



# **Frankfurt University of Applied Sciences**

–Faculty of Computer Science and Engineering–

## **Evaluation von Speech Recognition Systemen und Erstellen eines Proof of Concept für Vorlesungen**

Abschlussarbeit zur Erlangung des akademischen Grades  
Bachelor of Science (B.Sc.)

vorgelegt von

**Tobias Maas**

Matrikelnummer: 1207828

Referent : Prof. Dr. Eicke Godehardt  
Korreferentin : Prof. Dr. Ute Bauer-Wersing

## ERKLÄRUNG

---

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

*Bad Vilbel, 05.07.2021*



---

Tobias Maas

## ABSTRACT

---

Das Ziel dieser Arbeit ist die Evaluation von Spracherkennungssystemen und die Ausarbeitung eines Proof of Concept zur Verarbeitung von Vorlesungen.

Für die Evaluation werden die Aufzeichnungen aufbereitet und mit statistischen Kennzahlen wird die Genauigkeit gemessen. Außerdem wird die Geschwindigkeit der Verarbeitung gestoppt. Bei dem Proof of Concept wird eine Konzeption und Implementierung für die Spracherkennung von Vorlesungen vorgestellt. Die Ergebnisse werden in unterschiedlichen Ausgaben dargestellt.

Die Evaluation zeigt, dass der Anbieter Rev.AI die beste Kombination von Geschwindigkeit und Genauigkeit anbietet. Google und Microsoft Azure folgen auf den nächsten Plätzen. Die beiden Systemen haben entweder die Geschwindigkeit oder die Genauigkeit, aber nicht die Kombination. Abgeschlagen folgt die Software von IBM mit der schlechtesten Geschwindigkeit und Genauigkeit. Das Proof of Concept kann die transkribierten Videodateien in eine PDF schreiben und/ oder als Untertitel in das Video einfügen. Bei der PDF werden zusätzlich zu dem Resultat der Transkription noch Informationen von der Position des Transkripts im Video ergänzt.

Die Ergebnisse der Evaluation zeigen, dass die Anbieter ihre Programme im Vergleich zu anderen Studien verbessert haben. Weiterführende Forschung bei der Evaluation muss mit dem Hinzufügen weiterer Anbieter von Spracherkennungssoftware und Aufzeichnungen mit anderen Themen getätigt werden. Das Proof of Concept gibt eine Möglichkeit die Aufzeichnungen der Vorlesung in verschiedenen Formaten an die Studierenden weiterzugeben. Dabei müsste noch weiter an der Verschiebung der Tonspur in gewisse Frequenzbereiche gearbeitet werden.

## DANKSAGUNG

---

An dieser Stelle gilt mein Dank all denjenigen, die mich während des Schreibprozesses unterstützt haben.

Außerdem danke ich den Lehrenden des Studiengangs Bachelor der Informatik für das Bereitstellen der Vorlesungsaufzeichnungen. Ohne diese Hilfe hätte die Arbeit in dieser Form nicht entstehen können.

Ebenfalls möchte ich mich bei den Korrekturlesern meiner Bachelorarbeit bedanken.

Tobias Maas  
Bad Vilbel, 05.07.2021

# INHALTSVERZEICHNIS

Abbildungsverzeichnis	viii
Tabellenverzeichnis	ix
Listings	x

## I THESIS

1	EINLEITUNG	1
1.1	Problemstellung und Motivation	1
1.2	Zielsetzung	2
1.3	Aufbau der Arbeit	2
2	THEORETISCHE RAHMEN	3
2.1	Definition	3
2.1.1	Spracherkennung	3
2.1.2	Evaluation	3
2.1.3	Proof of Concept	4
2.2	Forschungsstand zur Evaluation von Spracherkennungssystemen	4
2.3	Sprachtypen	5
2.3.1	Isolierte Sprache	5
2.3.2	Verbundene Worte	5
2.3.3	Durchgängige Sprache	5
2.3.4	Spontane Sprache	6
2.4	Geschichte	6
2.4.1	Erste Generation	6
2.4.2	Zweite Generation	6
2.4.3	Dritte Generation	7
2.4.4	Späte dritte Generation 1990er	7
2.4.5	Späte dritte Generation 2000er	8
2.5	Wahrscheinlichkeitstheorie der Spracherkennung	8
2.6	Aufbau der Spracherkennung	9
2.6.1	Einleitung	9
2.6.2	Akustisches Front-end	10
2.6.3	Akustische Modell	12
2.6.4	Sprach-Modell	13
2.6.5	Decoder	14
2.7	Algorithmen für die Spracherkennung	15
2.7.1	Akustisch-phonetisch	15
2.7.2	Muster-Erkennung	15
2.7.3	Artificial intelligence	17
2.8	Herausforderungen bei dem Design der Spracherkennung	18
2.8.1	Herausforderungen beim Erkennen der Wörter	18
2.8.2	Herausforderung für Systeme mit verschiedenen Sprachen	20

2.8.3	Unterschied Deutsch und Englisch . . . . .	21
2.9	Weiterentwicklungen von Spracherkennungssystemen . . . . .	22
3	EVALUATION VON SPEECH RECOGNITION SYSTEMEN . . . . .	24
3.1	Vorstellung der Programme . . . . .	24
3.2	Vorstellung der Daten . . . . .	25
3.2.1	Vorlesungen . . . . .	25
3.2.2	externe Vergleichsdaten . . . . .	25
3.3	Methoden zur Bestimmung der Genauigkeit . . . . .	25
3.3.1	Word Error Rate . . . . .	26
3.3.2	Match Error Rate . . . . .	26
3.3.3	Word Information Lost . . . . .	27
3.4	Methoden zum Bestimmen der Geschwindigkeit . . . . .	27
3.5	Durchführung der Evaluation . . . . .	27
3.5.1	Anbindung der Programme . . . . .	27
3.5.2	Vorbereitung der Videodateien . . . . .	28
3.5.3	Transkript für die Videos erstellen . . . . .	29
3.5.4	Transkripte an Vorlage anpassen . . . . .	29
3.5.5	Transkripte mit Vorlage vergleichen . . . . .	29
3.5.6	Dauer der automatisch erstellten Transkripte vergleichen . . . . .	30
3.6	Ergebnisse . . . . .	30
3.6.1	Genauigkeit Zusammenfassung . . . . .	30
3.6.2	Genauigkeit Englisch . . . . .	31
3.6.3	Genauigkeit Deutsch . . . . .	32
3.6.4	Geschwindigkeit . . . . .	33
3.7	Diskussion . . . . .	33
3.7.1	Vergleich zu den vorgestellten Studien . . . . .	34
3.7.2	Auslassung von CMU Sphinx . . . . .	34
3.7.3	Genauigkeit . . . . .	34
3.7.4	Geschwindigkeit . . . . .	36
3.7.5	Verknüpfung von Genauigkeit und Geschwindigkeit . . . . .	36
3.7.6	Probleme bei der Auswertung . . . . .	38
4	PROOF OF CONCEPT . . . . .	40
4.1	Bestimmung der Aufgaben . . . . .	40
4.2	Konzeption des Programms . . . . .	40
4.2.1	Design der Struktur . . . . .	40
4.2.2	Auswahl des Spracherkennungsprogramms . . . . .	41
4.2.3	Erstellen der Timestamps . . . . .	41
4.3	Implementierung des Proof of Concept . . . . .	42
4.3.1	Start des Programms . . . . .	42
4.3.2	Umwandlung von Video- in Audioformat . . . . .	43
4.3.3	Vorbereitung der Audiodatei . . . . .	43
4.3.4	Transkription . . . . .	45
4.3.5	Vorbereitung Ausgabe . . . . .	46
4.3.6	Ausgabe PDF . . . . .	47
4.3.7	Ausgabe Video mit Untertitel . . . . .	49
4.3.8	Löschung Audiodateien . . . . .	51

4.4	Test des Proof of Concept . . . . .	51
4.4.1	Geschwindigkeit des Programms . . . . .	52
4.4.2	Konsolen Interface . . . . .	52
4.4.3	Untertitel des Videos . . . . .	53
4.4.4	PDF mit Informationen . . . . .	54
4.5	Diskussion . . . . .	55
4.5.1	Erkennen von Stille in der Tonspur . . . . .	55
4.5.2	Geschwindigkeit der Erstellung des Videos . . . . .	55
4.5.3	Verhältnis Tonspur zu Untertitel . . . . .	56
4.5.4	Verhältnis Gesagten zu Transkription . . . . .	56
4.5.5	Speicherung der Audio-Elemente . . . . .	57
5	FAZIT . . . . .	58
5.1	Zusammenfassung . . . . .	58
5.2	Weiterführende Forschung . . . . .	59
 <b>II ANHANG</b>		
A	PROOF OF CONCEPT CODE . . . . .	x
B	TABELLEN GENAUIGKEIT . . . . .	xvii
B.1	Technische Informatik . . . . .	xvii
B.2	Statistik . . . . .	xviii
B.3	Betriebssysteme . . . . .	xix
C	TABELLEN GESCHWINDIGKEIT . . . . .	xx
 LITERATUR . . . . .		
		xxi

## ABBILDUNGSVERZEICHNIS

---

Abbildung 2.1	Ablauf der Spracherkennung [6] . . . . .	9
Abbildung 2.2	Aufbau der Spracherkennung [8] . . . . .	10
Abbildung 2.3	Extrahierung der Eigenschaften [9] . . . . .	11
Abbildung 4.1	Komponentendiagramm . . . . .	40
Abbildung 4.2	Eingabe Konsole . . . . .	52
Abbildung 4.3	Konsole Help Funktion . . . . .	53
Abbildung 4.4	Video mit Untertitel . . . . .	53
Abbildung 4.5	Video mit Fehlermeldung als Untertitel . . . . .	54
Abbildung 4.6	PDF mit Überschrift und Transkription . . . . .	54
Abbildung 4.7	PDF mit Fehlermeldung . . . . .	55



## TABELLENVERZEICHNIS

---

Tabelle 3.1	Table Cloud Programme . . . . .	24
Tabelle 3.2	Zusammenfassung der Ergebnisse . . . . .	31
Tabelle 3.3	Cloud Computing . . . . .	31
Tabelle 3.4	Ted Talk . . . . .	31
Tabelle 3.5	Betriebssysteme Zusammengefasst . . . . .	32
Tabelle 3.6	Statistik Zusammengefasst . . . . .	32
Tabelle 3.7	Technische Informatik Zusammengefasst . . . . .	32
Tabelle 3.8	Vergleich der Geschwindigkeit englischer Aufzeichnungen . . . . .	33
Tabelle 3.9	Vergleich der Geschwindigkeit deutscher Aufzeichnungen . . . . .	33
Tabelle 3.10	Vergleich der Genauigkeit nach Sprachen . . . . .	35
Tabelle 3.11	Geschwindigkeit Verhältnis . . . . .	36
Tabelle 3.12	Vergleich der Programme nach Platzierungen . . . . .	37
Tabelle 3.13	Vergleich der Programme nach Verhältnis . . . . .	38
Tabelle B.1	Technische Informatik Anfang . . . . .	xvii
Tabelle B.2	Technische Informatik Mitte . . . . .	xvii
Tabelle B.3	Technische Informatik Ende . . . . .	xvii
Tabelle B.4	Statistik Anfang . . . . .	xviii
Tabelle B.5	Statistik Mitte . . . . .	xviii
Tabelle B.6	Statistik Ende . . . . .	xviii
Tabelle B.7	Betriebssysteme Anfang . . . . .	xix
Tabelle B.8	Betriebssysteme Mitte . . . . .	xix
Tabelle B.9	Betriebssysteme Ende . . . . .	xix
Tabelle C.1	Vergleich der Geschwindigkeit Vorlesung Betriebssysteme . . . . .	xx
Tabelle C.2	Vergleich der Geschwindigkeit Vorlesung Statistik . . .	xx
Tabelle C.3	Vergleich der Geschwindigkeit Vorlesung Technische Informatik . . . . .	xx

## LISTINGS

---

Listing 3.1	Wit Verbindungsbeispiel . . . . .	28
Listing 3.2	Vergleich Transkription zu Vorlage . . . . .	30
Listing 4.1	Implementierung Konsolen Interface . . . . .	42
Listing 4.2	Umwandlung Video- zu Audioformat . . . . .	43
Listing 4.3	Einlesen und Aufteilen der Audiodatei . . . . .	43
Listing 4.4	Zusammenfügen der Audiodateien zu längeren Teilstücken . . . . .	44
Listing 4.5	Anpassung der Länge der Teilstücke . . . . .	44
Listing 4.6	Abspeichern für die Transkription . . . . .	45
Listing 4.7	Transkription . . . . .	45
Listing 4.8	Erstellen der Videoabschnitte . . . . .	46
Listing 4.9	Erstellen Teiluntertitel . . . . .	47
Listing 4.10	Erstellen der PDF Initialisierung . . . . .	47
Listing 4.11	Erstellen der PDF Schleife Teil 1 . . . . .	48
Listing 4.12	Erstellen der PDF Schleife Teil 2 . . . . .	49
Listing 4.13	Erstellen der Untertitel und Zusammenfügen der Videoabschnitte Teil 1 . . . . .	49
Listing 4.14	Erstellen der Untertitel und Zusammenfügen der Videoabschnitte Teil 2 . . . . .	51
Listing 4.15	Löschen der vorübergehend gespeicherten Audiodateien . . . . .	51
Listing A.1	Kompletter Python Code . . . . .	x

Teil I

THESIS

## EINLEITUNG

---

Überall, wo der Mensch mit Computersystemen interagiert, kann die mechanische Eingabe über die Tastatur durch akustische Inputs der Sprache getauscht werden. Dort kommen Spracherkennungssysteme zum Einsatz, die die akustischen Signale wahrnehmen und in Buchstaben umwandeln. Dadurch können Spracherkennungssysteme in allen Branchen eingesetzt werden. Die Fähigkeiten der Spracherkennung haben sich soweit entwickelt, dass sie in den Bereich der breiten Bevölkerung vorgedrungen sind.

Spracherkennung kann dafür verwendet werden, Menschen mit besonderen Bedürfnissen in die Gesellschaft zu integrieren. Viele Leute haben Probleme beim Verfolgen der Vorlesungen oder können diese schwerer verstehen. Die Aufzeichnungen der Vorlesungen zu transkribieren ermöglicht jedem eine gleiche Chance auf seinem Bildungsweg.

Die IEEE Signal Processing Society zählt diese Punkte als Vorteile für die Spracherkennung auf:

- Zu Robotern sprechen
- Digitale Geräte kontrollieren
- Visuell und akustisch Eingeschränkte helfen
- Handfreie Technologie ermöglichen [46]

### 1.1 PROBLEMSTELLUNG UND MOTIVATION

Was macht ein gutes Spracherkennungs-System aus ? Wenn man nach IBM geht, dann sind das folgende Punkte:

- Sprach-Gewichtung: Verbesserte Genauigkeit beim stärkeren Gewichten von spezifischen Wörtern, die häufig gesprochen werden
- Sprecher-Markierung: Eine Ausgabe der Transkription, welche jeden Sprecher in einer Mehr-Sprecher- Konversation zitiert oder markiert
- Akustisches Training: An der akustischen Seite des Business teilnehmen. Das System so trainieren, dass es eine akustische Umwelt und Sprecher-Style anpasst
- Profanität filtern: Nutzung von Filtern um gewisse Wörter oder Phrasen zu identifizieren und das Transkript zu bereinigen. [47]

Bei der Evaluation von den Speech Recognition Systemen soll der Fokus auf Aufzeichnungen aus Vorlesungen liegen, bei denen vor allem eine sprechende Person zuhören sein wird. In dieser Arbeit werden aufgezeichnete

Vorlesungen aus Online Meeting Software verwendet. Diese werden weniger Hintergrundgeräusche als Aufzeichnungen aus einem Saal haben. Ein Augenmerk der Evaluation ist es, das "richtige" oder "beste" System für unser Proof of Concept zu finden.

Bei dem Proof of Concept soll ein Programm entstehen, welches auf die Bedürfnisse von Universitäten und Fachhochschulen ausgelegt ist. Die Applikation soll Videodateien aus Vorlesungen als Input verarbeiten, ein Transkript erstellen und als ein lesbares und verteilbares Format abspeichern. In der Outputdatei sollen Zeitmarkierungen vorhanden sein, sodass die zeitliche Stelle des Gesagten erkennbar ist.

## 1.2 ZIELSETZUNG

Die Bachelorarbeit hat zwei Ziele. Die erste Aufgabe ist eine Auswahl an Spracherkennungssoftware zusammenzustellen und diese miteinander zu vergleichen. Dabei beruht die Evaluation auf zwei verschiedenen Methoden. Die Systeme werden auf ihre Schnelligkeit untersucht als auch auf ihre Genauigkeit. Dabei werden für die Bestimmung der Genauigkeit verschiedene Fehlerkennzahlen verwendet. Für die Einordnung der Ergebnisse von den Vorlesungen wird eine vergleichbare Audiodatei verwendet. Diese Datei ist ein Talk mit einem ähnlichem Informatikthema wie die Vorlesungen. Um die Fähigkeiten der Programme in verschiedenen Sprachen zu überprüfen, werden Vorlesungen der deutschen und englischen Sprache verwendet.

Der zweite Teil der Arbeit wird ein Proof of Concept für das automatische Transkribieren von Vorlesungen erstellt. Das Programm unterstützt verschiedene Videoformate und die Transkripte werden in einem Format ausgegeben, dass leicht zu verarbeiten ist. Der Fokus des Teils liegt auf der Konzeption und der technischen Umsetzung.

## 1.3 AUFBAU DER ARBEIT

Im ersten Abschnitt werden die theoretischen Konzepte der Spracherkennung, die Geschichte und die Herausforderungen bei dem Design von Spracherkennungssystemen erklärt.

Im folgenden Teil wird eine Auswahl an Spracherkennungssystemen evaluiert. Dabei werden die Systeme anhand ihrer Genauigkeit und Geschwindigkeit bewertet und im Anschluss werden die Ergebnisse diskutiert.

Der Proof of Concept wird im nächsten Kapitel beschrieben. Der Fokus liegt auf dem Erstellen von Transkripten von Aufzeichnungen aus Vorlesungen. Im Anschluss wird auf mögliche Vor- und Nachteile des Proof of Concept hingewiesen.

Abschließend folgt ein Fazit zu den Erkenntnissen.

## THEORETISCHE RAHMEN

---

In dem theoretischem Rahmen wird auf bisherige Forschungsarbeiten zu dem Thema Evaluation eingegangen. Außerdem wird die Struktur der Spracherkennung besprochen und die Herausforderungen erklärt.

### 2.1 DEFINITION

Der Abschnitt Definition bestimmt die Begriffe Spracherkennung, Evaluation und Proof of Concept, die für diese Arbeit eine Bedeutung haben.

#### 2.1.1 *Spracherkennung*

Levis und Suvorov [15] definieren in ihrem Papier Spracherkennung als "Automatische Spracherkennung ist ein unabhängiger, auf Maschinen basierender Prozess vom Decodieren und Transkribieren von oraler Sprache". Außerdem stellen sie in ihrer Definition klar, dass es Unterschiede zwischen der Spracherkennung und dem Sprachverstehen, als auch zwischen Spracherkennung und Stimmerkennung gibt. Bei dem Sprachverstehen oder Sprachidentifikation wird die Bedeutung des Gesagtem versucht zu verstehen und bei der Stimmerkennung versucht man die sprechende Person zu identifizieren. Die Forschenden nennen in ihrer Definition die wissenschaftlichen Felder, in denen die Spracherkennung angesiedelt ist. Die Spracherkennung verwendet Eigenschaften aus der Linguistik, Informatik und Elektrotechnik. [15]

Die Website DeepAI.org [25] bezeichnet Automatische Spracherkennung als "[...] ein Teilfeld der rechnergestützten Linguistik, welche von der Erkennung und Übersetzung von gesprochener Sprache in Text durch Computer handelt, manchmal wird der Prozess als 'speech to text' bezeichnet". [25]

Aus den vorherigen Definitionen geht hervor, dass die Spracherkennung als Verarbeitung von Audiosignalen mit dem Ziel des Verstehens von dem Gesagten verstanden werden kann. Dabei ist die Art der Aufnahme des Gesagten, die verwendete Vorgehensweise zur Bearbeitung und die Ausgabe der Daten für die Definition, nicht von Bedeutung.

#### 2.1.2 *Evaluation*

Regina Nissen [20] hat für das Wirtschaftslexikon des Springer Gabler Verlaages das Evaluieren als "Sammelbezeichnung für den systematischen Einsatz von Methoden, die dazu dienen, die Erreichung eines vorab festgelegten Ziels einer Intervention [...] nach deren Durchführung zu überprüfen." definiert.

Das Evaluieren wird in der Arbeit als ein Bewerten von Methoden oder Programmen durch statistische Mittel festgelegt. Dabei ist das Vorgehen reproduzierbar und somit überprüfbar.

### 2.1.3 *Proof of Concept*

Catherine Kendig [7] beschreibt in ihrem Papier Proof of Concept als "[...] Forschung in den anfänglichen Stufen, auf dem neusten Stand von neuen Applikationen oder Technologien und ist ein Schlagwort um wissenschaftliche Forschung als potentiell erweiterbar und/ oder skalierbar zu kennzeichnen". Außerdem bestimmt Sie den Begriff *Proof of Concept Forschung* als "eine bestimmte Art von Forschung, welche das Ziel eine Frage zu beantworten ist, dessen Antwort eine weite Anwendungsmöglichkeit in Bereichen des Getesteten hat. Auf diese Weise bietet die Forschung eine Begründung in der Ausführung von der potentiellen Transportfähigkeit der Forschung". [7] Der Begriff Proof of Concept beschreibt ein Vorgehen, in dem ein neues theoretisches Konstrukt erarbeitet wird, das auf einen spezifischen Fall reduziert wird. Im nächsten Schritt wird dieses Konstrukt zu einem Programm entwickelt, bei dem der Fokus auf die Lauffähigkeit des Programms ausgelegt ist und weniger auf das Anwenden von Entwicklungsrichtlinien oder das Testen des Programms.

## 2.2 FORSCHUNGSSTAND ZUR EVALUATION VON SPRACHERKENNUNGSSYSTEMEN

In den folgenden wissenschaftlichen Arbeiten wurde von unterschiedlichen Gruppen Spracherkennungssysteme evaluiert.

Kim et al. [14] haben in ihrer Arbeit "A Comparison of Online Automatic Speech Recognition Systems and the Nonverbal Responses to Unintelligible Speech" fünf Spracherkennungssysteme miteinander verglichen und ausgewertet. Dabei wurde in der Arbeit der Zusammenhang von nonverbalen Verhalten zu nicht intelligenter Sprache untersucht. Diese hätte sich durch eine erhöhte Fehlerrate angedeutet. In der wissenschaftliche Arbeit waren die verwendeten Systeme Google Cloud, IBM Watson, Microsoft Azure, Trint und Youtube. In der Untersuchung kam heraus, dass Youtube die beste Genauigkeit erzielt hat und die durchschnittliche WER <sup>1</sup> bei 17,4% lag. [14]

Maglogiannis et al. [12] sind in ihrer Arbeit "A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems" aus dem Jahr 2020 auf die Genauigkeit und Geschwindigkeit von Spracherkennungssystemen eingegangen. Für die Untersuchung der Systeme IBM Watson, Google und WIT haben sie die Methoden WER, Hper und Rper<sup>2</sup> eingesetzt. Sie sind zu dem Ergebnis gekommen, dass die Spracherkennung von Google am besten abgeschnitten hat. Google erzielte in diesem Versuch eine durchschnittliche WER bei drei verschiedene Sprechern von 20.63% und die durchschnittliche WER

---

<sup>1</sup> (siehe 3.3.1)

<sup>2</sup> Hper und Rper sind Fehlerkennzahlen zur Bestimmung der Genauigkeit

von allen Programmen lag bei 31,57%. [12]

Haojin Yang et al. [11] haben in ihrer Untersuchung "German Speech Recognition: A Solution for the Analysis and Processing of Lecture Recordings" aus dem Jahr 2011 verschiedene Spracherkennungssysteme für das Erstellen von Transkripten miteinander verglichen. Dabei wurden fünf verschiedene lokal laufende Programme in der deutschen Sprache getestet. In der Untersuchung hat das Programm Julius den höchsten Score vor Sphinx4 erreicht. [11]

In der folgenden Evaluation der Programme werden die genannten Ergebnisse überprüft.

## 2.3 SPRACHTYPEN

Es gibt vier Kategorien, in der unser Sprachverhalten eingeordnet werden kann. Die Spracherkennungsprogramme bauen auf den entsprechenden Eigenschaften der Sprachtypen auf. Dabei steigt die Komplexität des Gesagten mit jeder Kategorie an.

### 2.3.1 *Isolierte Sprache*

Bei einem Programm zur Spracherkennung, welches isolierte Worte verarbeiten kann, muss das Wort mit einer Pause von folgenden Wörtern abgegrenzt werden. Zu der isolierten Sprache gehören nicht nur einzelne Wörter sondern auch Äußerungen oder Phrasen. Die Systeme sind auf ein Wort oder ein Kommando ausgelegt. Dadurch ergeben sich die zwei Zustände "Hören" und nicht "Hören" im System, sodass eine Pause zwischen den Äußerungen erzwungen wird. [21]

### 2.3.2 *Verbundene Worte*

Bei verbundenen Wörtern liegt der Unterschied zur isolierten Sprache in der geringeren Pausenzeit zwischen den Äußerungen. Separate Äußerungen wie verbundene Äußerungen können zusammen verarbeitet und wahrgenommen werden. [21]

### 2.3.3 *Durchgängige Sprache*

Durchgängige Spracherkennung kann geübtes Gesagtes oder abgelesenes Sprechen erkennen. Dabei muss das System die einzelnen Wörter voneinander unterscheiden können, ohne dass die sprechende Person eine Pause oder eine andere akustische Trennung macht. Die Systeme sind komplexer im Vergleich zur isolierten oder verbundenen Wörtern. [21]



### 2.3.4 Spontane Sprache

Spontane Sprache klingt natürlicher und wurde vor dem Sprechen nicht geübt. Dadurch beinhaltet sie Lückenfüller wie "uhm" oder "ähm", die zur Bedeutung des Satzes nicht beitragen. Diese zu erkennen und herauszufiltern ist eine Herausforderung. Außerdem muss das System mit den Merkmalen der spontanen Sprache wie zusammengezogener Wörter, falschem Satzstart, nicht perfekte Sätze, Husten oder Lachen umgehen. [21]

## 2.4 GESCHICHTE

In dem Buch "Speech Technology" stellen Chen und Jokinen in ihrem ersten Kapitel die bisherige Geschichte der Spracherkennung vor. Dabei unterteilen sie die Geschichte in folgende Kategorien:

- Erste Generation: erste Versuche in den 1950er und 1960er
- Zweite Generation: Vorlagen-basierte Technologie in den späten 1960er und 1970er
- Dritte Generation: statistische Modelle-basierte Technologie in den 1980er
- Späte dritte Generation: Fortschritte in der dritten Generation in den 1990er und 2000er [10]

### 2.4.1 Erste Generation

In der ersten Generation werden die Merkmale der akustischen Phonetik<sup>3</sup> verwendet, um Wörter in einer Aufzeichnung zu erkennen. Trotz eingeschränkter Leistungsfähigkeit der Systeme, versuchten Forschende die Schwingungen beim Sprechen auszunutzen. Mit Hilfe von analogen Filtern und logischen Schaltungen wurde versucht, die Schwingungen zu erkennen, die während des Sprechens in der Vokalregion entstehen.

Zu dem ersten System zählt ein Zahlenerkennungssystem aus dem Bell Labor aus dem Jahr 1952. Dieses war auf eine einzelne sprechende Person und isolierte Zahlen ausgelegt. Die Zahlen lassen sich durch unterschiedliche Frequenzen aus der Vokalregionen unterscheiden. Mit der Entwicklung einer Vorgehensweise zur Erkennung von Sprachpausen, konnten konstantere Ergebnisse gemessen werden. Die Länge der Sprachpause ist sehr variabel und wird in der Wissenschaft "Time normalization" oder Zeitnormalisierung genannt.[10]

### 2.4.2 Zweite Generation

Um Sprachsignale effektiver miteinander anzugleichen, wurde Ende der 60er Jahre das Verwenden von dynamischen Programmiermethoden zur Dis-

---

<sup>3</sup> Phonetik ist die Lehre der Laute

kussion gestellt. Der Algorithmus wird *dynamische Zeitnormierung* oder "dynamic time warping" genannt.

Das Verwenden von Algorithmen der Vorlagen-basierten Erkennung an isolierten Sätzen wurde durch Grundlagenforschung in Japan und der ehemaligen Sowjet-Union ermöglicht.

An der Carnegie Mellon Universität wurde von Forschenden ein System zur Erkennung von durchgängiger Sprache als Eingangssignal entwickelt. Dabei werden die Phoneme <sup>4</sup> des Gesagten automatisch vom System erkannt.

Außerdem wurde an ersten Systemen zur sprecherunabhängigen Erkennung geforscht. Dabei musste eine Auswahl an Vorlagen ausgewählt werden, die für möglichst viele Menschen zutrifft. Um diese Auswahl zutreffen, wurden verschiedenen Algorithmen zur Erkennung von Mustern ausgewählt.[10]

### 2.4.3 Dritte Generation

Die Verarbeitung von zusammenhängenden gesprochenen Äußerungen zu verbessern, war das Ziel der Wissenschaft der 80er Jahre.

Der nächste Schritt in der Entwicklung der Spracherkennungssysteme war die Implementierung statistischer Modelle als Algorithmen, welches die Vorlagen-basierende Modell ablösen. Statistische Modelle werden immer noch in Spracherkennungssystemen eingesetzt.

Mit dem Hidden-Markov-Modell wurde ein Gerüst entwickelt, in dem versucht wird, ein zukünftig eintretendes Ereignis eine Wahrscheinlichkeit zu geben, dabei ist aber die innere Zustand des Modells unbekannt.

Außerdem wurden Methoden zur statistischen Analyse der Wahrscheinlichkeit des Auftretens von Wörtern in einer spezifischen Reihenfolge entwickelt. Bei der Methode N-Gramme<sup>5</sup> wird das Gesagte in seine Wörter unterteilt.

Neuronale Netzen wurden durch Verbesserungen von Computern wieder als möglicher Algorithmus für die Spracherkennung vorgeschlagen, nachdem auch der Zusammenhang mit der Vorlagen-basierten Algorithmen besser verstanden wurde. [10]

### 2.4.4 Späte dritte Generation 1990er

In den vorherigen Algorithmen wurden alle Kombinationen von Wörtern für das beste Ergebnis ausprobiert. Durch die gestiegene Anzahl an Wörtern, die getestet werden müssen, ist es aus Kostengründen nicht mehr möglich, alle Wörter zu überprüfen. Die angepassten Algorithmen basieren nicht mehr auf den Satz von Bayes<sup>6</sup> beziehungsweise die Bayes Entscheidungstheorie. Die Entscheidung wird in 2.5 genauer beschrieben. Aus diesem Vorgehen sind Konzepte wie entscheidungsbasierendes Lernen und Kernel-basierende Methoden entstanden.

4 Laut Uni Graz[37] sind Phoneme die kleinsten bedeutungsunterscheidenden Einheiten einer Sprache

5 Reihe von Wörter aus einem Text [36]

6 ermöglicht die bedingte Wahrscheinlichkeit zu errechnen [41]

Außerdem wurde zu möglichen Vorgehen bei der Fehlerverminderung in Systemen, die von einem falschen Verhältnis von Trainings zu Test Bedingungen kommen können, geforscht.[10] Erklärt in 2.8

#### 2.4.5 Späte dritte Generation 2000er

Aktuelle Fortschritte erfolgten in den Gebieten: spontane Sprache, multimodale Erkennung und robustere Spracherkennung. Für die Verbesserung von der Erkennung spontaner Sprache wurden Probleme bei der akustischen Modellierung, Erkennung von Sprachpausen und Akzenten gelöst. Außerdem wurde an den Sprachmodellen und der Zusammenfassung des Gesagten geforscht.

Mit visuellen Informationen lässt sich die Genauigkeit der Systeme verbessern, da eine größere Auswahl an Daten für die Auswertung verfügbar ist. Dies ist hilfreich in Situation, bei der eine komplexe Nachricht übermittelt wird oder die Störgeräusche die Sprachqualität stark beeinträchtigen. Weitere Informationen lassen sich aus dem Gesicht und über die Lippen erkennen. Um die Systeme noch genauer zu gestalten, sollen die Systeme sagen können, wie sicher sie bei der Auswertung des Gesagten sind und ob sie das Gesagte richtig aufgenommen haben. Außerdem wird versucht, wichtige von unwichtigen Informationen einer spontanen Sprachaufnahme zu unterscheiden. Damit nur die wichtigen Stellen verwendet werden.[10]

### 2.5 WAHRSCHEINLICHKEITSTHEORIE DER SPRACHERKENNUNG

Die Spracherkennungssoftware versucht die am besten passende diskrete Reihe von Wörtern aus der Sprache  $L$  zu finden. Dabei steht dem Programm als Eingangswert die Reihe  $O$  zur Verfügung. Das Eingangssignal wird folgenderweise definiert:

$$O = O_1, O_2, O_3, \dots, O_t \quad (2.1)$$

Das Resultat kann als folgende Reihe von Wörtern fest gelegt werden:

$$W = W_1, W_2, W_3, \dots, W_n \quad (2.2)$$

Die Vorgehensweise zum Finden der passenden Wörter zu dem Eingangssignal wird definiert als:

$$\hat{W} = \operatorname{argmax} P(W|O) \quad \forall W \in L \quad (2.3)$$

Die Wahrscheinlichkeit der Übereinstimmung von einer Reihe von Wörtern und dem Eingangssignal lässt sich mit der gegebenen Formel berechnen. Die Wahrscheinlichkeit  $P(W|O)$  kann nicht direkt errechnet werden, weil dafür alle möglichen Wortkombinationen betrachtet werden müssten. Deswegen wird mit Hilfe des Satzes von Bayes die Formel umgewandelt:

$$P(W|O) = \frac{P(O|W)P(W)}{P(O)} \quad (2.4)$$

$P(W)$  ist die Wahrscheinlichkeit, wie häufig die Wortreihenfolge bereits aufgetreten ist, wofür man das Wissen von vorherigen Erkennungen benötigt.  $P(O)$  kann für die Max-Entscheidung weggelassen werden, da  $P(O)$  sich für alle Reihen von Wörtern nicht ändert, folgt daraus:

$$\hat{W} = \operatorname{argmax} \frac{P(O|W)P(W)}{P(O)} = \operatorname{argmax} P(O|W)P(W) \quad \forall W \in L \quad (2.5)$$

Die Beobachtungshäufigkeit oder akustischer Score wird die Häufigkeit  $P(O|W)$  genannt, welche das Verhältnis des Eingangssignales anhand einer Reihe vorgeschlagen Wörtern  $W$  bestimmt wird. [23]

## 2.6 AUFBAU DER SPRACHERKENNUNG

Für die Erklärung des Aufbaus wird ein Überblick über die benötigten Module gegeben. Danach folgen die Erläuterungen zu den Modulen.

### 2.6.1 Einleitung

Die Einleitung gibt einen Überblick auf den Aufbau eines Spracherkennungssystems. Dabei wird die Struktur anhand von zwei Schaubildern erläutert, welche sich in ihrem Detaillierungsgrad unterscheiden. Das Bild 2.1 zeigt den kompletten Prozess einer Spracherkennung von der Entstehung der Aufnahme bis zu der Ausgabe der Wörter. Die Darstellung 2.2 veranschaulicht den inneren Aufbau der Spracherkennung und unterteilt diese in verschiedene Module mit spezifischen Aufgaben.

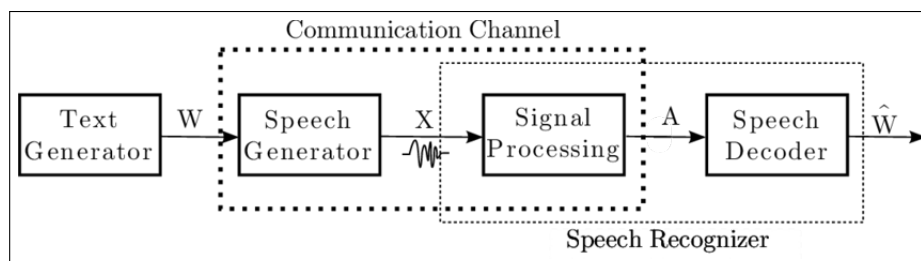


Abbildung 2.1: Ablauf der Spracherkennung [6]

Auf dem Schaubild 2.1 ist zu erkennen, dass in dem Ablauf der Erkennung zwei Bereiche markiert sind.

Im Kommunikationskanal findet die Übertragung der Äußerungen von dem Menschen zu der Spracherkennungssoftware statt. Dabei nimmt ein Mikrofon die analogen Signale auf und wandelt diese in ein digitales Format um. Diese werden über ein Medium zum Beispiel ein Kabel zu dem Computer geschickt und von der Software verarbeitet. In dem Kommunikationskanal kann die Qualität der Aufnahme durch Störsignale verringert werden. Außerdem werden die Signale mit der Länge der Übertragung schwächer, womit diese schwerer zu verarbeiten sind.

In dem Bereich Spracherkennung zeigt das Schaubild 2.1 die Module Signalverarbeitung und Sprachumwandler. Diese beiden Module sind für die Spracherkennung essentiell. Bei der Signalverarbeitung werden die ankommenden digitalen Signale für die Weiterverarbeitung aufbereitet. Der Sprachumwandler erkennt aus den aufbereiteten Daten die Wörter und gibt diese als Antwort mit den richtigen Satzzeichen aus.

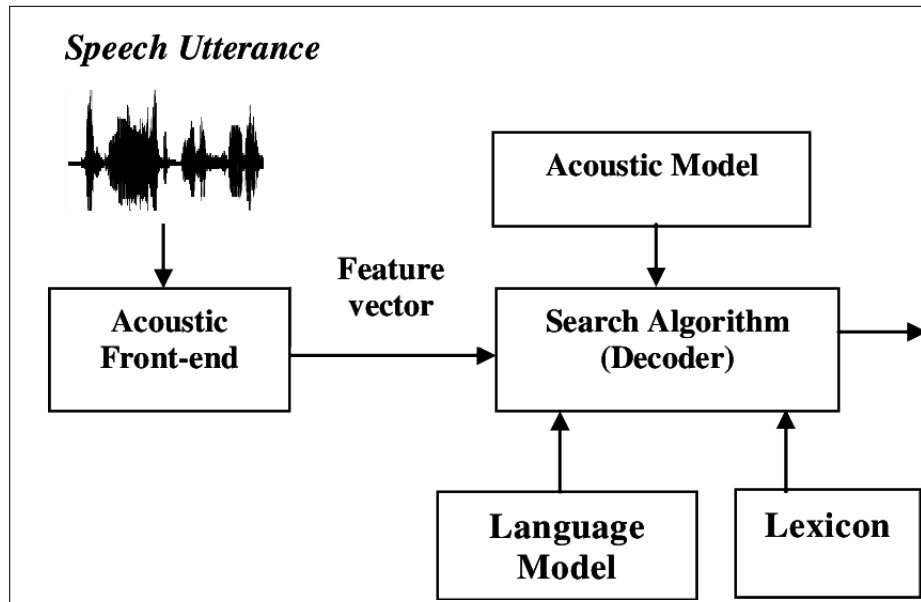


Abbildung 2.2: Aufbau der Spracherkennung [8]

In der Grafik 2.2 sind die Module der Spracherkennung abgebildet. Es zeigt den Weg des Signales innerhalb eines Programms und welche Module in welcher Reihenfolge verwendet werden. Wie in dem Abbild 2.1 ist in der Darstellung 2.2 das System in zwei Hauptbereiche unterteilt. Das akustische Front-End ist mit der Signalverarbeitung aus Schaubild 2.1 gleich zusetzen. Der Decoder beziehungsweise Suchalgorithmus ist mit dem Sprachumwandler aus der Grafik 2.1 vergleichbar. Der Suchalgorithmus wird durch ein akustisches Modell, Sprach-Modell und ein Lexikon erweitert. Die Darstellung 2.2 dient als weitere Vorlage, um den Aufbau der Spracherkennung zu erklären.

### 2.6.2 Akustisches Front-end

Das akustische Front-end ist der erste Schritt in der Erkennung von Audiosignalen. Dabei werden in dem Abschnitt die benötigten Merkmale aus der Audiospur ausgelesen und in ein weiterverarbeitendes Format gebracht. Das Auslesen der Merkmale wird in drei Schritte unterteilt:

1. Das Sprachsignal wird in kurze Abschnitte unterteilt, aus denen die groben Merkmale der Höhen und Tiefen des Gesagten ausgelesen werden. Dieser Prozess wird als Sprachanalyse oder akustisches Front-end bezeichnet.

2. Die Ergebnisse der Sprachanalyse werden mit dynamischen und statischen Merkmalen in einen neuen Vektor geschrieben.
3. Die Größe des Vektors wird für eine bessere Weiterverarbeitung komprimiert.

In der Wissenschaft gibt es keinen eindeutigen Grundsatz, wie ein "Merkmal-Set" aufgebaut sein muss. Stattdessen hat man sich auf drei Richtlinien verständigt.

1. Das Programm kann gleich klingende Wörter differenzieren.
2. Das System legt neue akustische Modelle für unbekannte Äußerungen an, ohne dafür Stunden zu trainieren.
3. Es muss in der Lage sein, Statistiken über akustische Modelle zu erstellen, bei denen sprechende Personen und Umgebungen kaum einen Einfluss auf die Modelle haben.

Um aussagekräftige Ergebnisse aus dem Gesagten zuziehen, vereinfacht das Programm die Daten in die wichtigsten Parameter und Eigenschaften. Anhand der komprimierten Parameter werden die Abschnitte des Audiosignals nach ihren Eigenschaften sortiert. Diese Gruppierung wird die "feature extraction method" genannt.

Eine der Methoden um die Eigenschaften darzustellen, ist der "mel-frequency cepstral coefficient". [23]

Der Koeffizient basiert auf der Fourier Analyse und besteht aus folgenden Stufen:

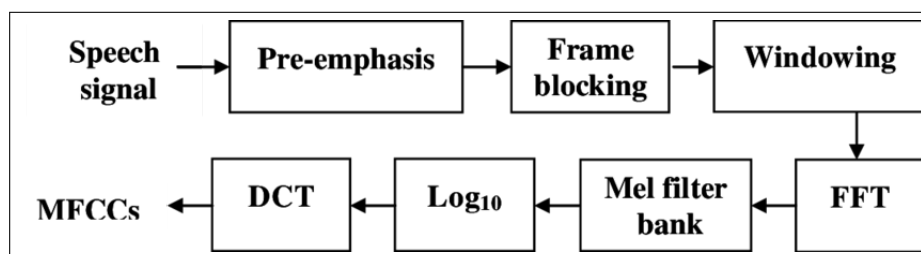


Abbildung 2.3: Extrahierung der Eigenschaften [9]

Pre-emphasis oder Vorverzerrung:

In dem Block werden die ankommenden Sprachsignale gefiltert, sodass die höheren Frequenzen stärker gewichtet werden. Dadurch werden die höheren Signale mit einer kürzeren Amplitude an die normale Amplitude angepasst, um die Erkennbarkeit zu verbessern. [23]

Framing & Windowing:

Bei Framing wird das ankommende Sprachsignal in kleinere Abschnitte mit gleichen Abstand zerteilt. So entstehen Abschnitte, die kaum statistisch schwanken. Um negativen Effekte an den Schnittstellen beziehungsweise Kanten auszuschließen, wird die "Hamming window" Funktion verwendet. [18] [23]

Diskrete Fourier Transformation:

Die DFT wird für jedes Teilstück berechnet. Es resultieren die Größenordnung und Phase des Gesagten. Es wurde bewiesen, dass die Informationen über die Phasen nicht für die Erkennung benötigt werden. [18] [23]

Mel<sup>7</sup>-Filterreihe:

Die Menschen können Frequenzen über 1000 Hertz schlechter wahrnehmen. Deswegen wird das Ergebnis der diskreten Fourier Transformation auf jeder Frequenzlage gleich verteilt. Daraus folgend müssen die Frequenzen über 1000 Hertz mit einer Logarithmischen-Mel-Skala verringert werden. Über sogenannte Dreiecksfilter werden die Frequenzen über 1000 Hertz logarithmisch gestaucht und die Frequenzen unter 1000 beständig verteilt. Als Mel-Spektrum wird das Resultat des Bearbeitungsprozess bezeichnet. [17] [18] [23]

Logarithmus:

Der Mel-Spektrum-Koeffizient ergibt sich, wenn von dem Mel-Spektrum der Logarithmus genommen wird. [23]

Diskrete Kosinus-Transformation:

Wenn die diskrete Kosinus-Transformation auf den Mel-Spektrum-Koeffizienten angewendet wird, folgt als Resultat die MFCC oder auch "Mel Frequency Cepstral Coefficient" genannt. [23]

MFCC Eigenschaften:

Durch die erste und zweite Ableitungen werden aktive Wechsel des Signals bestimmt. Die resultierenden Koeffizienten heißen Delta und Doppelt Delta. [18] [23]

### 2.6.3 Akustische Modell

Das akustische Modell hat die Aufgabe, die aus dem Front-end generierten Daten, die Eigenschaften und Merkmale herauszulesen.

Die Eigenschaften und Merkmale werden statistisch berechnet und in einer Reihe von Vektoren gespeichert. Die Entscheidung für die fundamentale Einheit des Modells ist eine Herausforderung. Anhand der zu erkennenden Sprache kann festgelegt werden, welche der unterschiedlichen Unterworteinheiten für dieses Modell ausgewählt wird.

Um die Unterworteinheiten auseinander halten zu können, muss für jedes eine statistische Darstellung in dem Modell abgespeichert werden. Die Darstellungen werden mit einem Label beschriftet, um diese schneller zu identifizieren. Die Unterworteinheiten werden später wieder zu einem Wort zusammengesetzt. Mit Hilfe einer Datenbank, in der zahlreiche Laute einer Sprache gespeichert sind, und einem Algorithmus, der auf dieses Szenario spezialisiert ist, wird das akustische Modell aufgebaut. [23]

Außerdem werden in den akustischen Modellen auch "Aussprache Modelle" verwendet. Dieses Modell hilft bei der Darstellung von Elementen wie Wörter aus Unterworteinheiten in einem Satz. Durch Information vom Decoder über Vektoren kann das akustische Modell diese anpassen, um eine bessere

<sup>7</sup> definiert die psychoakustische Wahrnehmung der Tonhöhe [34]



Erkennung von Wörtern in verschiedenen Umgebungen zu gewährleisten. Die Prozesse aus dem akustischen Modell berufen sich auf die theoretische Wahrscheinlichkeit, beschrieben in 2.5. Die erkannten Laute müssen mit einem Vergleichsobjekt aus dem Modell verglichen und auf Übereinstimmung überprüft werden. Dafür werden die Wörter in ihre Einzelteile aufgeteilt. [55]

Für den Aufbau eines akustischen Modells gibt es unterschiedliche Modelle. Zu diesen zählen das Hidden-Markov-Modell, das Segmental Modell, das Super Segmental Modell oder Neuronale Netze.

Bei den Hidden-Markov-Modell bekommt jedes Laut einer Sprache ein Hidden-Markov-Modell zugewiesen. In dem Prozess des Erkennens werden in dem akustischen Modell die Hidden-Markov-Modelle mit der Äußerung abgeglichen. [23]

#### 2.6.4 Sprach-Modell

Der Mensch hat einen Vorteil gegenüber Spracherkennungssystemen. Er weiß mit dem gegebenen Kontext, wie wahrscheinlich es ist, dass als nächstes ein gewisses Wort oder Phrase gesagt wird.

Die Aufgabe des Sprach-Modells ist das Zurverfügungstellen des Kontextes. In dem Sprach-Modell wird fest gelegt, wie die Wörter in einer gewissen Sprache aussehen und in welchem Ablauf eines Satzes diese auftreten. Dazu zählen auch Wörter, die ein vergleichbares Klangbild besitzen.

Außerdem versucht der Algorithmus, wie der Mensch ein Wort vorherzusagen, welches Wort als nächsten gesagt wird. Dabei schaut der Algorithmus vor allem auf erwähnte Wörter und kann bei Veränderungen auch das Sprach-Modell anpassen. Dieses Wissen und Vorgehen beeinflusst die Präzision des Systems auf eine positive Weise. [23]

Das Sprach-Modell liefert das Ergebnis der Wahrscheinlichkeitsrechnung  $P(W)$  aus der Gleichung 2.5. Eine Unteraufgabe des Sprach-Modells ist die Bewertung der Grammatik eines Satzes. Dabei geht es um die Bewertung der Richtigkeit einer Sequenz von Wörtern in einer gewissen Sprache.

Die Methoden zur Bestimmung der Grammatik lassen sich in deterministisches Vorgehen und statistische Berechnung der Wahrscheinlichkeit von aufeinander folgenden Worten unterteilen.

Bei der deterministische Grammatik wird der Satzbau anhand der vorgegebenen grammatischen Regeln untersucht und entweder als korrekt oder fehlerhaft eingestuft.

Bei der statistischen Berechnung wird mit einer großen Datenmenge von korrekten Sätzen die grammatischen Regeln abgeleitet und verallgemeinert. So wird bei einer späteren Auswertung die Richtigkeit der statistische Aussagen garantiert. Bei dieser Art des Erlernens von den Regeln ist die Vorgabe der korrekten grammatikalischen Regeln mit ihren Ausnahmen nicht notwendig. [55]

Für die statistische Auswertung der Sprache wird häufig das N-Gramm Modell verwendet.



Bei der N-Gram Methode wird die Häufigkeit von Wortsequenzen in einer Datenmenge gezählt. Dabei wird versucht die Perplexität von Wörtern zu verringern. Perplexität bedeutet, dass es Wörter mit mehreren Bedeutungen gibt und somit in verschiedenen Kontexten auftreten wird. Aus den unterschiedlichen Häufigkeiten der Wortsequenzen werden die Wahrscheinlichkeiten für die N-Gramme bestimmt. In den Sprach-Modellen werden Bi- oder Trigramm Methoden verwendet. Mit dieser Methoden wird die Wahrscheinlichkeit von zwei oder drei aufeinander folgenden Wörtern bestimmt. Ein Beispiel für ein Bi-Gramm ist (Meine, Arbeit) und für ein Tri-Gramm (Meine, Arbeit, macht). [23]

### 2.6.5 Decoder

Das Erkennungsmodul vergleicht das eingegangene Signal mit den ermittelten Worten. Dabei wird es von dem akustischen- und Sprach-Modell unterstützt.

Dabei werden nicht alle Pfade berechnet, sondern Pfade mit einer großen Übereinstimmung zu dem Eingangssignal für die Erkennung verwendet. [23]

Der Decoder bestimmt eine Reihe von Wörter, die die größte Wahrscheinlichkeit zu  $P(W|O)$  und dem Eingangssignal haben.

Die Algorithmen sind abhängig von der Größe des Wortschatzes und den Regeln des Sprach-Modells der verwendeten Sprache. Umso größer der Wortschatz und umso mehr Regeln vorhanden sind, desto größer ist der Einfluss auf den Algorithmus und desto aufwendiger ist die Suche. Wenn der Wortschatz eine Größe erreicht hat, dann ist es nicht mehr möglich beziehungsweise zu kostenintensiv alle Pfade zu berechnen. Die Suchalgorithmen müssen flexibel mit Situation wie unbekannte Anzahl an Wörtern oder ab wann beginnt und endet ein Wort in einer Aufnahme umgehen können. Diese Eigenschaften stehen am Anfang der Suche nicht fest.

Zwei Algorithmen für die Pfadsuche sind A\* Stapel und Viterbi.

Der A\* Stapel Algorithmus hat seine Stärken bei der Suche in Datenmengen, bei denen das A\* Kriterium häufig verwendet werden kann.

Der Viterbi Algorithmus durchsucht nach dem richtigen Wort die Pfade. [55]

Um die Problematik der Durchsuchung bei großen Datenmengen zu umgehen, wurde der Algorithmus mit einer "beam" Suche erweitert. Dabei werden die Pfade, die niedriger bewertet als ein vorgegebener Wert sind, nicht mehr betrachtet. In der nächsten Stufe werden nur die Pfade weiter durchsucht, die über den Grenzwert liegen. Bei der Variante werden die Pfade nicht berücksichtigt, die in der vorherigen Stufe aussortiert wurden, aber am Ende der Suche die höchste Übereinstimmung mit dem Eingangssignal haben. Für diesen Fall wurden die beiden Methoden "extending Viterbi" und "vorwärts-rückwärts" geschrieben. [23]

## 2.7 ALGORITHMEN FÜR DIE SPRACHERKENNUNG

Der Algorithmus bei einer Spracherkennung ist essenziell für die Genauigkeit und Geschwindigkeit. Die Algorithmen lassen sich in die Kategorien Akustisch-phonetisch, Muster-Erkennung und künstliche Intelligenz unterteilen.

### 2.7.1 Akustisch-phonetisch

Die akustisch-phonetische Vorgehensweise beruht auf dem Finden von Spracheigenschaften in dem Eingangssignal. Voraussetzung dafür ist, dass die erkennende Sprache eine endliche und fixe Menge an Lauten hat.

Die akustischen Eigenschaften lassen sich in die Gruppen binär und durchgängig unterteilen. Zu den binären Eigenschaften zählen Nasalisierung<sup>8</sup> oder Frikativ<sup>9</sup>. Formant<sup>10</sup>-Positionen und unterschiedliche Frequenzen sind Eigenschaften der durchgängigen akustischen Eigenschaften. [1] [54]

Die Vorgehensweise des Algorithmus lässt sich in folgende Schritte unterteilen.

1. Mit der Spektralanalyse und Merkmalerkennung werden die spektralen Eigenschaften der zuerkennenden Sprache ausgelesen. Diese Eigenschaften geben die Merkmale der unterschiedlichen phonetischen Einheiten wieder.
2. Das Sprachsignal wird in stabile akustische Bereiche getrennt und entsprechend den Bereichen mit Markierungen versehen. Das Ergebnis des Vorgehens wird "phoneme lattice characterization" genannt.
3. Mit den angebrachten Label wird versucht, ein passendes Wort oder Wortsequenz zu finden. [23]

Diese Vorgehensweise hat keinen weitreichenden kommerziellen Erfolg. [1]

### 2.7.2 Muster-Erkennung

Der Mustervergleich beziehungsweise "Pattern-matching" besteht aus folgenden Stufen:

In der ersten Stufe werden die Muster mit einem Algorithmus und Daten kreiert. Die zweite Stufe vergleicht das eingehende Sprachsignal mit den vorhanden Muster auf Übereinstimmung.

Bei der Vorgehensweise gibt es mathematische Strukturen, um die Muster aus präparierten Daten mit einem Trainingsalgorithmus zu erstellen. Das Spracherkennungsprogramm kann auf verschiedene Längen von Wörtern ausgerichtet werden. Die Länge der Wörter reicht von einer Silbe bis zu einer Phrase.

<sup>8</sup> Laute, die durch die Aussprache von den Mund und Nase gleichzeitig passieren [52]

<sup>9</sup> Wörter, die durch die Verwendung von Lippen, Zähnen oder Gaumen entstehen [50]

<sup>10</sup> typischer Zwischenton eines Lautes [28]

Die Muster-Erkennung kann in die beiden Kategorien Vorlagen-basiert und stochastisch-basiert aufgeteilt werden. [1] [23]

#### 2.7.2.1 Vorlagen-basiert

Die Eigenschaften der möglichen Kandidaten werden bei dem Vorlagen-basierten Algorithmus in Referenzmuster eingeteilt. Diese bestehen aus einer Ansammlung von Sprachmustern. Während des Erkennungsprozesses wird das Eingangssignal mit jedem Referenzmuster verglichen und die Referenz mit der höchsten Übereinstimmung wird ausgewählt. Für Vergleichbarkeit und Genauigkeit muss das System Referenzmuster von den unterschiedlichsten Wörtern bilden. [1] [54]

Der Algorithmus profitiert von sehr akkuraten Wortmodellen. Der Nachteil an dem Vorgehen ist, dass die Modelle vorgegeben sind. Außerdem müssen neue Modelle hinzugefügt werden, wenn Abweichungen von den eigenen Modellen und dem Gesagten registriert werden. Durch den erhöhten Aufwand im Training und geringer Geschwindigkeit wegen mehr Vergleichen gibt es ein Limit an Wörtern, die aufgenommen werden können. Durch die Einsortierung von Wörtern in Referenzen können Fehler verhindert werden. Kleinere Worte sind akustisch variabler und die fehlerhafte Aufteilung von längeren Wörtern wird vermieden. [3]

Für die Auswahl des Musters werden zwei Methoden angewendet.

Die eine ist die Berechnungen des Mittelwerts und die andere ist die Messung des lokalen spektralen Abstands.

Mit der dynamischen Programmierung lassen sich die Muster überlappen. Damit werden Unterschiede in der Sprachgeschwindigkeit und Wortwiederholungen einer Person ausgeschlossen. [1] [54]

#### 2.7.2.2 Stochastik-basiert

Stochastische Vergleichsmodelle nutzen wahrscheinlichkeitstheoretische Methoden zum kompensieren von fehlerhaften Daten. Die Gründe für fehlerhaften Daten werden in Kapitel 2.8 beschrieben.

Um mit fehlerhaften Daten besser umgehen zu können, verwendet der Algorithmus vorhersagende Modelle. Die Vorhersage erhöht die Genauigkeit des Algorithmus.

Durch seinen Aufbau ist das Hinzufügen neuer Trainingsdaten für den Algorithmus kein Problem.

Ein statistisches Vorgehen ist das Hidden-Markov-Modell. Dieses basiert auf einem Markov-Modell mit endlichen Zuständen und verwendet eine Reihe von Ausgabeverteilungen. Das Hidden-Markov-Modell besteht aus zwei verschiedenen Schwankungen, welche die Spracherkennungsfunktion ausmachen. Die temporäre Schwankung nutzt die Überleitungseinschränkungen aus dem Markov-Ketten-Modell. Die Einschränkungen aus der Ausgabeverteilungsfunktion werden für die spektrale Schwankung verwendet.

Die stochastischen Algorithmen sind mathematischer und umfangreicher als Muster-basierte Vorgehensweisen. Außerdem versuchen statistische Mo-

delle Wörter vorherzusehen, dass zu einer höheren Genauigkeit gegenüber Muster-basierten Algorithmen führt. [1] [54]

### 2.7.3 *Artificial intelligence*

Bei der künstlichen Intelligenz soll das Verhalten der Menschen kopiert werden. Für fehlerfreie und akkurate Spracherkennung sind Methoden basierend auf der künstlichen Intelligenz die effektivsten. Dabei wird die Sprache konvertiert und transformiert und in Muster abgespeichert. Dieses Vorgehen wird in beide Richtungen vollführt.

Die Methoden der akustischen-Phonetik und Mustervergleichs werden zu einem Vorgehen kombiniert. Deswegen wird diese Methode auch als hybrid bezeichnet. Ein weiterer Name für dieses Vorgehen ist Wissens-basiert. In den Algorithmen werden Informationen aus den Bereichen Spektrogramme<sup>11</sup>, Phonetik oder Linguistik eingesetzt.

Außerdem werden die Methoden der künstlichen Intelligenz für das Design neuer Algorithmen, Vorstellung von Spracheinheiten und Darstellungen von richtigen und angebrachten Eingaben verwendet.

Das Ausdrücken von erreichten Erkenntnissen ist schlechter als bei anderen Methodiken. [3]

#### 2.7.3.1 *Neuronale Netzwerke*

Neuronalen Netzwerke gehört zu dem Bereich der künstlichen Intelligenz. Das Kopieren des menschlichen Gehirns ist die Idee hinter Neuronalen Netzwerken. Joe Tebelskis [16] zählt als Eigenschaften der Neuronalen Netzwerke folgendes auf: "Trainierbarkeit, Verallgemeinerung, Nichtlinearität, Robustheit, Gleichmäßigkeit und Parallelität".

Die einzelnen Neuronen eines Netzwerkes übernehmen die Verarbeitung von Daten. Die Neuronen basieren auf einer vereinfachten Struktur, wie sie im Gehirn vorzufinden sind. Ein Neuronales Netz lernt, indem es die Gewichtung, ab wann ein Neuron auslöst, verändert.

Die Ein- und Ausgabeschichten sind bei den Netzwerken bekannt. Die Neuronen in den mittleren Schichten sind nicht bekannt beziehungsweise "hidden".

Zu den elementaren Einheiten, die in jedem Netz vorhanden sind, zählen verarbeitende Einheiten, Verbindungen und rechnende und trainierende Prozedere.

Es werden hauptsächlich drei Lernverfahren angewendet: überwachtes, unüberwachtes und bestärktes Lernen. [16]

Die Neuronalen Netzwerke sind effektiv im Bestimmen von kurzen Spracheinheiten wie Phoneme oder isolierter Wörter. Das Erkennen von durchgängiger Sprache ist ein Problem, da sie temporäre Abhängigkeiten nicht darstellen können. [23]

---

<sup>11</sup> graphische Darstellung von Frequenzen [42]

### 2.7.3.2 *Deep Learning*

Laut IBM [24] ist der Unterschied zwischen einem Deep Learning und einem Neuronalen Netzwerk die Anzahl an Schichten in einem Modell. Ein Deep Learning Algorithmus ist jedes Neuronales Netzwerk, das mehr als drei Schichten besitzt.

Deep Learning wird zum führenden Algorithmus in der Spracherkennung. Die Methodik lässt sich in folgende Architekturen unterteilen.

Die erste Architektur nennt sich "generative deep architecture". Damit werden mit dem Daten die "high-order" Korrelations-Charakteristiken erstellt oder es wird von visuellen Daten die verbundene statistische Verteilung mit den dazugehörigen Gruppen bestimmt. Diese Architektur kann mit dem Satz von Bayes in eine differenzierende Methodik umgewandelt werden.

Differenzierend wird die zweite Architektur genannt, weil dort bei der Zuordnung von Mustern differenzierende Entscheidungen getroffen werden. Außerdem werden anhand der visuellen Daten mit einer späteren Verteilung die Klassenlabel charakterisiert.

In der dritten Architektur wird der erste und zweite Ansatz kombiniert. Diese Vorgehensweise wird "hybrid" genannt. Dabei wird versucht, die Differenzierung mit den Resultaten aus der Kategorie zwei zu verbessern. Die erzeugenden Modelle werden für die Verbesserung der Differenzierung verwendet. [23]

## 2.8 HERAUSFORDERUNGEN BEI DEM DESIGN DER SPRACHERKENNUNG

In dem Abschnitt Herausforderungen wird auf Themen eingegangen, die das Ergebnis der Spracherkennung beeinflussen. Dabei wird auf die Herausforderungen des Erkennens der Wörter eingegangen. Außerdem werden die Probleme bei dem Anbieten von unterschiedlichen Sprachen in der Spracherkennung beschrieben.

### 2.8.1 *Herausforderungen beim Erkennen der Wörter*

Die Spracherkennung muss mit den folgenden Herausforderungen die Wörter erkennen. Die Probleme reichen von Doppeldeutigkeit bis zur Aufnahmequalität.

#### 2.8.1.1 *Ausdruck von Wörtern*

Bei den Ausdrücken von Wörtern kommt es vor, dass ein Wort verschiedene Bedeutungen hat. Dann muss das System entscheiden, in welchem Kontext das Wort verwendet wurde. Von dem Kontext abhängig entscheidet das System die Bedeutung. Außerdem gibt es akustisch gleich klingende Wörter mit unterschiedlichen Bedeutungen. Dann muss das System ebenfalls anhand von Merkmalen entscheiden, welches Wort gesagt wurde. [3] [21]

### 2.8.1.2 Abhängigkeiten von sprechenden Personen

Keine sprechende Person ist gleich. Sie unterscheiden sich in den Eigenschaften der Stimme. Daraus lassen sich die Spracherkennungssysteme in zwei Kategorien abhängig der Anzahl der trainierten Stimmen einordnen. Bei sprecherabhängigen Systemen wird ein Kompromiss eingegangen. Sie haben eine höhere Genauigkeit im Vergleich zu sprecherunabhängigen Systemen für eine bestimmte Person. Dafür sind sie weniger flexibel einsetzbar. Sprecherunabhängige Systeme erreichen die Unabhängigkeit von einer sprechenden Person mit einer Menge von Daten, die unterschiedlichen Eigenschaften haben. Dabei starten die Systeme mit einzelnen Sprechern. Durch die große Anzahl an Daten werden verschiedene Merkmale in der Sprache erkannt. Die Genauigkeit bleibt niedriger als bei sprecherabhängigen Systemen. Dafür haben die Systeme einen größeren Anwendungsbereich. [3] [21]

### 2.8.1.3 Wortschatz

Der Wortschatz ist die Ansammlung von Wörtern oder Äußerungen, mit welchen das System das Gesagte abgleicht.

Das Eingangssignal wird auf Kandidaten mit dem Wortschatz verglichen. Bei kleineren Wortschätzen lassen sich alle Kandidaten abgleichen. Mit steigender Zahl von Wörtern im Wortschatz steigt die Komplexität, sinkt die Genauigkeit und verändert die Voraussetzungen zur Ausführung. Dabei ist die Anzahl der Wörter für einen Eintrag nicht auf ein Wort beschränkt, sondern es kann auch ganze Phrasen oder Sätze enthalten. [21]

Ayushi Y. Vadwala et al unterteilen die Wortschätze in folgende Gruppen ein:

- **Kleiner Wortschatz** - Zehner von Wörtern
- **Mittlerer Wortschatz** - Hunderte von Wörtern
- **Großer Wortschatz** - Tausende von Wörtern
- **Richtig Großer Wortschatz** - Zehntausende von Wörtern
- **Außerhalb ein Wortschatzes** - Verknüpfungen eines Wortes von dem Wörterbuch zu einem unbekannten Wort. [3]

### 2.8.1.4 Aufnahmequalität

Der Ausschlag des Sprachsignals verringert sich, wenn im Hintergrund der Aufnahme Geräusche zuhören sind. Dabei hat die Dichte der Hintergrundgeräusche einen Einfluss. Wenn die Dichte des Gesagten geringer als die der Hintergrundgeräusche ist, dann können Informationen des Gesagten verloren gehen.

Die Hintergrundgeräusche haben einen negativen Einfluss auf die Spracheffizienz. Die sprechende Person muss bei einer Konversation mit Hintergrundgeräuschen sein Sprachverhalten anpassen, sodass der Partner in der Konversation das Gesagte versteht.

Die Qualität der Sprache geht zurück, wenn die sprechende Person sich neben dem Sprechen noch auf andere Aufgaben fokussieren muss. Der Einfluss verstärkt sich, wenn die Nebentätigkeit sich auf die Kapazitäten der Sauerstoffaufnahme auswirken.

Bei der Übertragung werden die Sprachssignale durch das Übertragungsmedium verringert. Diese Problematik ist nicht vorhersehbar und hat einen großen Einfluss auf die Genauigkeit des Programms.

Eine Veränderung des Abstandes von Mikrofon und sprechender Person hat negative Auswirkungen, weil Mikrophone die veränderten Sprachwellen anders wahrnehmen.

Außerdem entstehen Echos und diese können sich zu einem Nachhall steigern. Diese Signale interferieren mit den eigentlichen Sprachsignalen. Dann werden diese abgeschwächt und die Erkennung erschwert. [3]

### 2.8.2 Herausforderung für Systeme mit verschiedenen Sprachen

Die folgenden Herausforderungen zeigen die Unterschiede zwischen Sprachen auf. Diese müssen für eine hohe Genauigkeit in die Spracherkennung eingearbeitet werden.

#### 2.8.2.1 Skripte und Schriftarten

Die unterschiedlichen Schreibsysteme der Sprachen lassen sich in die Gruppen ideografisch und phonologisch einteilen.

Bei ideografischen Schriftzeichen stehen die Zeichen für eine Idee, Vorstellung oder Meinung und geben nicht die Aussprache vor. Ein Beispiel ist die ägyptische Hieroglyphe.

In Silben-basierte und alphabetische Schriftzeichen kann das phonologische Schreibsystem weiter unterschieden werden.

Jede Silbe steht bei der Silben-basierten Schriftart für ein Graphem<sup>12</sup>. Bei alphabetischen Schriftsystemen sind die Grapheme einzelne Laute. Sowohl die deutsche als auch die englische Schriftsprache lassen sich der alphabetischen Schriftweise zuordnen.

Für die Spracherkennung ist es leichter ein Aussprache-Wörterbuch für phonologische Schriftzeichen aufzubauen. In vielen Sprachen gibt es Regeln um die Grapheme in Phoneme umzuwandeln. Die Anzahl der Regeln unterscheidet sich von jeder Sprache und für diese Regeln gibt es Ausnahmen, die die Erstellung komplizierter machen. Diese Konvertierungsregeln sind für ideografische Schriftzeichen nicht anwendbar, da für diese Schriftzeichen keine Regeln existieren. [4]

#### 2.8.2.2 Romanisierung und Segmentierung

Die deutschen und englischen Wörter sind voneinander getrennt, sodass diese direkt als Einträge in das Wörterbuch verwendet werden. So müssen die

---

<sup>12</sup> verwendetes Schriftzeichen einer Sprache [30]



Wörter nicht weiter in ihre einzelnen Bestandteile aufgeteilt werden. Deutsche und englische Sprache hat kleinere Wörter, welche zur Speicherung in Wörterbüchern für die Spracherkennung genutzt werden. Die Wörter unterscheiden sich in der Aussprache um nicht verwechselt zu werden. Dabei dürfen sie nicht zu lange sein, weil die kürzeren Wörter zusammengesetzt werden, um längere Wörter abzubilden. [4]

#### 2.8.2.3 *Prosodische Struktur*

Die Prosodie ist die Eigenschaft der Wort- und Satzbetonungen. In der deutschen und englischen Sprache werden die einzelnen Bestandteile eines Wortes bei der Aussprache betont. Im Japanischen werden mit der Hilfe von verschiedenen Tonhöhen einzelne Wörter voneinander getrennt. Die Form der Prosodie spielt bei der Erkennung von den Wörtern eine untergeordnete Rolle. Dafür ist die Prosodie für die Ausgabe der Wörter wichtig. [4]

#### 2.8.2.4 *Morphologie*

Die deutsche und englische Sprache lassen sich in der Morphologie unterscheiden. Englisch zählt zu den Sprachen mit einem einfachen morphologischen Aufbau. Einen komplexeren morphologischen Aufbau hat die deutsche Sprache. Im Deutschen gehören die Eigenschaften der Verb-Konjugationen, Nomen-Deklinationen und zusammengesetzten Wörter zu den Charakteristiken. Deswegen wird Deutsch als eine gebogene oder gebeugte Sprache bezeichnet. Durch die zusammengesetzten Wörter entstehen "neue" Wörter, die nicht in einem Wörterbuch eingetragen sind. Das führt zu einer schlechteren Genauigkeit bei der Erkennung und Einordnung der Wörter. Außerdem sind die Systeme von den richtigen Testdaten abhängig. [4]

#### 2.8.3 *Unterschied Deutsch und Englisch*

In dem Abschnitt werden die Unterschiede der deutschen und englischen Sprache besprochen. Die Alphabete der beiden Sprachen unterscheiden sich bei den deutschen Umlauten.

Beide Sprachen haben Ähnlichkeiten bei ihrem Klangbild und den Stress- und Betonungsmuster. Die Aussprache unterscheidet sich bei den Buchstaben und den deutschen Umlauten.

Grammatikalische Unterschiede lassen sich in die Gruppen Zeitformen der Verben und Wortausrichtung im Satz einordnen.

Im deutschen gibt es weniger Konjugationsformen für Verben als im Englischen. Dafür verwendet die englische Sprache die Zeitform "continuous tense", die in der deutschen Sprache nicht vorhanden ist. Die Sätze müssen bei Übersetzungen umformuliert werden.

Außerdem wird im Deutschen die abgeschlossene Gegenwart verwendet, wenn über geschehene Events in der Vergangenheit berichtet wird. Diese Ver-



wendung verstößt gegen englische Grammatikregeln. Ein Beispiel ist: "Dann habe ich meine Freunde besucht". Eine falsche Übersetzung ist: "I visited my friends.". Nach Anwendung der grammatikalischen Regeln heißt der Satz: "I have visited my friends.". Im Englischen werden die Sätze, bei denen das Event in der Zukunft liegt, mit "will" aufgebaut. Dafür wird im Deutschen das Präsens verwendet. Der deutsche Satz lautet: "Wenn ich ihn treffe, dann sage ich ihm ...". Im Englischen heißt der Satz: "If I see him, i will tell him ...".

In der englischen Sprache ist der Aufbau eines Satzes nach der Regeln S-P-O regelt. In drei Fällen unterscheiden sich die Anordnungen der beiden Sprachen. Im Deutschen gelten folgende Regeln:

1. Das Verb steht in einem unabhängigen Satz an zweiter Stelle, sodass das Verb auch vor dem Subjekt stehen kann.
2. Das Verb ist bei unabhängigen Sätzen in der Vergangenheit kommt an der letzten Position im Aufbau.
3. Unabhängig von der Zeitform kommt in einem Nebensatz das Verb an letzter Position.

Außerdem gibt es verwandte Wörter, die nicht die selbe Bedeutung haben müssen. Ein weiterer Unterschied sind die unterschiedlichen Artikel. Im Englischen gibt zwei verschiedene Artikel, während es im Deutschen für jedes Geschlecht einen Artikel gibt.

Im Deutschen gibt es zusammengesetzte Wörter, strengere Zeichensetzung und die Schreibweise der Nomen die sich nicht mit dem Englischen überschneiden. [5] [45]

## 2.9 WEITERENTWICKLUNGEN VON SPRACHERKUNNGSSYSTEMEN

Aus dem Buch "An Overview of Modern Speech Recognition" von Huang und Deng [55] werden fünf Punkte erwähnt, an denen in der Zukunft gearbeitet werden sollte.

Die Fehlervermeidung der Spracherkennungssysteme gegenüber äußeren Einflüssen. Dieser Punkt wird in 2.8.1.4 ausführlich besprochen. Natürliche Sprache hat komplizierte Variabilitäten, die mit dem Kontext des Gesagten verbunden sind. Daraus folgt eine komplexe Aufgabe die zu meistern ist.

Mit Hilfe größerer Datenmengen und darin enthalten unterschiedlichen Sprechern kann die Genauigkeit der Spracherkennung verbessert werden.

Den Algorithmen soll beigebracht werden, sich selber weiterzuentwickeln und an neue Gegebenheiten anpassen sollen. Dadurch werden die Algorithmen weniger auf eine Sprache zugeschnitten, sodass sich das Erlernen neuer Sprachen vereinfacht.

Die Robustheit der Systeme wird durch das Erkennen von Wörtern gesteigert, auch wenn diese nicht im Wortschatz vorhanden sind. Daraus folgt eine gesteigerte Genauigkeit.

Anpassen der Algorithmen an das menschliche Lernen. Es muss in die Algorithmen eingearbeitet werden, wie der Mensch das Gesagte wahrnimmt und verarbeitet. Außerdem wie der Mensch sich an neue Dialekte anpasst oder eine Sprach sich wieder beibringt. [55]

## EVALUATION VON SPEECH RECOGNITION SYSTEMEN

Die Evaluation wird in folgende Schritte unterteilt. Die Programme und Daten werden vorgestellt. Für die Bestimmung der Genauigkeit und Geschwindigkeit werden Methoden eingeführt. Die Durchführung wird mit Codebeispielen erklärt. Danach folgt die Vorstellung der Ergebnisse mit anschließender Diskussion.

### 3.1 VORSTELLUNG DER PROGRAMME

Die Programme auf dem Markt müssen folgende Kriterien erfüllen, damit diese für die Evaluation geeignet sind.

- Kostenloser Account
- mindestens fünf Stunden kostenlosen Service
- API Schnittstelle oder lokal laufend

Mit der Aussage kostenloser Account ist die kostenlose Anmeldung und Nutzung des Systems gemeint. In der Tabelle 3.1 sind alle Programme enthalten, die die oben genannten Regeln erfüllen und die in der Cloud laufen.

Untersuchte Cloud Programme		
Programme	Kapazitäten pro Monat	maximale Dateigröße
IBM Watson [32]	10.000 Zeichen	4 MB
Google Cloud [29]	60 Minuten	1 Minute
Wit.Ai [48]	kostenlos	20 Sekunden
Rev.Ai [40]	5 Stunden	Keine Limitierungen
Microsoft Azure [35]	5 Stunden	Keine Limitierungen

Tabelle 3.1: Table Cloud Programme

In dem weiteren Verlauf dieser Arbeit wird IBM Watson als IBM, Google Cloud als Google, Wit.Ai als Wit, Rev.Ai als Rev und Microsoft Azure als Azure angegeben.

Als lokal laufende Alternative wurde Sphinx-4 von CMUSphinx [27] implementiert. Laut der Herstellerseite ist das Programm "state-of-the-art". Dieses wurde in der Programmiersprache Java entwickelt. [26]

### 3.2 VORSTELLUNG DER DATEN

Für die Evaluation werden sowohl Vorlesungen als auch ein Talk verwendet. In dem Abschnitt werden die Vorlesungen vorgestellt. Außerdem werden die Kriterien zu der Auswahl des Talks erklärt.

#### 3.2.1 *Vorlesungen*

Die Audiodateien aus den Vorlesungen dienen zur Bestimmung der Genauigkeit von den Systemen. Dazu werden sowohl deutsch- als auch englischsprachige Vorlesungen verwendet. Die Themen der Vorlesungen sind aus dem Bachelor-Studiengang der Informatik. Diese sollen die Szenarien an der Universität abbilden und vergleichbar machen. Außerdem unterscheiden sich die deutsche von der englischen Sprache, wie in 2.3 beschrieben.

#### 3.2.2 *externe Vergleichsdaten*

Zur Evaluierung der Genauigkeit wird für die englische Vorlesung eine weitere Quelle benötigt.

Es wurde sich für einen Vortrag von einem TED Talk entschieden, da die Bibliothek von TED hat eine Auswahl an Vorträgen mit informatischen Themen. Die Kriterien für die Auswahl sind:

1. Es muss ein Thema aus der Informatik sein, welches nicht von einer anderen Vorlesung abgedeckt wird
2. Die sprechende Person muss eine Frau sein, da die anderen Vorlesungen männliche Sprecher haben.

Die Website von TED hat weitere Vorteile. Für die Vorträge der Talks werden Transkription zu Verfügung gestellt. Es gibt eine Auswahl an sprechenden Personen und Themen. Die Videos lassen sich herunterladen. Die TED Talks sind für die weitere Verwendung und Veröffentlichung zugelassen, sodass es keine Probleme mit dem Urheberrecht gibt. [44]

Es wurde sich für einen Talk von Margaret Gould Stewart zu dem Thema "How giant websites design for you (and a billion other, too)" aus dem Jahr 2014 entschieden. Dieser Talk hat eine ähnliche Länge wie die englische Vorlesung. [43]

### 3.3 METHODEN ZUR BESTIMMUNG DER GENAUIGKEIT

Bei dem Erstellen eines Transkripts kann ein Spracherkennungssystem drei Fehler machen.

Der erste Fehler ist die Verwechslung oder "substitution". Dabei wird ein Wort aus dem Referenztext durch ein anderes Wort ersetzt.

Die Löschung, Auslassung oder im Englischen "deletion" ist der zweite Fehler. Hier werden Wörter aus dem Referenztext nicht mit in das Transkript

übernommen.

Fehler drei ist die Einsetzung im Englischen die "insertion". Dabei wird in das Transkript ein Wort eingefügt, welches in dem Referenztext nicht vorhanden ist.

Diese Fehler werden für die Analyse der Genauigkeit bei verschiedenen Methoden anders ins Verhältnis gesetzt. Außerdem haben die Fehler nicht die selbe Wertigkeit im Sinne des Einflusses auf die Lesbarkeit. [2]

McCowan et al. [13] definieren in ihren Papier verschiedene Eigenschaften, die eine Evaluationsmethode erfüllen sollte. Beim direkten Messen wird an der Komponente für die Spracherkennung gemessen. Ein weiteres Kriterium ist das objektive automatisierte Messen. Das nächste Kriterium besagt, dass die Messungen interpretierbar sein müssen. Das bedeutet, dass das Ergebnis der Messung eine Aussage über die Genauigkeit des Systems geben. Außerdem muss die Differenz der Messergebnisse in einer fairen Relation zueinander stehen. Die Modularität der Methoden ist das letzte Kriterium. Die Metriken müssen genaue spezifische Messungen für ein Spracherkennungssystem erlauben.

### 3.3.1 Word Error Rate

Die WER vergleicht die Wörter von einem vorgegebenen Text mit der Transkription des Systems. Die Buchstaben in der Formel haben folgende Bedeutung: I ist die Anzahl der eingesetzten Wörter, D ist die Anzahl der gelöschten Wörter und S ist die Anzahl der ersetzten Wörter. Die Zahlen werden addiert und durch die Anzahl der Wörter in der Transkription geteilt. Die WER wird in Prozent umgerechnet. Die WER wird in der Spracherkennung am häufigsten verwendet. [19]

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

Die WER hat Nachteile. Sie gibt keine korrekte Prozentzahl an, weil das Ergebnis 100% übersteigen kann und somit kein oberes Limit hat. Damit lässt sich nicht aussagen, ob ein System eine hohe Genauigkeit hat, sondern der Wert lässt sich nur mit anderen vergleichen.

Der WER verstärkt Ersetzungen gegenüber Löschungen und hat somit kein gleiches Verhältnis. [19]

### 3.3.2 Match Error Rate

Die MER untersucht, ob ein Wort gegenüber dem Vergleichswort falsch ist und gibt dafür eine Wahrscheinlichkeit an. Dafür verwendet Sie normalisierte Werte. Somit ist das Resultat eine korrekte Prozentangabe im Gegensatz zur WER. [12] [22]

Andrew C. Morris et. al [2] sagen in ihrer Arbeit, dass die MER Ergebnisse in der Größenordnung zwischen WER und WIL liegen.

$$MER = \frac{(S + D + I)}{(N = H + S + D + I)} = 1 - \frac{H}{N}$$

### 3.3.3 *Word Information Lost*

Der WIL basiert auf den "Relative Information Lost (RIL)". Dabei wird dieser vereinfacht, sodass die WIL weiter verbreitet ist als die RIL. Dabei wird versucht, einen Wert für den Verlust der Information eines Wortes in einem Satz zu finden. Der WIL verwendet dafür eine probabilistische Vorgehensweise. [12] [2] [22]

$$WIL = 1 - \frac{H^2}{(H + S + D)(H + S + i)}$$

## 3.4 METHODEN ZUM BESTIMMEN DER GESCHWINDIGKEIT

Für die Bestimmung der Geschwindigkeit der Systeme werden vor und nach dem Aufruf der Transkription die Zeit genommen. Im Anschluss wird die Endzeit von der Startzeit abgezogen, um die reine Laufzeit zu erhalten. Die Verarbeitungszeit wird für jede Transkription abgespeichert. Für die Auswertung werden die Zeiten nach der Länge der Aufnahme gruppiert, da sonst die Ergebnisse nicht miteinander vergleichbar sind.

## 3.5 DURCHFÜHRUNG DER EVALUATION

Bei der Durchführung werden alle Schritte von dem Start bis zu den Ergebnissen der Evaluation erklärt. Der Start umfasst die Anbindung der Programme und die Vorbereitungen der Daten. Das Ende beinhaltet die statistische Auswertung von den Ergebnisse der Anbieter.

### 3.5.1 *Anbindung der Programme*

Die Anbindung der Programme lässt sich in verschiedene Stufen unterteilen.

Als erstes muss ein Konto bei dem Anbieter erstellt werden.

Danach wird mit Hilfe der Dokumentation des Anbieters die Verbindung eingerichtet. Dabei muss ein API-Schlüssel erstellt werden. Dieser wird zu den herunter geladenen Komponenten hinzugefügt.

Im nächsten Schritt wird die Verbindung und Funktionalität des Anbieters mit einer Testdatei überprüft. Dabei wird die Struktur des Ergebnisses gespeichert, damit die Daten leichter verarbeitet werden könnten.

Daraufhin wird sich die Limitation der maximalen Größe des Uploads der Anbieter angeschaut. Daraus werden Schlüsse gezogen, in wie weit die Dateien für jeden Anbieter vorbereitet werden müssen.

Im letzten Schritt wird die Platzierung der Timestamps für jedes Programm überlegt, sodass am Ende alle Zeiten miteinander vergleichbar sind und alle den gleichen Prozess gemessen haben. Für die Timestamps wird aus dem Python Modul time die Funktion time() verwendet. Laut der Dokumentation von Python ist die time() Funktion die Genaueste. [38]

Die Programme werden ausschließlich in der ursprünglichen Version verwendet. Das bedeutet, bei dem Aufruf werden keine Wörter zur Hilfe mitgegeben, die in der Transkription vorkommen. Außerdem werden den Programmen bei der Übertragung keine Themen als Hilfe genannt. Wenn die Anbieter Informationen über Wörter oder Themen erhalten hätten, dann ist die Vergleichbarkeit nicht mehr gegeben.

```

1 import time
2 from wit import Wit
3
4 access_token = "API-Key"
5
6 client = Wit(access_token=access_token)
7 resp = None
8
9 t_start = time.time()
10 try:
11     for x in range(70):
12         with open('examples_english.wav', 'rb') as f:
13             resp = client.speech(f, {'Content-Type': 'audio/wav'})
14             print('response: ' + str(resp) + '; ')
15             print(resp["text"])
16 except:
17     print("error")
18
19 t_end = time.time()
20 print(t_end - t_start)

```

Listing 3.1: Wit Verbindungsbeispiel

In dem Codebeispiel ist die Anbindung eines Spracherkennungssystems zu sehen. Bei diesem Beispiel handelt es sich um die Implementierung von Wit.

### 3.5.2 Vorbereitung der Videodateien

Um die Videodateien für die Spracherkennung vorzubereiten, muss die Tonspur ausgelesen und abgespeichert werden.

Danach werden die Audiodateien der deutschen Vorlesungen gekürzt. Diese haben eine Länge von 1 Stunde und 30 Minuten und sind damit zulange, um sie komplett auszuwerten. Aus den Vorlesungen werden drei Abschnitte mit einer Dauer von fünf Minuten verwendet. Damit haben sie in der Addition eine ähnliche Dauer wie die englischen Vorlesungen. Die verwendeten Abschnitte sind die ersten 5 Minuten, von Minute 33:30 und die letzten 5 Minuten. Im Anschluss wird geschaut, ob an den Kanten Wörter angeschnitten wurde. Wenn dies der Fall ist, dann wird das Wort komplett aus der Datei entfernt.

Die letzte Vorbereitung ist die Aufteilung der Audiodatei in kleinere Elemente. Das hat den Hintergrund, weil Anbieter Grenzen für die maximale Länge einer Audiodatei haben. Deswegen müssen die Dateien mit Hilfe der

Funktion `split_on_silence` von `pydub` in kleinere Elemente zerteilt werden. Danach werden die Stücke zu der passenden Länge zusammengesetzt. Die Funktion wird in dem Abschnitt 4.3.3 besprochen und in 4.5.1 wird auf die Probleme mit der Funktion eingegangen.

### 3.5.3 *Transkript für die Videos erstellen*

Für die Evaluation müssen für die Audiodateien Transkripte erstellt werden. Dafür werden die Dateien angehört und es werden alle Wort und Äußerung aufgeschrieben, die von der Spracherkennung verarbeitet werden. Dabei wird auf die Formulierung von Wörtern geachtet. So werden Zahlen und Symbole wie zum Beispiel das Prozentzeichen ausgeschrieben.

### 3.5.4 *Transkripte an Vorlage anpassen*

Die von der Spracherkennung erstellten Transkripte werden an die Vorlagen in Schritten angepasst.

Die Wörter der Transkription werden auf ihre Schreibweise überprüft und angepasst. Das umfasst folgende Regeln:

- Wörter werden angeglichen, zum Beispiel werden Wörter zusammengefügt.
- Zahlen werden in ausgeschriebene Form gebracht.
- Alle Wörter müssen klein geschrieben werden, da nicht alle Programme ihre Antworten in Sätzen zurück geben. Sonst würden Fehler am Satzanfang entstehen.
- Es werden doppelte Leerzeichen entfernt.
- Alle Satzzeichen müssen entfernt werden, da nicht alle Systeme diese als Antwort geben haben.
- Es werden mögliche Leerzeichen nach jeder Reihe entfernt.
- Abkürzungen werden ausgeschrieben, zum Beispiel `that's` zu `that is`.

Die Formate der Transkripte werden aneinander angepasst. Die Transkripte werden in Zeilen unterteilt, sodass die miteinander verglichen werden. So sind statistische Auswertung innerhalb des Transkriptes möglich.

### 3.5.5 *Transkripte mit Vorlage vergleichen*

Die Antworten der Programme werden aus einer Textdatei für die Verarbeitung eingelesen. Der Vergleich wird mit dem Python-Modul `JiWER` [33] gemacht. In dem Modul sind die drei Kennzahlen `WER`, `MER` und `WIL` implementiert. Die Kennzahlen wurden im Abschnitt 3.3 erklärt. Außerdem verfügt das Modul Funktionen zur Bereinigung der Antworten. Im Anschluss werden die Antworten ausgegeben und können gespeichert werden.



```

1 ground_truth_file = open("new_txt.txt", "r")
2 hypothesis_file = open("txt.txt", "r")
3 ground_truth = []
4 hypothesis = []
5
6 transformation = jiwer.Compose([
7     jiwer.RemoveMultipleSpaces(),
8     jiwer.Strip(),
9     jiwer.RemoveEmptyStrings(),
10    jiwer.RemovePunctuation(),
11    jiwer.ExpandCommonEnglishContractions(),
12    jiwer.ToLowerCase()
13 ])
14
15 for _ in range(5):
16     ground_truth.append(ground_truth_file.readline())
17     hypothesis.append(hypothesis_file.readline())
18
19 for x in range(5):
20     measures = jiwer.compute_measures(ground_truth[x], hypothesis[x],
21                                     truth_transform=transformation, hypothesis_transform=
22                                     transformation)
21     wer = measures['wer']
22     mer = measures['mer']
23     wil = measures['wil']
24     print(str(wer) + ';' + str(mer) + ';' + str(wil))

```

Listing 3.2: Vergleich Transkription zu Vorlage

### 3.5.6 Dauer der automatisch erstellten Transkripte vergleichen

Für den Vergleich der Geschwindigkeit ist keine Implementierung vonnöten. Die Zeiten werden in Sekunden abgespeichert und miteinander verglichen.

## 3.6 ERGEBNISSE

Die Ergebnisse sind nach Genauigkeit und Geschwindigkeit unterteilt. Als nächstes folgt eine Zusammenfassung des Abschnittes. Danach werden die Ergebnisse nach der Sprache getrennt. In den Tabellen der Genauigkeit sind alle Ergebnisse in Prozent umgewandelt worden. Die Resultate der Geschwindigkeit in den Tabellen werden in Sekunden angegeben. Die Ergebnisse sind auf die zweite Nachkommastelle gerundet.

### 3.6.1 Genauigkeit Zusammenfassung

Wenn die Resultate von den Programmen und Sprachen zusammengefasst werden, dann ergibt sich eine WER von 12.34, eine MER von 11.92 und eine WIL von 14.79.

Die Tabelle 3.2 zeigt die Zusammenfassung der Ergebnisse für jedes Programm.

Genauigkeit Zusammengefasst			
Programme	WER	MER	WIL
IBM	14.89	14.11	18.70
Google	10.80	10.45	13.67
Wit	21.78	21.37	23.99
Azure	6.59	6.34	8.13
Rev	7.65	7.32	9.44

Tabelle 3.2: Zusammenfassung der Ergebnisse

### 3.6.2 Genauigkeit Englisch

In den folgenden Tabellen werden die Ergebnisse der Vorlesung Cloud Computing und des TED Talks von Margaret Gould Stewart vorgestellt.

Vergleich von Vorlesung Cloud Computing			
Programme	WER	MER	WIL
IBM	16.26	14.92	19.29
Google	20.79	19.90	26.49
Wit	38.42	37.78	43.92
Azure	9.15	8.71	11.61
Rev	10.19	9.59	12.54

Tabelle 3.3: Cloud Computing

Vergleich von Ted Talk			
Programme	WER	MER	WIL
IBM	2.46	2.43	3.20
Google	2.93	2.89	3.80
Wit	42.58	41.97	43.91
Azure	1.10	1.08	1.24
Rev	1.17	1.15	1.38

Tabelle 3.4: Ted Talk

### 3.6.3 Genauigkeit Deutsch

Die Ergebnisse der Vorlesungen Betriebssysteme, Statistik und Technischer Informatik werden in diesem Abschnitt vorgestellt. Die Durchführung der Messung ist in Kapitel 3.5 beschrieben. Die Tabellen beinhalten die Zusammengefassten Resultate. Die einzelnen Messungen der fünfminütigen Teilstücke ist im Anhang unter B nachzulesen.

Vorlesung Betriebssysteme Zusammenfassung			
Programme	WER	MER	WIL
IBM	11.53	11.06	14.39
Google	5.93	5.75	7.43
Wit	6.65	6.53	7.76
Azure	4.67	4.48	5.48
Rev	5.93	5.74	7.12

Tabelle 3.5: Betriebssysteme Zusammengefasst

Vorlesung Statistik Zusammenfassung			
Programme	WER	MER	WIL
IBM	24.22	23.21	30.51
Google	13.38	13.02	16.24
Wit	9.16	8.83	10.21
Azure	9.11	8.78	10.75
Rev	10.69	10.24	13.01

Tabelle 3.6: Statistik Zusammengefasst

Vorlesung Technische Informatik Zusammenfassung			
Programme	WER	MER	WIL
IBM	19.99	18.95	26.12
Google	15.50	15.09	20.08
Wit	12.11	11.74	14.15
Azure	8.94	8.63	11.55
Rev	10.28	9.90	13.17

Tabelle 3.7: Technische Informatik Zusammengefasst

### 3.6.4 Geschwindigkeit

In den Tabellen sind die Ergebnisse der Geschwindigkeit. Die Ergebnisse werden nach der Sprache der Aufzeichnung getrennt. Bei der Geschwindigkeit ist es nicht möglich, einen Gesamtwert für die Aufzeichnungen zu errechnen. Die Resultate sind absolute Werte und die Dauer der Aufzeichnungen sind nicht gleich. Die Ergebnisse der deutschen Vorlesungen sind Zusammenfassungen der Messungen. Im Anhang unter C befinden sich die Resultate der Abschnitte.

Vergleich der Geschwindigkeit in Sekunden		
Programme	TED Talk	Cloud Vorlesung
IBM	1366.70	1185.16
Google	242.01	320.67
Wit	362.34	375.09
Azure	477.58	500.32
Rev	318.55	375.7
CMU	14901.73	16985.22

Tabelle 3.8: Vergleich der Geschwindigkeit englischer Aufzeichnungen

Vergleich der Geschwindigkeit in Sekunden zusammen addiert			
Programme	Betriebssysteme	Statistik	Technische Informatik
IBM	1.400,74*	1.284,1	1.257,36
Google	287,71	328,17	283,69
Wit	465,86*	303,24	305,57
Azure	535,26	532,04	536,19
Rev	300,39	296,53	309,9

\* Resultieren aus einer hohen Anzahl von Aufrufen an den Anbieter

Tabelle 3.9: Vergleich der Geschwindigkeit deutscher Aufzeichnungen

## 3.7 DISKUSSION

In der Diskussion wird auf die fehlende Verwendung von CMU Sphinx eingegangen. Außerdem werden die Ergebnisse von der Genauigkeit und Geschwindigkeit weiter untersucht.

### 3.7.1 Vergleich zu den vorgestellten Studien

Die Studien zu der Evaluation von Spracherkennungssystemen 2.2 haben Ergebnisse von 17.4 und 20.63 gemessen. Die gemessenen Ergebnisse in 3.6.1 zeigen einen Wert von 12.34. Somit sind die Resultate deutlich niedriger. Die Unterschiede von den Resultaten kann aus verschiedenen Gründe entstanden sein. Die Anbieter haben ihre Programme in der Zwischenzeit verbessert. Außerdem wurden bei den Studien nicht die gleichen Programme verwendet, sodass das ein Grund für unterschiedliche Ergebnisse ist. Bei dieser Untersuchung wurde dazu eine weitere Sprache getestet, von der die Ergebnisse auch in das Resultat eingeflossen sind. Außerdem kann die Vorbereitung der Daten, in 3.5.4 beschrieben, einen Einfluss auf das Ergebnis haben.

### 3.7.2 Auslassung von CMU Sphinx

Die Spracherkennungssoftware CMU Sphinx wurde für die Aufzeichnungen der Cloud Vorlesung und des TED Talks verwendet. Das Programm hat keine zwei hintereinander folgende Wörter erkannt und die Ergebnisse waren nicht auswertbar. Außerdem war die Geschwindigkeit langsamer als bei den anderen Programmen 3.8. Dieser Fakt war nicht der Grund für die Aussortierung des Programms. Das Programm wurde in der englischen Sprache mit der vorinstallierten Bibliothek verwendet.

### 3.7.3 Genauigkeit

Die in den folgenden Abschnitten gezeigten Ergebnisse wird der WER verwendet.

#### 3.7.3.1 WER, MER und WIL

Andrew C. Morris et al [2] schreiben in ihrer Arbeit, dass die drei Vergleichsmethoden bei niedrigen Resultaten auf ähnliche Ergebnisse kommen, sodass die ungenügende statistische Herleitung der WER keinen Effekt hat.

In den Ergebnissen lag die MER im Vergleich zu der WER weniger als 1 Prozentpunkt niedriger. Die Resultate der WIL sind um 21 Prozentpunkte im Durchschnitt bei der zusammengefassten Genauigkeit 3.2 höher als bei der WER. Die Werte bleiben konstant zwischen 23 und 27 Prozentpunkten. Wit ist bei dieser Rechnung die Ausnahme mit einem Verhältnis von 10 Prozentpunkten.

Durch die niedrigen Werte bei der Auswertung lässt sich mit diesem Datensatz die Schwäche der WER nicht darstellen.

#### 3.7.3.2 Deutsch und Englisch getrennt

In der Tabelle 3.10 sind die Durchschnittswerte nach Sprache und Programm aufgeschlüsselt.

Vergleich der Genauigkeit nach Sprachen getrennt		
Programme	Deutsch	Englisch
IBM	18,58	9,36
Google	11,6	11,86
Wit	9,31	40,5
Azure	7,57	5,13
Rev	8,97	5,68

Tabelle 3.10: Vergleich der Genauigkeit nach Sprachen

Bei einer geringeren Anzahl an Werten ist der Durchschnittswert leichter zu beeinflussen. Deswegen fällt der Unterschied bei IBM zwischen Deutsch und Englisch so Groß aus. IBM erreichte mit dem TED Talk ein Ergebnis, dass den Durchschnitts fast halbiert hat. Ansonsten wäre das Resultat der englischen Aufzeichnungen auf einem Wert wie bei den deutschen Vorlesungen. Wit hat bei diesen Messungen zwischen den Sprachen den größten Unterschied. Es ist nicht zu beobachten, dass alle Anbieter bessere Ergebnisse in einer Sprache erzielt haben.

### 3.7.3.3 TED Talk

Die Ergebnisse des TED Talks 3.4 unterscheiden sich von den anderen Resultaten, da die Anbieter bei dem Talk bessere Ergebnisse erzielt haben. Für die folgenden Berechnungen wurde der Anbieter Wit nicht berücksichtigt. Die durchschnittliche WER liegt bei 1.92. Der durchschnittliche WER von allen Aufzeichnungen liegt bei 9.98 3.2. Werden diese Werte miteinander verglichen, dann ergibt sich, dass die Ergebnisse von TED mehr als 5 mal niedriger sind.

Wenn der TED Talk mit der englischen Vorlesung gegenüber gestellt wird, dann sind die Resultate der Vorlesung 7 mal höher. Die Vorlesung kommt auf eine durchschnittliche WER von 14.1.

Die niedrigsten gemessen Werte hat die Vorlesung Betriebssysteme. Dort liegt der Durchschnittswert der WER ohne Wit bei 7.01 und mit Wit bei 6.94. Damit sind die Werte des TED Talks 3.5 mal niedriger.

Es lässt die Vermutung aufkommen, dass die Anbieter der Software ihre Modelle mit Talks von TED trainiert haben. Diese sind einfach herunterzuladen und Transkription wurden angefertigt.

Der WER von IBM ist im Vergleich zu den anderen Ergebnissen niedrig. Die WER von IBM liegt für den TED Talk bei 2.46 und die Genauigkeit für alle Ergebnisse ist bei 14.89. Damit ist das Resultat des TED Talks 6 mal niedriger. Der absolute Verlust von den Prozentpunkten wurde mit 12.43 gemessen. Damit ist das Resultat der größte absolute Unterschied von den Programmen zwischen TED Talk und der Zusammenfassung der Genauigkeit.

### 3.7.4 Geschwindigkeit

Bei dieser Tabelle 3.11 wird das Verhältnis von Gesamtlaufzeit der Transkription und Dauer der Aufzeichnung betrachtet. Damit lassen sich die Werte miteinander vergleichen.

Verhältnis von Dauer der Transkription zu Länge der Vorlesung			
Programme	Deutsch	Englisch	Kombination
IBM	87.27	99.59	93,43
Google	19.99	21.83	20,91
Wit	23.89	28.73	26,31
Azure	35.63	38.06	36,85
Rev	20.15	26,98	23,57

Tabelle 3.11: Geschwindigkeit Verhältnis

Für die deutschen Vorlesungen lassen sich die Programme in drei Kategorien einordnen. Rev, Wit und Google haben die schnellsten Zeiten. Azure wird in die mittlere Kategorie eingeordnet und IBM ist das Schlusslicht. Es ergibt sich ein Muster. Die schnellsten Programme sind doppelt so schnell wie das mittlere Programm. Wobei dieses doppelt so schnell wie das langsamste Programm ist.

Bei den englischen Vorlesungen hat Google die beste Geschwindigkeit. Es folgen Wit und Rev wobei der Abstand zu Azure kleiner als bei der deutschen Sprache ist. Bei der englischen Aufzeichnungen ist IBM ebenfalls das langsamste Programm.

Die Audiodateien in der englischen Sprache haben eine Länge von 12:47 und 13:23 Minuten und sind damit ähnlich lang. Für IBM wurden für den kürzeren TED Talk um die 200 Sekunden mehr gemessen. Damit ist IBM der einzige Anbieter, der für den kürzeren Talk mehr Zeit benötigt hat.

### 3.7.5 Verknüpfung von Genauigkeit und Geschwindigkeit

Es wurde keine wissenschaftliche Studie gefunden, die die Geschwindigkeit und Genauigkeit in ein Verhältnis setzt, um ein Gesamtleistung zu errechnen. In dem Abschnitt werden zwei Methoden vorgestellt, mit denen die Programme verglichen werden. Die Methoden können nur für einen Vergleich untereinander verwendet werden und geben keine generelle Aussage, ob ein Anbieter ein genauen und schnellen Service anbietet.

Die erste Methode verwendet die Platzierung in den Tabellen von schnell zu langsam beziehungsweise von genau zu ungenau. Bei der zweiten Methode wird das Verhältnisse der Ergebnisse zu dem besten Resultat berechnet, um den Abstand einordnen zu können.

## 3.7.5.1 Methode 1

Regeln der ersten Methode:

1. Es gibt ein Ranking von 1 bis 5.
2. Wenn der Unterschied zwischen der WER von Programmen weniger als 1 beträgt, dann wird der gleiche Platz notiert.
3. Wenn die Geschwindigkeiten zwischen zwei Anbietern weniger als 1% gemessen wurde, dann erfolgt die gleiche Platzierung.
4. Angenommen zwei Programme bekommen die selbe Platzierung, dann wird eine Platzierung übersprungen.
5. Das Zwischenergebnis der Programme ist der Durchschnitt aus den 5 Platzierungen.
6. Das Endergebnis wird aus dem Durchschnitt der Zwischenergebnisse errechnet.

Vergleich nach Platzierung			
Programme	Geschwindigkeit	Genauigkeit	Kombination
IBM	5	4.2	4.6
Google	1.4	3.4	2.4
Wit	2.6	3.2	3
Azure	4	1	2.5
Rev	2.2	1.8	2

Tabelle 3.12: Vergleich der Programme nach Platzierungen

## 3.7.5.2 Methode 2

Regeln für Methode 2:

1. Das beste Ergebnis erhält den Wert 1.
2. Die weiteren Ergebnisse sind das Verhältnis des Programms zum schnellsten oder genauesten Programm.
3. Zahlen werden auf die zweite Nachkommastelle gerundet.
4. Das Zwischenergebnis ist der Durchschnitt aus den 5 Resultaten.
5. Das Endergebnis ist der Durchschnitt von den Zwischenergebnissen.



Vergleich nach Verhältnis			
Programme	Geschwindigkeit	Genauigkeit	Kombination
IBM	4.57	2.28	3.43
Google	1.06	1.88	1.47
Wit	1.28	9.37	5.33
Azure	1.81	1	1.41
Rev	1.12	1.15	1.14

Tabelle 3.13: Vergleich der Programme nach Verhältnis

### 3.7.5.3 Diskussion der Ergebnisse

Bei den Betrachtungsarten kommen ähnliche Platzierungen heraus. Beide Methoden kommen zu dem Ergebnis, dass Azure und Google in der Kombination von Geschwindigkeit und Genauigkeit nahe zu identische Ergebnisse haben. Dabei sind die Ergebnisse bei Methode 1 mit 2.4 und 2.5 und Methode 2 mit 1.41 und 1.47 fast identisch.

Der große Unterschied zwischen den Ergebnissen der Methoden ist Wit. Diese Unterschiede resultieren von den WER-Werten bei den englischen Aufzeichnungen. Wenn nur die Platzierung als Ergebnis betrachtet wird, dann haben Ausreißer einen kleineren Ausschlag und "verfälschen" das Ergebnis weniger. Daher kommt die Platzierung von Wit bei der ersten Methode. Wit hat bei Methode 1 den selben Abstand zu den Zweitplatzierten wie der Zweitplatzierte zu den Bestplatzierten hat. Wird der selbe Abstand auf die zweite Methode übertragen, dann hätte Wit ein Ergebnis um 1.7.

### 3.7.6 Probleme bei der Auswertung

Nachteile bei der Verwendung von dem Verhältnis:

Bei der Genauigkeit des TED Talks 3.4 gibt es WER-Werte von 2.46 und 1.1. Das Verhältnis der Werte liegt bei 2.24. Damit ist der absolute Abstand zwischen den Werten kleiner als das Verhältnis.

Das Verhältnis funktioniert bei größeren Werten besser als bei kleineren Werten. Bei kleineren Werten haben absolute Unterschiede einen großen Einfluss auf das Verhältnis und damit auf das Ergebnis.

In den Messdaten befinden sich unterschiedlich große Werte, die von 1 bis 44 reichen. Damit ist es nicht möglich, die perfekte Auswertungsmethode zu finden.

Verhältnis von Geschwindigkeit zu Genauigkeit:

In den Auswertungen wurde das Verhältnis von jeweils fünfzig Prozent angenommen. Dieses Verhältnis wurde gewählt, weil Vorlesungen von einer Dauer bis zu 90 Minuten ein wesentlicher Faktor für die Performance des Proof of Concept sind. Je länger die Aufzeichnungen sind, umso größer

muss die Gewichtung der Geschwindigkeit sein. Deswegen hängt das Verhältnis von dem Einsatzgebiet des Programms ab.

## PROOF OF CONCEPT

In dem Abschnitt Proof of Concept werden die Konzeption und Implementierung vorgestellt. Danach werden die Testergebnisse gezeigt und auf Probleme eingegangen.

### 4.1 BESTIMMUNG DER AUFGABEN

Das Programm wird über die Kommandozeile aufgerufen und verarbeitet die Aufzeichnungen der Vorlesungen. Als nächstes wird die Tonspur aus-gelesen und transkribiert. Die Transkription wird als Videountertitel und/oder PDF ausgegeben. In der PDF stehen die Dauer des Abschnitts, der Start- und Endzeitpunkt und das Transkript des Abschnittes.

### 4.2 KONZEPTION DES PROGRAMMS

In der Konzeption wird das Design des Programms, die Auswahl der Spracherkennungssoftware und die Erstellung der Zeitabschnitte besprochen.

#### 4.2.1 Design der Struktur

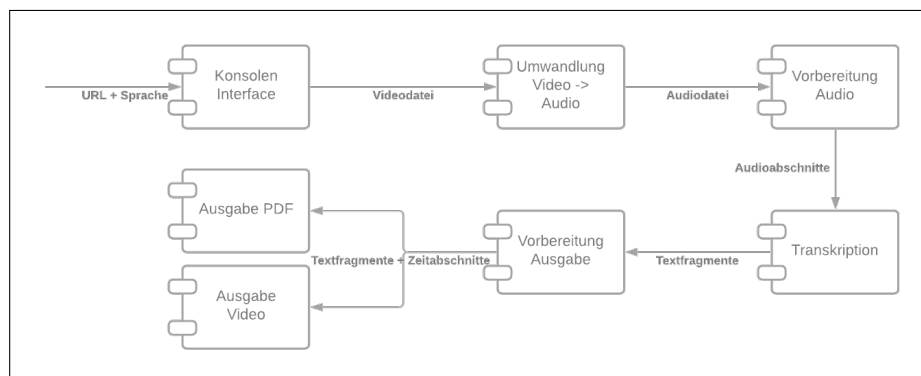


Abbildung 4.1: Komponentendiagramm

Das Programm wird aus verschiedenen Blöcken bestehen. Der erste Block ist das Interface in der Kommandozeile. Dort wird sicher gegangen, dass alle notwendigen Informationen für den Ablauf vorhanden sind. Im zweiten Block wird die Videodatei in eine WAV-Datei umgewandelt. Die Aufteilung der Aufzeichnungen in kleinere Abschnitte für die Transkription wird im dritten Block gemanagt. Die Transkription der Tonspuren passiert im vierten Block. Im fünften Block werden die Ergebnisse für die Ausgabe aufbereitet.

Die Erstellung der PDF mit ihren Eigenschaften wird im sechsten Block gemacht. Der letzte Block ist das Hinzufügen der Untertitel zu der Videodatei.

#### 4.2.2 *Auswahl des Spracherkennungsprogramms*

Mit den folgenden Kriterien wird das Spracherkennungsprogramm für den Proof of Concept ausgesucht.

- Kostenlose Erstellung eines Accounts.
- Dauerhaftes kostenloses erstellen von Transkripten.
- Anbindung des Programms über eine API-Schnittstelle.

Diese Kriterien erfüllt die Software Wit. Wit hat sowohl für die deutsche Sprache 3.10 als auch bei der Geschwindigkeit 3.11 das dritt beste Ergebnis erzielt.

Über Wit können verschiedene Audioformate wie Wav, mpeg3, ogg und ulaw übertragen und verarbeitet werden. [31]

In diesem Proof of Concept wird ausschließlich das Format Wav implementiert und verwendet.

#### 4.2.3 *Erstellen der Timestamps*

Es gibt verschiedene Methoden zur Erstellung der Timestamps. Diese Abschnitte könnten zur Einteilung der Videodatei verwendet werden. Dabei wird bei den Methoden auf die Vor- und Nachteile eingegangen. Diese Methoden sind nicht in den Proof of Concept implementiert worden.

##### 4.2.3.1 *Timestamps pro einer Zeiteinheit*

Dabei wird das Video bei der Erstellung von den Timestamps in gleichmäßige Stücke aufgeteilt. Die Länge der Abschnitte ist frei wählbar und kann als Beispiel 20 Sekunden betragen. Ein Vorteil ist die einfache Implementierung der Methodik. Der Nachteil ist die Fehleranfälligkeit. Die Sprechgeschwindigkeit variiert während der Aufnahme und es sind Sprachpausen vorhanden. Damit ist der Anteil des Gesagten unterschiedlich für jeden Abschnitt. Außerdem werden Wörter teilweise abgeschnitten, sodass die Datei nachgebessert werden muss. An den Kanten der Abschnitte müssten lautlose Abschnitte hinzugefügt werden, damit die Systeme besser Sprachpausen erkennen.

##### 4.2.3.2 *Timestamp pro Slide einer Vorlesung*

Die Bilder in einer Videodatei würden miteinander verglichen werden, um Timestamps anhand von Folienwechseln zu erstellen. Dabei muss beachtet werden, dass die lehrende Person in der Videodatei eingeblendet sein kann. Außerdem ist eine Problematik, wenn es einen Szenenwechsel gibt. Damit

ist gemeint, wenn die lehrende Person von einem kleinem Bild zu einem Vollbild wechselt.

Dabei muss bei diesem Konzept die Anzahl der verarbeiten Bilder überlegt werden. Bei einem Video mit 60 Bildern in einer Sekunde und einer Dauer von 45 Minuten, dann kommen mit folgender Gleichung: Bilder pro Sekunde \* Sekunden \* Minuten entsprechend viele Vergleiche zustande:  $60 \times 60 \times 45 = 162,000$ . Wenn die Anzahl der verwendeten Bilder auf zwei verringert wird, dann kommt es zu folgendem Ergebnis:  $2 \times 60 \times 45 = 5,400$ . Dabei sinkt die Anzahl der benötigten Bilder auf  $3,3\%$ .

#### 4.3 IMPLEMENTIERUNG DES PROOF OF CONCEPT

In diesem Proof of Concept werden die drei externe Bibliotheken pydub [53], moviepy [51] und reportlab [39] verwendet.

Pydub und moviepy werden für die Erkennung und Verarbeitung der Video-beziehungsweise Audiodateien benötigt. Außerdem generiert moviepy die Untertitel. Reportlab wird für die Erstellung der PDF verwendet.

Für pydub ist die Bibliothek ffmpeg Voraussetzung. Pydub ist ein Tool um Audioformate einzulesen und diese zu bearbeiten. Mit der Bibliothek werden die Audiospuren zusammengefügt und getrennt. Außerdem bietet die Bibliothek mit der Funktion `split_on_silence` die Fähigkeit an, die Audiospur anhand von Stille zu zerteilen.

Um Untertitel in Videodateien mit moviepy zuschreiben, wird ImageMagick benötigt. In dem Proof of Concept wird moviepy zum Auslesen der Audiospur aus einer Videodatei verwendet. Die Audiospur wird dann mit moviepy als Audiodatei gespeichert. Mit moviepy lassen sich Texte und Effekte in eine Videodatei schreiben. Die Bibliothek kann unterschiedliche Formate einlesen und verarbeiten.

Mit Reportlab lassen sich neue PDFs erstellen und dort verschiedene Elemente einfügen.

##### 4.3.1 Start des Programms

```

1 parser = argparse.ArgumentParser()
2
3 parser.add_argument('-f', action='store', dest='filename', required=True
4     ,
5     help='Name of the file')
6 parser.add_argument('-l', action='store', dest='language', required=True
7     ,
8     help='Set language. Supported is de and en')
9 parser.add_argument('-d', action='store', dest='dBFS', type=int, default
10     =-25,
11     help='Can be be between -16 and -25')

```

```

12 parser.add_argument('-o', action='store', dest='output', default="both",
13                     help='Set Output. Can be video, pdf or both')
14
15 parser.add_argument('--version', action='version', version='%(prog)s 0.1
16                     ')
17 results = parser.parse_args()

```

Listing 4.1: Implementierung Konsolen Interface

Für die Implementierung des Starts wird das `argparse` Modul von Python verwendet. In diesem Modul ist der Konsolenbefehl `help` implementiert, der im Code spezifiziert werden muss. Dieser listet die definierten Parameter auf. In der Auflistung folgt die Flagge, der Variablenname und die Beschreibung.

Bei der Implementierung wird der parser initialisiert. Danach werden die einzelnen Argumente definiert. Die Argumente werden mit Flaggenamen, Aktion, Variablennamen, Variablentyp, Standardwert und Beschreibung hinzugefügt. Wenn das Programm startet, dann werden die Variablen über die Konsole abgefragt und für den weiteren Verlauf verwendet.

#### 4.3.2 Umwandlung von Video- in Audioformat

```

1 def convert_to_wav(src):
2     video_function = mp.VideoFileClip(src)
3     video_function.audio.write_audiofile("testtest.wav")
4     return video_function

```

Listing 4.2: Umwandlung Video- zu Audioformat

In dem Code 4.2 ist zusehen, wie die Videodatei von dem Programm eingelesen wird. Dabei muss die Datei im selben Ordner liegen, von wo aus das Programm ausgeführt wird. Anschließend wird die Audiospur aus der Videodatei ausgelesen und in einer Datei in dem Ordner zwischen gespeichert. Im Anschluss wird die Videodatei von der Funktion zurück gegeben, um später Merkmale aus der Datei auszulesen.

#### 4.3.3 Vorbereitung der Audiodatei

```

1 def read_wav_file_and_split(src):
2     audio_file_function = AudioSegment.from_wav(src)
3     print("audio file duration: " + str(audio_file_function.
4         duration_seconds)) # complete
5
6     dBFS = audio_file_function.dBFS
7     # split file
8     chunks_function = split_on_silence(
9         audio_file_function,

```

```

9         min_silence_len=250,
10        silence_thresh=dbFS - 25,
11        keep_silence=150,
12    )
13
14    return chunks_function, dBFS, audio_file_function

```

Listing 4.3: Einlesen und Aufteilen der Audiodatei

Der Codeabschnitt 4.3 zeigt, wie die Audiodatei eingelesen wird, die im oberen Teil erstellt wurde. Daraufhin wird der dBFS<sup>1</sup> ausgelesen. Anhand diesem erkennt die Funktion `split_on_silence`, wann nichts gesagt wird und unterteilt die Audiodatei anhand der stillen Stellen. Die Rückgabe der Funktion enthält die erkannten Sprachteile, die dBFS und die Audiodatei.

```

1 def add_audio_together(chunks_function, target_length_function):
2     output_chunks_function = [chunks_function[0]]
3     for chunk in chunks_function[1:]:
4         if len(output_chunks_function[-1]) < target_length_function:
5             output_chunks_function[-1] += chunk
6         else:
7             # if the last output chunk is longer than the target length,
8             # we can start a new one
9             output_chunks_function.append(chunk)
10    return output_chunks_function

```

Listing 4.4: Zusammenfügen der Audiodateien zu längeren Teilstücken

Die Länge der erkannten Teilstücke variiert. Das Programm läuft schneller, umso weniger Teilstücke an den Service verschickt werden. Deswegen werden die Teilstücke zu längeren Abschnitten zusammen gesetzt. Dabei ist die minimale Länge der Abschnitte frei bestimmbar. Die Funktion gibt die zusammengesetzten Teilstücke wieder an das Hauptprogramm zurück.

```

1 def fix_length_audio(chunks_, dBFS, audio_file_function, test_function):
2     new_list = []
3     target_length_function = 10 * 1000
4     for chunk in chunks_:
5         if chunk.duration_seconds > 20:
6             # split audio
7             # split file
8             chunks_function = split_on_silence(
9                 chunk,
10                min_silence_len=50,
11                silence_thresh=dbFS - 23,
12                keep_silence=150,
13            )
14            new_list.extend(add_audio_together(chunks_function,
15                target_length_function))
16        else:

```

---

<sup>1</sup> weitere Informationen in 4.5.1

```

16         new_list.append(chunk)
17     # create last snippet
18     t_finish = audio_file_function.duration_seconds
19     t_start = (t_finish - test_function) * 1000
20     splitted_audio = audio_file_function[-t_start:]
21     new_list.append(splitted_audio)
22     return new_list

```

Listing 4.5: Anpassung der Länge der Teilstücke

Durch das Vorgehen mit der minimalen Länge passiert es, dass die Abschnitte zu lang werden. Dies muss vermieden werden, da der Service von Wit nur eine Dauer um 20 Sekunden erlaubt. Wenn ein Teilstück zu lange ist, dann antwortet Wit mit einer Fehlermeldung. Für die Vermeidung wird die Funktion `split_on_silence` verwendet. Mit dieser werden die zu langen Teilstücke aufgeteilt und in kleinere Teilstücke abgespeichert. Für die Umwandlung wird die Methode aus dem Codeabschnitt 4.4 verwendet.

```

1 def save_chunks_temporarily(output_chunks_function):
2     number_function = 0
3     list_chunk_length_function = []
4     for x_function, added_chunk in enumerate(output_chunks_function):
5         added_chunk.export("test_test{o}.wav".format(x_function), format
6                             ="wav")
7         number_function = x_function
8         list_chunk_length_function.append(round(added_chunk.
9                                             duration_seconds, 2))
10    return number_function, list_chunk_length_function

```

Listing 4.6: Abspeichern für die Transkription

Die Abschnitte werden zwischengespeichert, da Wit einen Pfad zu einer Datei benötigt. Dafür wird eine Funktion von `pydub` verwendet, die die Teilstücke in dem Wav-Format abspeichert. Die abgespeicherten Teilstücke werden nach der Ausführung gelöscht, siehe 4.3.8.

#### 4.3.4 Transkription

```

1 def transcribe_snippets(repetitions):
2     access_token = "API-Key"
3     # deutsch: API-Key
4     # englisch:
5
6     client = Wit(access_token=access_token)
7
8     resp = None
9
10    list_transcriptions_function = []
11

```



```

12     for x_function in range(repetitions):
13         try:
14             with open('test_test' + str(x_function) + '.wav', 'rb') as f
15                 :
16                 resp = client.speech(f, {'Content-Type': 'audio/wav'})
17                 # print('response: ' + str(resp) + '; ')
18                 list_transcriptions_function.append(resp["text"])
19         except:
20             # print("error")
21             list_transcriptions_function.append("error")
22     return list_transcriptions_function

```

Listing 4.7: Transkription

Für die Transkription muss der Client initialisiert werden. Dafür wird bei Wit nur der Access-Token benötigt. Danach iteriert eine Schleife über die zwischengespeicherten Audiodateien. Dabei wird die Audiodatei eingelesen und an Wit verschickt. Die Sprache wird über den verwendeten Access-Token geregelt. Das bedeutet, wenn in seinem Programm Deutsch und Englisch verarbeitet wird, dann muss für beide Sprachen einen Access-Token generiert werden. Außerdem muss eine Routine zum Mappen von Sprache und Access-Token implementiert werden. Die Antwort wird in eine Liste gespeichert. Wenn es in diesem Prozess zu einem Fehler kommt, dann wird statt der Antwort der Text "error" in die Liste eingefügt.

#### 4.3.5 Vorbereitung Ausgabe

```

1  def create_video_chunks(video_function, number_function,
2      added_chunk_list_function):
3      video_length = video_function.duration
4      clip_list_function = []
5      for x_function in range(number_function):
6          if x_function == 0:
7              clip_list_function.append(video_function.subclip(0,
8                  added_chunk_list_function[x_function]))
9          else:
10             clip_list_function.append(video_function.subclip(
11                 added_chunk_list_function[x_function - 1],
12                 added_chunk_list_function[x_function]))
13     return clip_list_function

```

Listing 4.8: Erstellen der Videoabschnitte

Die Videodatei muss bei der Vorbereitung von der Ausgabe noch in die selben Abschnitte wie die Audiodatei unterteilt werden. Dafür wird die Liste mit den Informationen über die Grenzen der Audioabschnitten verwendet. Anhand dieser Grenzen wird die Videodatei in Abschnitte geteilt, in einer Liste abgespeichert und zum Hauptprogramm zurück gegeben.

```

1 def generate_text_chunks(text, length):
2     splitted = text.split(" ")
3
4     text_list = []
5     text = ""
6     for splits in splitted:
7
8         new_string_length = len(text) + len(splits)
9
10        if new_string_length < length:
11            text = text + " " + splits
12        else:
13            text_list.append(text)
14            text = splits
15    text_list.append(text)
16
17    return text_list

```

Listing 4.9: Erstellen Teiluntertitel

Für die Untertitel werden die Abschnitte in einer bestimmten Länge benötigt. Dabei wird der Untertitel in seine einzelnen Wörter zerlegt. Danach wird mit einer Schleife über die einzelnen Wörter gegangen, sodass bei jedem Durchgang das nächste Wort mit einem bisherigen Wort ergänzt wird. Wenn das zusammenhängende Wort und das folgende Wort kleiner als die vorgegebene Länge ist, dann wird das folgende Wort an das zusammengefügte Wort angehängt. Wenn die Kombination der Wörter größer ist, dann wird das zusammengefügte Wort in eine Liste gespeichert und das folgende Wort bildet den Stamm für das nächste zusammenhängende Wort.

#### 4.3.6 Ausgabe PDF

```

1 def generate_pdf(new_list, audio_split_times, transcriptions):
2     # draw to pdf with time
3     canvas = Canvas("test.pdf", pagesize=A4)
4     canvas.setFontSize(20)
5     canvas.drawString(2 * cm, 28.5 * cm, "test pdf")
6     canvas.setFontSize(10)
7     canvas.drawString(2 * cm, 27 * cm,
8                       "Automatisch generiertes PDF Dokument. Es können
9                       Fehler in der Transkription vorhanden sein.")
10
11    page_size = 26

```

Listing 4.10: Erstellen der PDF Initialisierung

In dem Code 4.10 ist zu erkennen, wie das Canvas für die PDF initialisiert wird. Auf die erste Seite wird eine Überschrift und ein kleiner Text eingefügt. Das Schreiben in die PDF funktioniert nach dem Prinzip, dass x und y Koordinaten mit dem gewünschten Text angegeben werden. Die Besonder-

heit ist, dass die (0,0) Koordinate nicht oben links zu finden ist, sondern die Zählung von unten links beginnt.

```

1      for x_function, chunk in enumerate(new_list):
2
3          if page_size < 1.8:
4              page_size = 28
5              canvas.showPage()
6              canvas.setFontSize(10)
7
8          if x_function == 0:
9              if len(transcriptions[x_function]) > 80:
10                 string_list = generate_text_chunks(transcriptions[
11                     x_function], 80)
12
13                 canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
14                     format(new_list[x_function].duration_seconds, 2)
15                     + "; " + "o : " + str(
16                         audio_split_times[0]) + "; " + str(
17                             string_list[0]))
18
19                 for string_number in range(1, len(string_list)):
20                     canvas.drawString(2 * cm, (page_size - (0.6 *
21                         string_number)) * cm, str(string_list[
22                             string_number]))
23
24                 page_size -= 0.6 * len(string_list)
25             else:
26                 canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
27                     format(new_list[x_function].duration_seconds, 2)
28                     + "; " + "o : " + str(
29                         audio_split_times[0]) + "; " + str(
30                             transcriptions[x_function]))
31
32                 page_size -= 0.6

```

Listing 4.11: Erstellen der PDF Schleife Teil 1

Am Anfang des Codes 4.11 ist zusehen, wie über eine Liste mit Zeitangaben der einzelnen Videoabschnitte iteriert wird. Dabei ist die erste Kontrolle in der Schleife, ob das Ende der PDF erreicht ist. Wenn das zutrifft, dann wird die Variable, die die Position in der PDF beobachtet, auf den Anfang der PDF ausgerichtet. Außerdem wird eine neue Seite ausgewählt und die Größe der Schrift wird korrigiert.

Die nächste Überprüfung ist, ob der Zähler an der ersten Stelle ist. Wenn dies zutrifft, dann wird im nächsten Schritt geschaut, ob die Transkription länger als 80 Zeichen ist. Angenommen die Transkription ist länger als 80 Zeichen, so wird diese mit der Funktion `generate_text_chunks` in kleinere Teilstücke aufgeteilt.

In der ersten Reihe jeder Audiodatei wird die Dauer des Segments, Beginn des Segments im Video und das Ende des Abschnittes geschrieben. In den

nächsten Reihen folgen die Teilstücke der Transkription. Die letzte Aufgabe ist die Anpassung der Variable der Position.

```

1      else:
2          if len(transcriptions[x_function]) > 80:
3              string_list = generate_text_chunks(transcriptions[
4                  x_function], 80)
5
6              canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
7                  format(new_list[x_function].duration_seconds, 2)
8                      + "; " + str(audio_split_times[
9                          x_function - 1]) + " : " +
10                         str(audio_split_times[x_function]) + "
11                         ; " + str(string_list[0]))
12
13              for string_number in range(1, len(string_list)):
14                  canvas.drawString(2 * cm, (page_size - (0.6 *
15                      string_number)) * cm, str(string_list[
16                          string_number]))
17
18              page_size -= 0.6 * len(string_list)
19
20      else:
21          canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
22              format(new_list[x_function].duration_seconds, 2)
23                  + "; " + str(audio_split_times[
24                      x_function - 1]) + " : " +
25                  str(audio_split_times[x_function]) + "
26                  ; " + str(transcriptions[
27                      x_function]))
28
29          page_size -= 0.6
30
31      canvas.save()

```

Listing 4.12: Erstellen der PDF Schleife Teil 2

Der Unterschied des Codes von dem Segment 4.11 und 4.12 liegt in der "drawString" Funktion. Diese verwendet als Angabe die Position des Abschnittes im Video. In dem Codesegment 4.11 ist der erste Wert mit null fest implementiert und der zweite Wert ist das erste Element in der Liste.

#### 4.3.7 Ausgabe Video mit Untertitel

```

1  def generate_video(clip_list_function, transcriptions):
2      list_text_clip = []
3      new_video = None
4
5      for x_function, video_clip in enumerate(clip_list_function):
6          # only if length is greater than 65
7          if len(transcriptions[x_function]) > 65:

```

```

8         sub_subtitle = generate_text_chunks(transcriptions[
          x_function], 65)
9
10        sub_parts_list = [0]
11
12        chunk_sub_parts = video_clip.duration / sub_subtitle.__len__
          ()
13
14        for x_2 in range(sub_subtitle.__len__()): # todo "
          uberarbeiten
15            if x_2 < len(sub_subtitle) - 1:
16                sub_parts_list.append((x_2 + 1) * chunk_sub_parts)
17
18        sub_parts_list.append(video_clip.duration)
19
20        for y, sub_subtitles in enumerate(sub_subtitle):
21            text_name = str(sub_subtitles)
22            txt_clip = mp.TextClip(text_name, fontsize=25, color="
          red")
23            txt_clip = txt_clip.set_position('bottom')
24
25            txt_clip = txt_clip.set_duration(chunk_sub_parts)
26
27            sub_clip = video_clip.subclip(sub_parts_list[y],
          sub_parts_list[y + 1])
28
29            list_text_clip.append(mp.CompositeVideoClip([sub_clip,
          txt_clip]))

```

Listing 4.13: Erstellen der Untertitel und Zusammenfügen der Videoabschnitte Teil

1

In dem Codeabschnitt 4.13 ist der erste Teil der for-Schleife zusehen. Bei der Generierung der Untertitel wird auf die Länge des Textes geachtet, da auf dem Video nur 65 Zeichen abgebildet werden können. Wenn der Text länger als 65 Zeichen ist, dann sind Teile des Untertitels außerhalb des Videos. Wenn die Transkription des Abschnittes länger als 65 Zeichen ist, dann wird diese mit der Funktion `generate_text_chunks` unterteilt. Die Funktion gibt eine Liste an Teilstücken zurück.

Daraufhin wird die Länge der zukünftigen Videoabschnitte berechnet. Dafür wird die Videolänge des Abschnittes durch die Anzahl der Elemente in der Liste geteilt. Daraus erfolgen gleichmäßige Stücke. Mit der Anzahl von den gleichmäßigen Stücken und einer Schleife wird eine Liste mit den Werten des Start- und Endpunkten für die zukünftigen Videoabschnitte berechnet. Die folgende Beispielrechnung veranschaulicht die Vorgehensweise. Wenn die Dauer des Videoabschnitts 10 Sekunden ist und die Anzahl der Teilstücke 5 ist, dann ergibt sich der Wert 2 für die gleichmäßigen Stücke. Danach wird mit einer Schleife die Liste so gefüllt, dass sich daraus die Start- und Endpunkte für die 5 Abschnitte ergeben. Die Liste sieht dann folgenderweise aus: [0, 2, 4, 6, 8, 10].

Mit Hilfe einer for-Schleife wird über alle Untertitel gegangen. Dabei werden die Untertitel abgespeichert. Die TextClips werden mit dem Untertitel, Schriftgröße, Farbe, Position und Dauer initialisiert. Danach wird der Videoabschnitt in Segmente mit den vorher bestimmten Grenzen unterteilt. Der generierte TextClip wird mit dem Videosegment verschmolzen.

```

1      else:
2          text_name = str(transcriptions[x_function])
3          txt_clip = mp.TextClip(text_name, fontsize=25, color="red")
4          txt_clip = txt_clip.set_position('bottom')
5
6          if x_function == clip_list_function.__len__() - 1:
7              txt_clip = txt_clip.set_duration(list_chunk_length[
8                  x_function])
9              list_text_clip.append(mp.CompositeVideoClip([video_clip,
10                  txt_clip]))
11          else:
12              txt_clip = txt_clip.set_duration(list_chunk_length[
13                  x_function])
14              list_text_clip.append(mp.CompositeVideoClip([video_clip,
15                  txt_clip]))
16
17      new_video = mp.concatenate_videoclips(list_text_clip, method="
18          compose")
19      print(new_video.duration)
20      new_video.write_videofile("new_video.mp4")

```

Listing 4.14: Erstellen der Untertitel und Zusammenfügen der Videoabschnitte Teil 2

Wenn die Länge des Untertitels kleiner als 65 Zeichen ist, wird lediglich der TextClip erstellt und mit den Videoabschnitt verschmolzen. Die Resultate werden zu einem Video zusammengefügt und als MP4 abgespeichert.

#### 4.3.8 Löschung Audiodateien

```

1  for x in range(number):
2      path = "test_test" + str(x) + ".wav"
3      os.remove(path)

```

Listing 4.15: Löschen der vorübergehend gespeicherten Audiodateien

Für die Löschung der gespeicherten Audiodateien wird mit einer Schleife der Name der Datei erstellt und mit der "remove" Funktion gelöscht.

## 4.4 TEST DES PROOF OF CONCEPT

In dem Abschnitt Test wird die Geschwindigkeit des Programms dargestellt. Daraufhin folgt die Darstellung des Konsoleninterfaces. Die Untertitel des

Video und die generierten PDF werden mit erfolgreicher und falscher Transkription gezeigt.

#### 4.4.1 *Geschwindigkeit des Programms*

Die Geschwindigkeit des Programms wird in zwei Kategorien unterteilt. Die erste Kategorie ist ohne Transkription. Bei der zweiten Kategorie wird die Transkription mitgemessen. Dabei wird bei diesem Test nicht die Genauigkeit der Transkription getestet. Die Genauigkeit ist in Abschnitt 3.6.1 zu finden. Die Geschwindigkeit des Programms wird mit zwei Dateien getestet. Das erste Video hat eine Dauer von 12 Minuten und 47 Sekunden und die zweite Aufzeichnung ist 13:23 Minuten lang.

Die Geschwindigkeit ohne Transkription ist für die Verarbeitung mit ausschließlich der PDF bei 1:12 und 1:23 Minuten. Die Verarbeitungszeit mit Video schwankte von 22 Minuten und 23 Sekunden bis zu 49 Minuten. Der Unterschied kommt durch andere Nutzungen an dem verwendeten Computer. Bei dem schnelleren Durchgang gab es keine anderen Tätigkeiten. Bei der langsameren Zeit wurde der Computer nebenbei normal verwendet.

Die Laufzeit des Programms mit Transkription und nur der Erstellung einer PDF liegt zwischen 9:30 und 10:30 Minuten. Die Laufzeit mit Video schwankt ebenfalls wie bei der Erstellung mit PDF und Video. Die gemessenen Zeiten variieren von 32 Minuten bis zu 1 Stunde. Zu den Gründen kommt dazu, dass das Programm die Abschnitte über das Internet an den Anbieter verschickt. Wenn gleichzeitig andere Services über das Internet verwendet werden, sinkt die maximale Upload-Geschwindigkeit für das Programm.

#### 4.4.2 *Konsolen Interface*

```
(venv) C:\Users\tobia\OneDrive\Desktop\Bachelorarbeit\programm\evaluation>python poc.py -f test.file -l language
filename      = test.file
language      = language
dBFS          = -25
output        = both
```

Abbildung 4.2: Eingabe Konsole

In dem Bild 4.2 ist die Eingabe in die Kommandozeile zusehen. Die Felder filename und language müssen beim Start angegeben werden. Das Felder dBFS und output sind optional. Wenn für das Feld dBFS kein Wert angegeben wird, dann verwendet das Programm -25. Bei output ist das Erstellen des Videos und der PDF Standard.

```
(venv) C:\Users\tobia\OneDrive\Desktop\Bachelorarbeit\programm\evaluation>python poc.py --help
usage: poc.py [-h] -f FILENAME -l LANGUAGE [-d DBFS] [-o OUTPUT] [--version]

optional arguments:
  -h, --help            show this help message and exit
  -f FILENAME            Name of the file
  -l LANGUAGE            Set language. Supported is de and en
  -d DBFS                Can be be between -16 and -25
  -o OUTPUT              Set Output. Can be video, pdf or both
  --version              show program's version number and exit
```

Abbildung 4.3: Konsole Help Funktion

Das Schaubild 4.3 zeigt die Auflistung der help-Funktion. Diese gibt einen Überblick über die notwendigen und optionalen Parameter dieses Programms.

#### 4.4.3 Untertitel des Videos

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

## Evolution of the Clients at the Client-Server Model

- In the early days of the client-server era, clients were pure terminals
- With the increasing computing power, the clients became more and more powerful and more and more tasks have been outsourced to them
- But computing power is a resource of which there is never enough available

Do you know the 3 options to reduce the time, which is required to solve computation-intensive tasks?


35/70 Pro/Con: Reducing the Time to Compute | Show Full Slide

yeah but still no better how much computation power you have

Abbildung 4.4: Video mit Untertitel



Der Ausschnitt 4.4 aus dem Video stellt eine erfolgreiche Transkription anhand des eingefügten Untertitels dar. Bei dem Untertitel lässt sich die Farbe, Schriftgröße, Position und Umrandung ändern bzw. hinzufügen. Somit kann für jedes Video der richtige Untertitel eingefügt werden.




  
Image Source: Google

## Summary about the Clients

- The era of **X-Terminals** and **Text-Terminals** is over
- **Applet Clients** did fail in the 1990s but their popularity may grow in the next years
  - ⇒ Google Chrome OS

- **Fat Clients** are standard today
- **Thin/Zero Clients** are rarely used today
  - Things change slowly in the industry
  - Thin Clients are a hot topic again because of rising energy costs
  - Keyword: Green IT

34/70
Prof. Dr. Eicke Godehardt | Cloud Computing | Summer Semester 2020

Abbildung 4.5: Video mit Fehlermeldung als Untertitel

Wenn bei der Transkription ein Fehler auftritt, dann wird als Untertitel "error" eingeblendet. Dieser wird durch einen neuen Text ersetzt, wenn ein neuer Videoabschnitt beginnt. Wit gibt keine aussagekräftige Antwort zu Fehlern zurück, sodass diese nicht in dem Programm verwendet werden können. Es ist auch die Frage, ob ein Nutzer wissen möchte oder sollte, warum es genau da jetzt zu einem Fehler gekommen ist.

#### 4.4.4 PDF mit Informationen

## Transkription

Automatisch generiertes PDF Dokument. Es können Fehler in der Transkription vorhanden sein.

14.27; 0 : 14.27; what do you think of when i say the word design you probably think of things like this finely crafted objects that you can hold in your hand

Abbildung 4.6: PDF mit Überschrift und Transkription

Die PDF hat den Aufbau von einer Überschrift und einer Erklärung, bevor die Ergebnisse der Transkription zu lesen sind. In jedem Abschnitt einer Transkription sind folgende Informationen vorhanden: Länge des Abschnittes, Start und Ende im Video und die Transkription des Abschnittes. Wenn die Transkription länger als eine Anzahl an Zeichen ist, dann beginnt eine

neue Zeile mit dem restlichen Wörtern der Transkription. Somit kann sich ein Abschnitt über mehrere Zeilen strecken.

28.81; 234.81 : 263.62; error 22.94; 263.62 : 286.56; error 32.33; 286.56 : 318.89; error
---

Abbildung 4.7: PDF mit Fehlermeldung

Wenn es während der Transkription zu einem Fehler kommt, dann wird statt der Transkription ein "error" in die Zeile der PDF geschrieben. Somit ist auch gleich ersichtlich, zu welchem Zeitpunkt der Fehler passiert sind.

## 4.5 DISKUSSION

In der Diskussion werden Themen besprochen, die bei der Implementierung zu Problemen geführt haben oder die immer noch Probleme bereiten. Zu den Themen zählen das Erkennen von Sprechpausen, die Geschwindigkeit der Erstellung des Videos. In den einzelnen Abschnitten werden die Probleme genauer beschrieben und Lösungen vorgeschlagen.

### 4.5.1 Erkennen von Stille in der Tonspur

Die Aufteilung der Audiodatei in Abschnitte wird anhand des dBFS Wertes gemacht. Die Abkürzung steht für "Dezibel Full Scale" und gibt einen absoluten Wert für einen digitalen Tonpegel an [49]. Dieser Wert kann von Aufnahme zu Aufnahme schwanken.

Vorausgesetzt der richtige Wert wird ausgewählt, dann hat das keine Auswirkungen auf die Verarbeitung. Wenn der gewählte Wert von dem dBFS-Werten der Stille des Videos abweicht, dann erkennt die Funktion nicht alle Abschnitte. Anschließend wird nicht die komplette Datei ausgewertet und es fehlen Teilstücke. Der verwendete Wert basiert nicht auf Regeln, sondern ist durch ausprobieren entstanden. In dem Programm wird der dBFS-Wert aus der Audiospur ausgelesen. Um den stillsten Punkt zu treffen, wird ein gewählter Wert abgezogen, der hoffentlich zu den meisten Audioquellen passt.

### 4.5.2 Geschwindigkeit der Erstellung des Videos

Die Funktion concatenate\_videoclips von moviepy ist langsam. Die Geschwindigkeit der Funktion ist bei  $\approx 8$  Bilder pro Sekunde. Wenn ein Video 30 Bil-

der pro Sekunde hat, dann hat ein 15 minütiges Video folgende Anzahl an Bildern:

$$\begin{aligned}
 \text{Anzahl Bilder} &= \text{Anzahl} \frac{\text{Bilder}}{\text{Sekunden}} * \text{Länge Video in Sekunden} \\
 &= 30 \frac{\text{Bilder}}{\text{Sekunden}} * 15 \text{ Minuten} * 60 \\
 &= 30 \frac{\text{Bilder}}{\text{Sekunden}} * 900 \text{ Sekunden} \\
 &= 30 * 900 \frac{\text{Bilder} * \text{Sekunden}}{\text{Sekunden}} \\
 &= 30 * 900 \text{ Bilder} \\
 &= 27.000 \text{ Bilder}
 \end{aligned} \tag{4.1}$$

Mit den 8 verarbeiteten Bildern pro Sekunde ergibt sich folgende Rechnung zu der Dauer:

$$\begin{aligned}
 \text{Dauer in Sekunden} &= \text{Bearbeitung in} \frac{\text{Sekunden}}{\text{Bild}} * \text{Bilder pro Video} \\
 &= 125 \frac{\text{Sekunden}}{\text{Bild}} * 27.000 \text{ Bilder} \\
 &= 125 * 27.000 \frac{\text{Sekunden} * \text{Bilder}}{\text{Bilder}} \\
 &= 125 * 27.000 \text{ Sekunden} \\
 &= 3.375.000 \text{ Sekunden} \\
 &= 56,25 \text{ Minuten}
 \end{aligned} \tag{4.2}$$

In der Rechnung werden Annahmen zu der Leistung des Computers gemacht. Die Dauer gibt einen Richtwert der Erstellung für Videos. Mit einem schnelleren PC kann die Dauer reduziert werden. Bei einer Steigerung der Anzahl der Verarbeitenden Bilder pro Sekunde auf 12, würde sich die Dauer auf  $\approx 34$  Minuten senken.

#### 4.5.3 Verhältnis Tonspur zu Untertitel

Der Untertitel wird zu spät eingeblendet. Die Gründe für dieses Verhalten sind nicht bekannt. Eine Vermutung ist, wenn es viele Sprachpausen in der Tonspur gibt, dann werden diese ausgeschnitten. An den Kanten werden jeweils 150 Millisekunden an dem Anfang und am Ende angehängt. Wenn die Pause länger als 300 Millisekunden ist, dann wird in der Mitte der Tonspur Material ausgelassen. Deswegen passiert es, dass die Tonspur kleiner ist als die Dauer des Videos. Das ist von der Länge des Videos und den Längen der Sprachpausen abhängig.

#### 4.5.4 Verhältnis Gesagten zu Transkription

Die Untertitel sind nicht synchron zum Gesagten. Dass bedeutet, die angezeigten Untertitel werden nicht nach der Position des Gesagten eingeblendet.

Sondern der Abschnitt wird in gleichmäßige Stücke geteilt, auf die dann der Untertitel geschrieben wird. Die Abschnitte spiegeln nicht das Verhältnis der einzelnen Längen der Transkription ab. Die folgende Rechnung zeigt das Problem auf. Es wird davon ausgegangen, dass die Dauer des Abschnittes 10 Sekunden sind. Das Ergebnis der Transkription ist 70 Zeichen lang. Damit ist es länger als die maximale Grenze von 65. Die Transkription wird mit der Funktion `generate_text_chunks` in Segmente von 55 und 15 Zeichen unterteilt. Der Abschnitt des Videos wird in 2 Teile mit jeweils 5 Sekunden unterteilt. Diese 5 Sekunden repräsentieren nicht das Verhältnis der 55 zu 15 wieder. Somit passiert es, dass der nächste Untertitel eingeblendet wird, bevor in der Audiospur das Gesagte auch an diesem Punkt angelangt ist und umgekehrt.

#### 4.5.5 *Speicherung der Audio-Elemente*

Bei der Verarbeitung müssen die Audioabschnitte gespeichert werden, da Wit nur Dateien einlesen kann. Dafür werden in dem Programm alle Elemente mit einer Länge von maximal 20 Sekunden abgespeichert. Diese Lösung ist nicht optimal, da bei langen Dateien von über 1 Stunde viele Teilstücke abgespeichert werden.

Bei der Rechnung wird angenommen, dass eine Vorlesung mit einer Länge von 90 Minuten verarbeitet wird. Es wird eine optimalen Aufteilung von 3 Teilstücken pro Minute angenommen. Dann ergibt sich eine Anzahl von 270 Teilstücken. Wenn das Programm ausgeführt wird, dann werden die Dateien jeweils geschrieben und gelöscht. Dies ist für SSD mit limitierten Anzahl an Löschvorgängen nicht optimal.

## FAZIT

---

### 5.1 ZUSAMMENFASSUNG

Die Arbeit hat das Ziel, die folgenden Fragen zu beantworten:

- Die Evaluation von verschiedenen Spracherkennungssystemen.  
Dabei lag der Fokus auf lokalen und Cloud-basierten Systemen, die durch verschiedene Sprachen auf Genauigkeit und Geschwindigkeit verglichen wurden.
- Das Erstellen eines Proof of Concept eines Spracherkennungssystems zur Verarbeitungen von Vorlesungen.  
Bei dem Proof of Concept war es das Ziel, verschiedene Möglichkeiten zur Anwendung der Technologie bei Universitäten zu zeigen.

Bei der Evaluation der verschiedenen Spracherkennungssysteme wurde ein Sieger festgestellt. Mit Rev hat ein System mit der besten Kombination aus Geschwindigkeit und Genauigkeit mit Abstand gewonnen. Danach folgen mit Google und Azure zwei Systeme, die entweder relativ schnell waren und dann im Vergleich eine geringere Genauigkeit hatten oder eine hohe Genauigkeit und eine langsamere Verarbeitung. Wit ist bei der englischen Sprache ungenau, sonst hätte es in der Auswertung ähnlich gut wie Azure oder Google abgeschnitten. Das Spracherkennungssystem von IBM hat in diesem Test am Langsamsten und Ungenausten performt.

Die Resultate der Transkription von Sphinx4 waren für eine Auswertung nicht verwendbar und ein Vergleich mit der Vorlage nicht möglich. Die Geschwindigkeit hat hier für keine ausschlaggebenden Einfluss gehabt, obwohl diese mehr als 10x langsamer als die Software von IBM ist.

In dem Proof of Concept wurde ein Programm gestaltet, welches die Möglichkeit zur Erstellung von Transkriptionen von Vorlesungsaufzeichnungen auf zwei verschiedene Weisen zeigt.

Die erste Möglichkeit ist das Erstellen einer PDF mit Zeitdaten und Resultat der Transkription. Diese gibt einen Überblick über die komplette Vorlesung und ist schnell erstellt. Bei der zweiten Methoden werden Untertitel in die Videodatei der Aufzeichnung geschrieben. Nach dem Erstellen der Untertitel kann die Aufzeichnung an die Studierenden verteilt werden.

In den erwähnten Studien wurden Ergebnisse von 17.4 und 20.63 gemessen. Die vorgestellten Ergebnisse haben einen durchschnittlichen WER-Wert über alle Programme und Sprachen von 12.34. Damit sind die Ergebnisse niedriger als bei den vergleichbaren Studien.

Durch die Evaluation in dieser Arbeit wurde sichtbar, dass die Firmen ihre Spracherkennungssysteme verbessert haben. Außerdem konnte durch den Proof of Concept ein Programm zur Verwendung der Services mit dem Fokus auf zwei verschiedene Ausgaben gezeigt werden.

## 5.2 WEITERFÜHRENDE FORSCHUNG

Bei der Evaluation müssen mehr Programme betrachtet werden. Leider sind viele Programme nicht kostenlos oder haben keinen kostenlosen Probezeitraum. Außerdem müssen weitere Auswertungen mit anderen Testdaten gemacht werden, um andere Themengebiete und Sprachverhalten untersuchen zu können.

An dem Proof of Concept ist die wichtigste Änderung, die Audiodateien in einen gewissen dBFS-Bereich umzuwandeln. Damit kann das Programm die Aufzeichnungen besser und konstanter in kleinere Stücke aufteilen. Außerdem ergeben sich gleichmäßigere Abschnitte für die Transkription.

Teil II

ANHANG

## PROOF OF CONCEPT CODE

```
1 import os
2 import moviepy.editor as mp
3 from moviepy.video.tools.subtitles import SubtitlesClip
4 from pydub import AudioSegment
5 from pydub.silence import split_on_silence
6 from reportlab.pdfgen.canvas import Canvas
7 from reportlab.lib.pagesizes import A4
8 from reportlab.lib.units import cm
9 import time
10 import datetime
11 from wit import Wit
12 import argparse
13 import sys
14
15
16 def add_audio_together(chunks_function, target_length_function):
17     output_chunks_function = [chunks_function[0]]
18     for chunk in chunks_function[1:]:
19         if len(output_chunks_function[-1]) < target_length_function:
20             output_chunks_function[-1] += chunk
21         else:
22             output_chunks_function.append(chunk)
23     return output_chunks_function
24
25
26 def convert_to_wav(src):
27     video_function = mp.VideoFileClip(str(src))
28     video_function.audio.write_audiofile("testtest.wav")
29     return video_function
30
31
32 def read_wav_file_and_split(src):
33     audio_file_function = AudioSegment.from_wav(src)
34     print("audio file duration: " + str(audio_file_function.
35         duration_seconds))
36
37     dBFS = audio_file_function.dBFS
38     chunks_function = split_on_silence(
39         audio_file_function,
40         min_silence_len=250,
41         silence_thresh=dBFS - 25,
42         keep_silence=150,
43     )
44
45     return chunks_function, dBFS, audio_file_function
```



```

46
47 def save_chunks_temporarily(output_chunks_function):
48     number_function = 0
49     list_chunk_length_function = []
50     for x_function, added_chunk in enumerate(output_chunks_function):
51         added_chunk.export("test_test{o}.wav".format(x_function), format
                             ="wav")
52         number_function = x_function
53
54         list_chunk_length_function.append(round(added_chunk.
55         duration_seconds, 2))
56     return number_function, list_chunk_length_function
57
58 def create_video_chunks(video_function, number_function,
59     added_chunk_list_function):
60     video_length = video_function.duration
61     clip_list_function = []
62     for x_function in range(number_function):
63         if x_function == 0:
64             clip_list_function.append(video_function.subclip(0,
65             added_chunk_list_function[x_function]))
66         else:
67             clip_list_function.append(video_function.
68             subclip(added_chunk_list_function[x_function - 1],
69             added_chunk_list_function[x_function]))
70     return clip_list_function
71
72 def transcribe_snippets(repetitions):
73     if language == "englisch":
74         access_token = "API-Key"
75     elif language == "deutsch":
76         access_token = "API-Key"
77     else:
78         sys.exit("wrong language")
79     client = Wit(access_token=access_token)
80
81     resp = None
82     list_transcriptions_function = []
83
84     for x_function in range(repetitions):
85         try:
86             with open('test_test' + str(x_function) + '.wav', 'rb') as f
87                 :
88                 resp = client.speech(f, {'Content-Type': 'audio/wav'})
89                 list_transcriptions_function.append(resp["text"])
90         except:
91             list_transcriptions_function.append("error")
92
93     return list_transcriptions_function

```

```

92
93 def generate_video(clip_list_function, transcriptions):
94     list_text_clip = []
95     new_video = None
96
97     for x_function, video_clip in enumerate(clip_list_function):
98         if len(transcriptions[x_function]) > 65:
99             sub_subtitle = generate_text_chunks(transcriptions[
100                 x_function], 65)
101
102             sub_parts_list = [0]
103
104             chunk_sub_parts = video_clip.duration / sub_subtitle.__len__
105                 ()
106
107             for x_2 in range(sub_subtitle.__len__()):
108                 if x_2 < len(sub_subtitle) - 1:
109                     sub_parts_list.append((x_2 + 1) * chunk_sub_parts)
110
111             sub_parts_list.append(video_clip.duration)
112
113             for y, sub_subtitles in enumerate(sub_subtitle):
114                 text_name = str(sub_subtitles)
115                 txt_clip = mp.TextClip(text_name, fontsize=25, color="
116                     red")
117                 txt_clip = txt_clip.set_position('bottom')
118
119                 txt_clip = txt_clip.set_duration(chunk_sub_parts)
120
121                 sub_clip = video_clip.subclip(sub_parts_list[y],
122                     sub_parts_list[y + 1])
123
124                 list_text_clip.append(mp.CompositeVideoClip([sub_clip,
125                     txt_clip]))
126
127             else:
128                 text_name = str(transcriptions[x_function])
129                 txt_clip = mp.TextClip(text_name, fontsize=25, color="red")
130                 txt_clip = txt_clip.set_position('bottom')
131
132                 if x_function == clip_list_function.__len__() - 1:
133                     txt_clip = txt_clip.set_duration(list_chunk_length[
134                         x_function])
135                     list_text_clip.append(mp.CompositeVideoClip([video_clip,
136                         txt_clip]))
137                 else:
138                     txt_clip = txt_clip.set_duration(list_chunk_length[
139                         x_function])
140                     list_text_clip.append(mp.CompositeVideoClip([video_clip,
141                         txt_clip]))

```

```

134     new_video = mp.concatenate_videoclips(list_text_clip, method="
        compose")
135     print(new_video.duration)
136     new_video.write_videofile("new_video.mp4")
137
138
139 def fix_length_audio(chunks_, dBFS, audio_file_function, test_function):
140     new_list = []
141     target_length_function = 10 * 1000
142     for chunk in chunks_:
143         if chunk.duration_seconds > 20:
144             chunks_function = split_on_silence(
145                 chunk,
146                 min_silence_len=50,
147                 silence_thresh=dBFS - 23,
148                 keep_silence=150,
149             )
150             new_list.extend(add_audio_together(chunks_function,
151                 target_length_function))
152         else:
153             new_list.append(chunk)
154     t_finish = audio_file_function.duration_seconds
155     t_start = (t_finish - test_function) * 1000
156     splitted_audio = audio_file_function[-t_start:]
157     new_list.append(splitted_audio)
158     return new_list
159
160 def generate_text_chunks(text, length):
161     splitted = text.split(" ")
162
163     text_list = []
164     text = ""
165     for splits in splitted:
166
167         new_string_length = len(text) + len(splits)
168
169         if new_string_length < length:
170             text = text + " " + splits
171         else:
172             text_list.append(text)
173             text = splits
174     text_list.append(text)
175
176     return text_list
177
178
179 def generate_pdf(new_list, audio_split_times, transcriptions):
180     canvas = Canvas("test.pdf", pagesize=A4)
181     canvas.setFontSize(20)
182     canvas.drawString(2 * cm, 28.5 * cm, "Transkription")
183     canvas.setFontSize(10)

```

```

184 canvas.drawString(2 * cm, 27 * cm,
185                  "Automatisch generiertes PDF Dokument. Es koennen
                  Fehler in der Transkription vorhanden sein.")
186 page_size = 26
187
188 for x_function, chunk in enumerate(new_list):
189
190     if page_size < 1.8:
191         page_size = 28
192         canvas.showPage()
193         canvas.setFontSize(10)
194
195     if x_function == 0:
196         if len(transcriptions[x_function]) > 80:
197             string_list = generate_text_chunks(transcriptions[
198                 x_function], 80)
199
200             canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
201                             format(new_list[x_function].duration_seconds, 2)
202                             + "; " + "o : " + str(
203                                 audio_split_times[0]) + "; " + str(
204                                     string_list[0]))
205
206             for string_number in range(1, len(string_list)):
207                 canvas.drawString(2 * cm, (page_size - (0.6 *
208                     string_number)) * cm, str(string_list[
209                         string_number]))
210
211             page_size -= 0.6 * len(string_list)
212         else:
213             canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
214                             format(new_list[x_function].duration_seconds, 2)
215                             + "; " + "o : " + str(
216                                 audio_split_times[0]) + "; " + str(
217                                     transcriptions[x_function]))
218
219             page_size -= 0.6
220
221     else:
222         if len(transcriptions[x_function]) > 80:
223             string_list = generate_text_chunks(transcriptions[
224                 x_function], 80)
225
226             canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
227                             format(new_list[x_function].duration_seconds, 2)
228                             + "; " + str(audio_split_times[
229                                 x_function - 1]) + " : " +
230                             str(audio_split_times[x_function]) + "
231                             ; " + str(string_list[0]))
232
233             for string_number in range(1, len(string_list)):

```

```

220         canvas.drawString(2 * cm, (page_size - (0.6 *
221             string_number)) * cm, str(string_list[
222                 string_number]))
223
224     page_size -= 0.6 * len(string_list)
225
226     else:
227         canvas.drawString(2 * cm, page_size * cm, "{:.2f}".
228             format(new_list[x_function].duration_seconds, 2)
229             + "; " + str(audio_split_times[
230                 x_function - 1]) + " : " +
231             str(audio_split_times[x_function]) + "
232             ; " + str(transcriptions[
233                 x_function]))
234
235     page_size -= 0.6
236
237     canvas.save()
238
239     parser = argparse.ArgumentParser()
240
241     parser.add_argument('-f', action='store', dest='filename', required=True
242         ,
243         help='Name of the file')
244
245     parser.add_argument('-l', action='store', dest='language', required=True
246         ,
247         help='Set language. Supported is de and en')
248
249     parser.add_argument('-d', action='store', dest='dbfs', type=int, default
250         =-25,
251         help='Can be between -16 and -25')
252
253     parser.add_argument('-o', action='store', dest='output', default="both",
254         help='Set Output. Can be video, pdf or both')
255
256     parser.add_argument('--version', action='version', version='%(prog)s 0.1
257         ')
258
259     results_parser = parser.parse_args()
260
261     # convert to wav
262     video = convert_to_wav(results_parser.filename)
263     language = results_parser.language
264     output = results_parser.output
265
266     # read wav file
267     chunks, dbfs_function, audio_file = read_wav_file_and_split("testtest.
268         wav")
269
270     target_length = 15 * 1000

```

```

261 output_chunks = add_audio_together(chunks, target_length)
262
263
264
265 fixed_length_audio_list = fix_length_audio(output_chunks, dbfs_function,
      audio_file, test_3)
266
267
268
269 number, list_chunk_length = save_chunks_temporarily(
      fixed_length_audio_list)
270 number += 1
271
272
273 # generate list for time start
274 added_chunk_list = []
275 for x, value in enumerate(list_chunk_length):
276     if x == 0:
277         added_chunk_list.append(value)
278     else:
279         new_value = added_chunk_list[x - 1] + value
280         new_value = round(new_value, 2)
281         added_chunk_list.append(new_value)
282
283 # create video chunk from the list
284 clip_list = create_video_chunks(video, number, added_chunk_list)
285
286 # add chunks together
287 list_transcription = transcribe_snippets(number)
288
289 if output == "video":
290     print("generate video")
291     generate_video(clip_list, list_transcription)
292 elif output == "pdf":
293     print("generate pdf")
294     generate_pdf(fixed_length_audio_list, added_chunk_list,
      list_transcription)
295 elif output == "both":
296     print("generate video")
297     generate_video(clip_list, list_transcription)
298     print("generate pdf")
299     generate_pdf(fixed_length_audio_list, added_chunk_list,
      list_transcription)
300 else:
301     sys.exit("wrong output format")
302
303 # delete temporary files
304 # delete test_test
305 for x in range(number):
306     path = "test_test" + str(x) + ".wav"
307     os.remove(path)

```

Listing A.1: Kompletter Python Code

## TABELLEN GENAUIGKEIT

## B.1 TECHNISCHE INFORMATIK

Vorlesung Technische Informatik Anfang			
Programme	WER	MER	WIL
IBM	17.55	16.28	21.89
Google	10.95	10.71	14.41
Wit	8.39	8.09	9.18
Azure	6.51	6.37	8.29
Rev	6.85	6.61	8.54

Tabelle B.1: Technische Informatik Anfang

Vorlesung Technische Informatik Mitte			
Programme	WER	MER	WIL
IBM	20.33	19.64	27.44
Google	18.44	17.91	23.18
Wit	12.70	12.39	15.51
Azure	9.64	9.36	12.80
Rev	10.24	9.85	12.89

Tabelle B.2: Technische Informatik Mitte

Vorlesung Technische Informatik Ende			
Programme	WER	MER	WIL
IBM	22.10	20.94	29.04
Google	17.12	16.64	22.65
Wit	15.23	14.73	17.77
Azure	10.68	10.15	13.55
Rev	13.75	13.23	18.09

Tabelle B.3: Technische Informatik Ende

## B.2 STATISTIK

Vorlesung Statistik Anfang			
Programme	WER	MER	WIL
IBM	18.05	17.31	22.72
Google	9.73	9.40	11.78
Wit	9.22	8.90	10.14
Azure	7.59	7.29	8.72
Rev	7.65	7.32	8.91

Tabelle B.4: Statistik Anfang

Vorlesung Statistik Mitte			
Programme	WER	MER	WIL
IBM	32.60	30.99	41.46
Google	19.49	18.91	23.14
Wit	13.71	13.12	15.15
Azure	13.15	12.66	15.53
Rev	16.26	15.44	19.96

Tabelle B.5: Statistik Mitte

Vorlesung Statistik Ende			
Programme	WER	MER	WIL
IBM	22.01	21.34	27.34
Google	10.93	10.75	13.80
Wit	4.55	4.46	5.33
Azure	6.60	6.40	8.00
Rev	8.16	7.95	10.15

Tabelle B.6: Statistik Ende



## B.3 BETRIEBSSYSTEME

Vorlesung Betriebssysteme Anfang			
Programme	WER	MER	WIL
IBM	9.88	9.52	11.99
Google	5.77	5.66	6.92
Wit	7.83	7.74	8.65
Azure	5.83	5.55	6.17
Rev	7.39	7.14	8.28

Tabelle B.7: Betriebssysteme Anfang

Vorlesung Betriebssysteme Mitte			
Programme	WER	MER	WIL
IBM	10.70	10.18	13.35
Google	5.35	5.19	6.84
Wit	6.17	6.05	7.26
Azure	3.49	3.39	4.78
Rev	4.44	4.30	5.71

Tabelle B.8: Betriebssysteme Mitte

Vorlesung Betriebssysteme Ende			
Programme	WER	MER	WIL
IBM	14.02	13.48	17.84
Google	6.68	6.40	8.53
Wit	5.95	5.79	7.37
Azure	4.68	4.51	5.50
Rev	5.95	5.79	7.37

Tabelle B.9: Betriebssysteme Ende

## TABELLEN GESCHWINDIGKEIT

Vergleich Vorlesung Betriebssysteme in Sekunden			
Programme	Anfang	Mitte	Ende
IBM	448.13	488.87	463.74
Google	97.89	92.32	97.50
Wit	146.77	163.58	155.51
Azure	178.91	179.48	176.87
Rev	87.91	111.77	100.71

Tabelle C.1: Vergleich der Geschwindigkeit Vorlesung Betriebssysteme

Vergleich Vorlesung Statistik in Sekunden			
Programme	Anfang	Mitte	Ende
IBM	420.25	420.14	443.71
Google	112.85	118.70	96.62
Wit	102.97	97.37	102.90
Azure	174.70	178.39	178.95
Rev	101.10	100.81	94.62

Tabelle C.2: Vergleich der Geschwindigkeit Vorlesung Statistik

Vergleich Vorlesung Technische Informatik in Sekunden			
Programme	Anfang	Mitte	Ende
IBM	441.26	407.84	408.26
Google	93.73	97.07	92.89
Wit	108.47	98.96	98.14
Azure	179.02	179.43	177.74
Rev	99.57	110.83	99.50

Tabelle C.3: Vergleich der Geschwindigkeit Vorlesung Technische Informatik

## LITERATUR

---

- [1] A.Haridas, R.Marimuthu und V.Sivakumar. *A critical review and analysis on techniques of speech recognition: The road ahead*. URL: [https://www.researchgate.net/publication/323978495\\_A\\_critical\\_review\\_and\\_analysis\\_on\\_techniques\\_of\\_speech\\_recognition\\_The\\_road\\_ahead](https://www.researchgate.net/publication/323978495_A_critical_review_and_analysis_on_techniques_of_speech_recognition_The_road_ahead). [Online](accessed: 12.05.2021).
- [2] A.Morris, V.Meier und P.Green. *From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition*. URL: [https://www.isca-speech.org/archive/archive\\_papers/interspeech\\_2004/i04\\_2765.pdf](https://www.isca-speech.org/archive/archive_papers/interspeech_2004/i04_2765.pdf). [Online](accessed: 08.05.2021).
- [3] A.Vadwala, K.Suthar, Y.Karmakar und N.Pandya. *Survey paper on Different Speech Recognition Algorithm: Challenges and Techniques*. URL: [https://www.researchgate.net/profile/Msayushi-Vadwala/publication/320547133\\_Survey\\_paper\\_on\\_Different\\_Speech\\_Recognition\\_Algorithm\\_Challenges\\_and\\_Techniques/links/59eb1c744585151983c80ee0/Survey-paper-on-Different-Speech-Recognition-Algorithm-Challenges-and-Techniques.pdf](https://www.researchgate.net/profile/Msayushi-Vadwala/publication/320547133_Survey_paper_on_Different_Speech_Recognition_Algorithm_Challenges_and_Techniques/links/59eb1c744585151983c80ee0/Survey-paper-on-Different-Speech-Recognition-Algorithm-Challenges-and-Techniques.pdf). [Online](accessed: 08.05.2021).
- [4] A.Waibel, H.Soltau, T.Schultz, T.Schaaf und F.Metze. *Multilingual Speech Recognition*. URL: <http://www.cs.cmu.edu/~tanja/Papers/WSMmulti.6.pdf>. [Online](accessed: 10.05.2021).
- [5] B.Racoma. *German Interpreting: Key Differences Between German and English*. URL: [https://www.daytranslations.com/blog/german-english-key-differences/#:~:text=However%2C%20there%20are%20several%20similarities,and%2029%25%20similarity%20with%20French.%20Combining%20all%20speakers%20\(native%20speakers,to%20132%2C176%2C520%20who%20speak%20German..](https://www.daytranslations.com/blog/german-english-key-differences/#:~:text=However%2C%20there%20are%20several%20similarities,and%2029%25%20similarity%20with%20French.%20Combining%20all%20speakers%20(native%20speakers,to%20132%2C176%2C520%20who%20speak%20German..) [Online](accessed: 06.05.2021).
- [6] C.Guerrero. *Image: Source-channel model for a speech recognition system*. URL: [https://www.researchgate.net/figure/Source-channel-model-for-a-speech-recognition-system\\_fig8\\_312056224](https://www.researchgate.net/figure/Source-channel-model-for-a-speech-recognition-system_fig8_312056224). [Online](accessed: 20.05.2021).
- [7] C.Kendig. *What is Proof of Concept Research and how does it Generate Epistemic and Ethical Categories for Future Scientific Practice?* URL: [https://www.researchgate.net/publication/277251805\\_What\\_is\\_Proof\\_of\\_Concept\\_Research\\_and\\_how\\_does\\_it\\_Generate\\_Epistemic\\_and\\_Ethical\\_Categories\\_for\\_Future\\_Scientific\\_Practice](https://www.researchgate.net/publication/277251805_What_is_Proof_of_Concept_Research_and_how_does_it_Generate_Epistemic_and_Ethical_Categories_for_Future_Scientific_Practice). [Online](accessed: 28.06.2021).
- [8] E. Chandra. *Image: Speech Recognition Architecture*. URL: [https://www.researchgate.net/figure/Speech-Recognition-Architecture\\_fig1\\_302915903](https://www.researchgate.net/figure/Speech-Recognition-Architecture_fig1_302915903). [Online](accessed: 20.05.2021).

- [9] E.Chandra. *Imgae: MFCC Feature Extraction*. URL: [https://www.researchgate.net/figure/MFCC-Feature-Extraction\\_fig2\\_302915903](https://www.researchgate.net/figure/MFCC-Feature-Extraction_fig2_302915903). [Online](accessed: 20.05.2021).
- [10] F.Chen und K.Jokinen. *Speech Technology: Theory and Applications*. URL: [https://www.academia.edu/25976754/Speech\\_Technology](https://www.academia.edu/25976754/Speech_Technology). [Online](accessed: 10.05.2021).
- [11] H.Yang, C.Oehlke und C.Meinel. *German Speech Recognition: A Solution for the Analysis and Processing of Lecture Recordings*. URL: [https://hpi.de/fileadmin/user\\_upload/fachgebiete/meinel/papers/Web-University/2011\\_Yang\\_ICIS.pdf](https://hpi.de/fileadmin/user_upload/fachgebiete/meinel/papers/Web-University/2011_Yang_ICIS.pdf). [Online](accessed: 11.05.2021).
- [12] I.Maglogiannis, L.Iliadis und E.Pimenidis. *A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems*. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7256403/>. [Online](accessed: 10.05.2021).
- [13] I.McCowan, D.Moore, J.Dines, D.Gatica-Perez, M.Flynn, P.Wellner und H.Bourlard. *On the use of information retrieval measures for speech recognition evaluation*. URL: <https://infoscience.epfl.ch/record/83156>. [Online](accessed: 08.05.2021).
- [14] J.Kim, C.Liu, R.Calvo, K.Wu, K.McCabe, S.Taylor und B.Schuller. *A Comparison of Online Automatic Speech Recognition Systems and the Non-verbal Responses to Unintelligible Speech*. URL: <https://arxiv.org/ftp/arxiv/papers/1904/1904.12403.pdf>. [Online](accessed: 10.05.2021).
- [15] J.Levis und R.Suvorov. *Automatic Speech Recognition*. URL: [https://www.researchgate.net/publication/261287458\\_Automatic\\_Speech\\_Recognition](https://www.researchgate.net/publication/261287458_Automatic_Speech_Recognition). [Online](accessed: 02.06.2021).
- [16] J.Tebelskis. *Speech Recognition using Neural Networks*. URL: <https://isl.anthropomatik.kit.edu/pdf/Tebelskis1995.pdf>. [Online](accessed: 20.06.2021).
- [17] M.Anusuya und S.Katti. *Speech Recognition by Machine: A Review*. URL: <https://arxiv.org/ftp/arxiv/papers/1001/1001.2267.pdf>. [Online](accessed: 05.05.2021).
- [18] O.Khalifa, K.El-Darymli, A.Abdullah und J.Daoud. *Statistical Modeling for Speech Recognition*. URL: [https://www.researchgate.net/publication/272355418\\_Statistical\\_Modeling\\_for\\_Speech\\_Recognition](https://www.researchgate.net/publication/272355418_Statistical_Modeling_for_Speech_Recognition). [Online](accessed: 05.05.2021).
- [19] R.Errattahi, A.El Hannani und H.Ouahmane. *Automatic Speech Recognition Errors Detection and Correction: A Review*. URL: <https://www.sciencedirect.com/science/article/pii/S1877050918302187>. [Online](accessed: 20.06.2021).
- [20] R.Nissen. *Evaluation*. URL: <https://wirtschaftslexikon.gabler.de/definition/evaluation-32471>. [Online](accessed: 02.06.2021).
- [21] S.Cook. *Speech Recognition HOWTO*. URL: <https://www.iitk.ac.in/LDP/HOWTO/Speech-Recognition-HOWTO/>. [Online](accessed: 10.05.2021).

- [22] S.Kafle und M.Huenerfauth. *Evaluating the Usability of Automatically Generated Captions for People who are Deaf or Hard of Hearing*. URL: <https://arxiv.org/ftp/arxiv/papers/1712/1712.02033.pdf>. [Online](accessed: 20.06.2021).
- [23] S.Karpagavalli und E.Chandra. *A Review on Automatic Speech Recognition Architecture and Approaches*. URL: [https://www.researchgate.net/publication/302915903\\_A\\_Review\\_on\\_Automatic\\_Speech\\_Recognition\\_Architecture\\_and\\_Approaches](https://www.researchgate.net/publication/302915903_A_Review_on_Automatic_Speech_Recognition_Architecture_and_Approaches). [Online](accessed: 12.05.2021).
- [24] Unbekannt. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?* URL: <https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>. [Online](accessed: 20.06.2021).
- [25] Unbekannt. *Automatic Speech Recognition - DeepAI.org*. URL: <https://deepai.org/machine-learning-glossary-and-terms/automatic-speech-recognition>. [Online](accessed: 02.06.2021).
- [26] Unbekannt. *CMUSphinx - Versionen*. URL: <https://cmusphinx.github.io/wiki/versions/>. [Online](accessed: 20.06.2021).
- [27] Unbekannt. *CMUSphinx Website*. URL: <https://cmusphinx.github.io/wiki/>. [Online](accessed: 20.06.2021).
- [28] Unbekannt. *Formant Definition*. URL: <https://www.dwds.de/wb/Formant>. [Online](accessed: 30.06.2021).
- [29] Unbekannt. *Google Speech-to-Text Website*. URL: <https://cloud.google.com/speech-to-text?hl=de>. [Online](accessed: 30.06.2021).
- [30] Unbekannt. *Graphem Definition*. URL: <https://www.mediensprache.net/de/basix/lexikon/?qu=Graphem>. [Online](accessed: 30.06.2021).
- [31] Unbekannt. *HTTP API Reference - Wit*. URL: <https://wit.ai/docs/http/20200513/>. [Online](accessed: 02.06.2021).
- [32] Unbekannt. *IBM Watson Speech to Text Website*. URL: <https://www.ibm.com/de-de/cloud/watson-speech-to-text>. [Online](accessed: 30.06.2021).
- [33] Unbekannt. *JiWER Dokumentation*. URL: <https://pypi.org/project/jiwer/>. [Online](accessed: 20.06.2021).
- [34] Unbekannt. *Mel Definition*. URL: <https://de.wikipedia.org/wiki/Mel>. [Online](accessed: 30.06.2021).
- [35] Unbekannt. *Microsoft Azure Spracherkennung Website*. URL: <https://azure.microsoft.com/de-de/services/cognitive-services/speech-to-text/>. [Online](accessed: 30.06.2021).
- [36] Unbekannt. *N-Gramm Definition*. URL: <https://fortext.net/ueberfortext/glossar/n-gramm>. [Online](accessed: 30.06.2021).
- [37] Unbekannt. *Phonem - Linguistikarium Uni Graz*. URL: <https://www-gewi.uni-graz.at/gralis/Linguistikarium/Phonetik/Phonem-Wikipedia.html>. [Online](accessed: 30.06.2021).

- [38] Unbekannt. *Python Time Dokumentation*. URL: <https://docs.python.org/3/library/time.html>. [Online](accessed: 20.06.2021).
- [39] Unbekannt. *ReportLab Website*. URL: <https://www.reportlab.com/dev/intro/>. [Online](accessed: 20.06.2021).
- [40] Unbekannt. *Rev.ai Speech-to-Text Website*. URL: <https://www.rev.ai/>. [Online](accessed: 30.06.2021).
- [41] Unbekannt. *Satz von Bayes Definition*. URL: [https://de.wikipedia.org/wiki/Satz\\_von\\_Bayes](https://de.wikipedia.org/wiki/Satz_von_Bayes). [Online](accessed: 30.06.2021).
- [42] Unbekannt. *Spektrogramm Definition*. URL: <https://www.wortbedeutung.info/Spektrogramm/>. [Online](accessed: 30.06.2021).
- [43] Unbekannt. *TED Talk Margaret Gould Stewart*. URL: [https://www.ted.com/talks/margaret\\_gould\\_stewart\\_how\\_giant\\_websites\\_design\\_for\\_you\\_and\\_a\\_billion\\_others\\_too#t-1923](https://www.ted.com/talks/margaret_gould_stewart_how_giant_websites_design_for_you_and_a_billion_others_too#t-1923). [Online](accessed: 20.06.2021).
- [44] Unbekannt. *TED Terms of Service*. URL: <https://www.ted.com/about/our-organization/our-policies-terms/ted-talks-usage-policy>. [Online](accessed: 20.06.2021).
- [45] Unbekannt. *The differences between English and German*. URL: <http://esl.fis.edu/grammar/langdiff/german.htm>. [Online](accessed: 06.05.2021).
- [46] Unbekannt. *What Are the Benefits of Speech Recognition Technology?* URL: [WhatAretheBenefitsofSpeechRecognitionTechnology?](https://www.ibm.com/cloud/learn/speech-recognition#toc-key-featur-DdHYi0BA). [Online](accessed: 28.06.2021).
- [47] Unbekannt. *What Are the Benefits of Speech Recognition Technology?* URL: <https://www.ibm.com/cloud/learn/speech-recognition#toc-key-featur-DdHYi0BA>. [Online](accessed: 28.06.2021).
- [48] Unbekannt. *Wit Spracherkennung Website*. URL: <https://wit.ai/>. [Online](accessed: 30.06.2021).
- [49] Unbekannt. *dBFS - BET-Fachwörterbuch*. URL: <https://www.bet.de/lexikon/dbfs>. [Online](accessed: 20.06.2021).
- [50] Unbekannt. *frikativ Definition*. URL: <https://www.dwds.de/wb/frikativ>. [Online](accessed: 30.06.2021).
- [51] Unbekannt. *moviepy Website*. URL: <https://moviepy.readthedocs.io/en/latest/index.html>. [Online](accessed: 20.06.2021).
- [52] Unbekannt. *nasalieren Definition*. URL: <https://www.dwds.de/wb/nasalieren>. [Online](accessed: 30.06.2021).
- [53] Unbekannt. *pydub Website*. URL: <http://pydub.com/>. [Online](accessed: 20.06.2021).
- [54] W.Ghai und N.Singh. *Literature Review on Automatic Speech Recognition*. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.445&rep=rep1&type=pdf>. [Online](accessed: 02.06.2021).

- [55] X.Huang und L.Deng. *An Overview of Modern Speech Recognition*. URL: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Book-Chap-HuangDeng2010.pdf>. [Online](accessed: 05.05.2021).