Bachelorarbeit

Integration einer Sprachsteuerungsfunktion 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.

Inhaltsverzeichnis

1 Problem, Fragestellung, Vision **6**
- 1.1 Problemstellung . . . 6
- 1.2 Fragestellung . . . 6
- 1.3 Vision . . . 6

2 Grundlagen **7**
- 2.1 Audio . . . 7
  - 2.1.1 Sampling . . . 7
  - 2.1.2 Frames, Channels, Buffers . . . 7
  - 2.1.3 Buffers im Detail . . . 8
- 2.2 Audio APIs . . . 9
  - 2.2.1 Audio API für Analyse . . . 9
  - 2.2.2 Audio API für Integration . . . 9
- 2.3 Fourier-Analyse . . . 10
  - 2.3.1 Fourier-Transformation . . . 10
  - 2.3.2 Diskrete Fourier-Transformation . . . 11
  - 2.3.3 Aliasing . . . 13
- 2.4 Spektrogramm . . . 13
- 2.5 Machine Learning . . . 14
  - 2.5.1 Neuronale Netze . . . 14
  - 2.5.2 Convolutional Neural Networks . . . 15
  - 2.5.3 Recurrent Neural Networks . . . 15

3 Stand der Forschung **16**
- 3.1 Zeitliche Entwicklung der Spracherkennung . . . 16
- 3.2 Komparative Analyse von Sprachassistenten . . . 17
- 3.3 Funktionsweise von Siri . . . 17
  - 3.3.1 Takeaways . . . 18
- 3.4 In App Sprachsteuerung . . . 18
- 3.5 Marktanalyse - Benchmarking . . . 19
- 3.6 Diskussion . . . 19

4 Ideen und Konzepte **20**
- 4.1 Einleitung . . . 20
- 4.2 Grundlegende Idee . . . 20
- 4.3 Skizzenhafte Lösungsansätze . . . 20
- 4.4 Systemarchitektur . . . 20
- 4.5 Alternative Ansätze . . . 20
- 4.6 Mögliche Problempunkte . . . 20
- 4.7 Erste Überlegungen zu Tools und Technologien . . . 21
- 4.8 Zusammenfassung und Ausblick . . . 21

5 Methoden **22**
- 5.1 Vorgehensmodell . . . 22
- 5.2 Forschungsmethoden (falls anwendbar) . . . 22
- 5.3 Technische Evaluierung . . . 22
- 5.4 Integrationsmethode . . . 22
- 5.5 Teststrategie . . . 22

6 Realisierung **23**
6.1 Google Cloud Platform . . . 23
6.1.1 Recorder Webseite . . . 23
6.2 Datensatz . . . 23
6.2.1 Datensatz 'other . . . 23
6.2.2 Datensatz 'hey-fooby' . . . 23
6.3 Modell . . . 23
6.3.1 Modell 'CNN' . . . 23
6.3.2 Modell 'RNN' . . . 23

7 Evaluation und Validation **24**

8 Ausblick **25**

9 Anhang **26**
9.1 Projektmanagement . . . 26
9.1.1 Produkt Backlog . . . 26
9.1.2 Risikomanagement . . . 26
9.2 Grobplanung . . . 27

Glossar **28**

Glossar **28**

Abbildungsverzeichnis **28**

Tabellenverzeichnis **28**

Literaturverzeichnis **28**

1 Problem, Fragestellung, Vision

1.1 Problemstellung

Sprachsteuerungstechnologien haben ein grosses Potenzial und werden bisher vor allem als 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 gibt es eine Lücke zwischen eigenen Sprachassistenten und Mobile Apps. Diese Lücke soll mit dieser Arbeit geschlossen werden.

1.2 Fragestellung

Wie kann eine integrierte Sprachsteuerung für eine Mobile App entwickelt werden, die speziell das Erkennen von Triggerwörter oder Triggerwortsequenzen ermöglicht?

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 Learnings verwendet werden. Zum anderen soll diese 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 ist es ein Anliegen dieser Arbeit, dass das Thema Datenschutz und die ethischen Aspekte zu berücksichtigen sind.

2 Grundlagen

Um diese Arbeit fundiert anzugehen, ist ein Verständnis der Grundlagen in den Bereichen Audioverarbeitung und Machine Learning essenziell. Daher wird in diesem Kapitel ein Überblick über die wichtigsten Themen gegeben.

2.1 Audio

In der digitalen Welt repräsentiert Audio Schallwellen, die durch eine Reihe von numerischen Werten dargestellt werden. (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 in der Zeit. Diese Betrachtung ist vor allem in der Elektrotechnik von Bedeutung, da die Amplitude als Spannung angesehen werden kann. Grundsätzlich ist Audio ein kontinuierliches Signal. In der digitalen Welt können wir jedoch nur diskrete Werte darstellen. Daher wird das kontinuierliche Signal in diskrete Werte umgewandelt. 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 entwickelt, parallel zum Aufkommen der digitalen Telekommunikation (Deloraine und Reeves, 1965, p. 57). Im Grundsatz wird es heute noch in modernen Computersystemen nach dem gleichen Verfahren angewendet.

Es folgt eine formelle 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. Nun typische Sampling-Raten sind 44100 Hz oder 48000 Hz. Bei Sampling-Raten wird die Einheit *Hertz* verwendet. 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. Also $F_n = \frac{F_s}{2}$. Die Idee hinter der Nyquist-Frequenz ist, dass die Sampling-Rate mindestens doppelt so hoch sein muss wie die höchste Frequenz des Signals. Wenn diese Eigenschaft erfüllt ist, 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 dieser wird in digitalen Systemen durch eine bestimmte Anzahl von Bits dargestellt. Die Anzahl der Bits wird als *Bit-Depth* bezeichnet. Die Bit-Depth 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 angesehen 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 1 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.

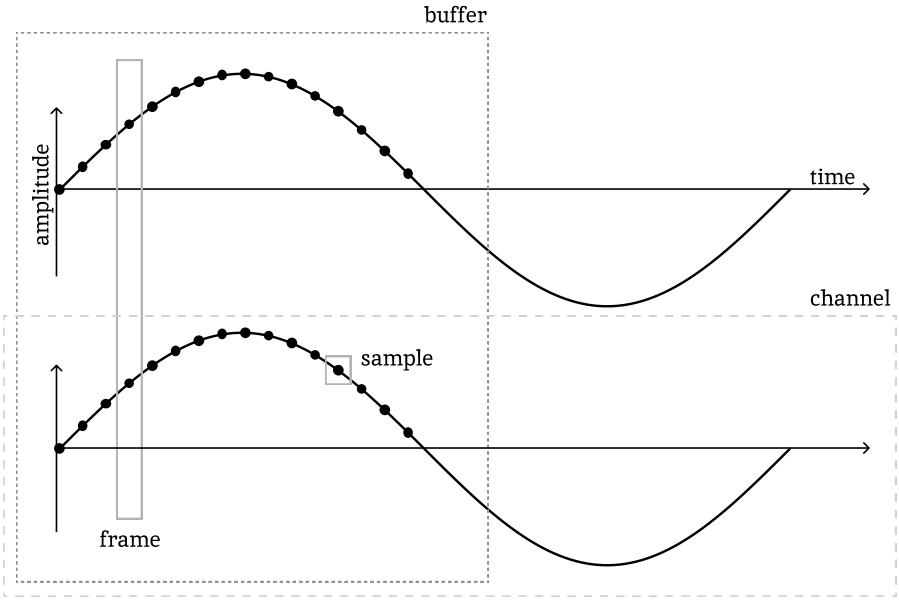

Abbildung 1: Frames, Channels und Buffers

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 den folgenden Abbildungen 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.

El sol es grande

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 die jeweiligen Eigenschaften und Möglichkeiten dieser APIs eingehen, insbesondere im Hinblick darauf, wie sie sich für die Aufnahme, Wiedergabe und Echtzeitverarbeitung von Audiodaten eignen.

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. TODO: PyAudio/SoundDevice/librosa vergleichen

2.2.2 Audio API für Integration

TODO: AVAudioEngine, AudioTrack und kurze Codebeispiele

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 $\dfrac{2\sin(\omega)}{\omega}$.

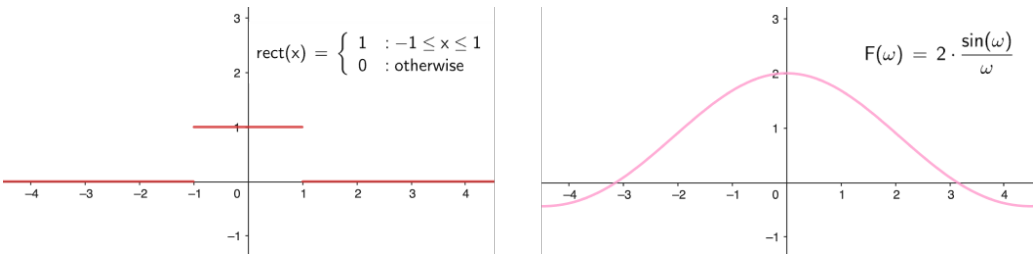

Abbildung 2: Rechteckfunktion und ihre Fourier-Transformation

Die Abbildung 2 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 3 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
```

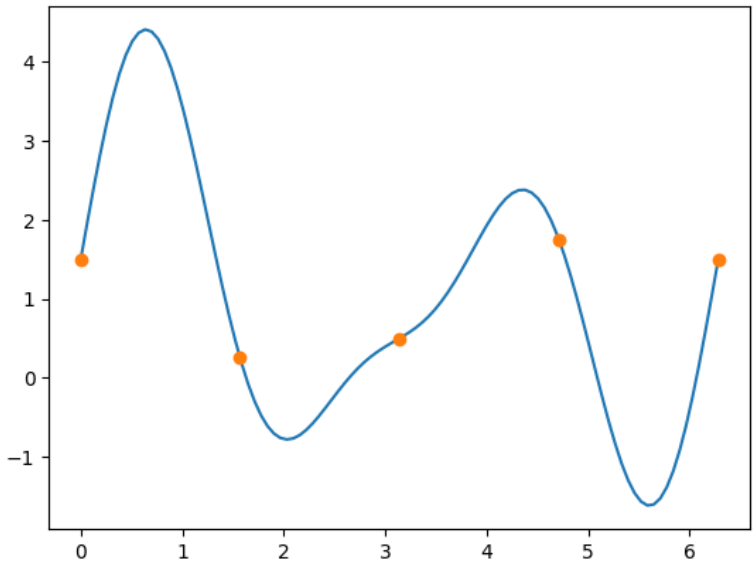

Abbildung 3: Funktion $f(x)$ mit 5 Samples

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
```

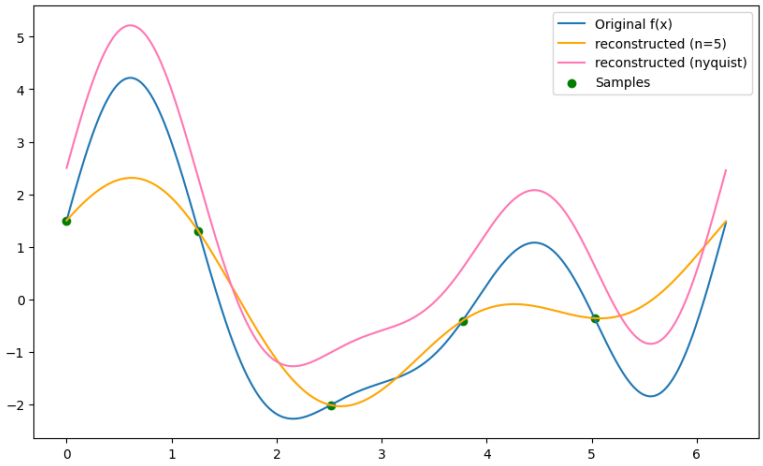

Abbildung 4: Rekonstruktion des Signals $f(x)$

Die Abbildung 4 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 5 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.

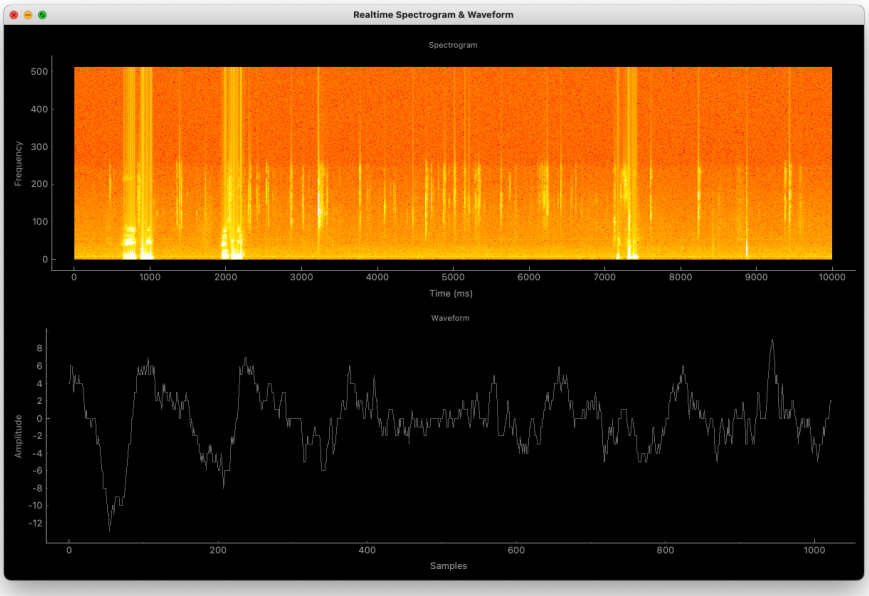

Abbildung 5: Spektrogramm

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 6 zeigt ein einfaches Beispiel eines Neurons.

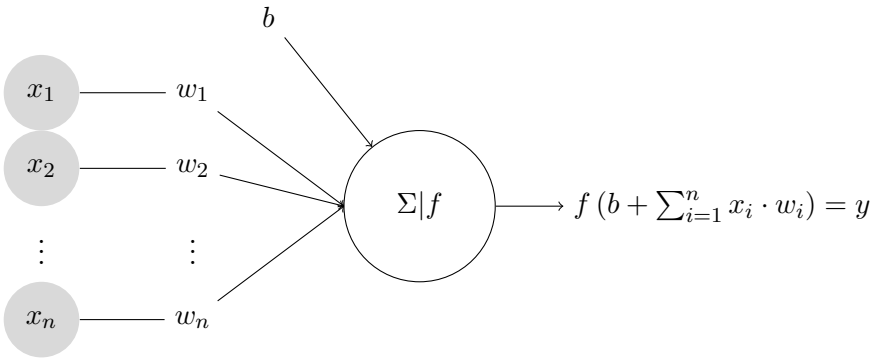

Abbildung 6: Neuronale Netze

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 7 soll das Konzept eines neuronalen Netzes veranschaulichen.

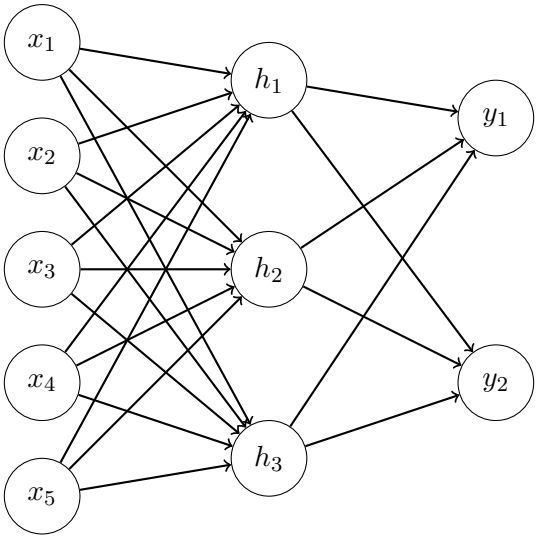

Abbildung 7: Ein neuronales Netzwerk mit drei Layern.

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 (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.

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 8 zeigt.

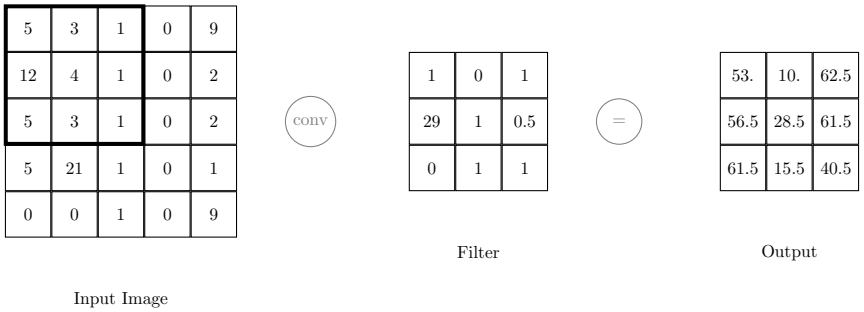

Abbildung 8: Convolution

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 8 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. Weiter werden die Sprachassistenten verglichen und die verwendeten Technologien aufgezeigt. Ein wichtiger Teil ist die Funktionsweise vom Sprachassistent Siri. Apple veröffentlichte seinen Sprachassistenten im Jahr 2011 mit dem Release von iOS 5. Ebenfalls werden InApp Sprachassistenten im Markt verglichen. 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. In der Allgemeinheit ist Spracherkennung vor allem durch die Sprachassistenten von Apple, Google und Amazon bekannt.

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 9 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).

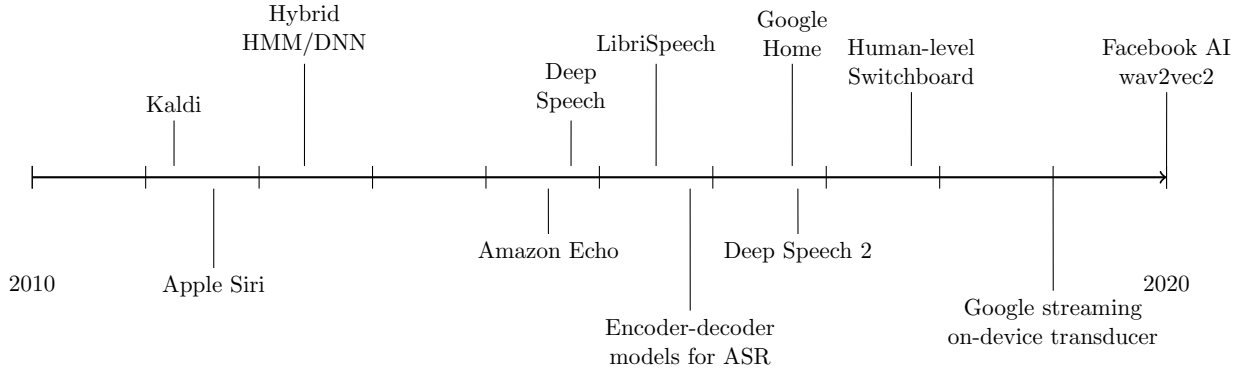

Abbildung 9: Timeline of Speech Recognition Developments (2010-2020)

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 Komparative Analyse von Sprachassistenten

TODO: - (**matarneh2017speechrecognition**) bietet eine komparative Analyse von Sprachassistenten. Die Analyse "Voice control implementation may be conditionally divided into parts: speech, recognition, translation, and execution of commands (Fig. 1)."

3.3 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 10 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).

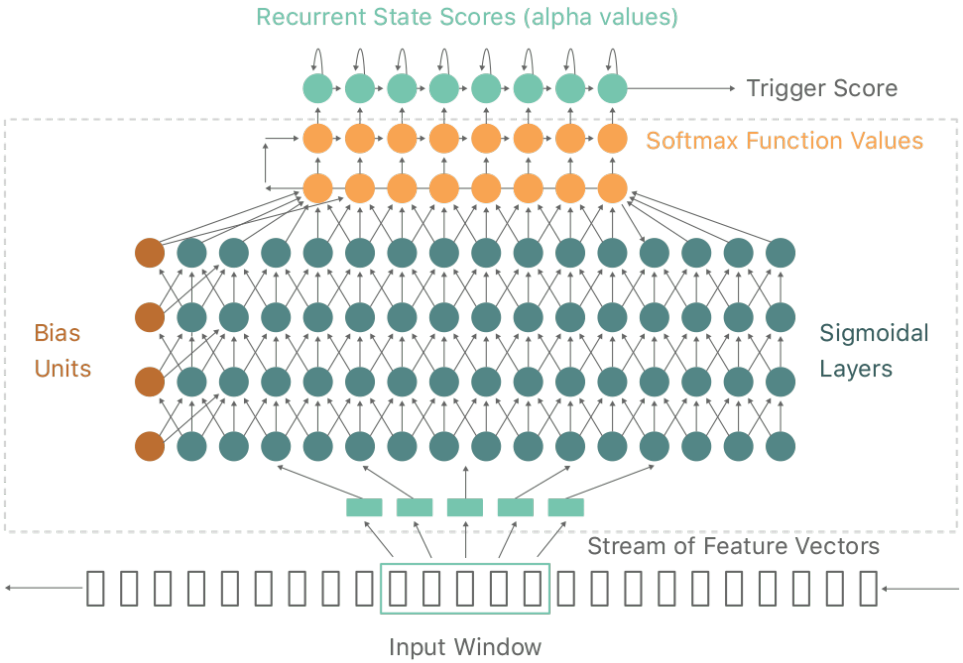

Abbildung 10: DNN Architektur von Siri

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."

3.3.1 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.

TODO: Offenlegung der Funktionsweise von Siri. Wie funktioniert Siri? Als Beispiel eines OS übergreifenden Sprachassistenten. (Siri-Team, 2017) und (Apple, 2023)

- Untersuchung und Darstellung der Funktionsweise von Siri.
- Nutzung der Quellen Siri-Team, 2017 und Apple, 2023 zur detaillierten Erklärung von Siris Arbeitsweise.
- Überprüfen der Links für zusätzliche Informationen:
  - https://machinelearning.apple.com/research/hey-siri
  - https://machinelearning.apple.com/research/voice-trigger

3.4 In App Sprachsteuerung

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:

- **Individuell**: Die Sprachsteuerung kann individuell auf die App angepasst werden.
- **Unabhängigkeit**: Die App ist nicht von Siri oder anderen Sprachassistenten abhängig.
- **Erweiterte Funktionen**: Die Sprachsteuerung kann erweiterte Funktionen bieten, welche von Siri nicht unterstützt werden.
- **Datenschutz**: Die Sprachsteuerung kann auf dem Gerät selbst laufen und somit keine Daten in die Cloud senden.
- **Branding**: Die Sprachsteuerung kann das Branding der App stärken.

3.5 Marktanalyse - Benchmarking

Während der Recherche für diese Arbeit wurden natürlich auch die bestehenden Lösungen auf dem Markt analysiert. Dabei wurde der folgende 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, welches Triggerwörter erkennt. Dazu wird ein Datensatz erstellt, welcher die Triggerwörter, sowie andere Wörter enthält.

In einem ersten Schritt wird ein eigenes Modell trainiert, welches Triggerwörter erkennt. Es gibt aber auch die Möglichkeit, ein bereits vortrainiertes Modell zu verwenden. Dieses Kapitel

4.1 Einleitung

- Kurze Wiederholung des Problems und der Vision für Ihr Projekt.
- Einführung in die Ideen und Konzepte, die Sie in Betracht ziehen, um dieses Ziel zu erreichen.

4.2 Grundlegende Idee

- Erklärung, wie eine integrierte Spracherkennung den Benutzererfahrungen in mobilen Apps verbessern könnte.
- Beispiel: Automatisierte To-Do-Liste-Einträge durch Sprachbefehle.

4.3 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.

4.4 Systemarchitektur

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

4.5 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.

4.6 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?

4.7 Erste Überlegungen zu Tools und Technologien

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

4.8 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

Am besten dieses Paper verwenden (Choi et al., 2018).

5.1 Vorgehensmodell

- Auswahl des passenden Vorgehensmodells zur Integration einer Spracherkennungsfunktion, z.B. iteratives Modell.
- Verweis auf den konkreten Terminplan mit Meilensteinen spezifisch für die Spracherkennungs-Integration (im Kapitel 5 oder im Anhang).

5.2 Forschungsmethoden (falls anwendbar)

- 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.

5.3 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.

5.4 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.

5.5 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

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.

6.1 Google Cloud Platform

TODO: Google Cloud Platform um Datensätze zu hosten auf Sorage Bucket. TODO: Recorder Webseite für Aufnahme von Sprachsamples (Warden, 2018

In den Sample zu VoiceCommands wird eine Recorder Webseite angeboten, mit welcher Sprachsamples aufgenommen werden können. Selbsterstellen einer kleinen Seite Hosting dieser Webseite auf GCP https://github.com/petewarden/open-speech-recording/tree/master

Fork (Warden, 2018)

6.1.1 Recorder Webseite

6.2 Datensatz

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

6.2.1 Datensatz 'other

Am besten von Datenset (Warden, 2018)

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

6.2.2 Datensatz 'hey-fooby'

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

6.3 Modell

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

6.3.1 Modell 'CNN'

- Aufbau des Modells
- TODO: Beschreibung des Modells

6.3.2 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

9.1 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.

9.1.1 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 11: Tabelle für das anfängliche Product Backlog

9.1.2 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

9.2 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 12 dargestellt.

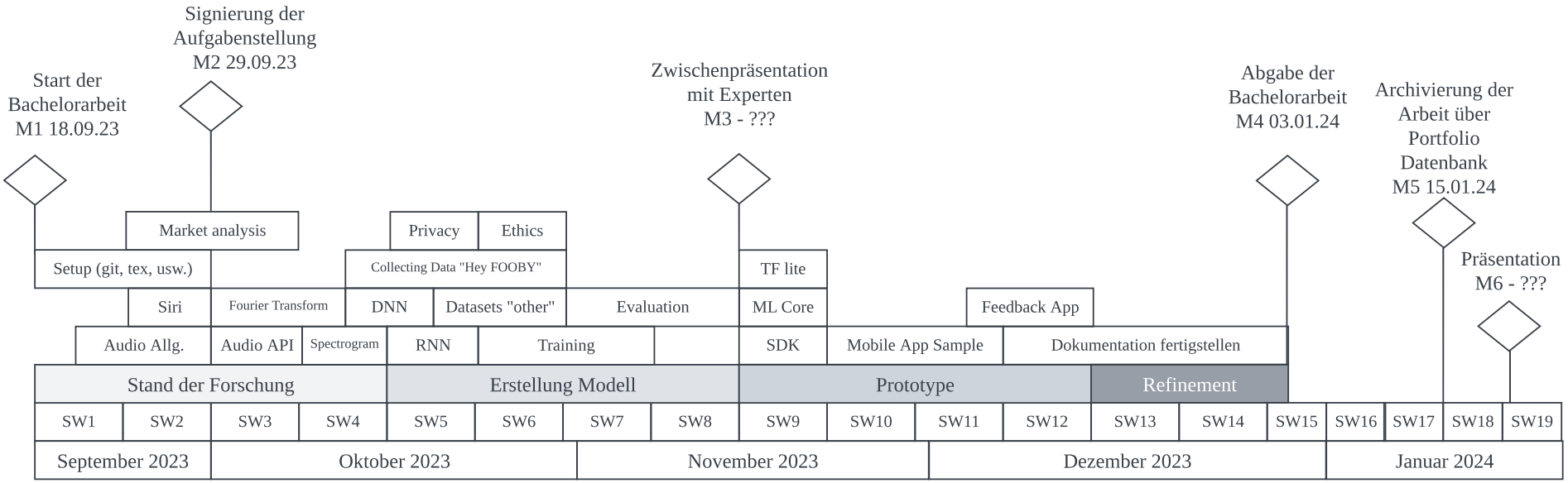

Abbildung 12: Grobplanung

Glossar

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

Abbildungsverzeichnis

1 Frames, Channels und Buffers 8
2 Rechteckfunktion und ihre Fourier-Transformation 10
3 Funktion $f(x)$ mit 5 Samples 11
4 Rekonstruktion des Signals $f(x)$ 12
5 Spektrogramm 13
6 Neuronale Netze 14
7 Ein neuronales Netzwerk mit drei Layern. 14
8 Convolution 15
9 Timeline of Speech Recognition Developments (2010-2020) 16
10 DNN Architektur von Siri 17
11 Tabelle für das anfängliche Product Backlog 26
12 Grobplanung 27

Tabellenverzeichnis

1 Frames in Interleaved und Non-interleaved Buffers 8
2 $f(x)$ und die DFT der 5 Samples 12
3 Identifizierte Risiken im Projekt 26

Literaturverzeichnis

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

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.

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 Sprachsteuerungstechnologien 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.