherkennungsmodelle stattfindet, müssen auch die Lernmethoden der neuronalen Netzen erwähnt werden. Wie die jeweiligen Modelle lernen, unterscheidet sich hauptsächlich durch die Anzahl von zur Verfügung stehenden beschrifteten Daten. So gibt es *supervised*, *self-supervised* und *unsupervised* Lernarten.

- *Supervised Learning*: bezieht sich auf eine Lernweise, bei der die Lerndaten vollständig beschriftet sind. Dieses Lernen wird hauptsächlich für die Klassifizierung der Daten

(Aufteilung in die Kategorien) und für die Regression (Suche nach den Abhängigkeiten zwischen den Variablen) verwendet.

- *Unsupervised Learning*: nutzt ML-Algorithmen zur Analyse und Gruppierung von nicht beschrifteten Datasets. Diese Algorithmen entdecken versteckte Muster oder Datengruppierungen, ohne dass die menschliche Steuerung erforderlich ist.
- *Self-Supervised Learning*: bei dieser Lernmethode wird eine kleine Anzahl von beschrifteten Daten verwendet, in welchen einige Teile vom Modell selbst maskiert werden. Diese Lernmethode ist besonders weit im NLP-Bereich verbreitet, da nicht alle Sprachen gut im Internet gesammelt und beschriftet sind. Somit stellt das self-supervised Learning eine Möglichkeit dar, die quantitative Barriere zu überwinden.

### 3 Modelle

Die in der Arbeit verwendeten ein- (Wav2Vec) sowie mehrsprachigen Spracherkennungsmodelle (XLSR) sind auf die Transformers-Architektur aufgebaut. Die Bestandteile von Transformers, sowie die geschichtliche Entwicklung wurden im Theorieteil beschrieben. In diesem Kapitel wird der Aufbau von den obenerwähnten Erkennungsmodellen kurz betrachtet.

#### 3.1 Wav2Vec

Wav2Vec ist ein Modell, das auf dem self-supervised-Algorithmus und der Transformer-Architektur basiert. Das Modell lernt anhand der Spracheinheiten, die aus einem rohen Audio entnommen worden sind. Diese Spracheinheiten sind kürzer als Phonemen, was das Modell ermutigt, sich nur auf die wichtigen Bestandteilen des Audios (also auf die Rede) zu konzentrieren. Somit erreicht der verwendete Ansatz eine bessere Erkennungsrate, was die Modellautoren ebenso erwähnen:

*"With just one hour of labeled training data, wav2vec 2.0 outperforms the previous state of the art on the 100-hour subset of the LibriSpeech benchmark — using 100 times less labeled data."* (8)

Ähnlich wie andere Transformer-Modelle, trainiert sich Wav2Vec durch die Vorhersage der Spracheinheiten der maskierten Teile des Audios. Ein wesentlicher Unterschied von Wav2Vec besteht darin, dass für das Training 25ms lange Audios gebraucht werden. Die Audioneinheiten werden mithilfe von CNN ausgelesen und weiter zum Quantizer und Transformer geleitet. Der Quantizer wählt eine Spracheinheit für die latente Audiodarstellung aus einem Bestand an gelernten Einheiten aus. Etwa die Hälfte der Audiodarstellung wird abgeschirmt, bevor sie dem Transformator zugeführt wird. Der Transformator fügt Informationen aus der gesamten Audiosequenz hinzu. Schließlich wird die Transformatorausgabe verwendet, um eine kontrastive Aufgabe zu lösen. Somit muss das Modell die richtig quantisierten Spracheinheiten für die maskierten Positionen identifizieren.

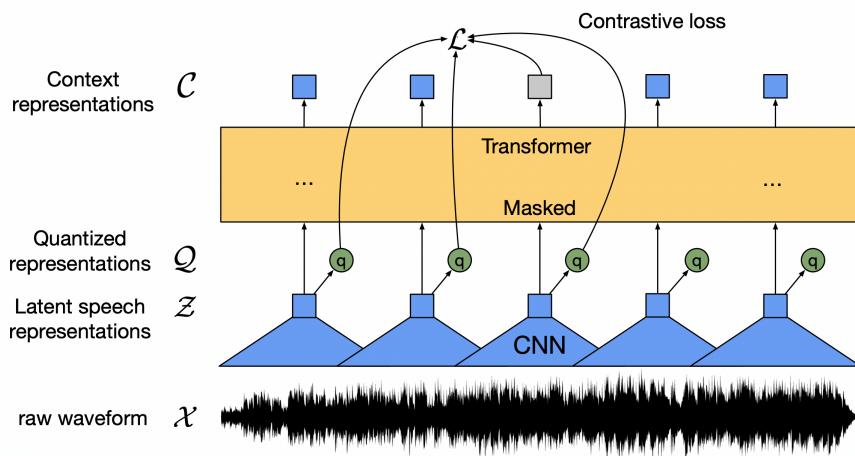


Abbildung 8: Wav2Vec-Architektur (Quelle: Baevski et. al 2020)

## 3.2 XLSR

Das Grundtraining von Wav2Vec auf mehreren Sprachen gleichzeitig hat zur Entstehung eines anderen Modell geführt. Der daraus resultierende Ansatz – XLSR – hat die Qualität der Spracherkennung der ressourcenarmen Sprachen deutlich verbessert (8). Ein gemeinsames Quantisierungsmodul über Merkmalskodierungsdarstellungen erzeugt mehrsprachige quantisierte Spracheinheiten. Die Einbettungen der Spracheinheiten werden dann als Ziele für einen durch kontrastives Lernen trainierten Transformer verwendet. Durch die sprachübergreifende gemeinsame Nutzung der diskreten Tokens, schafft das Modell Brücken zwischen den Sprachen zu erstellen.

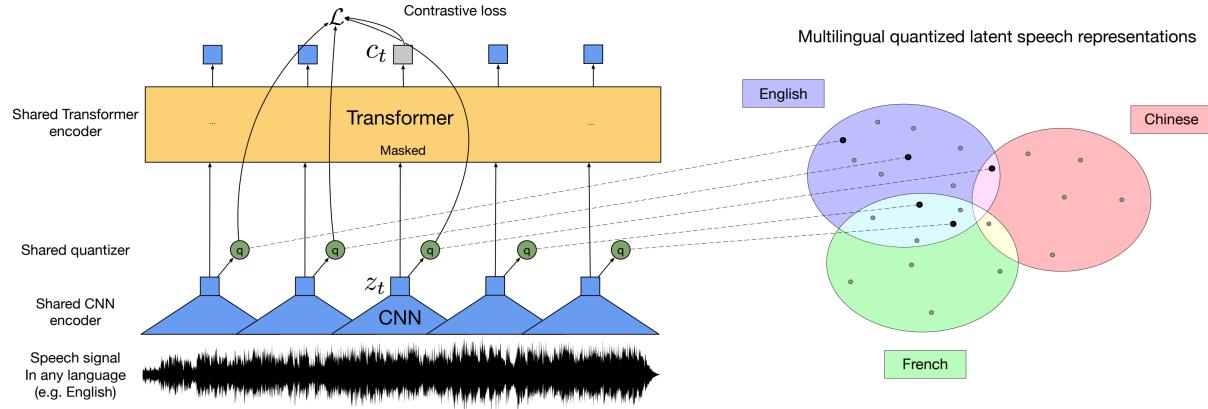


Abbildung 9: XLSR-Architektur (Quelle: Baevski et al., 2020)

## 3.3 XLSRDe

XLSRDe ist ein auf Hochdeutsch adaptiertes XLSR. Diese Adaptierung wurde von @jonatasgrosman<sup>1</sup> durchgeführt. Im Vergleich zu den anderen deutschen Modellen hat diese Modellvariante eine größere Anzahl von monatlichen Downloads: 4713 gegenüber 2157 bei Facebook<sup>2</sup> und 1448 bei einem anderen HuggingFace-Mitglied<sup>3</sup>. Darüber hinaus verfügt die Variante von @jonatasgrosman Evaluierungsergebnisse, was bei den anderen komplett fehlt. So hat das Modell 12% WER und 2,92% CER bei dem deutschen „Common Voice“-Dataset.

Aus diesen Gründen wurde entschieden, das XLSRDe von @jonatasgrosman in der Umsetzung zu verwenden.

<sup>1</sup> <https://huggingface.co/jonatasgrosman>

<sup>2</sup> <https://huggingface.co/facebook/wav2vec2-large-xlsr-53-german>

<sup>3</sup> <https://huggingface.co/maxidl/wav2vec2-large-xlsr-german>

## 4 Umsetzung

### 4.1 Ziele

Das Hauptziel der Arbeit ist es, einen minimalen Anteil der Vorarlberger Mundart mithilfe von XLSMR und Wav2Vec zu erkennen. Da es keine vorgefertigte vorarlbergerische Datasets gab, wurde ein Unterziel entdeckt - die Audiodateien maximal effektiv zu sammeln. Dafür wurden zwei Methoden verwendet: Email-Versand mit dem Link auf die Aufnahme-Webseite und eine manuelle Aufnahme der Dialektbeispiele. In der ersten Variante wurden 27 Personen (14 Frauen und 13 Männer) erreicht, die 21 im Hochdeutsch geschriebenen Sprachbefehle in ihrem Dialekt ausgesprochen haben. Bei der zweiten Variante wurden die Sprachbeispiele aus den öffentlichen Quellen, wie z.B. „Dornbirner Mundart Lexikon“<sup>1</sup> und „d'Sprôôch – Lustenauer Wörterbuch“<sup>2</sup> aufgenommen. Schlussendlich wurden die Audiodateien miteinander vermischt oder im Einzelnen in Form eines Datasets zu den Modellen geleitet. Das Fine-Tuning erfolgte mithilfe von GoogleColab, was im Grunde genommen ein Jupyter-Notebook mit Zugang auf Fern-GPUs von Google ist. Für ein besseres Verständnis und einen besseren Überblick wurden die Trainingsergebnisse auch graphisch dargestellt.

### 4.2 Datensammlung

#### 4.2.1 Client-Side

Die Webseite für die Sammlung der 21 Befehle wurde auf das Django-Framework von Python aufgebaut. Um ein auf Sicherheitsgründen beruhendes Misstrauen einer fremden Person zu vermeiden, wurde die Webseite mit „Microsoft Azure“ Cloud-Service ins globale Netz veröffentlicht. Microsoft Azure bietet einen sicheren Zugang zur Webseite durch das HTTPS-Protokoll und erfordert dazu kein eigenes Hardware-Mittel.

Die Struktur der Webseite besteht aus zwei Teilen: Umfrage- und Aufnahmeteil. Bei der Umfrage wurde das Geschlecht, die Altersgruppe und die Assoziation mit dem Dialekt der teilnehmenden Person abgefragt. Nachdem die Umfrage abgeschlossen war, wurden die Teilnehmenden gebeten, die 21 Befehle auszusprechen. Die Aufnahme erfolgte durch MediaRecorder-Interface und seine Konstruktor (beide sind in JavaScript implementiert). Während der Aufnahme wurde ein Data-Stream erstellt, in dem die Audio-Blöcke gespeichert wurden. Nach jeder Aufzeichnung wurde die Audiodatei zum lokalen Server mittels Fetch-API, bzw. durch die POST-Methode gesendet. Die durch die Umfrage gesammelten Daten zusammen mit der Nummer des Befehls wurden für die Bezeichnung des Audios verwendet. Nach einer positiven Antwort des Empfang-Servers, wurde der neue Befehl automatisch auf dem Bildschirm angezeigt.

#### 4.2.2 Server-Side

Der Server, der die Audiodateien empfangen hat, wurde einfacheitshalber mit Flask-Framework im Betrieb gesetzt. Dafür wurden nur 36 Zeilen von Python-Code gebraucht. Für einen externen Zugriff des Flask-Servers wurde ein HTTPS-Tunnel mit „PiTunnel“<sup>3</sup> erstellt. Im

---

<sup>1</sup> <https://mundartlexikon.dornbirn.at/>

<sup>2</sup> <https://www.lustenauer-mundart.at/>

<sup>3</sup> <https://www.pitunnel.com/doc>

Gegensatz zu Microsoft Azure wird PiTunnel direkt auf dem lokalen Rechner installiert. Der von PiTunnel zugewiesene URL kann nur im laufenden Zustand des Geräts erreicht werden, was auch einen Unterschied bei der Implementierung mit Microsoft Azure macht. Da der auf Raspberry Pi laufende Flask-Server nur für Empfangszwecke von Audiodateien vorgesehen wurde, hatte eine schnelle (im Vergleich zu Microsoft Azure) Bereitstellung des Servers Vorrang.

## 4.3 Datenvorbereitung

### 4.3.1 Beschriftung

Als die Daten gesammelt wurden, wurde jedes Audio durchgehört. Somit wurde sichergestellt, dass das Audio der geforderten Frequenz in Höhe von 16kHz entspricht und auch richtig nummeriert worden ist. Während des Durchhörens wurde eine Nichtübereinstimmung zwischen der Nummerierung des Audios und dem Befehl, der ausgesprochen wurde, entdeckt. Für Korrekturzwecke wurde das „Audacity“-Software verwendet.

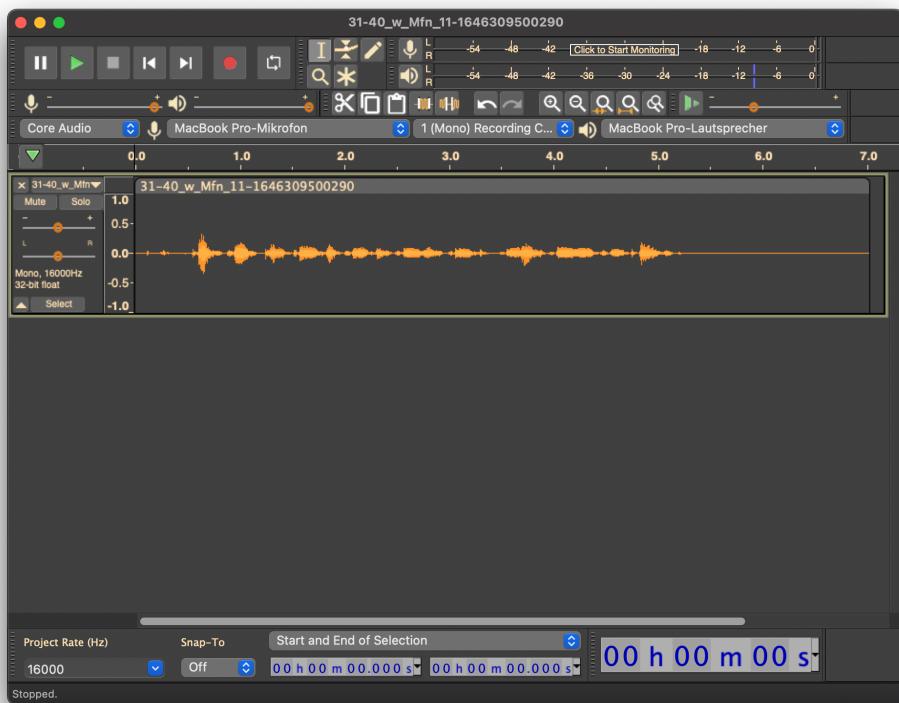


Abbildung 10: Screenshot aus Audacity (Quelle: Autor)

### 4.3.2 Datasets Erstellen

Für das Training des Modells muss ein vom Framework abhängiges Dataset erstellt werden. Im Falle von Wav2Vec und XLSMR ist das Framework PyTorch. So wird eine von `torch.utils.data.Dataset` abgeleitete Klasse erstellt, die zwei Methoden hat: `__len__`, die die Anzahl der Audios (jeweils Training- und Testaudios) zurückgibt und `__getitem__`, die das Audio mit einem Index herausnimmt. Unter dem Herausnehmen des Audios wird eine Rückgabe des Audiovektors verstanden, der mit der `read`-Methode der `scipy`-Bibliothek

durch das Auslesen des Audios erstellt wurde. Für eine bessere Repräsentation wurden insgesamt folgende Datasetkombinationen erstellt:

1. 21DL: Dataset der 21 Befehle + Dataset Dornbirn + Dataset Lustenau;
2. 21L: Dataset der 21 Befehle + Dataset Lustenau;
3. 21: Dataset der 21 Befehle;
4. DL: Dataset Dornbirn + Dataset Lustenau;
5. L: Dataset Lustenau;

Diese Kombinationen wurden so zusammengestellt, dass die Evaluierungsergebnisse zwischen supervised- und self-supervised-Lernen der 3 Modelle verglichen werden können. So zählt sich das Lernen anhand der Datasets DL und L zum self-supervised Lernen, währenddessen der Dataset der 21 Befehle ein Beispiel des supervised Lernens ist, da sich der Inhalt im Training- und Testset (außer die Stimme) nicht unterscheidet. Mithilfe von Dataset-DL wird analysiert, ob eine Verkomplizierung des Datasets bei einer niedrigen Anzahl von Trainingdaten positiven oder negativen Einfluss auf die Erkennungsqualität der Testdaten hat. Durch das Hinzufügen des Datasets-21 zu den Dataset-L und Dataset-D ist eine theoretische Abnahme der WER zu erwarten. Welches Dataset die beste Erkennungsrate liefert, ist jedoch schwierig vorherzusagen.

#### 4.3.3 Audio pro Wort

Um die Gründe für die Gesetzmäßigkeiten der späteren Ergebnisse zu finden, wurde entschieden, eine eigene Metrik zu erstellen, die „Audio pro Wort (des Vokabulars)“ oder kürzer „APW“ heißen würde. Mathematisch betrachtet, ist die Metrik durch die Division der Anzahl von Trainingaudios durch die Größe des Vokabulars berechnet. Unter der Größe des Vokabulars wird die Anzahl aller Wörter im Trainingdataset verstanden. Die Wiederholung eines Wortes, das schon bei der Berechnung verwendet wurde, wird nicht mitgezählt. Seien es zwei gleiche Sätze, wird die Größe des Vokabulars nur durch das Zählen der Wörter in einem Satz berechnet.

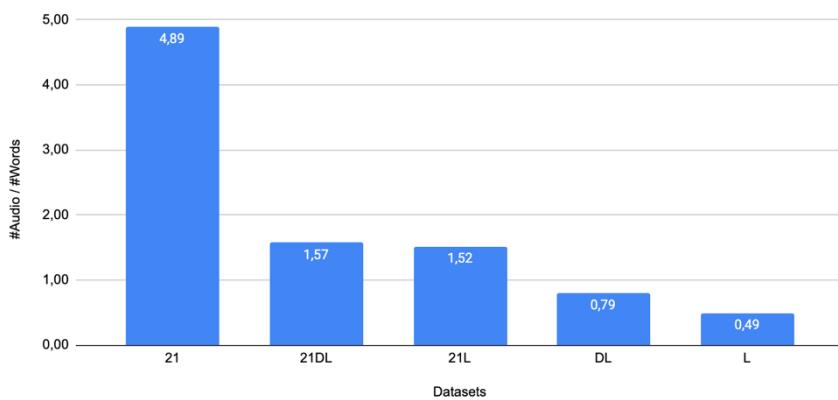


Abbildung 11: APW nach Dataset (Quelle: Autor)

#### 4.4 Fine-Tuning

Für das Fine-Tuning wurden Wav2vec als auch XLSR-Varianten von HuggingFace direkt im GoogleColab mit der `from_pretrained`-Methode heruntergeladen. Mit dieser Methode

wurde auch das Attention-Dropout auf 0,5 gesetzt. Damit wurde versucht, den Lernvorgang zu verbessern und einen frühere Überanpassung (eng. „Overfitting“) des Modells zu vermeiden. Das heruntergeladene Modell wird im von GoogleColab zugeteilten Speicherraum gespeichert.

```
[ ] 1 from transformers import Wav2Vec2ForCTC
2
3 model = Wav2Vec2ForCTC.from_pretrained(
4     "facebook/wav2vec2-xls-r-300m",
5     attention_dropout=0.5,
6     hidden_dropout=0.0,
7     feat_proj_dropout=0.0,
8     mask_time_prob=0.05,
9     layerdrop=0.0,
10    ctc_loss_reduction="mean",
11    pad_token_id=processor.tokenizer.pad_token_id,
12 )
```

Abbildung 12: Das Herunterladen von wav2vec in GoogleColab (Quelle: Autor)

Für die Evaluierung wurde WER verwendet. Die Metrik wurde zuerst mithilfe der Standardmethode der datasets-Klasse heruntergeladen. Danach erfolgte ein Vergleich zwischen dem vorhergesagten Wert und dem Referenzwert, indem die Textstrings direkt übergeben wurden.

```
wer = wer_metric.compute(predictions=pred_str, references=label_str)
return {"wer": wer}
```

Abbildung 13: Berechnung von WER (Quelle: Autor)

Nach einer Reihe von Einstellungen der Parameter, wird das Training mit einer train-Methode gestartet. Die Trainings- sowie Evaluierungsergebnisse werden periodisch nach einer bestimmten Anzahl von Lernschritten in Form einer Tabelle dargestellt.

## 5 Ergebnisse

Abbildung 14 und Abbildung 11 zeigen, dass bei einer zunehmenden APW eine Abnahme der WER stattfindet. So wurde von allen Modellen mit dem Dataset-21 die kleinste WER erreicht. Das zeigt auch, dass die stimmbezogene Information, wie z.B. Geschlecht, Dialektassoziation, Altersgruppe bei größeren Modellen keinen Einfluss auf die Erkennung macht, falls die Daten komplett beschriftet sind.

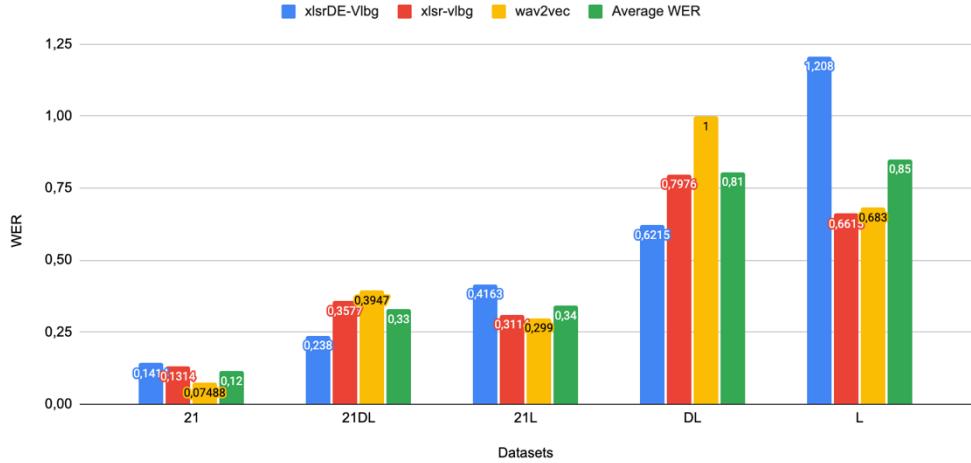


Abbildung 14: WER nach Dataset (Quelle: Autor)

Eine Verkomplizierung des Datasets-Lustenau durch das Hinzufügen des Datasets-Dornbirn hat zwar die APW von 0,49 auf 0,79 erhöht, hat aber zu keinem wesentlichen Unterschied der durchschnitts-WER geführt: von 0,85 beim Dataset-Lustenau auf 0,81 beim Dataset-Dornbirn-Lustenau. Die Abnahme der WER liegt XLSRDe zugrunde, da sich die Erkennungsqualität nur bei diesem Modell verdoppelt hat. Bei den anderen Modellen hat sich die WER erhöht: von 0,683 auf 1 (Wav2Vec) und von 0,6619 auf 0,7976 (XLSR-Vlbg). Der Grund liegt höchstwahrscheinlich an dem Fine-Tuning auf Hochdeutsch, das die Modellparameter vorher spezifisch für diese Sprache eingestellt hat.

Die genauere Erklärung der eben erläuterten Ergebnisse erfolgt anhand von Abbildung 15. Es fällt auf, dass im Vergleich zu den anderen Modellen die Erkennungsperformanz von XLSRDe bei einer Vokabulargröße größer als 700 Wörter am besten ist. Es ist auch zu erwähnen, dass bei kleineren Datasets, deren Vokabulargröße kleiner als 700 ist, eine besonders gute Erkennungsrate von Wav2Vec erreicht wird.

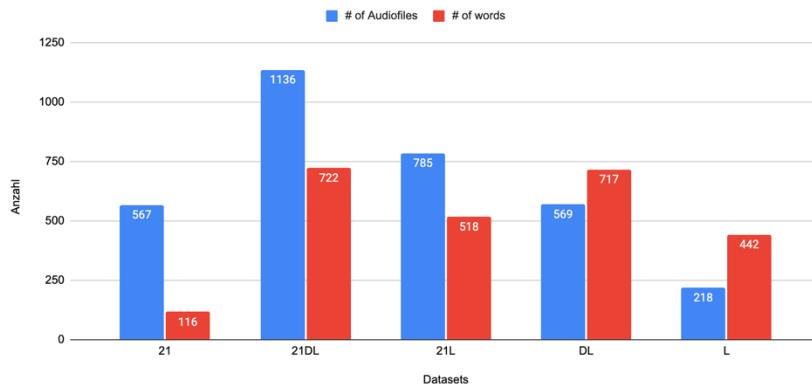


Abbildung 15: Anzahl der Audiodateien und die Größe des Vokabulars (Quelle: Autor)

Abbildung 16 stellt die WER-Verteilung nach Dataset dar. Zwar ist ersichtlich, dass XLSR-Vlbg im Allgemeinen die kleinste durchschnittliche WER i.H.v. 0,45 und eine dichtere Verteilung der Erkennungswerte hat. Das Wav2Vec steht an Platz 2 mit einer durchschnitts-WER gleich 0,49. Das XLSRDe hat sowohl die größte Differenz zwischen maximalen und minimalen Erkennungswerten, als auch die größte durchschnitts-WER.

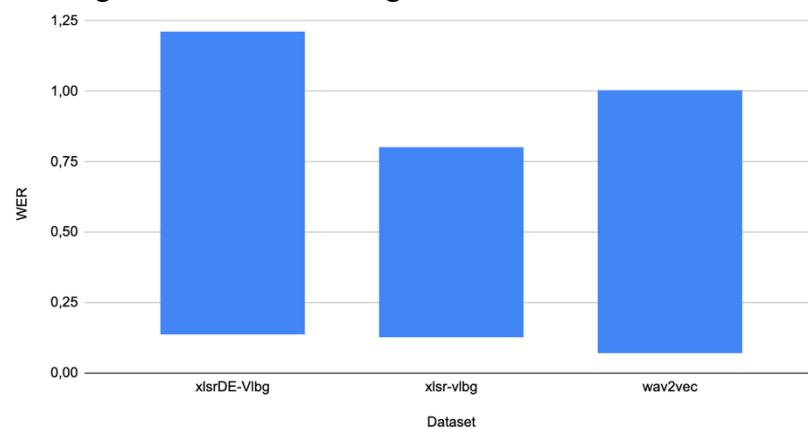


Abbildung 16: Verteilung der WER-Werte nach Dataset (Quelle: Autor)

## 6 Zusammenfassung und Ausblick

Mit dieser Arbeit wurde versucht, einen kleinen Teil des Wortbestands des Vorarlberger Dialektes zu implementieren. Die Audiodateien wurden sowohl anhand der Webseite als auch manuell aus den öffentlichen Quellen gesammelt. Die gesammelten Daten wurden im Form eines Datasets zusammengefasst und miteinander vermischt. Anhand der Datasets wurden drei Modelle für den Dialekt adaptiert. Eine indirekte Proportionalität zwischen APW und dem durchschnitts-WER ist zu sehen. Die Erkennungsperformanz der Modelle darf aber im Allgemeinen nicht betrachtet werden, da eine Abhängigkeit zwischen den Erkennungswerten der Dataset bestand. So wurde bei einem komplett beschrifteten Dataset-21 die kleinste WER von einem einsprachigen Wav2Vec erreicht, währenddessen das mehrsprachige XLSR immer die „zweitbeste“ war. Es kann aber behauptet werden, dass XLSR für einen allgemeinen Fall optimal geeignet ist. Falls es um verkomplizierte Datasets geht, kann XLSRDe als eine bessere Lösung betrachtet werden, da die WER dieser Modellvariante am kleinsten (im Vergleich zu den anderen) ist.

Der wesentliche Arbeitsaufwand entstand durch das Sammeln der Audiodaten und der Be-schriftung derer. Große Trainingszeiten und zufällige Ausfälle von GoogleColab haben dem kontinuierlichen Arbeitsfortschritt auch entgegengewirkt. Nichtsdestotrotz ist die Arbeit der erste Schritt in Richtung Dialekterkennung in Vorarlberg. Das gesammelte Dataset kann vervollständigt werden, indem bspw. die Autoren der öffentlichen Wörterbücher der Dornbirner und Lustenauer Mundart Sprachbeispiele zur Verfügung stellen würden. Eine andere Möglichkeit wäre, ein offizielles Crowdsourcing-Projekt in Form einer Webseite zu starten, indem verschiedene Menschen die gegebenen Texte aussprechen würden und ggf. die automatische Erkennung korrigieren. Ein Beispiel dafür ist „Common-Voice“ von Mozilla, womit schon Audiodaten von knapp 14000 Stunden in 76 Sprachen gesammelt wurden.

Als nächsten Schritt würde es sich auch lohnen, einen Übersetzer zu implementieren, der im Vorarlberger Dialekt geschriebenen Text ins Hochdeutsche (sowie in die andere Richtung) übersetzen kann. Dafür ist jedoch mehr Aufwand nötig, da die Schreibweise der Dialekte von Ort zu Ort unterschiedlich ist.

Es wäre auch interessant, die Ähnlichkeiten der Dialekte für das jeweilige Modell herauszufinden. Also welche Dialekte des Vorarlberger Raums mehr, und welche weniger Zusammenhang haben. Die daraus folgenden Ergebnisse können auch mithilfe der Linguistik begründet werden, was eine resultative Kooperation verspricht.

## Literaturverzeichnis

1. **Wikipedia-Autoren.** Geschichte der künstlichen Intelligenz. *Wikipedia*. [Online] 6. November 2001. [Zitat vom: 7. Juni 2022.]  
[https://de.wikipedia.org/wiki/Geschichte\\_der\\_k%C3%BCnstlichen\\_Intelligenz](https://de.wikipedia.org/wiki/Geschichte_der_k%C3%BCnstlichen_Intelligenz).
2. **Steffen Schneider, Alexei Baevski, Ronan Collobert, Michael Auli.** *Wav2vec: Unsupervised Pre-training For Speech Recognition*. [arxiv.org] s.l. : Facebook AI, 2019. arXiv:1904.05862v4.
3. **Alexis Conneau, Alexei Baevski, Ronan Collobert, Abdelrahman Mohamed, Michael Auli.** *Unsupervised Cross-lingual Representation Learning For Speech Recognition*. [arxiv.org] s.l. : Facebook AI, 2020. arXiv:2006.13979v2.
4. **Krishna D N, Pinyi Wang, Bruno Bozza.** *Using Large Self-Supervised Models for Low-Resource Speech Recognition*. [Interspeech 2021] India : Freshworks Inc, 2021. doi: 10.21437/Interspeech.2021-631.
5. **Laricchia, Federica.** *statista*. [Online] Speechmatics, 08. 02 2022. [Zitat vom: 17. 05 2022.] <https://www.statista.com/statistics/1208460/global-voice-technology-future-industries/>.
6. **Yano Research.** *statista*. [Online] 11. April 2022. [Zitat vom: 17. Mai 2022.]  
<https://www.statista.com/statistics/1301301/japan-speech-recognition-market-size/>.
7. **Ahrens, Sandra.** *statista*. [Online] 28. Jänner 2020. [Zitat vom: 17. Mai 2022.]  
<https://de.statista.com/statistik/daten/studie/1091481/umfrage/anwendung-von-kuenstlicher-intelligenz-in-deutschland-oesterreich-und-der-schweiz/>.
8. **Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin.** *Attention Is All You Need*. s.l. : arxiv.org, 2017. arXiv:1706.03762.
9. **Baevski, A., Conneau, A., Auli, M.** Meta AI. *Wav2vec 2.0: Learning the structure of speech from raw audio*. [Online] Facebook AI, 24. September 2020. [Zitat vom: 31. Mai 2022.] <https://ai.facebook.com/blog/wav2vec-20-learning-the-structure-of-speech-from-raw-audio/>.
10. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting.** **Srivastava, Nitish, et al.** 15, s.l. : Journal of Machine Learning Research, 2014.

## Anhang

### Fragebogen aus der Webseite

Servus! Mein Name ist Bogdan Khamelyuk. Ich bin Student im Studiengang „Elektrotechnik-Dual“ der FH Vorarlberg. Zusammen mit den Betreuern arbeite ich an meiner Bachelorarbeit welche sich um das Thema der „Erkennung von in Dialekt gesprochenen Sätzen“ dreht. Dafür brauche ich möglichst viele im Dialekt gesprochene Sätze – und somit deine Hilfe. Im folgendem wirst Du aufgefordert 21 Sätze, Befehle oder Wörter in deiner Mundart zu sprechen. Diese werden anonym abgelegt. Die Webpage speichert nur die folgenden Angaben die mir bei der Auswertung helfen:

Wie alt bist du?

Was ist dein Geschlecht?

Mit welchem Dialekt assoziierst du dich am ehesten?

[zum Text](#)

### Aufnahmeteil der Webseite

← → C 🔒 dialektsammler.azurewebsites.net/recorder/

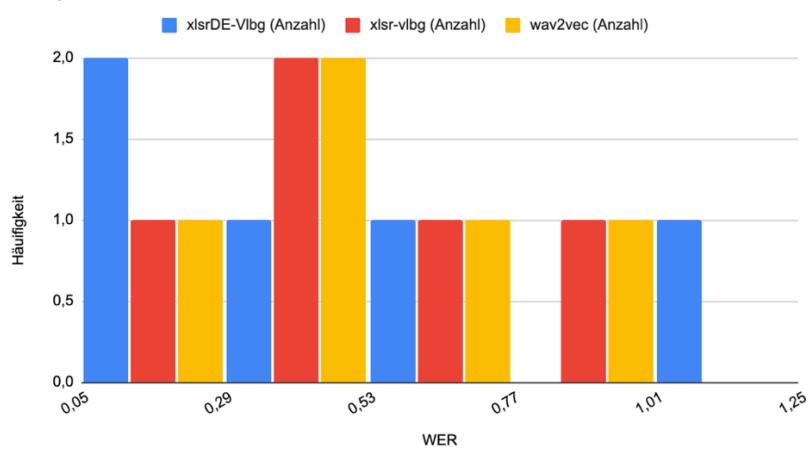
Sprich den gegebenen Satz in deinem Dialekt aus:

**wie ist das Wetter?**

[Start](#) bereit zu aufnehmen

1 /21

### Frequenz von Modell-WER in einem bestimmten Bereich



## **Eidesstattliche Erklärung**

Ich erkläre hiermit an Eides statt, dass ich vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.  
Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 08. Jun. 2022

 Unterschrift Verfasser\*in