Bachelorarbeit

Integration von Sprachsteuerung in Mobile Apps

Rubén Nuñez

Herbstsemester 2023

Bachelorarbeit an der Hochschule Luzern – Informatik

Titel: Integration einer Sprachsteuerungsfunktion in Mobile Apps

Studentin/Student: Ruben Nuñez

Studiengang: BSc Informatik

Jahr: 2023

Betreuungsperson: Dr. Florian Herzog

Expertin/Experte: xxx

Auftraggeberin/Auftraggeber: Stefan Reinhard, Bitforge AG

Codierung / Klassifizierung der Arbeit:
☒ Öffentlich (Normalfall)
☐ Vertraulich

Eidesstattliche Erklärung Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt habe, alle verwendeten Quellen, Literatur und andere Hilfsmittel angegeben habe, wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht habe, das Vertraulichkeitsinteresse des Auftraggebers wahren und die Urheberrechtsbestimmungen der Hochschule Luzern respektieren werde.

Ort / Datum, Unterschrift _____

Abgabe der Arbeit auf der Portfolio Datenbank:
Bestätigungsvisum Studentin/Student
Ich bestätige, dass ich die Bachelorarbeit korrekt gemäss Merkblatt auf der Portfolio Datenbank abgelegt habe. Die Verantwortlichkeit sowie die Berechtigungen habe ich abgegeben, so dass ich keine Änderungen mehr vornehmen kann oder weitere Dateien hochladen kann.

Ort / Datum, Unterschrift _____

Verdankung gibt ein separiertes Kapitel dazu

Ausschliesslich bei Abgabe in gedruckter Form: Eingangsvisum durch das Sekretariat auszufüllen

Rotkreuz, den _____ Visum: _____

Abstract

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Wake-up Wörter innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

El sol es grande

Inhaltsverzeichnis

1 Problem, Fragestellung, Vision

1.1 Problemstellung

Die Sprachsteuerung birgt grosses Potenzial und wird bisher vor allem für Sprachsteuerungsassistenten genutzt. Während es etablierte Sprachassistenten wie Siri, Google oder Alexa gibt, fehlt es an Lösungen für eine integrierte Sprachsteuerung in Mobile Apps, insbesondere in Bezug auf das Erkennen von Triggerwörtern. Daher besteht ein grosses Ausbaupotenzial für Sprachassistenten in Mobile Apps.

Für die Problemstellung dieser Arbeit ist es wichtig, die verschiedenen Disziplinen der Spracherkennung zu differenzieren. Diese Arbeit befasst sich vorallem mit der *Trigger Word Detection (TWD)*. Die nachfolgende Abbildung 1 zeigt die wichtigsten Teilgebiete der Spracherkennung und soll einen Überblick über die Thematik vermitten.

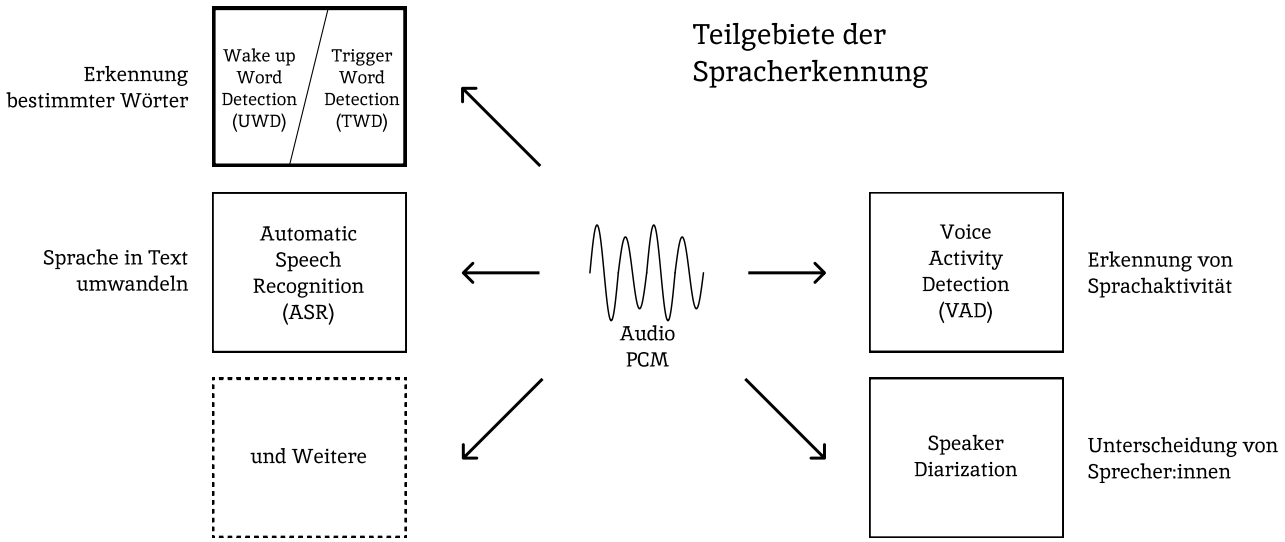

Abbildung 1: Teilgebiete der Spracherkennung

1.2 Fragestellung

Wie kann eine integrierte Sprachsteuerung für eine Mobile App entwickelt werden, welche das Erkennen von Triggerwörtern ermöglicht und wie kann diese Sprachsteuerung in eine Mobile App integriert werden?

1.3 Vision

Als Vision dieser Arbeit soll eine Grundlage geschaffen werden, um ein Triggerwort oder eine Sequenz von Triggerwörtern in der akustischen Sprache erkennen zu können. Dabei sollen Methoden und Werkzeuge aus dem Bereich des Machine Learning verwendet werden. Zudem sollen die gewonnenen Erkenntnise in eine mobile Plattform wie iOS oder Android integriert werden. Für den Rahmen dieser Arbeit genügt die Integration in eine der genannten Plattformen. Weiterhin soll auch das Thema Datenschutz und die ethischen Aspekte zu berücksichtigen werden.

1.4 Rechtfertigung

Siri sowie Google selbst stellen API's zur Verfügung, um Apps zu öffnen oder auch direkte Befehle auszuführen. Daher stellt sich die Frage, warum überhaupt eine eigene In App Sprachsteuerung? Wenn ein Sprachassistent wie Siri bereits API's zur Verfügung stellt, um beispielsweise eine App zu öffnen oder auch direkte Befehle auszuführen. Als Beispiel hat Spotify eine Integration mit Siri. Es können

Befehle wie “Hey Siri, play music on Spotify” verwendet werden. Um nur einiges zu nennen, gibt es folgende Gründe, weshalb eine eigene In App Sprachsteuerung sinnvoll ist:

- **Individualisierbar**: Anpassbare Sprachsteuerung für die App.
- **Unabhängig**: Keine Abhängigkeit von externen Sprachassistenten.
- **Erweiterte Funktionen**: Bietet Funktionen, die über Siri hinausgehen.
- **Datenschutz**: Läuft lokal für mehr Datensicherheit.
- **Branding**: Verstärkt das Markenbild der App.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für die Arbeit erläutert. Dabei wird auf die Themen Audioverarbeitung und Machine Learning eingegangen.

2.1 Audio

In der digitalen Welt werden Schallwellen, die durch eine Reihe von numerischen Werten repräsentiert. (Somberg et al., 2019, p.9) beschreibt Audio als: „Fundamentally, audio can be modeled as waves in an elastic medium. In our normal everyday experience, the elastic medium is air, and the waves are air pressure waves.“ Audiosignale werden durch die Funktion $A(t)$ repräsentiert, wobei t die Zeit und $A(t)$ die Amplitude zum Zeitpunkt t angibt. Die Amplitude ist die Stärke des Signals und die Zeit repräsentiert die Position des Signals. Grundsätzlich ist Audio ein kontinuierliches Signal, in der digitalen Welt können jedoch nur diskrete Werte dargestellt werden. Das kontinuierliche Signal muss somit in diskrete Werte umgewandelt werden. Dieser Vorgang wird als *Sampling* bezeichnet (Tarr, 2018, Chapter 3.1).

2.1.1 Sampling

Ein früher Ansatz zur digitalen Darstellung von analogen Signalen war die Pulse-Code-Modulation (PCM). Dieses Verfahren wurde bereits in den 1930er Jahren von Alec H. Reeves, parallel zum Aufkommen der digitalen Telekommunikation, entwickelt (Deloraine und Reeves, 1965, p. 57). Im Grundsatz wird es heute noch in modernen Computersystemen nach dem gleichen Verfahren angewendet.

Es folgt eine Definition von Sampling. Ein kontinuierliches Signal $A(t)$ wird in bestimmten Zeitintervallen T_s gesampelt. Diese Zeitintervalle werden auch als Sampling-Periode bezeichnet. Die Sampling-Rate $F_s = \frac{1}{T_s}$ gibt die Anzahl der Samples pro Sekunde an. Angenommen wir haben ein Signal mit einer Sampling-Periode von $T_s = 0.001$. Um nun die Sampling-Rate zu berechnen, müssen wir den Kehrwert der Sampling-Periode berechnen. $F_s = \frac{1}{0.001} = 1000$. Somit erhalten wir eine Sampling-Rate von 1000 Samples pro Sekunde. Typische Sampling-Raten sind 44100 Hz oder 48000 Hz. Ein Hertz entspricht einer Frequenz von einem Sample pro Sekunde. Ein weiter wichtiger Begriff ist die *Nyquist-Frequenz*. Die Nyquist-Frequenz F_n ist die Hälfte der Sampling-Rate $F_n = \frac{F_s}{2}$ und muss mindestens doppelt so hoch sein wie die höchste Frequenz des Signals. Ist diese Eigenschaft erfüllt, kann das Signal ohne Informationsverlust rekonstruiert werden (Tarr, 2018, Chapter 3.1). Mehr dazu folgt im Unterkapitel *Fourier-Analyse*.

Weiter ist es wichtig zu verstehen, dass ein Sample ein diskreter Wert ist und in digitalen Systemen durch eine bestimmte Anzahl von Bits dargestellt wird. Die Anzahl der Bits wird als *Bit-Depth* bezeichnet und bestimmt die Auflösung des Signals. Typische Bit-Depth Werte sind 16 oder 24 Bit (Somberg et al., 2019, p.10).

2.1.2 Frames, Channels, Buffers

Ebendfalls wichtig ist das Verständnis von Frames, Channels und Buffers. Da diese Arbeit sich mit Audio-Systemen beschäftigt, ist es wichtig, die Begriffe *Frame*, *Channel* und *Buffer* zu verstehen. Fangen wir mit dem Begriff *Channel* an. Ein Channel kann als ein einzelnes Audio-Signal verstanden werden. Ein Mono-Signal hat genau nur einen Channel. Ein Stereo-Signal hat zwei Channels. Ein Surround-Signal hat mehr als zwei Channels. usw. Nun zum Begriff *Frame*. Ein Frame entspricht einem Sample pro Channel. Weiter sind Frames in Buffers organisiert. Ein Buffer ist eine Sammlung von Frames. Typischerweise werden Buffers in Grössen von 64, 128, 256, 512 oder 1024 Frames organisiert. Die Abbildung 2 zeigt die Beziehung zwischen Frames, Channels und Buffers. Die Abbildung wurde basierend auf (Somberg et al., 2019, p.10) erstellt und verdeutlicht die Beziehung zwischen Frames, Channels und Buffers.

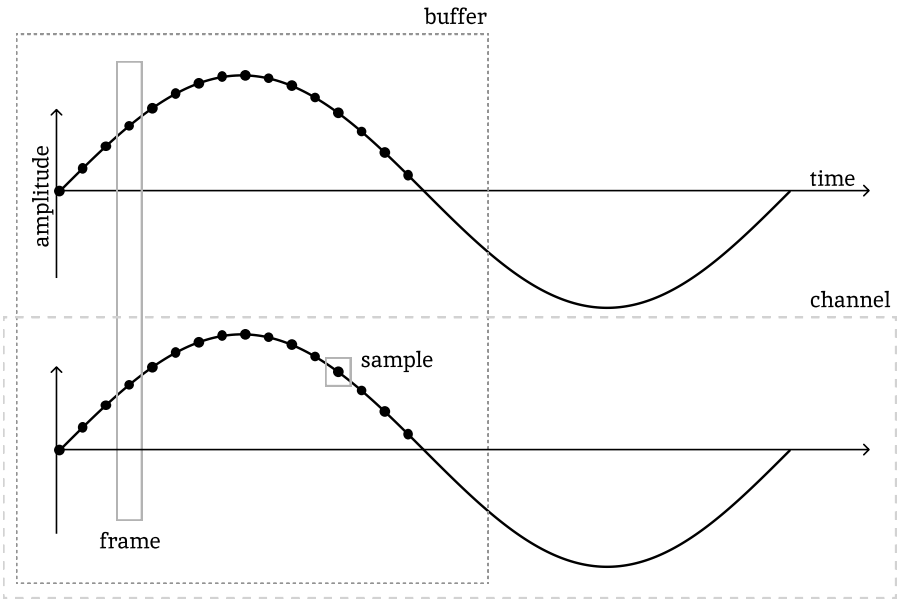

Abbildung 2: Frames, Channels und Buffers

2.1.3 Buffers im Detail

Ein Buffer im Kontext von Audio ist eine aufeinanderfolgende Sammlung von Frames. Die bereits angesprochene Grösse eines Buffers bestimmt im wesentlichen die Latenzzeit des Systems. Kleine Buffer-Grössen haben eine geringe Latenzzeit, während grosse Buffer-Grössen eine hohe Latenzzeit haben (Somberg et al., 2019, p.10). Der Trade-Off ist dass kleine Buffer-Grössen zu einer höheren CPU-Auslastung führen, während bei grossen Buffer-Grössen das nicht der Fall ist. Das liegt daran, dass bei kleinen Buffer-Grössen die CPU häufiger aufgerufen wird, um die Buffers zu verarbeiten.

Nun betrachten wir die mögliche Anordnung eines Buffers, wie in der folgenden Tablle 1 dargestellt. Es gibt zwei Möglichkeiten, wie Buffers angeordnet werden können: *Interleaved* und *Non-Interleaved*. Bei der *Interleaved*-Anordnung werden die Samples der einzelnen Channels nacheinander in sequentieller Reihenfolge in den Buffer geschrieben. Im Gegensatz dazu werden bei der *Non-Interleaved*-Variante die Samples eines Channels nacheinander in den Buffer geschrieben, bevor die Samples des nächsten Channels hinzugefügt werden. Dieser Vorgang wird für jeden Channel wiederholt. Die Tabelle 1 zeigt die Unterschiede zwischen den beiden Anordnungen. Jede Zelle der Tabelle entspricht einem Sample. L und R stehen exemplarisch für die Channels Left und Right. Die erste Zeile entspricht der *Interleaved*-Anordnung und die zweite Zeile der *Non-Interleaved*-Anordnung. Die Abbildung wurde basierend auf (Somberg et al., 2019, p.11) erstellt.

L	R	L	R	L	R	L	R
L	L	L	L	R	R	R	R

Tabelle 1: Frames in Interleaved und Non-interleaved Buffers

Mit diesem Wissen kennen wir nun die Unterschiede zwischen den beiden Anordnungen. Für die Anwendung ist es wichtig zu verstehen, mit welcher Anordnung die verwendete API arbeitet.

2.2 Audio APIs

Audio APIs sind im Bereich der Audioverarbeitung von essentieller Bedeutung. Sie bieten eine Schnittstelle, welche den Zugriff auf vielfältige Audiofunktionen erlaubt. Ohne solche APIs müssten Entwickler die Audioverarbeitung von Grund auf neu implementieren.

In der Erarbeitungsphase dieser Arbeit kristallisierten sich zwei primäre Anwendungsgebiete heraus.

Erstens die intensive Auseinandersetzung mit Audioverarbeitung in Python, um tieferes Verständnis für die Materie zu entwickeln. Zweitens die Notwendigkeit, eine Audio API in eine mobile Applikation zu integrieren. Im Kontext dieser Arbeit werden sie als *Audio API für Analyse* und *Audio API für Integration* bezeichnet.

Im Bereich der *Analyse* fiel die Wahl auf folgende APIs:

- **PyAudio:** Eine verbreitete Schnittstelle in Python zur Audioverarbeitung.
- **SoundDevice:** Eine vielseitige Python-Bibliothek für Audioverarbeitungsaufgaben.
- **librosa:** Eine Bibliothek, die speziell auf die Analyse von Audiosignalen ausgerichtet ist.

Im Kontext der *Integration* standen folgende APIs im Fokus:

- **AVAudioEngine:** Eine leistungsfähige Schnittstelle primär für die Plattformen iOS und macOS.
- **AudioTrack:** Eine spezialisierte API für Audioanwendungen auf Android-Geräten.

Die folgenden Abschnitte werden tiefer auf jeweils eine API pro Anwendungsgebiet eingehen. Insbesondere wird auf die Echtzeitverarbeitung von Audiodaten eingegangen und Beispiele dazu gezeigt.

2.2.1 Audio API für Analyse

Python zeichnet sich durch eine beeindruckende Auswahl an Bibliotheken für datenanalytische Aufgaben aus, zu denen auch NumPy, SciPy, Pandas und Matplotlib gehören. In dieser Arbeit wurde zunächst **PyAudio** in Erwägung gezogen. PyAudio ist als Schnittstelle zur PortAudio-Bibliothek bekannt, die plattformübergreifende Audioverarbeitungsfunktionen bereitstellt. Trotz ihrer intuitiven Funktionen für Aufnahme und Wiedergabe wurde PyAudio letztlich aufgrund von Inkompatibilitäten mit der gewählten Entwicklungsumgebung verworfen.

In dieser Arbeit wurde **SoundDevice** als Alternative zu PyAudio verwendet. Daher wird eine kurze Implementierung von SoundDevice gezeigt, welches das kontinuierliche Streamen von Audio in Chunks ermöglicht.

```
import sounddevice as sd

def stream_audio(chunk_duration, samplerate=16000):
    """Stream audio in chunks."""
    with sd.InputStream(samplerate=samplerate, channels=1) as stream:
        while True:
            audio_chunk, _ = stream.read(int(samplerate * chunk_duration))
            yield audio_chunk
```

Mit praktisch fünf Zeilen Code kann ein Audio-Stream in Chunks mit einer bestimmten Dauer erzeugt werden. Diese Einfachheit und Abstraktion ist ein grosser Vorteil von SoundDevice. Wie wir sehen werden, wird die Implementierung mit der **AVAudioEngine** API für iOS und macOS nicht so einfach sein.

2.2.2 Audio API für Integration

Dieser Abschnitt befasst sich mit dem Thema Audio API für Integration in eine iOS App. Die Wahl fiel auf die **AVAudioEngine** API für iOS und macOS. Im folgenden Code-Beispiel wird gezeigt, wie ein kontinuierlicher Audio-Stream mit AVAudioEngine aufgenommen werden kann.

```swift
public func startTappingMicrophone() {
    let inputNode = audioEngine.inputNode
    let inputFormat = inputNode.outputFormat(forBus: 0)

    guard let recordingFormat = AVAudioFormat(
        commonFormat: .pcmFormatFloat32,
        sampleRate: Double(sampleRate),
        channels: 1,
        interleaved: false
    ), let formatConverter = AVAudioConverter(from:inputFormat, to: recordingFormat) else { return }

    // installs a tap on the audio engine and specifying the buffer size and the input format.
    inputNode.installTap(
                    onBus: 0,
                    bufferSize: AVAudioFrameCount(recordingFormat.sampleRate * bufferTimeInterval),
                    format: inputFormat) {
        buffer, _ in

        self.conversionQueue.async { [self] in
            guard let pcmBuffer = AVAudioPCMBuffer(
                pcmFormat: recordingFormat,
                frameCapacity: AVAudioFrameCount(recordingFormat.sampleRate * bufferTimeInterval)
            ) else { return }

            let inputBlock: AVAudioConverterInputBlock = { _, outStatus in
                outStatus.pointee = AVAudioConverterInputStatus.haveData
                return buffer
            }

            var error: NSError?
            formatConverter.convert(to: pcmBuffer, error: &error, withInputFrom: inputBlock)

            if let error = error {
                print(error.localizedDescription)
                return
            }
            if let channelData = pcmBuffer.floatChannelData {
                let channelDataValue = channelData.pointee
                let channelDataValueArray = stride(
                    from: 0,
                    to: Int(pcmBuffer.frameLength),
                    by: buffer.stride
                ).map { channelDataValue[$0] }

                self.delegate?.audioInputManager(self, didCaptureChannelData: channelDataValueArray)
            }
        }
    }
}
```

Abbildung 3: AVAudioEngine

Im Gegensatz zu SoundDevice ist die Implementierung mit AVAudioEngine nicht so abstrakt. Man muss einige Konzepte verstehen, um die API effektiv nutzen zu können. Beispielsweise wird in diesem Code eine Konvertierung der Samplerate von 48000 Hz auf 16000 Hz durchgeführt. Ebenfalls müssen die Buffer-Grösse, Bit-Depth, Channels und die Anordnung der Frames definiert werden.

2.3 Fourier-Analyse

Die Fourier-Analyse befasst sich mit der Zerlegung von Funktionen in Frequenzkomponenten. Die Fourier-Analyse ist ein wichtiges Konzept in der Signalverarbeitung und findet breite Anwendung in der Audioverarbeitung. Daher ist ein Grundverständnis für diese Arbeit relevant.

2.3.1 Fourier-Transformation

Die Fourier-Transformation ist ein zentrales Werkzeug der Fourier-Analyse. Sie ermöglicht die Zerlegung von Funktionen in ihre Frequenzkomponenten und die Rekonstruktion von Funktionen aus diesen Komponenten. Dies wird als Fourier-Analyse und Fourier-Synthese bezeichnet. Dieses Konzept wird auch von Prof. Dr. Weitz in seinem Video zu Fourier-Analyse erläutert (Weitz, 2023, 2:20). Mathematisch ausgedrückt wird die kontinuierliche Fourier-Transformation eines Signals $f(t)$ wie folgt definiert (Hansen, 2014, Chapter 5):

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt$$

$F(\omega)$ ist die Fourier-Transformation von $f(t)$ (Weitz, 2023, 49:27). Als kleines Rechenbeispiel betrachten wir die Fourier-Transformation der Rechteckfunktion $\text{rect}(x)$, die wie folgt definiert ist:

$$\text{rect}(x) = \begin{cases} 1 & \text{für } -1 \leq x \leq 1 \\ 0 & \text{sonst} \end{cases}$$

Die Fourier-Transformation der Funktion $\text{rect}(x)$ kann wie folgt berechnet werden:

$$\begin{aligned} F(\omega) &= \int_{-\infty}^{\infty} \text{rect}(x) e^{-i\omega x}\, dx \\ &= \int_{-1}^{1} e^{-i\omega x}\, dx \\ &= \frac{1}{-i\omega} \left[e^{-i\omega x} \right]_{-1}^{1} \\ &= \frac{1}{-i\omega} \left(e^{-i\omega} - e^{i\omega} \right) \\ &= \frac{1}{-i\omega} \left(\cos(\omega) - i\sin(\omega) - \cos(\omega) - i\sin(\omega) \right) \\ &= \frac{1}{-i\omega} \left(-2i\sin(\omega) \right) \\ &= \frac{2\sin(\omega)}{\omega} \end{aligned}$$

Somit ist die Fourier-Transformation der Rechteckfunktion $\text{rect}(x)$ gleich $\dfrac{2\sin(\omega)}{\omega}$.

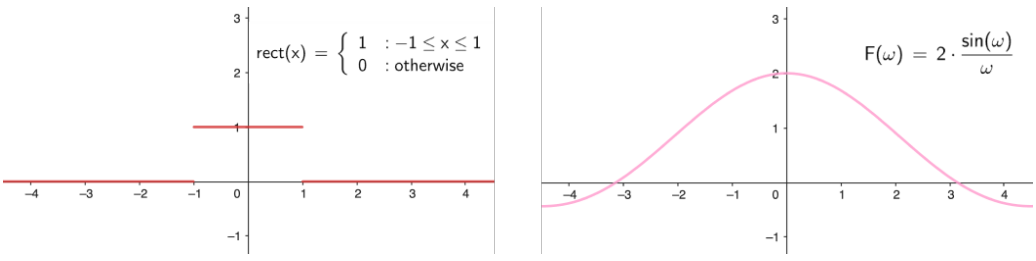

Abbildung 4: Rechteckfunktion und ihre Fourier-Transformation

Die Abbildung 4 stellt die $\text{rect}(x)$ Funktion und ihre Fourier-Transformation, die als $\text{sinc}(\omega)$ bezeichnet wird, dar. Die Nullstellen $\pm\pi, \pm 2\pi, \pm 3\pi, \ldots$ der $\text{sinc}(\omega)$ Funktion deuten darauf hin, dass die $\text{rect}(x)$

Funktion bei diesen Frequenzen keine Energie besitzt. Die primäre Energie der Funktion liegt bei $\omega = 0$. Beispiel adaptiert von (Hansen, 2014, Chapter 5 - Example 5.1).

2.3.2 Diskrete Fourier-Transformation

Die diskrete Fourier-Transformation (DFT) stellt eine diskrete Variante der kontinuierlichen Fourier-Transformation dar und wird speziell auf diskrete Signale angewendet. In digitalen Systemen sind Signale typischerweise diskret und bestehen aus einzelnen Samples, weshalb die DFT besonders relevant für solche Anwendungen ist. Die mathematische Definition der DFT ist (Hansen, 2014, Chapter 3):

$$F(k) = \sum_{n=0}^{N-1} f(n) \cdot e^{-\frac{2\pi i}{N} kn}$$

Zur Veranschaulichung betrachten wir ein Code-Beispiel. Wir haben eine Funktion $f(t)$ und unterteilen diese in N Samples. Die DFT berechnet nun die Frequenzkomponenten des Signals. Die Abbildung 5 zeigt ein Beispiel für ein Signal $f(t)$ mit $N = 5$ Samples.

$$f(t) = 1.5\cos(t) + 0.25\sin(t) + 2\sin(2t) + \sin(3t)$$

.

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return 1.5 * np.cos(x) + 0.25 * np.sin(x) + 2 * np.sin(2*x) + np.sin(3*x)

N_SAMPLES = 5

x_curve = np.linspace(0, 2*np.pi, 100) # 100 Punkte zwischen 0 und 2pi
x_points = np.linspace(0, 2*np.pi, N_SAMPLES) # 5 Punkte zwischen 0 und 2pi

plt.plot(x_curve, f(x_curve))
plt.plot(x_points, f(x_points), 'o') # Plotte die 5 Punkte
```

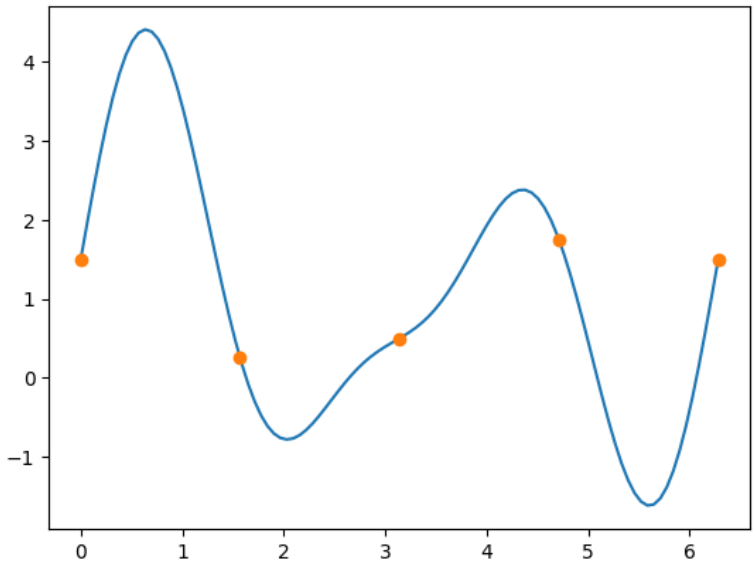

Abbildung 5: Funktion $f(x)$ mit 5 Samples

Nun berechnen wir die DFT der 5 Samples. Dazu verwenden wir die `fft` Funktion aus der `numpy` Bibliothek. Das Resultat ist ein Array mit N komplexen Zahlen. Die Tabelle 2 zeigt die Funktion $f(x)$ und die DFT der 5 Samples.

```
fhat = np.fft.fft(f(x_points), N_SAMPLES)
```

	0	1	2	3	4
$f(x)$	1.5	0.25	0.5	1.75	1.5
dft	$5.50 + 0.00i$	$0.22 + 1.92i$	$0.78 - 0.45i$	$0.78 + 0.45i$	$0.22 - 1.92i$

Tabelle 2: $f(x)$ und die DFT der 5 Samples

Mit den Frequenzkomponenten der DFT können Signale manipuliert werden, etwa durch Filtern bestimmter Frequenzbereiche. Um das ursprüngliche Signal wiederzuerlangen, wenden wir die inverse DFT an. Die Formel der inversen DFT, welche das Signal rekonstruiert, lautet (Hansen, 2014, Chapter 3):

$$f(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) \cdot e^{\frac{2\pi i}{N} kn}$$

```
reconstructed_manual = np.zeros_like(x_points, dtype=np.complex128)

dt = x_points[1] - x_points[0] # Abstand zwischen zwei Punkten
T = N_SAMPLES * dt # Periode des Signals

for n in range(N_SAMPLES):
    # Rekonstruiert Signal mit Fourier-Koeffizienten, neg. Freq. bei n > N_SAMPLES/2.
    freq = n / (2*np.pi) if n <= N_SAMPLES//2 else (n - N_SAMPLES) / (2*np.pi)
    reconstructed_manual += fhat[n] * np.exp(1j * 2 * np.pi * freq * x)

reconstructed_manual = (reconstructed_manual / N_SAMPLES).real
reconstructed = np.fft.ifft(fhat).real # Rekonstruiertes Signal mit ifft
```

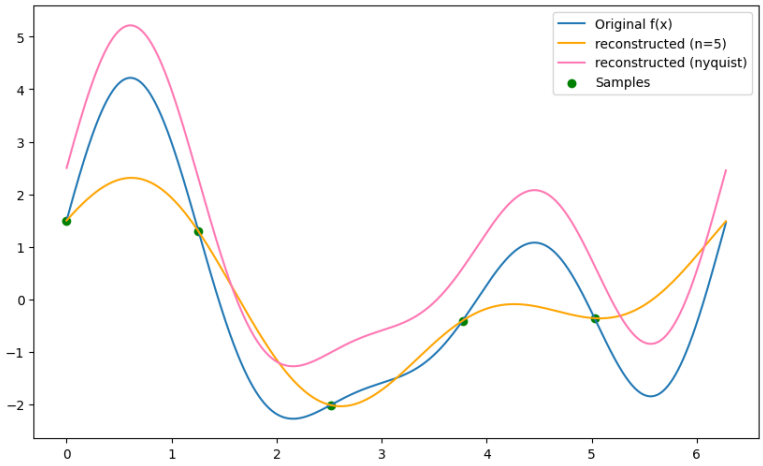

Abbildung 6: Rekonstruktion des Signals $f(x)$

Die Abbildung 6 zeigt die ursprüngliche Funktion $f(x)$ und die rekonstruierte Funktionen `reconstructed (n=5)` und `reconstructed (n=nyquist)`. Die Annäherung der mit der inversen DFT rekonstruierten Funktion an die ursprüngliche Funktion ist bei $n = 5$ deutlich sichtbar. Bei $n = nyquist$ ist die Annäherung nahezu perfekt, wenn nicht sogar perfekt. Die Sampling-Rate bei der Nyquist angenäherten Funktion ist das Doppelte der höchsten Frequenz des Signals. Das ist in diesem Fall $2 \cdot 3 + 1 = 7$.

2.3.3 Aliasing

Aliasing tritt auf, wenn ein Signal bei einer nicht ausreichend hohen Samplingrate digital erfasst wird, wodurch Frequenzen des Signals fehlinterpretiert werden können. Als allgemeines Beispiel wenn ein Sinussignal mit einer Frequenz von 1200 Hz betrachtet wird und dieses mit einer Samplingrate von nur 1000 Hz aufgenommen wurde, könnte das digitalisierte Signal so aussehen, als ob das ursprüngliche Signal eine Frequenz von 200 Hz hätte. Das ist, als ob man ein sich schnell drehendes Rad filmt und auf dem Video wirkt es, als würde es sich langsamer oder sogar rückwärts drehen. Um solche Fehler zu verhindern, sollte die Samplingrate stets mindestens das Doppelte der höchsten Frequenz des Signals betragen, ein Grundsatz, der als Nyquist-Kriterium bekannt ist. (Weitz, 2023).

2.4 Spektrogramm

Mit einem Verständnis der Grundlagen der Fourier-Analyse können wir die Bedeutung des Spektrogramms erfassen. Ein Spektrogramm bietet eine visuelle Darstellung der verschiedenen Frequenzen, die in einem Signal über die Zeit hinweg vorhanden sind. Ein Spektrogramm wird wie folgt definiert:

> "A spectrogram is a three-dimensional visualization of a signal's amplitude over frequency and time. Many audio signals are comprised of multiple frequencies occurring simultaneously, with these frequencies often changing over time." (Tarr, 2018, Chapter 15.2.1)

Im Bereich des Machine Learning, insbesondere bei der Spracherkennung, nimmt das Spektrogramm eine zentrale Position ein. Die Fourier-Transformation eines Audiosignals in seine Frequenzkomponenten resultiert in einem Verlust der zeitlichen Informationen durch die Anwendung der FFT. Für Aufgaben wie die Spracherkennung ist es jedoch von grundlegender Bedeutung, nicht nur die im Signal vorhandenen Frequenzen zu identifizieren, sondern auch den Zeitpunkt ihres Auftretens zu bestimmen. Hier schafft das Spektrogramm Abhilfe, da es die zeitliche Abfolge der Frequenzen sichtbar macht. Diese Fähigkeit ist insbesondere für das Erkennen der Sequenz gesprochener Wörter in einem Satz von Bedeutung (Chaudhary, 2020). Somit verknüpft das Spektrogramm zeitliche und frequenzbezogene Informationen, was es zu einem wichtigen Instrument für die Spracherkennung und andere Machine Learning-Anwendungen macht. Abbildung 7 zeigt ein Beispiel eines Spektrogramms, das im Rahmen dieser Arbeit entwickelt wurde. Das Spektrogramm wurde unter Verwendung der Python-Bibliotheken `PyQt6` für die Echtzeit-Visualisierung und `soundevice` für den Zugriff auf die Audio-Hardware generiert.

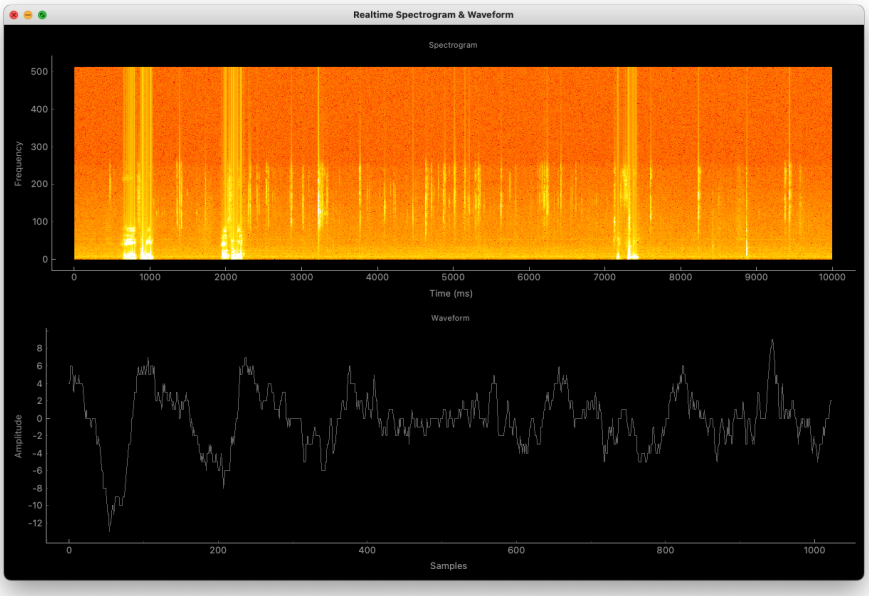

Abbildung 7: Spektrogramm

2.5 Machine Learning

Dieses Kapitel erläutert die grundlegenden Konzepte von Machine Learning, die für die Spracherkennung relevant sind. Es fokussiert sich auf Deep Neural Networks (DNN), Convolutional Neural Networks (CNN) und Recurrent Neural Networks (RNN). Um ein detailiertes Verständnis von neuronalen Netzen zu erhalten, empfiehlt sich die Lektüre von (Weidman, 2019).

2.5.1 Neuronale Netze

Unter Neuronalen Netzen versteht man im Allgemeinen eine Reihe von Berechnungen, die von der Funktionsweise des menschlichen Gehirns inspiriert sind. Im Grunde sind es mathematische Modelle, die aus einer Reihe von miteinander verbundenen Nodes bestehen, die als Neuronen bezeichnet werden. Im einfachsten Fall bestehen Neuronen aus Inputs $x_1, x_2, \ldots, x_n$, Gewichten $w_1, w_2, \ldots, w_n$, einem Bias b einer Aktivierungsfunktion f und einem Output y. Die Abbildung 8 zeigt ein einfaches Beispiel eines Neurons.

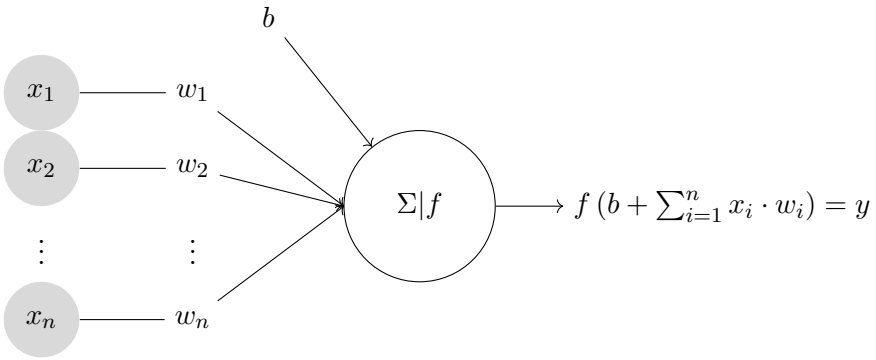

Abbildung 8: Neuronale Netze

Neuronale Netze sind gut geeignet, um komplexe Probleme zu lösen, die nicht besonders einfach mit traditionellen Algorithmen gelöst werden können. Ein Beispiel dafür ist die Spracherkennung oder auch die Bilderkennung. Netzwerke werden trainiert, um die richtigen Gewichte $w_1, w_2, \ldots, w_n$ und den Bias b zu finden, um die gewünschte Ausgabe zu erhalten.

Als nächtest betrachten wir ein kleines Beispiel eines neuronalen Netzes mit drei Layern. Die Abbildung 9 soll das Konzept eines neuronalen Netzes veranschaulichen.

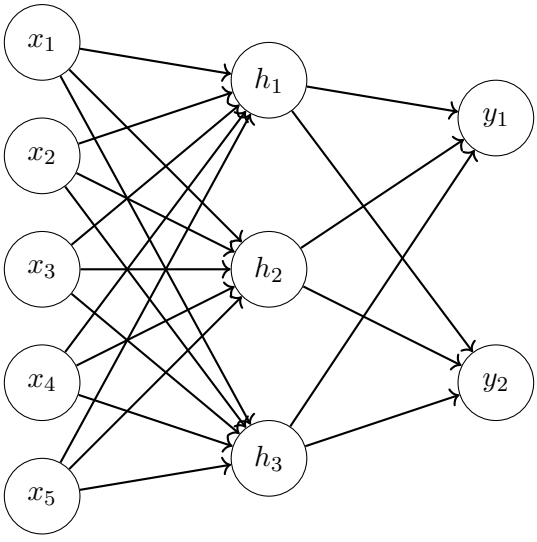

Abbildung 9: Ein neuronales Netzwerk mit drei Layern.

Aktivierungsfunktionen
Aktivierungsfunktionen sind ein wichtiger Bestandteil von neuronalen Netzen. Sie werden verwendet, um die Ausgabe eines Neurons zu berechnen. Es gibt verschiedene Arten von Aktivierungsfunktionen. Die bekanntesten sind die *Sigmoid*, *ReLU* und *Softmax* Funktionen. Bei den Aktivierungsfunktionen handelt es sich um nicht-lineare Funktionen (Weidman, 2019).

Forward Pass und Backward Pass
Zwei wichtige Konzepte in neuronalen Netzen sind der *Forward Pass* und der *Backward Pass*. Der Forward Pass ist der Vorgang, bei dem die Eingabe x durch das Netzwerk geleitet wird, um die Ausgabe y zu erhalten. Der Backward Pass ist der Vorgang, bei dem die Gewichte $w_1, w_2, \ldots, w_n$ und der Bias b optimiert werden. Das Optimieren der Gewichte und des Bias wird als *Training* bezeichnet. Das geschieht, indem der Fehler ausgedrückt als *Loss Function* minimiert wird.

Loss Function
Die Loss Function ist eine Funktion, die den Fehler zwischen der Ausgabe des neuronalen Netzes und dem gewünschten Output misst. Die Loss Function ist ein wichtiger Bestandteil des Trainingsprozesses eines neuronalen Netzes. Es gibt verschiedene Arten von Loss Functions. Die bekanntesten sind die *Mean Squared Error* und die *Cross Entropy* Funktionen.

2.5.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) sind eine spezielle Art von neuronalen Netzen, die vor allem in der Bilderkennung eingesetzt werden. Sie sind in der Lage, komplexe Muster in Bildern zu erkennen. In CNNs sind *Convolutional Layers* zentral. Sie nutzen einen Filter, der über den Input gleitet und mit diesem verrechnet wird, wie Abbildung 10 zeigt.

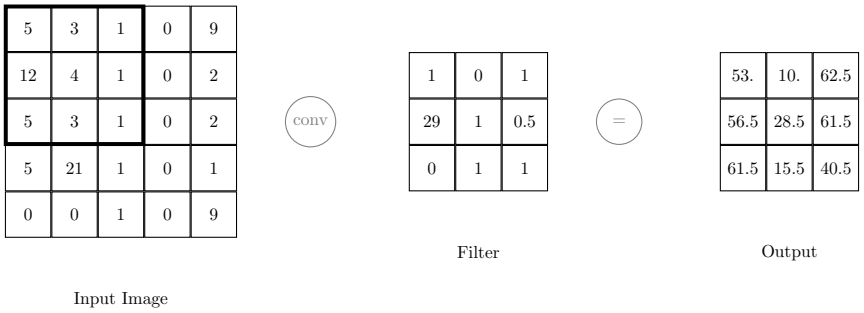

Abbildung 10: Convolution

Bei CNNs sind die Gewichte $w_1, w_2, \ldots, w_n$ und der Bias b die Filter. Filter werden während des Trainings optimiert, um die gewünschte Ausgabe zu erhalten. Der grosse Vorteil gegenüber herkömmlichen neuronalen Netzen ist, dass die Gewichte geteilt werden können. (Weidman, 2019) beschreibt CNNs wie folgt:

> "CNNs are the standard neural network architecture used for prediction when the input observations are images, which is the case in a wide range of neural network applications."

Speziell für CNNs sind die sogenannten *Convolutional Layers*. Diese Layer bestehen aus einem Filter, der über den Input von Links nach Rechts und von Oben nach Unten geschoben wird. Der Filter ist eine Matrix, die mit dem Input verrechnet wird. Die Abbildung 10 zeigt ein Beispiel einer Convolution.

2.5.3 Recurrent Neural Networks

3 Stand der Forschung

Der Stand der Forschung ist ein wichtiger Teil dieser Arbeit. Darin werden die wichtigsten zeitlichen Entwicklungen im Bereich der Spracherkennung dargestellt. Die Forschung im Bereich der Spracherkennung ist ein aktives Forschungsgebiet. Die Jahre 2010 bis 2020 haben laut (Hannun, 2021) einen grossen Fortschritt in der Spracherkennung erlebt. Dieser Fortschritt ist vor allem auf die Verwendung von Deep Learning zurückzuführen.Weiter wird der Aufbau von Sprachassistenten untersucht und verglichen. Ein wichtiger Teil ist die Funktionsweise vom Sprachassistent Siri. Apple veröffentlichte seinen Sprachassistenten im Jahr 2011 mit dem Release von iOS 5. Ebenfalls wird im Unterkapitel InApp Sprachassistenten über die Begründung von einem eigenen Sprachassistenten diskutiert.

3.1 Zeitliche Entwicklung der Spracherkennung

Die Spracherkennung entwickelte sich von 2010 bis 2020 mit enormen Fortschritten. In der heutigen Zeit wird die Spracherkennung in Form von Sprachassistenten wie Siri, Alexa und Google Assistant von vielen Menschen genutzt. Die Nutzung der Sprachassistenten wird vor allem für Suchanfragen verwendet. Angesichts des Fortschritts der letzten Jahre stellt sich die Frage, was die Zukunft bringen wird (Hannun, 2021).

Die Abbildung 11 zeigt eine Timeline der wichtigsten Entwicklungen im Zeitraum von 2010 bis 2020. Die Timeline wurde basierend auf der Timeline von (Hannun, 2021) erstellt aber mit einem Zusatz für das Jahr 2020, welches die neuste Entwicklung von Facebook AI's wav2vec2 aufzeigt (Baevski et al., 2020).

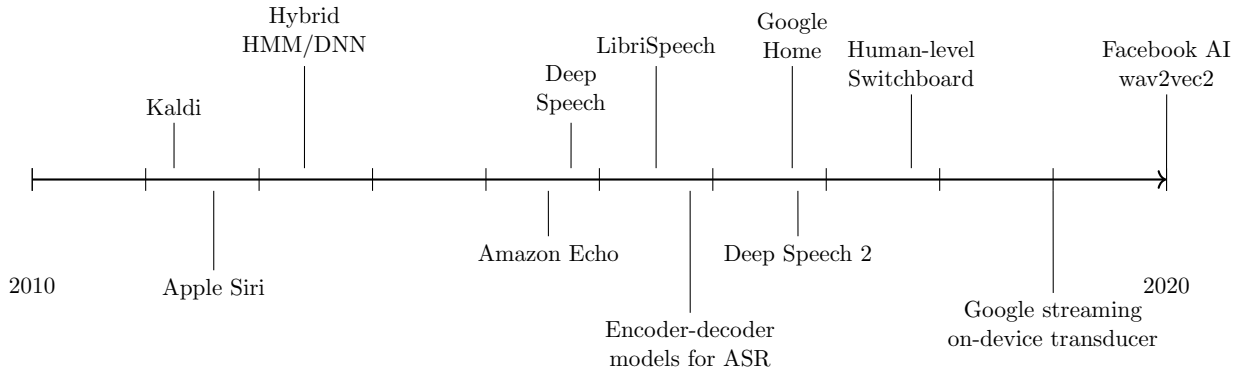

Abbildung 11: Timeline of Speech Recognition Developments (2010-2020)

Deep Learning hat zum grossen Teilen die Spracherkennung revolutioniert, insbesondere durch die Sammlung grosser transkribierter Datensätze und Fortschritte in der Hardware. Spätestens seit (*Deep Speech*) sind Fähigkeiten der akustischen Sprachmodelle mit denen von Menschen vergleichbar (Hannun, 2021).

Als offene Vorhersage für die Zukunft der Spracherkennung bezieht sich (Hannun, 2021) auf die Entwicklung bis 2030. Bis 2030 wird die Spracherkennungs-Forschung eine Verlagerung zu self-supervised Modellen und On-Device-Training erleben, wobei der Fokus auf kleine sparsame Modelle und personalisierten Modellen liegt. Die Wortfehlerrate wird weiter sinken, während die Sprachqualität steigt (Hannun, 2021).

3.2 Aufbau von Sprachassistenten

Sprachassistenten können laut (Matarneh et al., 2017) in folgende Komponenten unterteilt werden: Sprache, Erkennung, Übersetzung und Ausführung von Befehlen. Die nachfolgende Abbildung 12 zeigt

die Komponenten eines Sprachassistenten. Abbildung basiert auf (Matarneh et al., 2017). Zudem wurde aber die Komponente des *TWD* hinzugefügt.

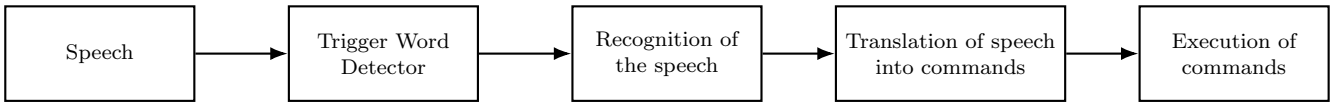

Abbildung 12: Voice control implementation.

Sprachassistenten funktionieren mehrheitlich nach dem gleichen Prinzip. Die Sprache wird durch ein Mikrofon aufgenommen und in ein digitales Signal umgewandelt. Das digitale Signal wird durch einen Trigger Word Detector (TWD) analysiert. Der TWD erkennt, ob das gesprochene Wort die Erkennung aktivieren soll. Für die effektive *Automatic Speech Recognition (ASR)* können verschiedene Ansätze verwendet werden. Eine Möglichkeit ist einen Cloud-basierten Ansatz zu verwenden. Anbieter wie Google, Amazon und Apple verwenden für ihre Sprachassistenten Cloud-basierte Ansätze (Matarneh et al., 2017).

3.2.1 Vergleich von Sprachassistenten

In der Arbeit von (Matarneh et al., 2017) werden verschiedene Spracherkennungssysteme anhand von Kriterien wie Genauigkeit, API-Qualität, Leistung, Echtzeitgeschwindigkeit, Antwortzeit und Kompatibilität verglichen. Die Beurteilung umfasst sowohl Systeme mit offenem (*Open-Source*) als auch geschlossenem Sourcecode (*Closed-Source*), mit einer detaillierten Untersuchung von deren Fähigkeiten und Einschränkungen.

Closed-Source:

- Dragon Mobile SDK
- Google Speech Recognition
- SIRI
- Yandex SpeechKit
- Microsoft Speech API

Open-Source:

- CMU Sphinx
- Kaldi
- Julius
- HTK
- iAtros
- RWTH ASR
- Simon

Zusammenfassend kann festgestellt werden, dass das Dragon Mobile SDK bei den Closed-Source-Systemen in Bezug auf Genauigkeit und Benutzerfreundlichkeit führend ist. Bei den Open-Source-Systemen sticht CMU Sphinx besonders hervor. Dennoch hat die Evaluation ergeben, dass aktuell keine der Systeme optimal und unkompliziert in mobile Apps integriert werden kann. Abgesehen von CMU Sphinx liegt der Schwerpunkt zudem nicht wirklich auf dem Trigger Word Spotting sondern auf der Automatic Speech Recognition (ASR).

3.3 Facebook AI wav2vec2

Im Jahr 2020 präsentierte Facebook AI mit wav2vec2 ein innovatives Modell im Bereich der Spracherkennung. Dieses Modell stellt einen signifikanten Durchbruch in der Technologie dar und nutzt eine Methode, die als *Self-Supervised Learning* bezeichnet wird. Durch diese Methode hat wav2vec2 das Potenzial, die Genauigkeit und Effizienz der Spracherkennung erheblich zu steigern (Baevski et al., 2020).

Für die Zielsetzung dieser Arbeit bietet wav2vec2 eine vielversprechende Perspektive. Obwohl es primär

nicht für die Triggerwort-Erkennung entwickelt wurde, besteht die Möglichkeit, eines seiner Basis-Modelle entsprechend zu adaptieren. Durch gezielte Anpassungen könnte wav2vec2 so modifiziert werden, dass es effektiv für die Triggerwort-Erkennung eingesetzt werden kann. Dies würde nicht nur die Genauigkeit der Erkennung verbessern, sondern auch die Effizienz des gesamten Systems steigern. Es lohnt sich daher, die Potenziale und Anpassungsmöglichkeiten von wav2vec2 für diese spezifische Anwendung weiter zu untersuchen.

Darüber hinaus zeigt wav2vec2.0 das grosse Potenzial des Vortrainierens auf nicht beschrifteten Daten für die Sprachverarbeitung. Selbst mit nur 10 Minuten beschrifteter Trainingsdaten erreicht das Modell beeindruckende Ergebnisse auf dem Librispeech-Test (Baevski et al., 2020). Es stellt auch einen neuen Standard in der Librispeech-Benchmark für verrauschte Sprache dar und übertrifft Modelle, die mit 100 Stunden Daten trainiert wurden, obwohl es 100-mal weniger beschriftete Daten verwendet.

Für das Vorhaben dieser Arbeit könnte wav2vec2 von grossem Nutzen sein. Da es einen signifikanten Durchbruch in der Technologie darstellt, könnte es die Genauigkeit und Effizienz der Spracherkennung erheblich steigern. Es ist zwar im Bezug auf die Trigger Word Erkennung nicht direkt anwendbar, aber durch adaptieren eines der Basis Modelle könnte es umfunktioniert werden.

3.4 Funktionsweise von Siri

Die Implementation von Siri ist nicht öffentlich zugänglich, aber Apple selbst dokumentiert das Grundlegende Konzept von Siri sehr detailliert. Der Beitrag "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant" (Siri-Team, 2017) von der ML Research Group von Apple gibt einen sehr detaillierten Einblick in die Funktionsweise von Siri. Siri besteht aus diversen Komponenten, welche mehrheitlich in der Cloud laufen. "Most of the implementation of Siri is in the Cloud, including the main automatic speech recognition, the natural language interpretation and the various information services." (Siri-Team, 2017). Der Trigger von Siri läuft jedoch auf dem Gerät selbst. Um diesen geht es im wesentlichen in diesem Kapitel.

Siri verwendet für den Voice Trigger ein Deep Neural Network (DNN) um das akustische Muster jedes Frames in eine Verteilung von Wahrscheinlichkeiten für jeden Phonem zu übersetzen. Die Phoneme sind die Bausteine der akustischen Sprache. Aus den Wahrscheinlichkeiten wird in einen zeitlichen Integrationsprozess bestimmt, wie sicher es ist, dass das Gesagte 'Hey Siri' war.

Das Mikrofon des Geräts wandelt das Audiosignal in einen kontinuierlichen Stream von 16000 Samples pro Sekunde um. Diese Samples werden anschliessend durch eine Spektralanalyse in eine Abfolge von Frames transformiert, wobei jeder dieser Frames das Spektrum von ungefähr 0,01 Sekunden beschreibt. Zwanzig Frames, die sich über 0,2 Sekunden erstrecken, werden dann als Eingabe für das DNN verwendet.

Die Architektur der für Siri verwendeten DNNs bestehen typischerweise aus fünf Layers. In der nachfolgenden Abbildung 13 ist die Architektur des DNNs dargestellt. Die Quelle der Abbildung ist der Beitrag "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant" (Siri-Team, 2017).

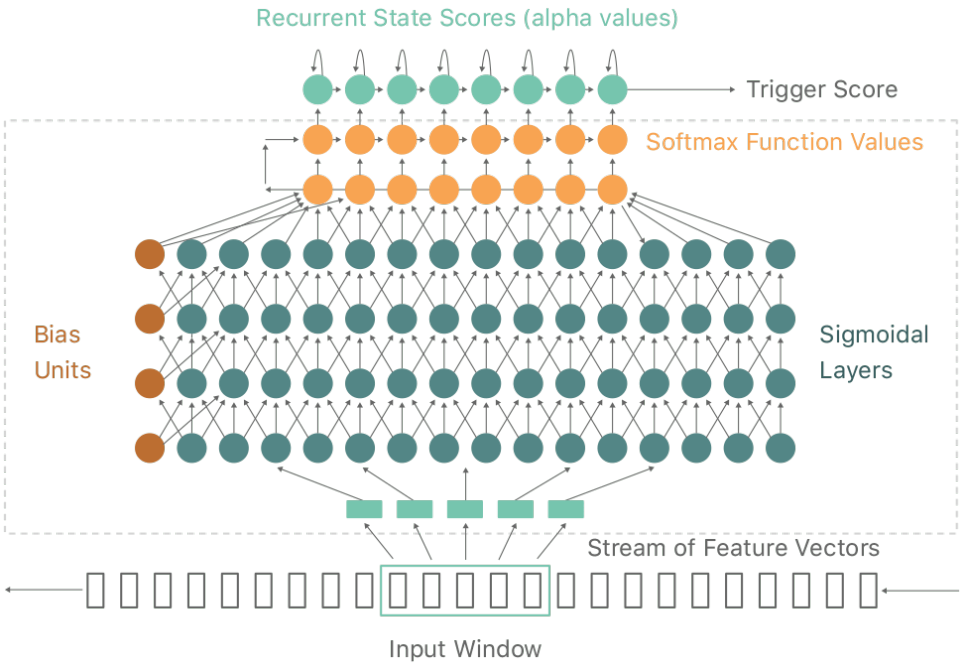

Abbildung 13: DNN Architektur von Siri

Die Siri DNNs werden je nach Gerätetyp und Leistung in verschiedenen Grössen implementiert. Typische Hidden Layer Grössen sind 32, 128 oder 192. Bei Siri werden mehrheitlich fully-connected Layers verwendet. Siri verwendet beim obersten Layer einen RNN Layer. Dies um die zeitliche Abfolge der Frames zu berücksichtigen und basierend darauf einen Score zu berechnen. Siri verwendet nicht nur ein Neuronales Netz, sondern gleich zwei. Das erste wird für die erste Erkennung von 'Hey Siri' verwendet. Das zweite dient als Bestätigung bzw. als zusätzlichen Checker.

Weiter wird im Beitrag “Hey Siri: An On-device DNN-powered Voice Trigger for Apple’s Personal Assistant” (Siri-Team, 2017) beschrieben, dass die Erkennung von ’Hey Siri’ nicht nur schnell sein muss, sondern auch sehr energieeffizient. Das folgende Zitat aus dem Beitrag beschreibt die Anforderungen an die Erkennung von ’Hey Siri’ sehr gut:

> “The Hey Siri detector not only has to be accurate, but it needs to be fast and not have a significant effect on battery life.”

3.4.1 Takeaways

Es ist sehr interessant zu sehen, wie der Trigger Mechanismus von Siri funktioniert. Der simple Aufbau mit einem DNN, welches die Wahrscheinlichkeiten für jedes Phonem berechnet, ist sehr interessant. Die Verwendung von zwei DNNs ist ebenfalls wertvoll und kann auch in dieser Arbeit in Betracht gezogen werden. Ebenfalls wird die Notwendigkeit von einer schnellen und energieeffizienten Erkennung von ’Hey Siri’ deutlich. Dies ist auch für die Umsetzung dieser Arbeit ein wichtiger Punkt. Die beiden Beiträge (Siri-Team, 2017) und (Apple, 2023) geben einen sehr guten Einblick in die Funktionsweise von Siri und sind essentiell für die Umsetzung dieser Arbeit.

3.5 Marktanalyse - Trigger Wort Erkennung

Während der Recherche für diese Arbeit wurden bestehende Lösungen auf dem Markt gesucht. Nebst den bereits erwähnten Lösungen im Unterkapitel 3.2.1 wurde eine Lösung gefunden. Es wurde ein Anbieter gefunden, welcher exakt eine Lösung für die Trigger Wort Erkennung anbietet. Der Anbieter heisst Picovoice und bietet eine Lösung mit dem Namen Porcupine an. Sie beschreiben ihre Lösung wie folgt:

> “Porcupine Wake Word is a wake word detection engine that recognizes unique signals to transition software from passive to active listening. Porcupine Wake Word enables enterprises to offer hands-free experiences by training and deploying custom wake words like Big Tech - but with superior technology.” (Picovoice, 2023)

Porcupine verspricht eine sehr hohe Genauigkeit und eine sehr schnelle Erkennung. Ebenfalls bieten sie eine grosse Auswahl an Integrationen für verschidenste Programmiersprachen bzw. Plattformen an. Um nur einige zu nennen: Python, C, Flutter, Java, JavaScript, Android, iOS, NodeJS, Unity, React und viele mehr. Die Integrationen sehen sehr vielversprechend aus und sind sehr einfach zu implementieren. Bei den Preisen bietet Picovoice drei verschiedene Modelle an. Individuell, Developer und Enterprise. Für ein grösseres Projekt mit dem Bedarf nach einer individuellen Triggerwort Erkennung könnte es genau die richtige Lösung sein. Dennoch ist gilt es zu erwähnen, dass die Lösung von Picovoice nicht Open Source ist. Die Preise sind ebenfalls nicht ganz ohne. Die Enterprise Lösung startet bei 2’500 USD pro Monat und ist somit für viele Projekte nicht finanzierbar. Die Preise sind auf der Webseite von Picovoice einsehbar (Picovoice, 2023).

3.6 Diskussion

Die Trigger Wort Erkennung ist ein aktives Forschungsgebiet. Es gibt viele verschiedene Ansätze und Lösungen. Die in diesem Kapitel betrachteten Lösungen und Ansätze haben einen guten Einblick in die Funktionsweise von Trigger Wort Erkennung gegeben. Die Funktionsweise von Siri hat hierbei einen besonderen Einblick gegeben.

4 Ideen und Konzepte

Dieses Kapitel beschreibt die Ideen und Konzepte, die für die Umsetzung der Arbeit verwendet werden. Es wird auch auf die verwendeten Technologien eingegangen. Die Grob Idee ist es im wesentlichen, ein eigenes Modell zu trainieren, ganz nach dem Vorbild von (Siri-Team, 2017), welches Triggerwörter erkennt. Dazu wird ein Datensatz erstellt, welcher die Triggerwörter, sowie andere Wörter enthält. Weiter sollen weitere Datensätze untersucht werden, um die Genauigkeit des Modells zu verbessern. Eines der gefundenen Datensätze ist der *Mozilla Common Voice* Datensatz (Ardila et al., 2020). Dieser Datensatz enthält unzählige Sprachaufnahmen von Menschen, welche freiwillig ihre Stimme aufgenommen haben.

4.1 Grundlegende Idee

Um die Grundlegende Idee zu beschreiben wird die Abbildung 14 verwendet. Darin wird illustrativ dargestellt, wie die Grundlegende Idee umgesetzt werden soll. Es startet mit der erarbeitung der Grundlagen in Bezug auf Audio und Machine Learning. Danach wird eine Möglichkeit erarbeitet, um Sprachaufnahmen zu machen. Ein Schritt weiter ist es, die Sprachaufnahmen aufzubereiten und in ein Format zu bringen, welches für das Training eines Modells verwendet werden kann. Ebenfalls soll eine Modelarchitektur erarbeitet werden, welche für die Triggerwort Erkennung verwendet werden kann. Nachdem das Modell trainiert wurde, soll es in eine App integriert werden.

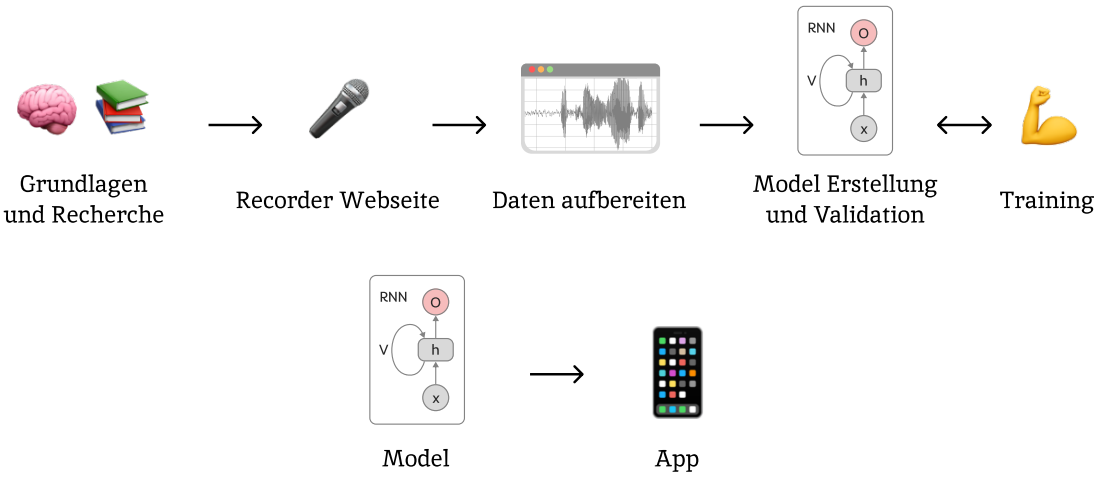

Abbildung 14: Grundlegende Idee

Das Ergebnis soll eine App sein, welche ein Triggerwort erkennt. Das Resultat dieser Bachelorarbeit ist sozusagen eine Teilkomponente einer Sprachsteuerung. Denn wie wir im Kapitel Stand der Forschung gesehen haben, besteht eine Sprachsteuerung aus mehreren Komponenten. Um einen möglichen Use Case zu nennen, könnte die App als Triggerwort Erkennung für eine Sprachsteuerung in einer Koch App verwendet werden. Die App könnte dann beispielsweise folgende Befehle verstehen: “Hey FOOBY, nächster Kochschritt” oder “Hey FOOBY, vorheriger Kochschritt”. Eine Sprachsteuerung für eine Koch App würde somit den Anwendern ermöglichen, die Hände frei zu haben und sich voll und ganz auf das Kochen zu konzentrieren. Die Nutzererfahrung könnte mit einem solchen Feature massiv verbessert werden.

4.2 Skizzenhafte Lösungsansätze

- Machine Learning: Verwendung von bereits trainierten Modellen oder Aufbau eigener Modelle zur Triggerwort-Erkennung.
- Cloud vs. Edge Computing: Ob die Spracherkennung in der Cloud oder direkt auf dem Gerät erfolgen sollte.

- Integration: SDKs oder APIs, die Entwicklern helfen, die Spracherkennungsfunktionalität in ihre Apps zu integrieren.

4.3 Systemarchitektur

- Überlegungen zur Systemarchitektur: Ist ein monolithisches System oder eine Microservice-Architektur sinnvoller?
- Vorteile und Herausforderungen beider Ansätze im Kontext der Spracherkennung.

4.4 Alternative Ansätze

- Untersuchung anderer Technologien oder Ansätze zur Spracherkennung, z.B. regelbasierte Systeme im Vergleich zu Machine Learning.
- Warum Sie sich für einen bestimmten Ansatz entschieden haben oder warum Sie einen anderen verworfen haben.

4.5 Mögliche Problempunkte

- Datenschutz: Wie wird die Privatsphäre der Benutzer gewahrt?
- Effizienz: Wie kann sichergestellt werden, dass die Spracherkennung schnell und akkurat ist?
- Integration: Wie kann die Kompatibilität mit einer Vielzahl von Apps und Plattformen gewährleistet werden?

4.6 Erste Überlegungen zu Tools und Technologien

- Welche Programmiersprachen oder Frameworks könnten verwendet werden?
- Eventuelle Betrachtung von Open-Source-Optionen für Spracherkennung.

4.7 Zusammenfassung und Ausblick

- Ein kurzer Überblick über die in diesem Kapitel vorgestellten Ideen und Konzepte.
- Ein Vorgeschmack darauf, wie diese Konzepte in den kommenden Kapiteln weiter verfolgt werden.

....

5 Methoden

Das Vorgehen dieser Arbeit kann als eine hybride Methode beschrieben werden. Aus der Aufgabenstellung ging schon hervor, dass ein eigenes Deep Learning Modell für die Triggerwort Erkennung erarbeitet werden soll. Zum andren soll das Modell in eine mobile App integriert werden. Diese zwei Bereiche wurden Bereits bei der Planung der Arbeit definiert und haben daher die Planung der Arbeit stark beeinflusst.

5.1 Projektphasen

Im Projektmanagement wurden die Projektphasen definiert. Diese bestehen aus vier Phasen. Die erste Phase widmet sich der *Stand der Forschung*. Darin wurden aber bereits einige Punkte der Planung definiert sowie dem Setup der Entwicklungsumgebung. Der Bestandteil liegt aber in der Erarbeitung der Grundlagen sowie der Recherche von bestehenden Lösungen. Die zweite Phase ist die *Erstellung des Modells*. In dieser Phase wird das Modell entwickelt, trainiert und evaluiert. Ebenfalls wird auch eine Möglichkeit erarbeitet um ein Datenstaz zu erstellen. In der dritten Phase namens *Prototype* wird das Modell in eine mobile App integriert. Die letzte Phase ist das *Refinement*. In dieser Phase wird die Arbeit sowie die Dokumentation fertiggestellt.

5.2 Projektmanagement

Das Projektmanagement spielt eine zentrale Rolle in der Vorbereitungsphase meiner Bachelorarbeit und bildet die Grundlage für den Erfolg des gesamten Vorhabens. Dabei geht es nicht nur um die reine Planung, sondern auch um eine effiziente Steuerung und kontinuierliche Kontrolle aller Arbeitspakete und derer Ergebnisse. Die besondere Herausforderung meiner Arbeit liegt darin, das umfangreiche Themengebiet, das für diese Bachelorarbeit relevant ist, innerhalb des engen Zeitrahmens von 14 Wochen sinnvoll und fundiert zu bearbeiten. Das Themengebiet umfasst diverse Bereiche der Informatik. Darunter fallen Audioverarbeitung, maschinelles Lernen, Softwareentwicklung und auch einiges an mathematischem Hintergrundwissen. Daher wurde ein agiles Vorgehensmodell gewählt. Dies bedeutet, dass sowohl die Planung als auch die Umsetzung in iterative Zyklen unterteilt sind. Während es zu Beginn eine grobe Struktur und Zielsetzung gibt, ermöglicht diese Herangehensweise Flexibilität in der Durchführung. Dadurch können Veränderungen oder unerwartete Ereignisse leichter integriert und die Bachelorarbeit fortlaufend optimiert werden.

5.2.1 Produkt Backlog

Der Product Backlog beinhaltet alle Anforderungen die für dir Erstellung des Spracherkennungssystems. Die Erstellung des Models, die Entwicklung der mobile App und die Dokumentation sind dabei die Hauptaufgaben.

Abbildung 15: Tabelle für das anfängliche Product Backlog

5.2.2 Risikomanagement

Als mögliche Risiken wurden im Projekt zum einen die Komplexität des Themas und zum anderen die begrenzte Zeit für die Bearbeitung identifiziert. Weitere Risiken, die während der Projektdurchführung aufgetreten sind, werden nachfolgend zusammengefasst. Die Tabelle 3 zeigt die identifizierten Risiken, deren Eintrittswahrscheinlichkeit sowie die Auswirkung auf das Projekt.

Risiko	**Eintrittswahrscheinlichkeit**	**Auswirkung**
Komplexität des Themas	Hoch	Gross
Begrenzte Zeit für Bearbeitung (14 Wochen)	Mittel	Kritisch
80% Arbeitspensum, 20h pro Woche am Wochenende	Hoch	Mittel
Grosser Release in aktueller Arbeit bis Ende Oktober	Hoch	Hoch
Qualität des Speech Recognizers	Mittel	Kritisch
Latenz und Performance des Erkenners	Hoch	Gross

Tabelle 3: Identifizierte Risiken im Projekt

5.3 Grobplanung

Die Grobplanung zeigt die wichtigsten Meilensteine sowie die anfängliche zeitliche Einteilung der einzelnen Themenbereiche die für die Bachelorarbeit relevant sind, aufgezeigt. Die Grobplanung ist in Abbildung 16 dargestellt.

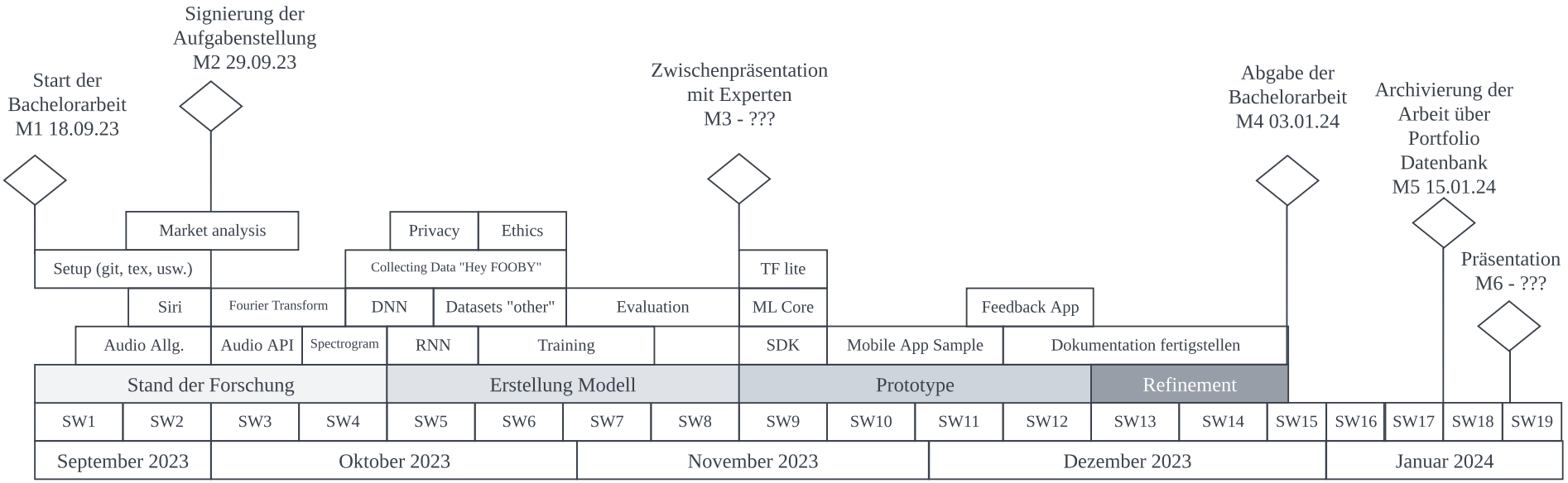

Abbildung 16: Grobplanung

5.4 Forschungsmethoden (falls anwendbar)

- Wichtigkeit von PyTorch
- Probleme beim erstellen des Modells
- Versuche mit wav2vec2
- Versuche mit LiteFEW
- Versuche mit ConvLSTM
- Realisierung der Integration
- Erwägung von Benutzertests, um zu prüfen, wie intuitiv und effizient die Spracherkennungsfunktion ist.
- Interviews mit potenziellen Nutzern der App, um ihre Anforderungen und Erwartungen in Bezug auf Spracherkennung zu verstehen.

5.5 Technische Evaluierung

- Vergleich verschiedener Spracherkennungs-APIs oder Frameworks hinsichtlich Genauigkeit, Latenz, Sprachunterstützung und Kosten.
- Technische Tests zur Prüfung der Performance und Genauigkeit der gewählten Spracherkennungslösung.

5.6 Integrationsmethode

- Beschreibung der geplanten Techniken zur Integration, z.B. Einbindung über API, Implementierung bestimmter Bibliotheken oder Nutzung nativer App-Funktionen.
- Diskussion über potenzielle Herausforderungen bei der Integration und wie sie angegangen werden sollen.

5.7 Teststrategie

- Entwurf einer Teststrategie speziell für die Spracherkennungsfunktion: Testen unter verschiedenen Bedingungen (z.B. Hintergrundgeräusche, verschiedene Sprachen und Akzente).
- Verweis darauf, dass die eigentliche Testdurchführung in einem anderen Kapitel oder Dokument beschrieben ist.

6 Realisierung

Die Realisierung dieser Arbeit besteht aus zwei Teilen. Zum einen wurde ein eigenes Model trainiert, welches Triggerwörter erkennt. Zum anderen wurde das Model in eine App integriert. Diese zwei Bereiche wurden Bereits bei der Planung der Arbeit definiert. Für die Realisierung des Modells wurde PyTorch verwendet. Dieses Framework hat sich als sehr hilfreich erwiesen.

6.1 Verwendete Architektur des Modells

Die Architektur des Models wurde in ersten Linie von (Siri-Team, 2017) inspiriert. Dennoch wurde schlussendlich eine andere Architektur umgesetzt die sich aber ähnlich in Bezug auf Performance und Genauigkeit verhält. Der Grundgedanke war es die zeitliche Komponente der Sprache zu berücksichtigen. Dies wurde mit Convolutional Layers und einem Long Short-Term Memory (LSTM) Layer umgesetzt. Die Kombination dieser beiden Layer hat sich als sehr effektiv erwiesen. Das Paper (Khamees et al., 2021) zeigt diesen Ansatz sehr gut auf. Es bezieht sich zwar nicht auf die Triggerwort Erkennung, aber auf die Klassifizierung von Music Genres. Die finale Architektur des Models ist in der Abbildung 17 dargestellt.

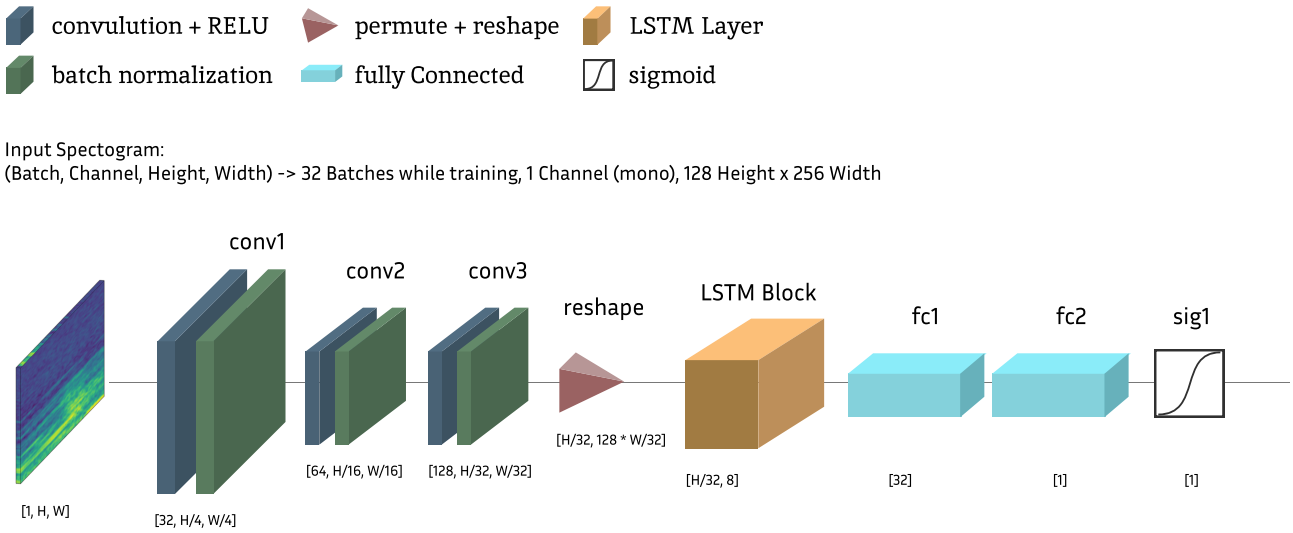

Abbildung 17: Architektur des Modells

Diese Art von Modelarchitektur setzt eine Transformation der Sprachdaten in ein Spektrogramm voraus. Die Transformation der Sprachdaten wurde im Kapiel 2 der Grundlagen bereits beschrieben.

6.2 PyTorch als Framework

Pytorch ist ein Open Source Machine Learning Framework, welches von Facebook entwickelt wurde. Es bietet eine grosse Flexibilität und ist sehr gut dokumentiert. Die Gründe für die Wahl von PyTorch waren, dass es zum einen eine sehr einfache und intuitive API bietet. Zum anderen bietet es die Möglichkeit, die trainierten Models in eine mobile App zu integrieren.

Am besten dieses Paper verwenden (Choi et al., 2018). Hat Grundwissern über Following their success in Computer Vision and other ar- eas, deep learning techniques have recently become widely adopted in Music Information Retrieval (MIR) research

Als Teil der Realisierung wurde ein eigenes Modell trainiert, welches Triggerwörter erkennt. Dazu wurden einige Tests wie auch Experimente durchgeführt. Projektphasen der Modell Erstellung, der Integration und der Evaluation.

6.3 Google Cloud Platform

Die Google Cloud Platform (GCP) ist eine Sammlung von Cloud Computing-Diensten. In dieser Arbeit wurde die GCP verwendet, um die Recorder Webseite zu hosten. Ebenfalls wurde in betracht gezogen, die Modelle auf der GCP zu trainieren. Die GCP bietet Services für Machine Learning an. Eines der Services ist Vertex AI. Darin können beispielsweise Jupiter Notebooks verwendet werden, um Modelle zu trainieren. Das Training kann beispielsweise auf CUDA-fähigen GPUs ausgeführt werden.

Die Sprachdaten wurden auf Google Cloud Storage (GCS) gespeichert. GCS ist ein Objektspeicher, der für die Speicherung und Verteilung von grossen Datenmengen optimiert ist. Die Sprachdaten wurden in einem Bucket gespeichert. Um mit der Datenschutzverordnung konform zu sein, wurde eine Löschrichtlinie für die Sprachdaten erstellt. Art. 17 Abs. 1 DSGVO besagt, dass Daten gelöscht werden müssen, wenn sie für den ursprünglichen Zweck nicht mehr erforderlich sind. Die Löschrichtlinie wurde so konfiguriert, dass die Sprachdaten nach einem Jahr gelöscht werden.

6.3.1 Recorder Webseite

Die Recorder Webseite stellt ein nützliches Tool dar, das die Aufnahme von Sprachsamples ermöglicht. Ursprünglich von Pete Warden entwickelt, bietet sie eine einfache und effiziente Möglichkeit, Sprachdaten zu sammeln Warden, 2018.

Im Kontext der VoiceCommands-Samples wird eine spezielle Version dieser Recorder Webseite bereitgestellt, die speziell für die Aufnahme von Sprachsamples konzipiert ist. Die ursprüngliche Webseite, die auf GitHub unter Pete Warden's Repository zu finden ist, wurde im Rahmen dieser Bachelorarbeit umfassend modifiziert und erweitert.

Einige der wesentlichen Anpassungen umfassen:

- **Hosting auf GCP**: Die Webseite wurde auf Google Cloud Platform (GCP) gehostet, um eine zuverlässige und skalierbare Lösung zu gewährleisten.
- **Dockerisierung und Aktualisierung**: Die Anwendung wurde dockerisiert, was die Bereitstellung und Skalierung erleichtert. Zudem wurde sie auf den neuesten Stand gebracht, um sicherzustellen, dass sie mit aktuellen Technologien kompatibel ist.
- **Technische Details**: Bei der zugrunde liegenden Anwendung handelt es sich um eine Python Flask App, die für ihre Leichtigkeit und Flexibilität bekannt ist.
- **Anpassungen für die Bachelorarbeit**: Die Zielwörter und die Länge der Aufnahmen wurden modifiziert, um den spezifischen Anforderungen dieser Arbeit gerecht zu werden. Darüber hinaus wurde die Webseite nahtlos in das Repository dieser Bachelorarbeit integriert.

Die überarbeitete Recorder Webseite stellte einen wichtigen Bestandteil dieser Arbeit dar. Sie ermöglichte es, einen eigenen Datensatz zu erstellen, der für das Training eines Modells zur Triggerwort-Erkennung verwendet werden konnte. Die nachstehende Abbildung gibt einen visuellen Überblick über das Erscheinungsbild und die Funktionalität der Webseite.

6.4 Datensatz

Um ein Modell zu trainieren, welches Triggerwörter erkennt, wird ein Datensatz benötigt.

6.4.1 Datensatz 'other

Am besten von Datenset (Warden, 2018)

- Datenset von Mozilla Common Voice
- Datenset von Google Speech Commands

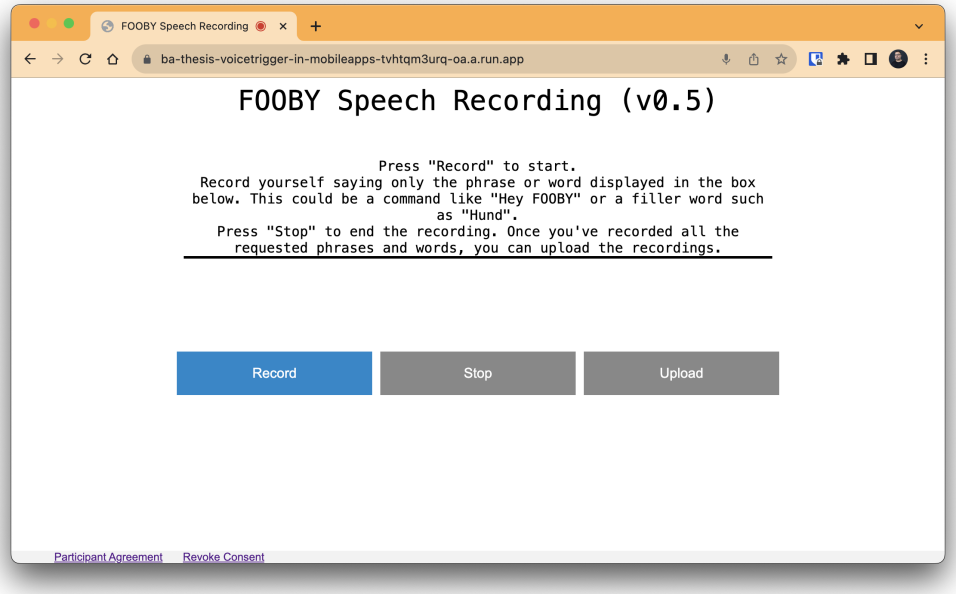

Abbildung 18: Die überarbeitete Recorder Webseite

6.4.2 Datensatz 'hey-fooby'

- TODO: Recorder Webseite für Aufnahme von Sprachsamples
- Synthetische Sprachsamples mit verschiedenen Stimmen Geräuschen und Hintergrundgeräuschen

6.5 Modell

- Auswahl des Modells
- Auswahl der Hyperparameter
- Training des Modells
- Evaluation des Modells

6.5.1 Modell 'CNN'

- Aufbau des Modells
- TODO: Beschreibung des Modells

6.5.2 Modell 'RNN'

- Aufbau des Modells
- TODO: Beschreibung des Modells

7 Evaluation und Validation

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

8 Ausblick

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

9 Anhang

Glossar

Wake-up Wort Ein Wake-up Wort ist ein Wort oder eine Wortsequenz, die ein System aktiviert.. 6

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Apple, M. L. J. (2023, August). *Voice Trigger System for Siri* [Zugriffsdatum: 10. September 2023]. https://machinelearning.apple.com/research/voice-trigger

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., & Weber, G. (2020). Common Voice: A Massively-Multilingual Speech Corpus.

Baevski, A., Zhou, H., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations.

Chaudhary, K. (2020). *Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System* [Zugriff am: 06.10.2023]. https://dropsofai.com/understanding-audio-data-fourier-transform-fft-and-spectrogram-features-for-a-speech-recognition-system/

Choi, K., Fazekas, G., Cho, K., & Sandler, M. (2018). A Tutorial on Deep Learning for Music Information Retrieval.

Deloraine, E. M., & Reeves, A. H. (1965). The 25th anniversary of pulse code modulation. *IEEE Spectrum*, *2*(5), 56–63. https://doi.org/10.1109/MSPEC.1965.5212943

Hannun, A. (2021). The History of Speech Recognition to the Year 2030.

Hansen, E. W. (2014, September). *Fourier Transforms: Principles and Applications*. Wiley.

Khamees, A. A., Hejazi, H. D., Alshurideh, M., & Salloum, S. A. (2021). Classifying Audio Music Genres Using CNN and RNN. In A.-E. Hassanien, K.-C. Chang & T. Mincong (Hrsg.), *Advanced Machine Learning Technologies and Applications* (S. 315–323). Springer International Publishing.

Matarneh, R., Maksymova, S., Lyashenko, V. V., & Belova, N. V. (2017). Speech Recognition Systems: A Comparative Review [Submission Date: 13-10-2017, Acceptance Date: 27-10-2017]. *OSR Journal of Computer Engineering (IOSR-JCE)*, *19*(5), 71–79. https://doi.org/10.9790/0661-1905047179

Picovoice. (2023). *Porcupine — Train custom wake words in seconds.* [Zugriff am 10. Oktober 2023]. https://picovoice.ai/platform/porcupine/

Siri-Team. (2017, Oktober). *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant* [Zugriffsdatum: 10. September 2023]. https://machinelearning.apple.com/research/hey-siri

Somberg, G., Davidson, G., & Doumler, T. (2019). A Standard Audio API for C++: Motivation, Scope, and Basic Design ["C++ is there to deal with hardware at a low level, and to abstract away from it with zero overhead." – Bjarne Stroustrup, Cpp.chat Episode #44]. *Programming Language C++*.

Tarr, E. (2018). *Hack audio : : an introduction to computer programming and digital signal processing in MATLAB* (1st edition). Routledge.

Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition.

Weidman, S. (2019). *Deep learning from scratch : : building with Python from first principles* (First edition.). O'Reilly.

Weitz, P. D. E. (2023). *Fourier-Analysis in 100 Minuten* [Zugriff am: 06.10.2023]. YouTube. https://www.youtube.com/watch?v=zXd743X6I0w

Aufgabenstellung

Integration von Sprachsteuerung in Mobile Apps, insbesondere zur Erkennung von Triggerwörtern.

Projektteam

- Student:in: Rubén Nuñez
- Betreuer:in: Herzog
- Firma: Bitforge AG

Auftraggeber

- Firma: Bitforge AG
- Ansprechperson: Stefan Reinhard
- Funktion: Head of Mobile
- Adresse: Zeughausstrasse 39, 8004 Zürich
- Telefon: +41 55 211 02 41
- E-Mail: stefan.reinhard@bitforge.ch
- Website: www.bitforge.ch

Sonstige Bemerkungen

Grundkenntnisse in Machine Learning, speziell im Bereich der Spracherkennung, sowie Erfahrung mit entsprechenden APIs sind erforderlich.