Bachelorarbeit

Integration von Sprachsteuerung in Mobile Apps

Rubén Nuñez

Herbstsemester 2023

Bachelorarbeit an der Hochschule Luzern – Informatik

Titel: Integration von Sprachsteuerung 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

Diese Bachelorarbeit konzentriert sich auf die Herausforderung, die Grundlage für eine integrierte Sprachsteuerung in mobilen Apps zu schaffen. Der Fokus liegt hierbei auf der Entwicklung einer Triggerwort-Erkennung, die es ermöglicht, bestimmte Wörter oder Wortsequenzen in der akustischen Sprache zu erkennen. Da integrierte Sprachsteuerungsfunktionen in mobilen Apps eher unüblich sind, bietet diese Arbeit eine innovative Lösung an. Der Ansatz umfasst drei Hauptkomponenten: die Erstellung eines ethisch und rechtlich einwandfreien Datensatzes, die Entwicklung und das Training eines effizienten Modells zur Erkennung von Triggerwörtern, und die Integration dieses Modells in eine mobile Anwendung. Grundsätzlich beschreibt diese Arbeit einen veralgemeinerten Ansatz wie Sprachsteuerungsfunktionen in mobile Apps integriert werden können. Dennoch wurde als möglicher Use Case eine Integration in die FOOBY App gewählt. Aus diesem Grund wurde der Datensatz mit Sprachsamples von 'Hey FOOBY' erstellt.

Die Erstellung des Datensatzes erfolgte durch die Aufnahme von Sprachdaten mit einer Webanwendung, die speziell für diesen Zweck entwickelt wurde. Bei Aufnahme der Daten wurde besonderes Augenmerk auf Datenschutz und ethische Aspekte gelegt. Das Ergebnis ist ein Datensatz, der die Grundlage für die Entwicklung eines Modells zur Erkennung von Triggerwörtern bildet. Die ethischen Aspekte befassen mit Überlegungen zu Diversität und Inklusion, sowie der Vermeidung von Diskriminierung. Zu den Datenschutzaspekten gehören die Einhaltung von Datenschutzrichtlinien und die Vermeidung von Datenschutzverletzungen.

Im Zentrum der Arbeit steht die Entwicklung eines Machine Learning-Modells, basierend auf einer Kombination aus Convolutional und Long-Short-Term-Memory (LSTM) Layern. Diese Architektur, implementiert in PyTorch, zeichnet sich durch Energieeffizienz und schnelle Inferenz aus. Das Modell, benannt als WakeupTriggerConvLSTM2s, wurde daraufhin trainiert, Triggerwörter aus akustischen Signalen zu erkennen. In einem Real-Time Verfahren werden die Sprachdaten in Spektrogramme transformiert, die als Input für das Modell dienen.

Ein entscheidender Aspekt der Arbeit war die Integration des Modells in mobile App. Hierfür wurde das Modell mithilfe von PyTorch Scripting in ein für mobile Apps geeignetes Format konvertiert und in einem Prototyp eingebettet, der einen kontinuierlichen Audio-Stream in Echtzeit verarbeitet. Die App kann Triggerwörter in 2-Sekunden-Abschnitten erkennen, wobei die Inferenz 10-mal pro Sekunde durchgeführt wird, um eine hohe Erkennungsgenauigkeit zu gewährleisten.

Insgesamt bietet diese Arbeit eine grundlegende Basis für die Entwicklung von Sprachsteuerungsfunktionen in mobilen Anwendungen, mit besonderem Augenmerk auf Effizienz, Datenschutz und ethische Aspekte. Die Ergebnisse können als Ausgangspunkt für die Implementierung vollständiger Sprachassistenten in mobilen Apps dienen.

Inhaltsverzeichnis

1 Problem, Fragestellung, Vision **6**
1.1 Problemstellung . . . 6
1.2 Fragestellung . . . 6
1.3 Vision . . . 6
1.4 Ziele . . . 6
1.5 Abgrenzung . . . 7
1.6 Rechtfertigung . . . 7

2 Grundlagen **8**
2.1 Audio . . . 8
2.1.1 Sampling . . . 8
2.1.2 Frames, Channels, Buffers . . . 8
2.1.3 Buffers im Detail . . . 9
2.2 Audio APIs . . . 9
2.2.1 Audio API für Analyse . . . 10
2.2.2 Audio API für Integration . . . 11
2.3 Fourier-Analyse . . . 12
2.3.1 Fourier-Transformation . . . 12
2.3.2 Diskrete Fourier-Transformation . . . 13
2.3.3 Aliasing . . . 15
2.4 Spektrogramm . . . 15
2.5 Machine Learning . . . 16
2.5.1 Neuronale Netze . . . 16
2.5.2 Convolutional Neural Networks . . . 17
2.5.3 Recurrent Neural Networks . . . 18
2.5.4 Metriken . . . 18

3 Stand der Forschung **20**
3.1 Zeitliche Entwicklung der Spracherkennung . . . 20
3.2 Aufbau von Sprachassistenten . . . 20
3.3 Vergleich von Sprachassistenten . . . 21
3.4 Facebook AI wav2vec2 . . . 21
3.5 Funktionsweise von Siri . . . 23
3.5.1 Takeaways . . . 24
3.6 Marktanalyse - Trigger Wort Erkennung . . . 24
3.7 Diskussion . . . 24

4 Ideen und Konzepte **25**
4.1 Grundlegende Idee . . . 25
4.2 Erstellung eines Datensatzes . . . 25
4.3 Ethische Überlegungen . . . 26
4.4 Datenschutz und Privatsphäre . . . 26
4.5 Erste Überlegungen zu Tools und Technologien . . . 26
4.6 Erste Überlegungen zur Modelarchitektur . . . 26
4.7 Versuche . . . 26
4.7.1 Versuch 1: Facebook AI wav2vec2 . . . 27
4.7.2 Versuch 2: Fine-Tuning des Facebook AI wav2vec2-Modells . . . 28
4.7.3 Versuch 3: LiteFEW . . . 29
4.7.4 Versuch 4: ConvLSTM . . . 29

5 Methoden **30**
5.1 Projektphasen . . . 30
5.2 Projektmanagement . . . 30
5.2.1 Produkt Backlog . . . 31
5.2.2 Risikomanagement . . . 31
5.3 Grobplanung . . . 32

6 Realisierung **33**
6.1 Aufbau des Datensatzes . . . 33
6.1.1 Recorder Webseite . . . 33
6.1.2 Google Cloud Platform als Unterstützung . . . 34
6.1.3 Augmentierung der Sprachdaten . . . 35
6.1.4 Datensatz 'hey-fooby' . . . 35
6.1.5 Datensatz 'other' . . . 35
6.2 Erstellung des Modells . . . 36
6.2.1 Verwendete Architektur des Modells . . . 36
6.2.2 Transformation, Preprocessing, Feature Extraction . . . 36
6.2.3 Training des Modells . . . 37
6.2.4 PyTorch Scripting . . . 38
6.2.5 Real Time Anwendung in Python . . . 38
6.3 Integration in eine mobile App . . . 40

7 Evaluation und Validation **41**

8 Ausblick **42**

9 Anhang **43**
9.1 Storyboard BA-thesis-video . . . 43

Abbildungsverzeichnis **45**

Tabellenverzeichnis **45**

Literaturverzeichnis **45**

1 Problem, Fragestellung, Vision

1.1 Problemstellung

Die Sprachsteuerung bietet grosses Potenzial in Mobile Apps und wird bisher vor allem für Sprachassistenten genutzt. Während Sprachassistenten wie Siri, Google Assistant oder Alexa von der Allgemeinheit genutzt werden, gibt es kaum bekannte 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 vor allem mit der *Trigger Word Detection (TWD)*. Die nachfolgende Abbildung 1 zeigt mögliche Teilgebiete der Spracherkennung und soll einen Überblick über die Thematik verschaffen.

Abbildung 1: Teilgebiete der Spracherkennung

1.2 Fragestellung

Wie lässt sich eine Sprachsteuerungsfunktion entwickeln, die effektiv Triggerwörter in akustischer Sprache erkennt und wie kann diese Funktion nahtlos in mobile Applikationen integriert werden?

1.3 Vision

Ziel dieser Arbeit ist es, eine solide Basis für die Erkennung von Triggerwörtern oder Sequenzen in akustischer Sprache zu schaffen, unter Verwendung von Methoden und Werkzeuge aus dem Bereich des Machine Learning. Ein wesentlicher Aspekt ist die Implementierung dieser Erkenntnisse auf einer mobilen Plattform wie iOS oder Android. Für den Umfang dieser Arbeit ist die Integration in eine dieser Plattformen ausreichend. Bei der Entwicklung des Datensatzes wird besonderes Augenmerk auf Datenschutz und ethische Aspekte gelegt.

1.4 Ziele

Die Ziele dieser Arbeit, nach SMART-Prinzip, werden in der folgenden Tabelle 1 aufgeführt.

Ziel	SMART-Kriterien
Erkennen von Triggerwörtern	**Spezifisch:** Entwicklung eines Modells zur Erkennung von von einem oder mehreren Triggerwörtern in der akustischen Sprache. **Messbar:** Erfolgsrate der korrekten Erkennung. **Ausführbar:** Umsetzung mit aktuellen Machine Learning-Methoden. **Realistisch:** Realisierbar mit bestehenden Technologien. **Terminiert:** Fertigstellung bis zum Ende der Bachelorarbeit.
Integration in eine Mobile App	**Spezifisch:** Einbindung des erarbeiteten Modells in eine iOS- oder Android-App. **Messbar:** Durch User-Tests und Messung der Erfolgsrate **Ausführbar:** Machbar mit vorhandenen Entwicklungswerkzeugen und Plattformen. **Realistisch:** Durchführbar innerhalb des Projektzeitrahmens. **Terminiert:** Integration bis zum Abschluss der Bachelorarbeit.
Datenschutz und Ethik	**Spezifisch:** Datenschutz und Ethik bei der Datensatzerstellung beachten. **Messbar:** Erfüllung spezifischer Datenschutzrichtlinien und ethischer Standards im Datenerhebungsprozess. **Ausführbar:** Einhaltung von Best Practices und Richtlinien während der Datensammlung und -aufbereitung. **Realistisch:** Umsetzbar durch sorgfältige Planung und Überprüfung der Prozesse. **Terminiert:** Kontinuierliche Überwachung und Anpassung während der Datensatzerstellung.

Tabelle 1: SMART Ziele der Bachelorarbeit

1.5 Abgrenzung

Diese Arbeit befasst sich mit der Erkennung von Triggerwörtern in der akustischen Sprache. Es ist kein Ziel dieser Arbeit, einen vollständigen Sprachassistenten zu entwickeln. Das würde den Rahmen einer Bachelorarbeit sprengen. Die Resultate dieser Arbeit könnten aber als Grundlage für eine vollständige Implementierung eines Sprachassistenten genutzt werden.

1.6 Rechtfertigung

Apple sowie Google stellen mit Siri und Google Assistant bereits Sprachassistenten 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 anbieten. Als Beispiel hat Spotify eine Integration mit Siri. Es können Befehle wie "Hey Siri, play music on Spotify" verwendet werden. Die Integration einer eigens entwickelten Sprachsteuerung in Apps ist trotz verfügbarer Lösungen wie Siri und Google Assistant aus mehreren Gründen vorteilhaft:

- **Personalisierung:** Sie ermöglicht eine massgeschneiderte Nutzererfahrung, die exakt auf die App und ihre Nutzer zugeschnitten ist.
- **Unabhängigkeit:** Apps bleiben unabhängig von den Schnittstellen und Einschränkungen externer Assistenten.
- **Funktionalität:** Es können spezifische Funktionen implementiert werden, die über das hinausgehen, was standardisierte Assistenten bieten.
- **Datenschutz:** Durch lokale Verarbeitung von Sprachdaten wird die Privatsphäre der Nutzer besser gewahrt.
- **Markenidentität:** Eine einzigartige Sprachsteuerung verstärkt das Markenbild und bietet ein differenziertes Nutzererlebnis.

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 2 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 2 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 2: 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. Der folgende Source Code (1) zeigt die Implementierung.

```python
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
```

Source Code 1: SoundDevice Audio Stream

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 der folgenden Abbildung 3 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 $\mathrm{rect}(x)$, die wie folgt definiert ist:

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

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

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

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

Abbildung 4: Rechteckfunktion und ihre Fourier-Transformation

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

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 Source Code Beispiel (2). 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.

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

.

```
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
```

Source Code 2: Unterteilung der Funktion $f(t)$ in 5 Samples

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 3 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 3: $f(x)$ 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: Neuron eines neuronalen Netzes

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, Chapter 4).

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 spezialisierte neuronale Netze, hauptsächlich eingesetzt in der Bilderkennung. Ihre Fähigkeit, komplexe Muster in Bildern zu erkennen, macht sie zu einem unverzichtbaren Werkzeug in der Bildanalyse. Zentral für CNNs sind die *Convolutional Layers*, die durch einen Filter gekennzeichnet sind. Dieser Filter gleitet über den Input, um spezifische Merkmale zu extrahieren, wie in Abbildung 10 dargestellt.

Abbildung 10: Convolution

Im Kontext von CNNs repräsentieren die Gewichte $w_1, w_2, \ldots, w_n$ und der Bias b die Elemente des Filters. Diese werden während des Trainings angepasst, um optimale Ergebnisse zu erzielen. Ein signifikanter Vorteil von CNNs im Vergleich zu traditionellen neuronalen Netzen ist die Fähigkeit, Gewichte zwischen verschiedenen Teilen des Netzwerks zu teilen, wodurch eine effizientere Mustererkennung ermöglicht wird. (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."

Die spezielle Funktionsweise von *Convolutional Layers* in CNNs ermöglicht eine effektive Verarbeitung von Bilddaten. Sie identifizieren und extrahieren relevante Merkmale direkt aus den Rohdaten, was sie besonders für Bilderkennungsaufgaben geeignet macht. Dieses effiziente Feature-Learning wird durch die einzigartige Struktur der Convolutional Layers ermöglicht, die in der Fähigkeit besteht, komplexe Muster in den Daten zu identifizieren und zu verarbeiten (Weidman, 2019 Chapter 5).

2.5.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) sind eine spezielle Art von neuronalen Netzen, die vor allem dazu verwendet werden, um Sequenzen zu verarbeiten. Gewöhnliche neuronale Netze verarbeiten Daten als unabhängige Einheiten, ohne die Beziehung zwischen den Daten zu berücksichtigen. RNNs hingegen sind so aufgebaut, dass sie die Beziehung zwischen den Daten berücksichtigen. Die Abbildung 11 zeigt ein Beispiel eines RNNs.

Abbildung 11: Recurrent Neural Network

Eine wichtige Erweiterung der RNNs sind Long Short-Term Memory Networks (LSTMs). LSTMs wurden entwickelt, um die Herausforderungen von RNNs, insbesondere das Problem des verschwindenden Gradienten bei langen Sequenzen, zu überwinden. LSTMs behalten die grundlegenden Eigenschaften von RNNs bei, führen jedoch eine zusätzliche Struktur ein: den Zellzustand. Neben dem versteckten Zustand (hidden state), der bei traditionellen RNNs verwendet wird, haben LSTMs einen Zellzustand (cell state), der es ihnen ermöglicht, Informationen über längere Zeiträume hinweg zu bewahren und zu verarbeiten.

Dieser Zellzustand arbeitet als eine Art "Gedächtnis"der LSTM, das es ihm ermöglicht, sowohl kurzfristige als auch langfristige Abhängigkeiten in Daten zu erkennen und zu lernen. Dies ist besonders nützlich in Anwendungen wie der Spracherkennung oder der Vorhersage von Zeitreihen, wo die Beziehung von Datenpunkten über längere Zeiträume hinweg entscheidend ist. (Weidman, 2019 Chapter 6).

2.5.4 Metriken

Neuronale Netze werden trainiert, um spezifische Ausgaben zu generieren. Zur Bewertung ihrer Performance dienen verschiedene Metriken, die in diesem Kapitel vorgestellt werden. Neben der bereits besprochenen Loss Function, die den Unterschied zwischen der tatsächlichen und der gewünschten Ausgabe quantifiziert, nutzen wir weitere Metriken. Die nachstehende Formel illustriert den *Mean Squared Error* (MSE), eine gängige Methode, um den Loss zu berechnen:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Vor der Betrachtung weiterer Metriken ist das Verständnis der *Confusion Matrix* wesentlich. Sie veranschaulicht die Anzahl korrekter und inkorrekter Vorhersagen des Modells. In dieser Arbeit konzentrieren wir uns auf die Klassifizierung eines Triggerworts, für welche die Confusion Matrix in Tabelle 4 abgebildet ist.

	Predicted	
Actual	**Triggerwort**	**Kein Triggerwort**
Triggerwort	True Positive (TP)	False Negative (FN)
Kein Triggerwort	False Positive (FP)	True Negative (TN)

Tabelle 4: Confusion Matrix

Mit der Confusion Matrix können wir nun die anderen Metriken betrachten. Die *Accuracy* zeigt den Anteil der korrekten Vorhersagen. Die Accuracy wird wie folgt berechnet:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Die *Precision* zeigt den Anteil der korrekten positiven Vorhersagen. Die Precision wird wie folgt berechnet:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Die *Recall* zeigt den Anteil der korrekten positiven Vorhersagen. Die Recall wird wie folgt berechnet:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Die *F1-Score* ist das harmonische Mittel zwischen Precision und Recall. Die F1-Score wird wie folgt berechnet:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3 Stand der Forschung

In diesem Kapitel werden die wichtigsten zeitlichen Entwicklungen im Bereich der Spracherkennung aufgezeigt. 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. Weiter betrachten wir eine kommerzielle Lösung von Picovoice, welche aufzeigen soll, dass es einen Bedarf für eine integrierte Spracherkennungslösung gibt.

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 12 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 12: Timeline of Speech Recognition Developments (2010-2020)

Deep Learning hatte einen grossen Einfluss auf die Revolutionierung der Spracherkennung, 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 13 zeigt

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

Abbildung 13: 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).

3.3 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).

3.4 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.

3.5 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 14 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 14: 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." (Siri-Team, 2017)

3.5.1 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.

3.6 Marktanalyse - Trigger Wort Erkennung

Während der Recherche für diese Arbeit wurden bestehende Lösungen auf dem Markt gesucht. Nebst den bereits erwähnten Lösungen im Unterkapitel 3.3 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).

3.7 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 15 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 15: 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 namens "FOOBY" 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 Erstellung eines Datensatzes

Die Erstellung eines Datensatzes ist ein wichtiger Bestandteil dieser Arbeit. Daher soll eine Möglichkeit erarbeitet werden, um Sprachaufnahmen zu machen. Der Datensatz soll aus Sprachaufnahmen bestehen, welche Triggerwörter enthalten. Ebenfalls sollen die Sprachaufnahmen auch andere Wörter enthalten. Neben den eigenen Sprachaufnahmen sollen auch bestehende Datensätze untersucht werden.

Ein sehr verbreiteter Datensatz ist der *Mozilla Common Voice* Datensatz (Ardila et al., 2020). Dieser Datensatz enthält unzählige Sprachaufnahmen von Menschen, welche freiwillig ihre Stimme aufgenommen haben.

4.3 Ethische Überlegungen

Das Erstellen eines Datensatzes hat eine gewisse Verantwortung. Es geht dabei nicht nur um die Einhaltung des Datenschutzgesetzes, sondern auch um ethische Aspekte. Ein Datensatz sollte vielfältige Sprachaufnahmen beinhalten und nicht nur Aufnahmen bestimmter Personengruppen. Das Paper von Papakyriakopoulos et al. (Papakyriakopoulos et al., 2023) betont die Notwendigkeit der Diversität in Sprachdaten und die Auswirkungen auf Fairness und Robustheit von Sprachtechnologien. Sie empfehlen, mehr Transparenz bei der Datensammlung zu schaffen, um ethische Aspekte zu dokumentieren und die Überlegungen zum sozialen Kontext der Datennutzung zu vertiefen.

In Bezug auf die Erstellung eines Datensatzes für diese Arbeit ist es beispielsweise wichtig, dass die Sprachaufnahmen aus diversen Sprachregionen stammen. In der Schweiz gibt es vier offizielle Sprachen und unzählige Dialekte. St. Galler Dialekt hat einen anderen Klang als der Zürcher Dialekt. Ebenfalls gibt es Personen mit Sprachfehlern oder Personen mit Migrationshintergrund, welche eine andere Sprache als Muttersprache haben. All diese Aspekte sollten bei der Erstellung eines Datensatzes berücksichtigt werden.

4.4 Datenschutz und Privatsphäre

Die Erstellung des Datensatzes und die Verwendung von Sprachaufnahmen soll unter Einhaltung des Datenschutzgesetzes erfolgen. Die Sprachaufnahmen sollen nicht an Dritte weitergegeben werden. Die Teilnehmer sollen über die Verwendung der Sprachaufnahmen informiert werden. Ebenfalls sollen die Sprachaufnahmen nach der Verwendung gelöscht werden.

4.5 Erste Überlegungen zu Tools und Technologien

Für die Entwicklung von Machine-Learning-Modellen stehen mehrere Frameworks zur Verfügung. Eine Schlüsselüberlegung für dieses Projekt war die Fähigkeit zur Integration in eine mobile App. Es gibt einge Frameworks, welche das ermöglichen. Um nur einige zu nennen: PyTorch, TensorFlow, CoreML. Wobei CoreML nur für Apple Ekosysteme verwendet werden kann. Da nicht nur die Integration in eine mobile App wichtig ist, sondern auch die Entwicklung von Prototypen, wurde PyTorch gewählt. TensorFlow wäre ebenfalls eine gute Wahl gewesen, aber laut (Valantis, 2023) hat PyTorch in den letzten Jahren an Popularität gewonnen. Ebenfalls ist PyTorch sehr beliebt für die Forschung und bietet eine Integration in mobile Apps. Daher wurde PyTorch als Framework für die Entwicklung der hier beschriebenen Arbeit gewählt.

4.6 Erste Überlegungen zur Modelarchitektur

Mit der Wahl von PyTorch als Framework wurde zwar eine wichtige Entscheidung getroffen, aber die Modelarchitektur ist noch nicht definiert. Die Inspiration für die Modelarchitektur kommt von (Siri-Team, 2017). Die Modelarchitektur von Siri ist sehr einfach und besteht aus einem DNN. Daher war ein erster Gedanke, eine ähnliche Architektur zu verwenden. Im folgenden werden die unterschiedlichen Versuche die im Rahmen dieser Arbeit gemacht wurden, beschrieben. Als Experiment stand immer die Erkennung von 'Hey FOOBY' im Vordergrund.

4.7 Versuche

Als Teil der Ideen und Konzepte wurden verschiedene Versuche beziehungsweise Experimente gestartet. Diese Versuche sind wichtig, um die Machbarkeit der Triggerwort Erkennung zu überprüfen. Da aus dem Kapitel Stand der Forschung hervorgeht, dass das Model wav2vec2 von Facebook AI

einen grossen Fortschritt in der Spracherkennung darstellt, basieren die ersten drei Versuche auf der Verwendung von wav2vec2.

4.7.1 Versuch 1: Facebook AI wav2vec2

Die ersten Versuche befassten sich mit der Verwendung von Facebook AI's wav2vec2. Dieses Modell ist zwar nicht für die Triggerwort Erkennung entwickelt worden, aber es hat das Potenzial diese Aufgabe zu erfüllen. Daher wurde ein Versuch gestartet, um zu sehen, ob es möglich ist mit wav2vec2 Triggerwörter zu erkennen. Die folgende Abbildung 16 zeigt den Code für die Verwendung von wav2vec2.

```python
processor = Wav2Vec2Processor.from_pretrained("./wav2vec2-base-960h")
model = Wav2Vec2ForCTC.from_pretrained("./wav2vec2-base-960h")

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

def transcribe_chunk(audio_chunk):
    if len(audio_chunk.shape) > 1:     # Ensure audio is mono
        audio_chunk = np.mean(audio_chunk, axis=1)


    input_values = processor( # Tokenize
        audio_chunk,
        return_tensors="pt",
        padding="longest",
        sampling_rate=16000
    ).input_values

    if torch.cuda.is_available():
        input_values = input_values.to("cuda")

    with torch.no_grad(): # Retrieve logits and decode
        logits = model(input_values).logits
    predicted_ids = torch.argmax(logits, dim=-1)
    transcription = processor.batch_decode(predicted_ids)

    return transcription

if __name__ == "__main__":
    CHUNK_DURATION = 1  # seconds
    OVERLAP_DURATION = 0.5  # seconds
    overlap_buffer = np.array([])

    for audio_chunk in stream_audio(CHUNK_DURATION):
        audio_chunk = np.squeeze(audio_chunk)  # Convert to 1D array
        audio_chunk_with_overlap = np.concatenate([overlap_buffer, audio_chunk])
        result = transcribe_chunk(audio_chunk_with_overlap)


        words = result[0].split() # Print latest word from transcription
        if words:
            print(words[-1])

        # Store overlap for next iteration
        overlap_buffer = audio_chunk[-int(OVERLAP_DURATION * 16000):]
```

Abbildung 16: Verwendung von wav2vec2

Die ersten Versuche mit wav2vec2 zeigten, dass es möglich ist, Triggerwörter zu erkennen, auch wenn es sich um ein Basismodell handelt, das noch nicht auf eine spezifische Aufgabe zugeschnitten wurde. In einer Testreihe mit drei 10-sekündigen Tests, in dem die Wörter "Hey FOOBY"mehrmals gesagt wurden, erkannte das Modell verschiedene Triggerwörter. Die zeitliche Abfolge dieser Erkennungen ist in Tabelle 5 dargestellt.

Zeitabschnitt	Test 1	Test 2	Test 3
0-2s	FOOBY	HEY FOOBE	E FOOBE
2-4s	F	B FOBE	E HA
4-6s	FHOBI	HEY	FUBI
6-8s	BE	FOOBE	HAL FUBI

Tabelle 5: Resultate der ersten Versuche mit wav2vec2

Diese Versuche haben gezeigt, dass es möglich ist ähnliche Wörter zu erkennen. Auch wenn die Resultate in erster Linie nicht sehr gut erscheinen, sind sie dennoch vielversprechend. Als Beispiel könnten die Wörter 'Hey FOOBY' und 'Hey FOOBE' als ähnliche Wörter betrachten werden. Durch die Berechnung der Levenshtein-Distanz könnte die Ähnlichkeit dieser Wörter berechnet werden. Die Levenshtein-Distanz ist eine Metrik zur Berechnung der Ähnlichkeit zweier Zeichenketten. Dieser Ansatz wurde jedoch nicht weiter verfolgt und stammt aus eigenen Überlegungen.

4.7.2 Versuch 2: Fine-Tuning des Facebook AI wav2vec2-Modells

Nach ersten Versuchen mit wav2vec2 wurde ein weiterer Ansatz verfolgt, der als Fine-Tuning bekannt ist. Hierbei wird ein bereits vortrainiertes Modell als Ausgangspunkt genutzt, welches dann für spezifische Aufgaben weiterentwickelt wird. Dies reduziert den ökologischen Fussabdruck durch geringeren Rechenaufwand (Hugging Face, 2023). Der Versuch umfasste die Anpassung des Basismodells durch Hinzufügen eines Layers, um das spezifische Triggerwort in Audio zu erkennen.

Das 'TriggerWordWav2Vec2Model' im Source Code 3 erweitert das wav2vec2-Modell um ein Sequential Layer. Diese Komponente besteht aus einem Linear Layer, welcher den Output des wav2vec2-Modells verarbeitet, und einer Sigmoid Activation Function. Der Linear Layer verarbeitet die Average Pooling-Repräsentation der Hidden States des wav2vec2-Modells, um zu entscheiden, ob das Triggerwort in der Audioeingabe vorhanden ist oder nicht. So die Grundidee.

```
class TriggerWordWav2Vec2Model(nn.Module):
        def __init__(self, config):
                super(TriggerWordWav2Vec2Model, self).__init__()
                self.wav2vec2 = Wav2Vec2ForCTC(config).wav2vec2
                self.classifier = nn.Sequential(
                        nn.Linear(config.hidden_size, 1),
                        nn.Sigmoid()
                )

        def forward(self, input_values):
                transformer_outputs = self.wav2vec2(input_values).last_hidden_state
                avg_pool = transformer_outputs.mean(dim=1)
                return self.classifier(avg_pool)
```

Source Code 3: TriggerWordWav2Vec2Model

Im Forward-Durchlauf des Modells werden die Audioeingaben durch das wav2vec2-Modell verarbeitet, das dabei relevante Merkmale aus den Audiodaten extrahiert. Vom Output des wav2vec2-Modells wird der Durchschnitt der Hidden States berechnet. Danch wird der Durchschnitt dem Linear Layer übergeben, welcher die Wahrscheinlichkeit berechnet, dass das Triggerwort in der Audioeingabe vorhanden ist. Die Wahrscheinlichkeit wird durch die Sigmoid Activation Function berechnet. Der Prozess wird im Beitrag "Fine-tuning XLS-R for Multi-Lingual ASR with Transformers" von Patrick von Platen beschrieben:

> "For fine-tuning, a single linear layer is added on top of the pre-trained network to train the model on labeled data of audio downstream tasks such as speech recognition, speech translation and audio classification" (von Platen, 2021)

Dieser Ansatz ermöglicht eine effiziente und zielgerichtete Anwendung des vortrainierten wav2vec2-Modells für die spezifische Aufgabe der Triggerwort-Erkennung. Es nutzt die fortschrittlichen Fähigkeiten des wav2vec2-Modells zur Merkmalsextraktion und kombiniert diese mit einem massgeschneiderten Klassifikationsmechanismus, um hohe Genauigkeit bei der Erkennung von Triggerwörtern zu erreichen. Dennoch wurde dieser Ansatz nicht weiter verfolgt, da das resultierende Modell nicht den Anforderungen entsprechen in Bezug auf die Integration in eine mobile App. Es ist zum einen zu gross in Bezug auf der Grösse in Megabytes und zum anderen zu energieintensiv beziehungsweise zu rechenintensiv in der Inferenz.

4.7.3 Versuch 3: LiteFEW

Nach den Misserfolgen mit wav2vec2 wurde ein weiterer vielversprechender Versuch gestartet. Dieser basiert auf dem Paper von lim et al. (Lim et al., 2023). Der Grundgedanke dieses Papers ist es, ein Modell auf Basis von wav2vec2 zu entwickeln, welches optimiert ist für die Triggerwort Erkennung auf mobilen Geräten. Das im Paper beschriebene Verfahren beinhaltet die Verwendung von wav2vec2 als Teacher-Model und ein kleineres Modell als Student-Model. Dieses Verfahren hätte den Vorteil die Problematik der im Versuch 2 beschriebenen Lösung zu umgehen. Doch an dieser Stelle wurde der Versuch abgebrochen, da die Implementation des Papers sehr komplex ist und kein Code veröffentlicht wurde.

4.7.4 Versuch 4: ConvLSTM

Nach den Misserfolgen mit wav2vec2 und LiteFEW war die Situation etwas frustrierend. Daher wurde ein weiterer Versuch gestartet, um ein eigenes Modell zu entwickeln. Dieser Versuch basiert auf dem Github Repository von Barazanji et al. (Barazanji und Spencer, 2023). Das Github Repository enthält eine Definition für ein Modell, welches mit TensorFlow implementiert wurde. Das von Barazanji et al. (Barazanji und Spencer, 2023) entwickelte Modell wurde für die Triggerwort Erkennung entwickelt und kann die Wörter 'Hey Ditto' erkennen. Ein Kritikpunkt an das Repository ist, dass es keine Dokumentation oder Quellenangaben enthält. Dennoch wurde das Modell als Grundlage für einen weiteren Versuch verwendet. Dazu wurde es in PyTorch übersetzt und entsprechend angepasst, sodass es auf den Anwendungsfall dieser Arbeit zugeschnitten ist. Das Verfahren selbst ist sehr interessant und basiert auf der Verwendung von Convolutional Layers in Kombination mit einem LSTM Layer. Hier konnte eine weitere Quelle gefunden werden, welche diese Art von Modelarchitektur verwendet. Das Paper von Khamees et al. (Khamees et al., 2021) beschreibt das Klassifizieren von Musik Genres mit einer ähnlichen Modelarchitektur. Eine detailierte Beschreibung der Modelarchitektur ist im Kapitel der Realisierung zu finden.

5 Methoden

Das Vorgehen dieser Arbeit kann als eine klassische Projektvorgehen beschrieben werden. Aus der Aufgabenstellung ging schon hervor, dass ein eigenes Machine 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 dem *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 der 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 der 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 klassisches Vorgehensmodell gewählt. Dies bedeutet, dass sowohl die Planung als auch die Umsetzung in klar definierte Arbeitspakete unterteilt sind.

5.2.1 Produkt Backlog

Der Product Backlog besteht primär aus den Arbeitspaketen, die in der Grobplanung definiert werden. Gewisse Arbeitspakete nehmen mehr Zeit in Anspruch als andere und erstrecken sich über mehrere Semesterwochen. Die Arbeitspakete werden in der Tabelle 6 aufgeführt.

Phase	**Semesterwoche**	**Beschreibung des Arbeitspakets**
Stand der Forschung	SW1/2	Setup (git, tes, usw.)
	SW2	Siri research
	SW2/3	Marktanlyse
	SW3	Audio Allg., Audio API
	SW3/4	Fourier Transform, Spektrogramm
Erstellung des Modells	SW4/5	DNN
	SW4/6	Collect Data "Hey FOOBY"
	SW5	RNNs verstehen
	SW5/6	Privacy und Ethik,
	SW5/6	Datasets 'other' evaluation
	SW6/7	Training
	SW7/8	Evaluation
Prototype	SW9	PyTorch Mobile integration
	SW10/11	Mobile Demo App Entwicklung
	SW12	Video für Bachelorarbeit
	SW11/12	Feedback der App sammeln
	SW12/14	Dokumentation fertigstellen

Tabelle 6: Hierarchische Auflistung der Arbeitspakete

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 7 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 7: 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 17 dargestellt.

Abbildung 17: Grobplanung

6 Realisierung

Die Realisierung dieser Arbeit besteht im wesentlichen aus drei Teilen. Welche die Ziele der Arbeit wiederspiegeln. Zum einen wurde ein Datensatz nach ethischen sowie rechtlichen Aspekten erstellt, welcher die Grundlage für die Experimente so wie das Training des Modells bildet. Weiter wurde ein Modell erarbeitet und anschliessend trainiert, welches Triggerwörter erkennt. Der letzte Teil der Realisierung ist die Integration dieses Modells in eine Sample App.

6.1 Aufbau des Datensatzes

Der Aufbau eines Datensatzes war zu Beginn der Arbeit geplant. Es war klar, dass ein Datensatz benötigt wird, um ein Modell zu trainieren. Der Datensatz wiederspiegelt die Grundlage des Problemfeldes. Es geht ja darum ein spezifisches Wort oder Wörter zu erkennen. Die nachfolgende Abbildung 18 zeigt die zwei Klassen des Datensatzes. Die Klasse 0 beinhaltet Sprachaufnahmen, welche das Triggerwort nicht enthalten. Die Klasse 1 beinhaltet Sprachaufnahmen, welche das Triggerwort enthalten.

Abbildung 18: Aufbau des Datensatzes

6.1.1 Recorder Webseite

Um den Datensatz zu erstellen, wurde eine Webseite entwickelt, welche die Aufnahme von Sprachsamples ermöglicht. Die Webseite wurde auf Google Cloud Platform (GCP) gehostet, um eine zuverlässige und skalierbare Lösung zu gewährleisten. Die Webseite 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. Bei der zugrunde liegenden Anwendung handelt es sich um eine Python Flask App, die für ihre Leichtigkeit und Flexibilität bekannt ist. Die 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).

Die Anpassungen der Webseite für diese Arbeit umfassten die folgenden Punkte:

- **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.

Mit der überarbeiteten Recorder Webseite wurden Sprachaufnahmen von zwei Sekunden Länge und einer Sample Rate von 48kHz erstellt. Bei den freiwilligen Teilnehmern handelte es sich um Personen aus dem Bekanntenkreis des Autors. Um den Datenschutz zu gewährleisten, wurden die Teilnehmer über die Nutzung der Sprachaufnahmen informiert. Das Paper von Papakyriakopoulos et al. (Papakyriakopoulos et al., 2023) betont die Notwendigkeit der Diversität in Sprachdaten. Die Aufnahmen beinhalten Aufnahmen von mindestens 10 verschiedenen Personen. Da die Variation der Sprachaufnahmen relativ gering war, wurden zusätzlich Augmentierungen der Sprachaufnahmen durchgeführt. Diese werden im Kapitel 6.1.3 beschrieben. Die nachstehende Abbildung gibt einen visuellen Überblick über das Erscheinungsbild und die Funktionalität der Webseite.

Abbildung 19: Die überarbeitete Recorder Webseite

6.1.2 Google Cloud Platform als Unterstützung

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.

6.1.3 Augmentierung der Sprachdaten

Um die Diversität und Robustheit der Sprachaufnahmen zu erhöhen und das Modell für die Erkennung von Triggerwörtern zu stärken, wurden gezielte Augmentierungstechniken eingesetzt. Dabei kamen Pitch Shifting zur Variation der Tonhöhe, Time Shifting zur Simulation zeitlicher Verschiebungen, Volume Control für Lautstärkeanpassungen und Noise Injection zur Ergänzung von Hintergrundgeräuschen zum Einsatz. Diese Massnahmen zielen darauf ab, die Generalisierungsfähigkeit des Modells zu verbessern und seine Leistung unter verschiedenen akustischen Bedingungen zu optimieren. Die Verwendung von Datenaugmentierung hat sich insbesondere bei einer begrenzten Anzahl von Samples als wirkungsvoll erwiesen und ist eine etablierte Methode in der Praxis. In Anlehnung an das Paper von Salamon et al. (Salamon und Bello, 2017), das den Einsatz verschiedener Augmentierungstechniken zur Verbesserung der Klassifikation von Umgebungsgeräuschen diskutiert, wurde das vorgeschlagene Verfahren angepasst, um die Herausforderungen bei der Erkennung von spezifischen akustischen Signalen, wie Triggerwörtern, zu meistern. Die effektive Kombination dieser Techniken im Trainingsprozess führte zu einem erweiterten und robusten Datensatz, der für die Entwicklung eines leistungsfähigen Erkennungssystems unerlässlich ist.

6.1.4 Datensatz 'hey-fooby'

Insgesamt wurden etwa 700 Sprachaufnahmen mit dem Recorder erstellt. 300 davon enthielten das Triggerwort 'Hey FOOBY', und 400 Aufnahmen bestanden aus anderen Wörtern. Durch Augmentierungstechniken wurde die Anzahl der Samples auf 10'000 erhöht, wobei ein Verhältnis von 1 zu 2 im Vergleich zum 'other' Datensatz beibehalten wurde. Die daraus resultierende Imbalance wurde bei dem Training der finalen Modellarchitektur berücksichtigt.

6.1.5 Datensatz 'other'

Der 'other' Datensatz umfasst Sprachaufnahmen, die andere Wörter als das Triggerwort beinhalten. Von den ursprünglich 400 Aufnahmen wurde die Menge durch Augmentierung und die Integration weiterer Datensätze, darunter auch solche von Mozilla Common Voice (Ardila et al., 2020), erhöht. Während des Trainings stellte sich heraus, dass das Modell auf Silence nicht adäquat reagierte, was die nachträgliche Hinzufügung von Silence Samples zum 'other' Datensatz motivierte. Gleiches galt für das Hinzufügen von Noise Samples, um die Reaktion auf Hintergrundgeräusche zu verbessern.

6.2 Erstellung des Modells

Bei der Erstellung des Modells wurde die Architektur von Barazanji et al. (Barazanji und Spencer, 2023) verwendet. Das Modell wurde in PyTorch implementiert als ein PyTorch Modul und anschliessend trainiert. Das Training wurde auf einem MacBook Pro mit einem M1 Pro Chip durchgeführt. Die Trainingszeit betrug für 120 Epochen etwa 8 Stunden. Neben der Architektur und dem Training des Modells wurde ebenfalls die Implementierung in einer Echtzeitanwendung konzipiert. Dabei war zu berücksichtigen, dass das Modell in eine mobile App integriert werden soll, weshalb besonderer Wert auf eine energieeffiziente Inferenz gelegt wurde. Die Echtzeitanwendung, entwickelt in Python, diente dazu, die Inferenzgeschwindigkeit des Modells zu testen, diese zu optimieren und entsprechende Leistungsmessungen durchzuführen.

6.2.1 Verwendete Architektur des Modells

Die Architektur besteht aus drei Convolutional Layers und einem Long-Short-Term-Memory (LSTM) Layer. Der Vorteil dieser Architektur ist, dass sie sehr energie effizient ist, wie auch sehr schnell in der Inferenz. Mehr dazu im Kapitel 7. Was besonders Markant ist, ist die Verwendung von Convolutional Layers in Kombination mit einem (LSTM) Layer, welcher die zeitliche Komponente der Sprache berücksichtigt. Die Idee der Kombination von Convolutional Layers und (LSTM) Layers ist nicht neu. Es gibt diverse Paper, welche diesen Ansatz verwenden. Beispielsweise verwendet (Khamees et al., 2021) diesen Ansatz für die Klassifizierung von Music Genres. Die finale Architektur des Models ist in der Abbildung 20 dargestellt. Für den Namen des Models wurde der Name *WakeupTriggerConvLSTM2s* gewählt. Der Name beinhaltet information darüber, welche Architektur verwendet wurde. Die Zahl 2s steht für die Anzahl Sekunden, welche das Model analysiert. Die Abbildung wurde mit Figma und Blender erstellt.

Abbildung 20: 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 und wird im anschliessenden Kapitel nochmals genauer beschrieben.

6.2.2 Transformation, Preprocessing, Feature Extraction

Wie in den Grundlagen bereits erläutert, werden Sprachdaten in ein Spektrogramm transformiert. Diese Transformation ist für die Architektur des Modells essentiell. Hierfür wurde ein spezifisches Pytorch-

Modul entwickelt, das für die Umwandlung der Sprachdaten zuständig ist. Eine kurze Einführung zum Konzept von PyTorch-Modulen: Diese erben von der Klasse nn.Module und bilden die Grundelemente in PyTorch. Sie bieten einen bedeutenden Vorteil, besonders bei der Integration in mobile Apps. Nähere Informationen hierzu folgen später. Das PyTorch-Modul, welches für die Transformation der Sprachdaten eingesetzt wird, ist in Abbildung 21 visualisiert.

Abbildung 21: PyTorch Modul für die Transformation der Sprachdaten

6.2.3 Training des Modells

Hier werden die Resultate des Trainings beschrieben. Der Datensatz wurde in Trainingsdaten und Testdaten aufgeteilt. Die Trainingsdaten wurden für das Training des Modells verwendet. Die Testdaten wurden verwendet, um das Modell nach dem Training zu evaluieren, das wird im Kapitel 7 beschrieben. Die Aufteilung der Daten erfolgte mit einem Verhältnis von 80% Trainingsdaten und 20% Testdaten.

Die nachfolgenden Abbildungen zeigen verschiedene Metriken, die während des Trainingsprozesses aufgezeichnet wurden:

Abbildung 22: Genauigkeit pro Batch während des Trainings

Abbildung 23: F1-Score pro Batch während des Trainings

Abbildung 24: Loss pro Batch während des Trainings

Abbildung 25: Genauigkeit pro Epoche

Abbildung 26: F1-Score pro Epoche

Abbildung 27: Loss pro Epoche

Die Grafiken zeigen eine umfassende Übersicht über die Entwicklung der Metriken während des Trainings. Die Modellgenauigkeit und der F1-Score steigen kontinuierlich an, während der Loss kontinuierlich abnimmt. Kurz vor der 40. Epoche ist eine Unterbrechung des Trainings zu erkennen, welche durch einen Fehler im Training Loop verursacht wurde. Es wurden fälschlicherweise bei jedem zweiten Sample 'Silence' hinzugefügt. Nach der Korrektur dieses Fehlers wurde das Training vom Punkt vor der Unterbrechung fortgesetzt, da das Modell bis dahin bereits vielversprechende Ergebnisse lieferte.

6.2.4 PyTorch Scripting

PyTorch Scripting ist eine Technik, die es ermöglicht, PyTorch-Module in ein Format zu konvertieren, das für die Integration in mobile Apps geeignet ist. Beziehungsweise für die Integration in die C++-API von PyTorch. Die C++-API von PyTorch ermöglicht die Verwendung von PyTorch-Modulen in anderen Programmiersprachen wie C++, Java oder Swift. Dies ist ein wichtiger Aspekt dieser Arbeit, da das Modell in eine mobile App integriert werden soll.

6.2.5 Real Time Anwendung in Python

Die Realtime Anwendung wurde in ersten Linie von (Siri-Team, 2017) inspiriert. Dies in Bezug auf den Aufbau der Anwendung. Die Anwendung verarbeitet einen kontinuierlichen Audio Stream, eines Mikrofons. Die Verarbeitung des Audio Streams erfolgt in Stücke von 2 Sekunden. Diese Stücke werden

alle 0.1 Sekunden verarbeitet. Zwei Sekunden sind 32000 Samples bei einer Sample Rate von 16kHz. Somit werden pro Sekunde 10 Inferenzen durch die Pipeline durchgeführt. Bei dieser Variante der Verarbeitung wird, falls das Triggerwort im Stream vorhanden ist, die Erkennung des Triggerwortes mehrfach erkannt. Das bietet den Vorteil einer gewissen Redundanz. Die Rate der Inferenzen könnte auch Variabel an die Systemleistung angepasst werden. Die nachfolgende Abbildung 28 zeigt schematisch die Realtime Anwendung. Dabei wird ein Audio Chunk von 2 Sekunden in ein Spektrogramm transformiert und anschliessend vom Modell verarbeitet. Das Resultat der Inferenz wird anschliessend in einer Progress Bar dargestellt. Die Progress Bar zeigt die Wahrscheinlichkeit an, dass das Triggerwort im Audio Chunk vorhanden ist.

Abbildung 28: Real Time Anwendung

Mit der Ausgabe der Wahrscheinlichkeit für die verarbeiteten Audio Chunks, kann eine Entscheidung getroffen werden, ob das Triggerwort vorhanden ist oder nicht. Dazu wird ein Gleitender Durchschnitt über die Wahrscheinlichkeiten der letzten n Audio Chunks berechnet. Ist der Durchschnitt über einem bestimmten Schwellenwert, wird das Triggerwort als vorhanden erkannt. Das ist auch als Simple Moving Average (SMA) bekannt. Die Formel für den Gleitenden Durchschnitt kann wie folgt beschrieben werden:

$$SMA = \frac{1}{n} \sum_{i=0}^{n} x_i$$

Die nachfolgende Abbildung 29 zeigt die Berechnung des Gleitenden Durchschnitts über die Wahrscheinlichkeiten der letzten 3, 5 und 10 Audio Chunks.

Abbildung 29: Berechnung des Gleitenden Durchschnitts

6.3 Integration in eine mobile App

Im Rahmen dieser Arbeit wurde das entwickelte Modell in eine mobile Anwendung integriert, wobei die Entscheidung auf eine iOS-App fiel. Diese Entscheidung basierte auf der bereits bestehenden Vertrautheit mit der iOS-API, insbesondere **AVAudioEngine**, die im Kapitel 2.2.2 'Audio API für Integration' thematisiert wurde. Die iOS-App wurde in Swift/Objective-C entwickelt, was eine optimale Nutzung der iOS-spezifischen Features und eine reibungslose Integration der PyTorch-Module ermöglichte.

Die grösste Herausforderung bei der Integration war die Anpassung der in PyTorch entwickelten Module für die Verwendung in der iOS-App. PyTorch Scripting spielte hierbei eine entscheidende Rolle, da es die Konvertierung der Module in ein Format ermöglichte, das mit der C++ API von PyTorch kompatibel ist. Diese Kompatibilität war essentiell für die Integration der Modelle in die mobile Anwendung.

Die Integration der C++ API von PyTorch in die iOS-App war ein komplexer und zeitintensiver Prozess. Die Herausforderung lag insbesondere in der Erstellung spezifischer Objective-C Bindings, um die Kommunikation zwischen der in PyTorch entwickelten Logik und der iOS-Anwendung zu ermöglichen. Trotz der Komplexität dieser Aufgabe wurde sie erfolgreich gemeistert, unter anderem durch die Nutzung von Anleitungen und Dokumentationen von PyTorch.

Während des Trainings des Modells wurden für jede Epoche Checkpoints erstellt. Für die endgültige Implementierung in der App wurde der Checkpoint der Epoche 120 ausgewählt. Dieser Schritt war entscheidend, um die bestmögliche Leistung und Genauigkeit des Modells in der App sicherzustellen. Für weiterführende Informationen zur Konvertierung eines PyTorch-Modells in TorchScript und dessen Ausführung in C++ wird auf das Tutorial *Loading a PyTorch Model in C++* („PyTorch Just-In-Time Compiler (JIT) Documentation“, 2023) verwiesen.

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 Storyboard BA-thesis-video

Die erste Skizze auf Abbildung 31 zeigt eine Küchenszene, in der jemand beginnt zu kochen. Sie zeigt ein Tablet als Kochhilfe und die Interaktion mit einem Sprachassistenten. Der erste Teil des Videos zeigt die Hands-free-Nutzung eines Sprachassistenten beim Kochen. Der Rest des Videos präsentiert die Bachelorarbeit 'Integration von Sprachsteuerung in mobilen Apps' als Slideshow. Das Storyboard besteht aus zwei Teilen: Einem Teaser-Clip mit Kochszene und einer anschliessenden Slideshow der BA.

Abbildung 30: Storyboard BA-thesis-video (Entwurf)

Die Skizze wurde in einem zweiten Schritt in ein Storyboard überführt. Das Storyboard ist in Abbildung 31 zu sehen. Es zeigt die einzelnen Szenen des Videos.

Abbildung 31: Storyboard BA-thesis-video

Erst wird der Titel der Bachelorarbeit gezeigt 1, wobei Fade-Transitions für einen übersichtlichen Übergang sorgen. In der ersten Videoszene 2 läuft jemand in die Küche, ohne dass das Gesicht gezeigt wird. Die Ka-mera bewegt sich in die entgegengesetzte Richtung, um die Laufrichtung zu betonen. Die Szene dau-ert maximal 3 Sekunden. Szene 3 zeigt eine Nahaufnahme vom Gemüseschneiden, um das Kochen zu betonen und eine schöne Kulisse zu schaffen. Diese Szene dauert maximal 2-3 Sekunden.In Szene 4 sagt der Protagonist 'Hey FOOBY', während das Tablet im Blickfeld ist und visuelles oder akustisches Feedback gibt. Danach folgt der Befehl 'Nächster Kochschritt'. Die letzte Szene 5 zeigt das Tablet von oben, mit den Kochschritten und der Ausführung des Befehls. Dann folgt die Slideshow, die eine Kurzfassung der Endpräsentation enthält. Diese beschreibt die erarbeitete Bachelorarbeit, insbesondere im Hinblick auf die Lösung.

Abbildungsverzeichnis

1 Teilgebiete der Spracherkennung . . . 6
2 Frames, Channels und Buffers . . . 9
3 AVAudioEngine . . . 11
4 Rechteckfunktion und ihre Fourier-Transformation . . . 12
5 Funktion $f(x)$ mit 5 Samples . . . 13
6 Rekonstruktion des Signals $f(x)$ . . . 14
7 Spektrogramm . . . 15
8 Neuron eines neuronalen Netzes . . . 16
9 Ein neuronales Netzwerk mit drei Layern. . . . 16
10 Convolution . . . 17
11 Recurrent Neural Network . . . 18
12 Timeline of Speech Recognition Developments (2010-2020) . . . 20
13 Voice control implementation. . . . 21
14 DNN Architektur von Siri . . . 23
15 Grundlegende Idee . . . 25
16 Verwendung von wav2vec2 . . . 27
17 Grobplanung . . . 32
18 Aufbau des Datensatzes . . . 33
19 Die überarbeitete Recorder Webseite . . . 34
20 Architektur des Modells . . . 36
21 PyTorch Modul für die Transformation der Sprachdaten . . . 37
22 Genauigkeit pro Batch während des Trainings . . . 37
23 F1-Score pro Batch während des Trainings . . . 37
24 Loss pro Batch während des Trainings . . . 38
25 Genauigkeit pro Epoche . . . 38
26 F1-Score pro Epoche . . . 38
27 Loss pro Epoche . . . 38
28 Real Time Anwendung . . . 39
29 Berechnung des Gleitenden Durchschnitts . . . 39
30 Storyboard BA-thesis-video (Entwurf) . . . 43
31 Storyboard BA-thesis-video . . . 44

Tabellenverzeichnis

1 SMART Ziele der Bachelorarbeit . . . 7
2 Frames in Interleaved und Non-interleaved Buffers . . . 9
3 $f(x)$ und die DFT der 5 Samples . . . 14
4 Confusion Matrix . . . 19
5 Resultate der ersten Versuche mit wav2vec2 . . . 28
6 Hierarchische Auflistung der Arbeitspakete . . . 31
7 Identifizierte Risiken im Projekt . . . 31

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.

Barazanji, O., & Spencer, P. (2023, Oktober). *Hey Ditto Activation Model* [Zugriffsdatum: 10. Oktober 2023]. url%7Bhttps://github.com/omarzanji/ditto%5C%5Factivation%7D

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/

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.

Hugging Face. (2023). *Fine-tune a pretrained model* [Zugriff am 21. November 2023]. Hugging Face. Verfügbar 21. November 2023 unter https://huggingface.co/docs/transformers/training

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.

Lim, H., Kim, Y., Yeom, K., Seo, E., Lee, H., Choi, S. J., & Lee, H. (2023). Lightweight feature encoder for wake-up word detection based on self-supervised speech representation [https://arxiv.org/abs/2303.07592 [Zugriff 15. Oktober 2023]].

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

Papakyriakopoulos, O., Choi, A. S. G., Thong, W., Zhao, D., Andrews, J., Bourke, R., Xiang, A., & Koenecke, A. (2023). Augmented Datasheets for Speech Datasets and Ethical Decision-Making. *2023 ACM Conference on Fairness, Accountability, and Transparency*. https://doi.org/10.1145/3593013.3594049

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

PyTorch Just-In-Time Compiler (JIT) Documentation [Zugriff am 20. November 2023]. (2023). *PyTorch*. https://pytorch.org/docs/stable/jit.html

Salamon, J., & Bello, J. P. (2017). Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification. *IEEE Signal Processing Letters*, *24*(3), 279–283. https://doi.org/10.1109/lsp.2017.2657381

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.

Valantis, K. (2023, Januar). Battle of The Giants: TensorFlow vs PyTorch 2023 [https://medium.com/@valkont/battle-of-the-giants-tensorflow-vs-pytorch-2023-fd8274210a38 [Zugriff 5. November 2023]].

von Platen, P. (2021, November). Fine-tuning XLS-R for Multi-Lingual ASR with Transformers [Zugriff am 22. November 2023].

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