FH Vorarlberg
University of Applied Sciences

Offline Speech To Text Engine For Delimited Context

In Combination With An Offline Speech Assistant

Master Thesis for obtaining the academic degree
Master of Science in Engineering

Vorarlberg University of Applied Sciences
Computer Science MSc

Supervised by:
Dipl.-Ing. (FH) Walter Ritter

Submitted by:
BSc. Pia-Maria Weiß

Dornbirn, Thursday 7th July, 2022

Statuary Declaration

Statuary Declaration I declare that I have developed and written the enclosed work completely by myself and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. This Master Thesis was not used in the same or in a similar version to achieve an academic degree, nor has it been published elsewhere.

Abstract

The inatura[1] museum in Dornbirn had planned an interactive speech assistant-like exhibit. The concept was that visitors could ask the exhibit several questions that they would like to ask a flower. Solution requirements regarding the functionalities were formulated, such as the capacity to run offline because of privacy reasons. Due to the similarity of the exhibit, open-source offline Speech To Text (STT) engines and speech assistants were examined. Proprietary cloud-based STT engines associated with the corresponding speech assistants were also researched. The aim behind this was to evaluate the hypothesis of whether an open-source offline STT engine can compete with a proprietary cloud-based STT engine. Additionally, a suitable STT engine or speech assistant would need to be evaluated. Furthermore, analysis regarding the adaption possibilities of the STT models took place. After the technical analysis, the decision in favour of the STT engines called "Vosk" was made. This analysis was followed by attempts to adapt the model of Vosk. Vosk was compared to proprietary cloud-based Google Cloud Speech to Text to evaluate the hypothesis. The comparison resulted in not much of a significant difference between Vosk and Google Cloud Speech to Text. Due to this result, a recommendation to use Vosk for the exhibit was given. Due to the lack of intent parsing functionality, two algorithms called "text matching algorithm" and "text and keyword matching algorithm" were implemented and tested. This test proved that the text and keyword matching algorithm performed better, with an average success rate of 83.93 %. Consequently, this algorithm was recommended for the intent parsing of the exhibit. In the end, potential adaption possibilities for the algorithms were given, such as using a different string matching library. Some improvements regarding the exhibit were also presented.

[1] https://www.inatura.at/

Kurzreferat

Das inatura Museum in Dornbirn hatte ein interaktives sprachassistentenähnliches Exponat geplant. Das Konzept sah vor, dass die Benutzenden dem Exponat verschiedene Fragen stellen können, die sie auch einer Blume stellen würden. Es wurden Lösungsanforderungen hinsichtlich der Funktionalitäten formuliert, wie z.B. die Fähigkeit, aus Datenschutzgründen offline zu laufen. Aufgrund der Ähnlichkeit des Exponats wurden Open-Source-Offline-STT-Engines und Sprachassistenten untersucht. Proprietäre Cloud-basierte STT-Engines in Verbindung mit den entsprechenden Sprachassistenten wurden ebenfalls untersucht. Ziel war es, die Hypothese zu evaluieren, ob eine Open-Source-Offline-STT-Engine mit einer proprietären Cloud-basierten STT-Engine konkurrieren kann. Zusätzlich sollte eine geeignete STT-Engine oder ein Sprachassistent evaluiert werden. Darüber hinaus wurde eine Analyse der Anpassungsmöglichkeiten der STT-Modelle durchgeführt. Nach der technischen Analyse fiel die Entscheidung zugunsten der STT-Engine namens "Vosk". Auf diese Analyse folgten Versuche, das Modell von Vosk anzupassen. Vosk wurde mit der proprietären Cloud-basierten Google Cloud Speech to Text verglichen, um die Hypothese zu bewerten. Der Vergleich ergab, dass es keinen signifikanten Unterschied zwischen Vosk und Google Cloud Speech to Text gibt. Aufgrund dieses Ergebnisses wurde empfohlen, Vosk für das Exponat zu verwenden. Aufgrund der fehlenden Intent-Parsing-Funktionalität wurden zwei Algorithmen namens "Text-Matching-Algorithmus" und "Text-and-Keyword-Matching-Algorithmus" implementiert und getestet. Dieser Test ergab, dass der Text-and-Keyword-Matching-Algorithmus mit einer durchschnittlichen Erfolgsquote von 83,93 % besser abschnitt. Folglich wurde dieser Algorithmus für das Intent-Parsing des Exponats empfohlen. Abschließend wurden potenzielle Anpassungsmöglichkeiten für die Algorithmen genannt, wie z.B. die Verwendung einer anderen String-Matching-Bibliothek. Es wurden auch einige Verbesserungen bezüglich des Exponats vorgestellt.

Contents

List of Acronyms

ADC Analogue to Digital Converter

APL Alexa Presentation Language

API Application Programming Interface

ARPA Advanced Research Projects Agency

ASG Auto Segmentation Criterion

ASK Alexa Skills Kit

ASR Automatic Speech Recognition

AVS Alexa Voice Service

AVX Advanced Vector Extension

AWS Amazon Web Services

BLSTM Bidirectional Long Short Term Memory

BSD Berkeley Software Distribution

CLI Command Line Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

CTC Connectionist Temporal Classification

CUDA Compute Unified Device Architecture

CuDNN Nvidia CUDA Deep Neural Network

DNN Deep Neural Network

FAIR Facebook AI Research

FMA FlexCast Management Architecture

FST Finite State Transducer

GMM Gaussian Mixture Modelling

GPU Graphical Processing Unit

GRPC Google Remote Procedure Call

HITL Human-in-the-loop

HMM Hidden Markov Model

IDE integrated development environment

JSON JavaScript Object Notation

KenLM Kenneth Heafield Language Model

LACE Layer-wise Context Expansion

LAS Listen Attend Spell

LM Language Model

LSTM Long Short Term Memory

MFCC Mel Frequency Cepstral Coefficients

MQTT Message Queuing Telemetry Transport

MS Microsoft

NLP Natural Language Processing

NPM Node Package Manager

ReLU Rectified Linear Unit

ResNet Residual Network

REST API Representational State Transfer Application Programming Interface

RNN Recurrent Neural Network

RNNLM Recurrent Neural Network Language Model

RNN-T Recurrent Neural Network Transducer

SDK Software Development Kit

SGMM Sparsified Gaussian Mixture Model

SSH Secure Shell

SRILM Stanford Research Institute Language Modeling Toolkit

STFT Short Time Fourier Transformation

STT Speech To Text

TCP Transmission Control Protocol

TDNN Time Delay Neural Network

TDNN-F Factorized Time Delay Neural Network

TDS Time Depth Separable

TTS Text To Speech

UI User Interface

WER Word Error Rate

WFST Weighted Finite State Transducer

WSL Windows Subsystem for Linux

WWDC World Wide Developers Conference

1 Introduction

The museum inatura - Erlebnis Naturschau in Dornbirn[1] had planned an interactive exhibit for children called "Sag's durch die Blume". The main idea of this exhibit was that children are able to ask several questions that they would like to ask a flower, via a button. These questions should be matched with predefined answers. The recording of a question will be processed with an speech to text (STT) engine that outputs a transcript. The resulting text is then matched to an answer that triggers the playback of a corresponding video. Since it is not possible to obtain consent from every user, in order to transmit the audio recording to a server or third-party provider, an offline solution was selected. Focusing on the case study from this interactive exhibit at inatura, the possibilities and limits of open-source offline speech recognition for triggering actions have been examined and compared with cloud-based variants.

1.1 Conceptual Formulation

As previously mentioned, the idea of the exhibit was to mimic a flower that could be asked questions by the visitors. Beforehand, a button triggers the recording of the question. To interpret the recording, an STT engine would be used for transforming the audio into text. This text results in the action of playing the answer video, as can be extracted from figure 1.1 below.

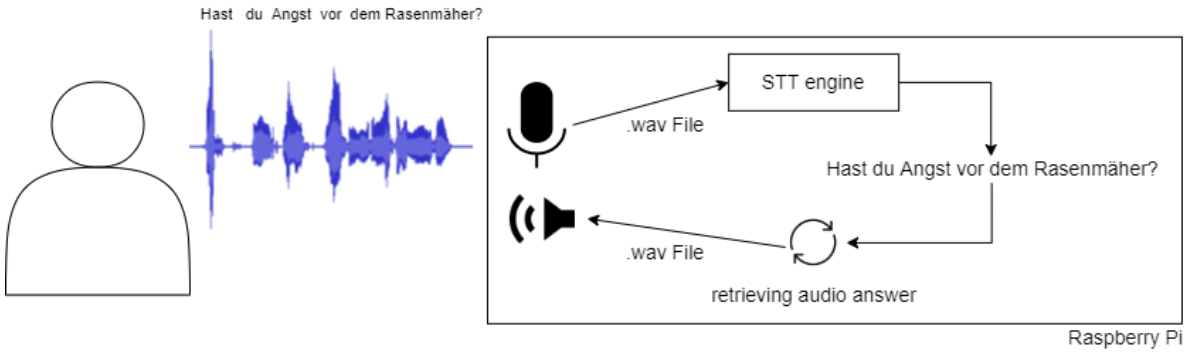

Figure 1.1: Conceptual structure of the core components of the exhibit.

Source: Prepared by the author

[1] https://www.inatura.at/

The underlying technologies of the exhibit concept and those of a speech assistant are equal. The two main components of a speech assistant are the wake word detector, the STT engine and the speech assistant itself. The task of the wake word detector is to register if the wake word has been said and to start the recording of the speech. Afterwards, the STT engine undertakes the job of converting the speech into text. At this point, either a cloud-based or an offline STT engine can be used depending on the used system. If a cloud-based STT engine is used, the recording will be transmitted to the cloud, whereas an offline STT engine processes this locally. From this point, only the text will be processed by the speech assistant. The text will be used to find and perform the action or task, which was given by the user. (Amazon.com Inc. 2010a)

The aim is to develop an application comparable to a speech assistant. Therefore, a suitable German STT engine must be researched. Furthermore, investigations regarding speech assistants and their functionality will take place. Due to adaptability and privacy reasons, open-source offline technologies are considered. The aim is to evaluate whether the open-source offline STT engine can compete with a proprietary cloud-based STT in a delimited context.

1.2 Definition of Objectives

The main objective was to examine if an offline STT engine and speech assistant can be a reliable alternative in comparison to a cloud-based solution. The plan was to test this with the previously mentioned exhibit functionality. The main reason for research was to to examine whether an offline open-source STT can keep up with the quality of a proprietary cloud-based STT engine. Furthermore, it should be researched if an offline open-source STT can be fine-tuned for the purpose of this exhibit regarding the recognized vocabulary. In addition, it will be examined whether an offline speech assistant with intent parsing or an individually implemented solution is better suited for the intended project. Subsequently, the software that has proven to be more reliable would then be recommended for the exhibit.

Another objective was that it should provide the ability to understand a spoken question with an appropriate accuracy, so that the question can be matched and therefore answered. Since the inatura[2] is located in Dornbirn, Vorarlberg, it was be an advantage if the solution was able to handle at least some input in dialect speech.
The target audience for the exhibit refers to children aged between 9-15 years. Because of this target audience, it was be desirable for the system to have a low sensitivity to syntax. This property allows the input of several sentence combinations and still be able to answer them.

The performance of an STT engine depends on the models used. The larger the model's vocabulary, the more words the engine can transcribe. Not included in this work is the development of an entirely new model. But changing the vocabulary will be covered in a separate chapter. For this reason, basic knowledge of machine learning is a benefit.

[2] https://www.inatura.at/

1.3 Solution Requirements

It was preferred that the system would be independent of internet usage, due to privacy concerns. Due to further cost concerns, open-source technology was the main focus of interest. Proprietary software, such as Amazon's "Alexa", transmits the audio records to an external server (Amazon.com Inc. 2010b). With an exhibition in a museum, privacy is an important point. For this reason, the audio recording is activated via push-to-talk and the speech processing is processed locally. This makes it a reasonable solution in terms of data protection. Regarding the language, it was important that it is able to understand standard German, with different syntax. A question asked in a German dialect that is not too complicated should also be understood, if possible. The solution should recognise the vast majority of asked questions, which are known by the software. In the case of a misunderstood question, there should also be a strategy to handle this situation. An asked question will be matched with the corresponding answer, which will then be given. In the example of the exhibit, a screen with videos of a flower should create an illusion that the flower has answered the question.

Here is a summary of the functional demands:

- The recorded speech should be transcribed with an open-source offline STT engine.
- The transcript of the question should be matched with the correct audio/video recording.
- It should be possible to easily expand the questions- and answer-pools.
- The software should operate on a Raspberry Pi.

Here is a summary of the non-functional demands:

- The transcription and matching should be carried out within a reasonable time, in comparison to a cloud-based solution.
- The system should be able to transcribe the speech, as well as a cloud-based solution.
- A strategy for not answerable questions should be applied.
- The application should run entirely offline.
- The scope includes neither the implementation of the exhibit, nor the implementation of the voice recording.
- The scope excludes the identification of potential questions of the visitors.

2 Related Work

It might be reasonably assumed that progress in the area of speech assistants and STT engines will further improve. To examine whether an offline STT engine and speech assistant can be a reliable alternative to a cloud-based solution, related work will be compared. The comparison will take place between proprietary and open-source STT engines. The proprietary STT engines will be the ones embedded in the most popular speech assistants like in Amazon Alexa (Adobe 2019). The comparison of open-source STT engines focuses on some of the most popular ones (Foster 2021) and (S. James 2020). Furthermore, some of the most popular proprietary and open-source speech assistants will be researched (Adobe 2019) and (yourtechdietAdmin 2021). The technologies will be compared regarding adaptability, costs, software and hardware requirements.

2.1 Speech to Text Engines

The STT engine is the essential component of the exhibit. It is needed to transcribe the spoken language into text. To work with speech it is necessary to convert the spoken words into a computer-understandable format. This is carried out via an Analogue to Digital Converter (ADC). The ADC samples the waves of the spoken words in regular intervals, which can be pictured in a histogram. The signal gets split into smaller sections and gets assigned to the known phonemes (Grabianowski 2006). Phonemes are the tiniest components of a language. In the case of German, there are about 40 phonemes (Tošović 2022). The following section will begin with an overview of the STT engines. Further subjects of comparison will be the documentation, the existing models of the engines and the possibility of creating or extending a model. Additionally, technological requirements for usage and training and a brief insight into how the engine works will be given.

2.1.1 Open-Source Software

The preference for using open-source engines is due to licensing, as the use of open-source STT engines is usually unrestricted. Additionally, the cost factors are in favour of open-source STT engines. For instance, the "Google Cloud Speech-To-Text" could be used, but it has added costs (Google Cloud 2022). "Mozilla Deepspeech", "Wav2letter/Flashlight", "Kaldi", "Coqui", "CMUSphinx" and "Vosk" are evaluated as representatives of the open-source STT engines (S. James 2020) and (Foster 2021).

2.1.1.1 Mozilla Deepspeech

As can be seen from Mozilla Deepspeech's Github repository, the STT engine project was launched in 2016. The vision of Mozilla Deepspeech is shaped by the goal of decreasing hardware effort and being able to operate on any Linux, Windows, Android or macOS machine (DeepSpeech 2020e). In August 2020, a reorganization at Mozilla was announced. At this point, the future of the Deepspeech project was unknown (Morais 2020). Later, in April 2021, further procedures were published and a recommendation for "Coqui" was given (Mlopatka 2021). They had planned to publish a how-to-use handbook and a clean version of the documentation. Moreover, Mozilla planned to stop the maintenance of the code. About 46.9% of the code is written in C++, therefore knowledge of this programming language is a perk. The current version of Deepspeech is 0.9.3. (DeepSpeech 2020e)

The current documentation of version r0.9 of Mozilla Deepspeech contains a detailed guide for installation, in combination with a pre-trained model. The installation can take place either via pip3 or Node Package Manager (NPM) where the Graphical Processing Unit (GPU) flag for Compute Unified Device Architecture (CUDA) can be provided. A Dockerfile is also provided to build it from scratch. Some preinstalled libraries are required (e.g. the deepspeech package for TensorFlow and deepspeech-tflite for TensorFlow Lite). Supported platforms are Windows, Linux, macOS and Android. In the case of Windows and Linux the GPU is supported. It is possible to use Deepspeech with programming languages like C, .NET, JavaScript, Java and Python. Code examples are given for demonstration purposes. Further examples are contributed by users in the documentation (DeepSpeech 2020d). A brief excursion regarding domain-specific vocabulary is provided. As well as optimization for parallelism and error codes are subjects of discussion (DeepSpeech 2020e).

There is already an existing German model offered on a GitHub repository that can be downloaded from a Google Drive. It provides a Word Error Rate (WER) of 21.5% (A. Agarwal 2022a). The documentation commits a chapter on the training of a new and the modification of an existing model. Other topics of the documentation are fine-tuning and transfer learning. For some languages, speech data is provided by the project Common Voice to train a new model. The project aims to advance the technology and makes datasets available to everyone. (Mozilla 2019) In the case of a Common Voice dataset, the documentation provides a script for the data preparation (DeepSpeech 2020c). To create a language model, 4 GB of audio samples is recommended (DeepSpeech 2020b). For creating or modifying an existing model, a Unix-based environment like Linux or iOS and Python 3.6 is required. A graphics card can be used to improve the learning speed of the model, otherwise a Central Processing Unit (CPU) is used. (DeepSpeech 2020c)

The engine's heart consists of a Recurrent Neural Network (RNN) with 55 hidden layer units. All the layers are recurrent, except for the first three which can be seen in figure 2.1. The first layer uses Mel Frequency Cepstral Coefficients (MFCC) frames for feature extraction. The second and third layers work with separate data. The third layer feeds the data into an Long Short Term Memory (LSTM). For the activation function, a Rectified Linear Unit (ReLU) was

utilized. The first recurrent layer uses an array of forwarding recurrent hidden units, which are displayed as the nodes above the LSTM pictured in figure 2.1. For the output layer, the character probabilities were used for predicting the output. The figure below represents the core structure of Deepspeech. (DeepSpeech 2020a)

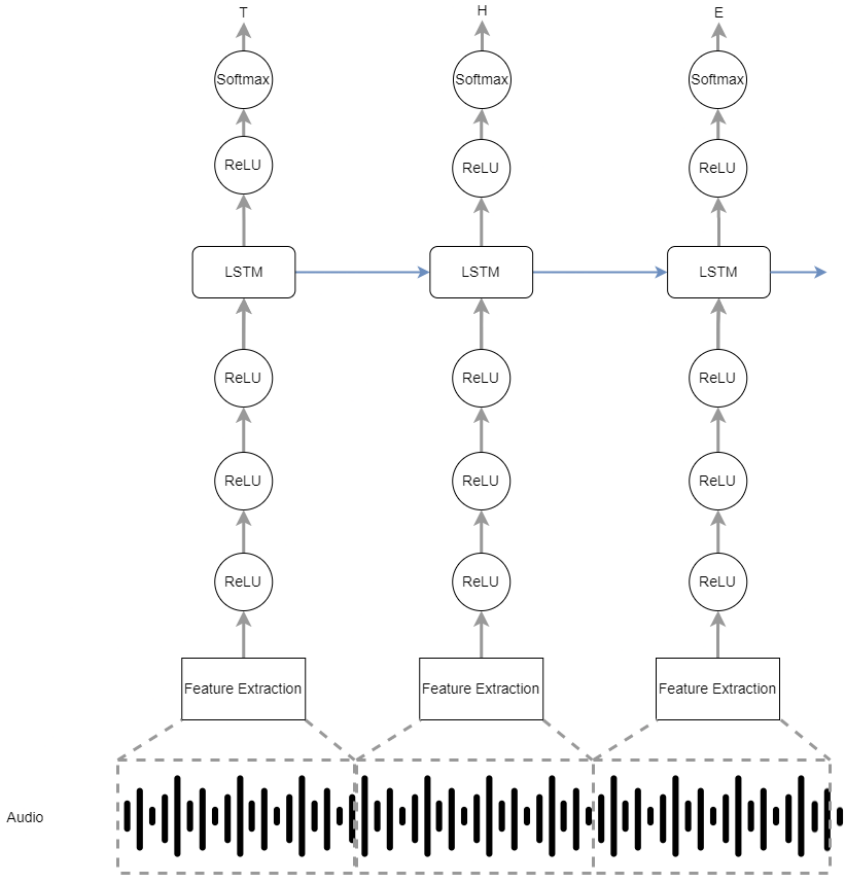

Figure 2.1: Structure of the Mozilla Deepspeech Model.

Source: Based on: `https://deepspeech.readthedocs.io/en/r0.9/DeepSpeech.html`, adapted by the author

Mozilla Deepspeech is licensed with Mozilla Public License 2.0.

The GitHub Repository for Deepspeech can be found at: `https://github.com/mozilla/DeepSpeech`

2.1.1.2 Flashlight

In April 2021, Meta AI presented "Flashlight" to create an open-source library for machine learning. This library should provide an opportunity to be adaptable for own needs (Meta AI 2021). The development of Flashlight and Wav2letter started in 2018, as the insights from GitHub suggest (wav2letter GitHub Repository 2022). Wav2letter focused on speech recognition and was merged into Flashlight. Further development of Wav2letter will take place in the Flashlight repository which represents itself as a flexible independent machine learning library. The current version of Flashlight is v0.3.1. (wav2letter GitHub Repository 2022)

The guides for installation and training are distributed on the GitHub repository of Flashlight and Wav2letter. To install Flashlight some system requirements need to be met. The machine needs to be a Linux-based system with a version of CMake and make is necessary. Since Flashlight and Wav2letter were implemented in C++, a corresponding compiler is mandatory. For the installation pip or the package manager vcpkg can be used. As listed in the installation guide on GitHub, CPU, as well as CUDA GPU, is supported. In addition, a brief manual on how to use it with Docker is given. Docker does support CUDA backends too. The documentation contains example usage of Flashlight in combination with C++. Likewise, there are also connections to use Flashlight with Python with a comprehensive step-by-step guide. (Flashlight GitHub Repository 2022a)

There are pre-trained models for eight languages including German (Flashlight GitHub Repository 2022c). For the training audio with transcriptions, there are predictions on the distributions of the words and a listing of the words in different sequences (Flashlight GitHub Repository 2022b).

Flashlight consists exclusively of convolutional layers. Thus, it is the first fully convolutional neural network speech recognition engine provided by theFacebook AI Research (FAIR) Team (Meta 2018). The structure consists of three parts which can be referred to as front-end, acoustic model and language model, combined with a beam search. Those three components are displayed in figure 2.2. The first part receives the input sound waves. The soundwaves are displayed above the first components of the front-end in figure 2.2. After feature extraction, mel-filterbank, low-pass and other filters are applied in the first processing section. The front-end is followed by the acoustic model. The acoustic model uses a Time Depth Separable (TDS) which causes a reduction of the parameters and therefore improves performance. The task of this part is to forecast the characters with Auto Segmentation Criterion (ASG), which is comparable to the Connectionist Temporal Classification (CTC). The last section of the structure is supplied with the output of the second section. It is composed of 14 1-D convolutional residual blocks. The applied gated linear units are used as activation functions. As a result, a transcript of possible contenders is the output of the acoustic model. The beam search's role is to produce word concatenations with the data received from the acoustic section. (Zeghidour et al. 2019) The figure below shows the sections of the Convolutional Neural Network (CNN) and their functionality:

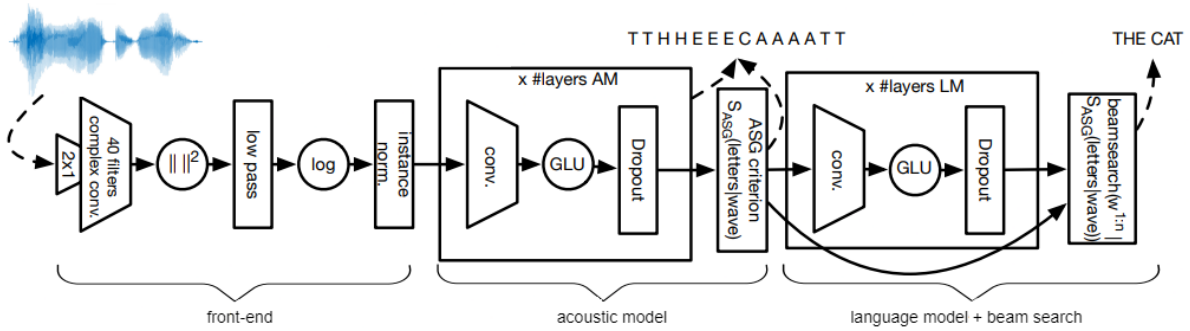

Figure 2.2: Components of the Flashlight STT engine.

Source: Based on: `https://arxiv.org/pdf/1812.06864.pdfl`, adapted by the author

Flashlight is under a MIT license.

The obsolete GitHub Repository for Flashlight can be found at: `https://github.com/flashlight/wav2letter`
The GitHub Repository for Flashlight can be found at: `https://github.com/flashlight/flashlight`

2.1.1.3 Kaldi

Kaldi was created in 2009 during a seminar at the John Hopkins University[1]. In 2011, the complete code was published. The code is still maintained by some of the creators. (Povey 2011a) There is a lot of code written in C++, although the majority are shell scripts. The current version of Kaldi is 5.0.0. (Povey 2022)

Kaldi is characterised by fine granular documentation. It covers topics such as the glossary of terms, deep neural networks and parallelism in combination with Kaldi (Povey 2011b). For the installation of Kaldi two different guides are offered. Fundamental knowledge for working with Kaldi is Git and Hidden Markov Model (HMM) - Gaussian Mixture Modelling (GMM). The requirement for installation is the wget package for downloading resources. On UNIX-based operating systems, which is also a prerequisite, wget[2] is mostly already installed. Additionally, experience with C++ or shell scripting is an advantage. (Povey 2011d) Furthermore, two Docker Images are provided in the GitHub Repository: one with GPU and the other with CPU support. Kaldi can be linked with Android by a Docker Image. Likewise, Kaldi can be used with a Python wrapper or can be cross-compiled for Web Assembly. (Povey 2022) The process of installation is separated into two sections based on the structure of the repository. In each sub-directory there is an installation guide. (Povey 2011c)

[1] https://www.cmu.edu/
[2] https://www.gnu.org/software/wget/

The chapter Kaldi Tutorial covers the topic of model creation. There is a pre-trained German model with a WER of 11.85 %. The model was created by the Technische Universität Darmstadt. After modifying a model, some retaining will be necessary to improve the accuracy. More details about the data sets, models and the used hardware can be found in the paper of Milde and Köhn (2018).

Kaldi is designed to be adaptable. This is noticeable in the documentation and description of the creator's paper. The construction of Kaldi consists of the feature extraction, the acoustic model, the phonetic decision tree and the language model. Not only the feature extraction offers different options to choose from, but the other parts are also flexible. As the default option for feature extraction, MFCC is used. The acoustic model is designed to work with conventional and Sparsified Gaussian Mixture Model (SGMM) models. For future progress, Kaldi can be altered to fit recent models. Afterwards, a phonetic decision tree is created using the HMM-State. The HMM-State connects the phonemes. The current phoneme is considered with the previous and the following phoneme. Thus, every phoneme has a decision tree. The language model is described as an Finite State Transducer (FST). As to be expected from Kaldi, there are tools for the customisation of the language model. For creating a decoding graph, Weighted Finite State Transducer (WFST) is used. Currently, the transitions between the phoneme arrays are handled. The figure 2.3 displays the single adjustable parts and further dependencies of Kaldi.(Povey et al. 2011)

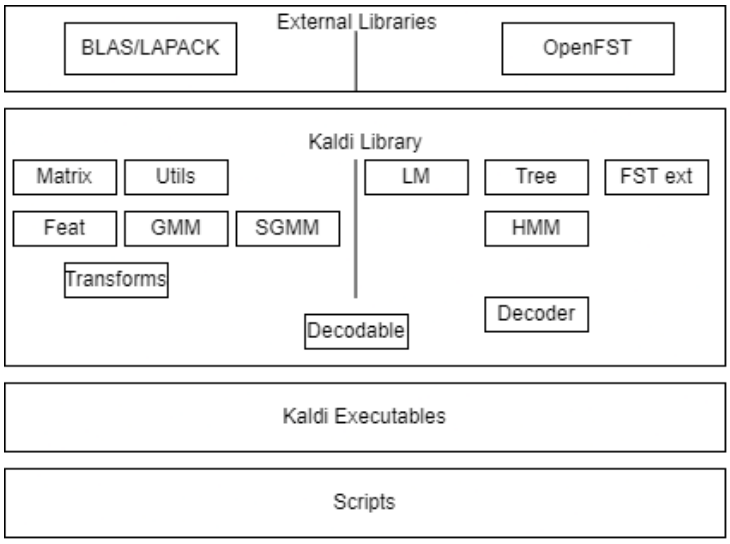

Figure 2.3: Individually adjustable parts of Kaldi.

Source: Based on:
`http://publications.idiap.ch/downloads/papers/2012/Povey_ASRU2011_2011.pdf`,
adapted by the author

Kaldi is under an Apache License 2.0.

The GitHub Repository for Kaldi can be found at: `https://github.com/kaldi-asr/kaldi`

2.1.1.4 Coqui STT

Coqui was founded in 2016 at Mozilla, due to a missing open-source technology for speech at that time. Consequently, an STT and Text To Speech (TTS) engine were developed. Next to the mentioned engines, a project for collecting voice data was launched. In the future, Coqui plans to continue support for the STT, TTS and additional current projects. (Coqui GmbH 2021a) The main part of this technology is written in C++. The current version is 1.3.0. (Coqui GitHub Repository 2022).

The documentation of Coqui includes introductory explanations for the basics. At the beginning, there is a guide on how to install Coqui with Python with the associated package manager. For demonstration purposes an English model was used. The models can be exchanged. Other ways to install Coqui are via Node.js, the Android AAR libstt package, which offers a C interface. In addition, it is also possible to use Coqui with Docker. Depending on the platform, some libraries are additionally necessary to install. Coqui is applicable under some Linux distributions, macOS, Android and Windows. As a hardware requirement, a CPU with Advanced Vector Extension (AVX) or FlexCast Management Architecture (FMA) architecture is supported. Application Programming Interface (API)s and code examples for Python, JavaScript, C, .NET and Java are provided. (Coqui GmbH 2021b)

Currently, 84 models are available including two German models. Some models were revised and released under a different version (Coqui GmbH 2022). The German model of Coqui and Mozilla Deepspeech are the same therefore, the WER is the same (A. Agarwal 2022b). How the models can be trained is explained in the documentation. For the training section, CUDA and Nvidia CUDA Deep Neural Network (CuDNN) are essential. The platforms supporting the training are restricted to Linux environments and Mac. The scripts for the training are written in Python. For this reason, a Python version of at least 3.6 is necessary. For training either a Dockerfile, Virtual Environment or a manual setup can be used. To train with new data audio recordings and the corresponding transcripts are required. The recordings should dispose of a sample rate of 16 kHz in a mono-channel Wav format with a length of at least five seconds. (Coqui GmbH 2021c)

In more detail, Coqui uses an MFCC for feature extraction. After the feature extraction, the data is fed into three fully connected non-recurrent layers. These are displayed as nodes h1, h2 and h3 in the figure 2.4 below. The next component is a bidirectional RNN layer, which uses LSTMs nodes and tanh as activation functions. The last component of the acoustic model is a fully hidden connected layer with ReLU activations. This is the h5 node in the figure 2.4. The output of the acoustic model is the probabilities of each letter. Afterwards, a CTC loss function predicts the arrangement of the letters. The language model called scorer recognises the probability of the words combined with their sequence. The Kenneth Heafield Language Model (KenLM) is in usage, but it is exchangeable. Coqui provides the functionality of changing the output mechanism. Some mechanisms are the alphabet base one, which is the default, and the bytes output. The last-mentioned output mode is experimental. The figure 2.4 below shows the structure of the acoustic model of Coqui and how the interaction takes place. (Coqui 2022)

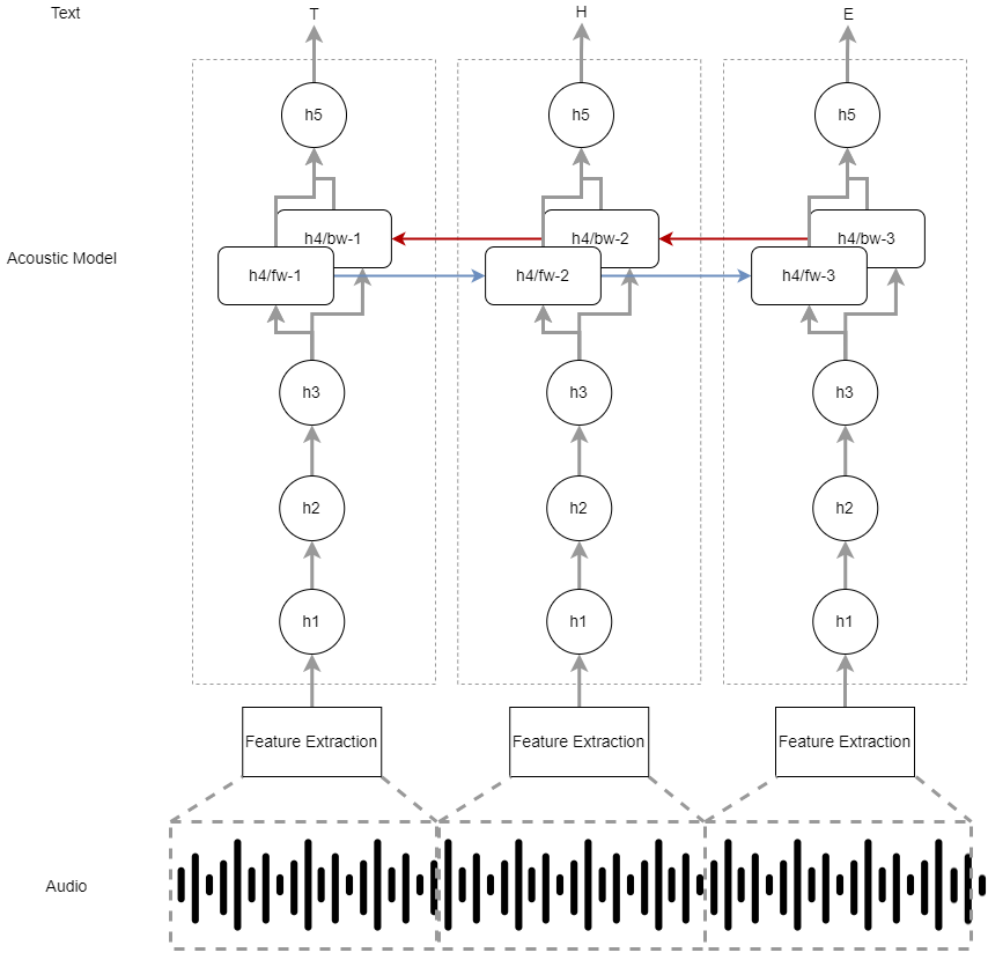

Figure 2.4: Structure and data flow of Coqui's acoustic model.

Source: Based on: `https://coqui.ai/blog/stt/a-journey-to-10-word-error-rate`, adapted by the author

Coqui is under a Mozilla Public License Version 2.0.

The GitHub Repository for Coqui STT can be found at: `https://github.com/coqui-ai/STT`

2.1.1.5 CMUSphinx

The first activities on the CMUSphinx Pocketsphinx's GitHub repository were in 2013. However, that was not exactly the start. CMUSphinx has gathered over 20 years of knowledge at Carnegie Mellon University[3]. This knowledge targets limited systems. In the same way, CMUSphinx provides various tools for speech recognition. (Shmyrev 2019a) On the website, CMUSphinx references their new library called "Vosk". This library offers seven languages (Shmyrev 2019f). Vosk is introduced in more detail later.

The documentation contains introductory information about CMUSphinx. The basic knowledge of the recognition procedure and its required components is discussed. The components of the CMUSphinx model are the acoustic, language model and phonetic dictionary. The acoustic model is used for the probabilities of the words (Shmyrev 2019i). In contrast to the acoustic model, the language model is used for the knowledge of potential word concatenations (Shmyrev 2019c). The phonetic dictionary contains the phonetic transcription of the known words (Shmyrev 2019e).
CMUSphinx consists of four different segments called Pocketsphinx, Sphinxbase, Sphinx4 and Sphinxtrain. The task of Pocketsphinx is to recognise speech and is written in the programming language C. Pocketsphinx depends on the library Sphinxtrain and Sphinxbase. The last segment is comparable to the one of Pocketsphinx except that it is written in Java and is called Sphinx4. (Shmyrev 2019h) For the usage of Sphinx4, tools such as Gradle or Apache Maven are recommended. All these segments are included in the latest release - 5prealpha. There are practical examples for integrations of sphinx4 and Pocketsphinx in applications and on how to use Pocketspinx with Android. Pocketsphinx is recommended for applications with low latency. For adaptability, Sphinx4 is suggested (Shmyrev 2019b). For the usage of Pocketsphinx, a UNIX-based or Windows system is a requirement. Additional installation dependencies are Python, gcc[4] and autoconf[5], for example. The guide contains helpful code with explanations. For the usage of Android devices, the dependencies differ. (Shmyrev 2019d).

There is a German model, amongst 15 other languages, and acoustic models are offered by CMUSphinx. To enhance the phonetic dictionary, a phonetic transcription of the words must be created with the library g2p-seq2seq[6] (Shmyrev 2019e). Alternatively, an existing dictionary can be used. The language model requires prepared text and training with an Advanced Research Projects Agency (ARPA) model and the toolkit Stanford Research Institute Language Modeling Toolkit (SRILM) (Shmyrev 2019c). The expansion of the acoustic model presumes a listing of the sentences, the phonetic dictionary and audio recordings of the new words. (Shmyrev 2018)

[3] https://www.cmu.edu/
[4] https://gcc.gnu.org/
[5] https://www.gnu.org/software/autoconf/
[6] https://github.com/cmusphinx/g2p-seq2seq

Sphinx4 consists of three main components stated as FrontEnd, Decoder and Linguist. The FrontEnd component employs MFCC as feature extraction by default. The feature extraction component is exchangeable. The task of the Linguist is to combine the information from the phonetic dictionary and acoustic model into a search graph. HMM is used for the implementation of the acoustic model which is arranged in the Linguist. To change the language, a phonetic dictionary, an acoustic and a language model of the desired language are necessary. The Decoder consults the search graph and the extracted features for the decoding process. Viterbi search[7] in the Decoder enables it to move forward or backwards in the features. This is comparable to a depth-first search. In figure 2.5 below the interaction between the individual components is visualised. (Walker et al. 2004)

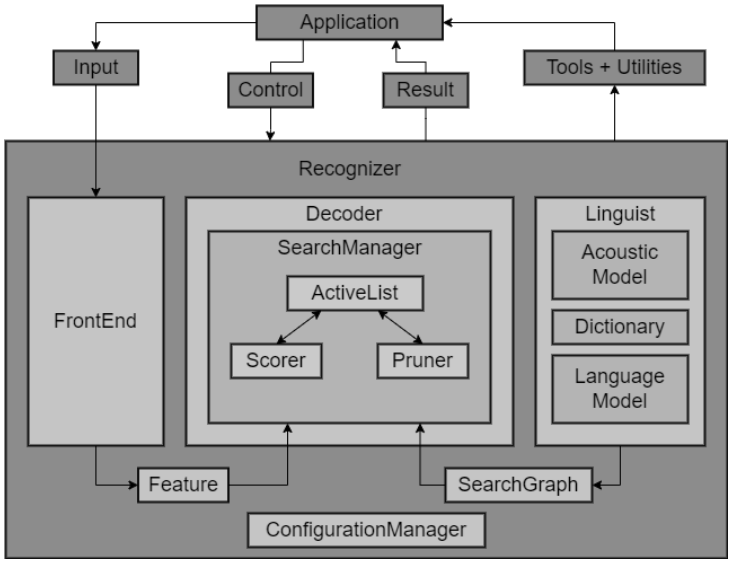

Figure 2.5: Interaction of the different components of Sphinx4.

Source: Based on: `https://www.researchgate.net/publication/228770826_Sphinx-4_A_flexible_open_source_framework_for_speech_recognition`, adapted by the author

A license comparable to Berkeley Software Distribution (BSD) is used (Shmyrev 2019a). Further details about the license are not mentioned.

The GitHub Repository for CMUSphinx can be found at: `https://github.com/cmusphinx`

2.1.1.6 Vosk

This STT engine is the new library of CMUSphinx and it is called Vosk (Shmyrev 2019f). It is an open-source offline STT engine with further functionalities. According to the GitHub repository, Vosk has been and is continually under development since 2019. The current version of the API is 0.3.32. It can be used for speech recognition, as an STT engine for smart home devices and as virtual assistance. (Alphacep GitHub Repository 2022)

[7] https://en.wikipedia.org/wiki/Viterbi_algorithm

The website promotes the positive aspects of using Vosk, which are the easy installation and use (Alpha Cephei Inc. 2016d). Moreover, connections for C, C#, Go, Java, Node.js and Python are given (Alphacep GitHub Repository 2022). The support of many languages is in favour of Vosk. It is possible to customize the vocabulary fast. Another benefit of Vosk is speaker identification. (Alpha Cephei Inc. 2016d) Vosk is supported on Linux, Raspbian, macOS, iOS, Android and Windows. For installation, the Python version 3.5-3.9 and a corresponding pip version of 20.3 or later are required. Additionally, a video with the installation and usage with Python is provided. Vosk can be used with a WebSocket server and Google Remote Procedure Call (GRPC) Server. (Alpha Cephei Inc. 2016a) The documentation gives a clear overview of the interesting topics. Currently, the documentation has been revised and improved a few times since the beginning of this thesis.

Vosk supports more than 20 languages, including German. For lightweight devices such as Android, iOS and Raspberry Pi different models are provided. The small model was specially made for Android or Raspberry Pis. The size of the large model-0.21 is about 1.9 GB, whereas the small one has the size of approximately 45 MB. (Alpha Cephei Inc. 2016c) The large model takes about 1 minute and 20 seconds for the initial loading. In contrast to the big model, the small one takes less than two seconds for the initial loading. Then, both models transcribe the audio in real-time. It is possible to change the vocabulary of this STT engine. To add new words to the vocabulary, the section Language Model (LM) adaption will be helpful. It explains the difference between the models of Vosk. The acoustic model provides the knowledge of the sounds of a language. The language model provides the flow of the words. The Phonetic dictionary includes the phonetic representation of the words known by Vosk. Depending on the language model, there are different representations. One representation is a graph that got statically compiled like Recurrent Neural Network Language Model (RNNLM). The other one is a dynamic language model. To build a language model, 100 Mb of text and their respective transcripts are recommended. For the acoustic model, 2000 hours of recordings are proposed. Hardware prerequisites to compile a new graph after adding new words are a 32 GB RAM and 100 GB of disk space on a Linux server. Kaldi, SRILM and Phonetisaurus are the software requirements. For adaptation, a special model is necessary. Currently, these can be downloaded in four languages, including German. (Alpha Cephei Inc. 2016b)

As can be extracted from the article by Nickolay Shmyrev, the idea of Multistream Time Delay Neural Network (TDNN) can be applied to the new Vosk models (Shmyrev 2021). In detail, a Multistream Factorized Time Delay Neural Network (TDNN-F) is used for the acoustic modelling presented in figure 2.6. The feature extraction utilises MFCC. Subsequently, five (not three, as shown in figure 2.6) layers of TDNN-F in a line without multistreaming are employed. These five layers receive the features as input. Afterwards, the features are split into the multistream. To be precise, these are split into a 23-pieced network. Each multistream line consists of a ReLU, batch normalization and a dropout tracked by 17 TDNN-F layers. (Han, Pan, Tadala, Ma, & Povey 2020)

For the language model, either SRILM or OpenGRM[8] can be used (Alpha Cephei Inc. 2016b). SRILM utilizes an RNNLM language model (Shmyrev 2019g). This structure is used to train the model for the German language (Shmyrev 2020).

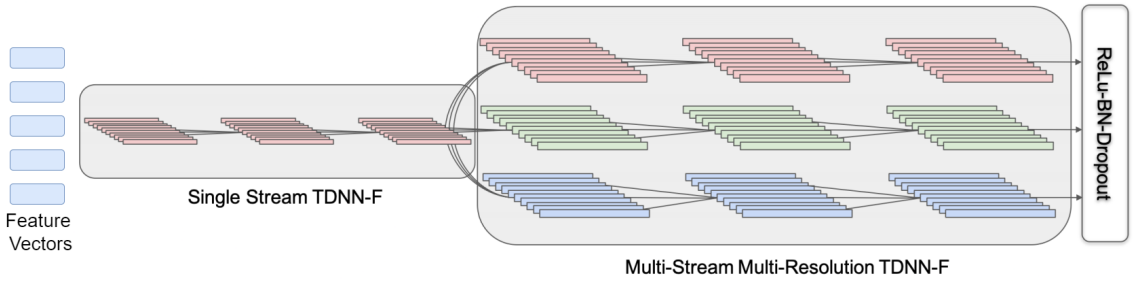

Figure 2.6: Data flow from feature input to Single Stream TDNN-F into multistream TDNN-F into the ReLU Dropout Layer.

Source: Based on: `http://danielpovey.com/files/2020_interspeech_multistream.pdf`, adapted by the author

Vosk is under an Apache-2.0 License.

The GitHub Repository for Vosk can be found at: `https://github.com/alphacep/vosk-api`

[8] https://www.opengrm.org/twiki/bin/view/GRM/WebHome

2.1.2 Proprietary Software

For a comparison between the performance and structure of open-source STT engines, a benchmark is necessary. For this reason, proprietary STT engines of proprietary speech assistants were researched. As a member of proprietary STT engines Alexa Voice Service respectively Alexa Voice Service Device Software Development Kit (SDK), Apple Speech, Microsoft Azure Cognitive Services - Speech to Text and Google Cloud Speech to Text will be examined. Due to the financial aspect and privacy concerns, the proprietary technologies were omitted from the exhibit.

2.1.2.1 Alexa Voice Service

The hidden tool in the Amazon Alexa is the Alexa Voice Service (AVS) (Marr 2021). This service allows access to the Alexa Voice Service Device SDK, or in short SDK to use the speech to text translation and further functionality provided by Alexa. The SDK is presently in version is 1.26.0 and was released in November 2021 (Amazon.com Inc. 2010b). As can be seen on GitHub, the SDK is written in the programming language C++. (Alexa AVS Device GitHub Repository 2022) AVS cloud-based service can be accessed for free. Consequently, if the internet connection should be lost, AVS will not work until the connection is re-established. (Amazon.com Inc. 2010b)

The extensive documentation disposes of tutorials for the usage of the SDK with devices that are not the default ones. A short outline of the terminology for better distinction between the terms AVS and AVS Device SDK is given. To summarise in short, AVS provides the Alexa functionality and AVS Device SDK is the library behind it (Amazon.com Inc. 2010d). For accessing AVS, an account should to be created and the device needs to be registered. The whole process is documented with comprehensive instructions. Topics like AVS Features or Concepts, Products Guidelines, and SDK and their Extensions are described with useful examples. Referring to the commissioning of a device with the SDK step-by-step guides are provided. The SDK runs on Android, iOS, Ubuntu, macOS, Raspberry Pi and Windows 64. (Amazon.com Inc. 2022e) The basic hardware requisites are a microphone, an audio source and an internet connection. (Amazon.com Inc. 2022h) Equally important are the software requirements for the SDK are Ubuntu 18.04 LTS, GCC or Clang compiler, the build tool CMake and other libraries which are presumed for the skills of AVS. Not all libraries are mandatory, as it depends on the device (Amazon.com Inc. 2010c).

In respect of the accuracy of the AVS, approximately 19 of 20 words are correctly detected by the current state-of-the-art leading proprietary tools. That is the result of successive learning. As a result of a misunderstood word, the input is used to improve the knowledge level, and therefore refine the performance. (Marr 2021)

The crucial component of the SDK is Automatic Speech Recognition (ASR). On the acoustic features, a Short Time Fourier Transformation (STFT) is applied. Additional components of the English acoustic model are two bidirectional LSTM layers and further five unidirectional

LSTM layers. The LSTMs contain the probabilities. The outcomes are the posteriors linked to the successive HMM results. These are mapped to words with the aid of a glossary and language model. (Swarup, Maas, Garimella, Mallidi, & Hoffmeister 2019)

Alexa Voice Service Device SDK is under an Apache-2.0 License.

The GitHub Repository for Alexa Voice Service Device SDK can be found at: `https://github.com/alexa/avs-device-sdk`

2.1.2.2 Apple Speech

With the announcement of iOS 10 in June 2016, the new API for speech recognition was introduced (Carman, Dzieza, & Zelenko 2016). This API can perform speech recognition from a file or in live scenarios. This technology is not entirely new, as Siri has already been using it since iOS 5. The release of iOS 10 is not just about the functionality used by Siri, it is a complete framework. For the task of speech recognition, access to the Apple servers are required and therefore an internet connection must be provided. This was state of the art in 2016. (Manson 2016) After the World Wide Developers Conference (WWDC) in 2019, the data is not sent to the servers and is therefore no longer dependent on the internet. The API is open for usage, despite a request restriction to keep the service accessible. In the case of an Apple device, the restriction is 1000 requests per hour. Otherwise, the restrictions relate to requests per day. (Apple Inc. 2017)

The documentation of Apple presents itself as organised and contains all the essential pieces of information, without being too overloaded. In this case, the relevant APIs are Speech, Core ML and Create ML. All three APIs are in the Xcode 13.0 respectively Xcode 13.3 release. Xcode is an integrated development environment (IDE) to develop apps and programs for iOS and other Apple products. Additionally, an Apple ID is required. (Apple Inc. 2022l)
Core ML, as well as the Speech APIs, are offered in Swift and Objective-C, while Create ML is only offered in Swift. For practical use cases, code examples of the APIs are available. (Apple Inc. 2022j) and (Apple Inc. 2022d) The Speech service is available from the major version and upwards iOS 10, iPadOS 10 and macOS 10 coupled with the hardware requirement of a microphone. Benefits of the framework apart from speech recognition are confidence levels and time information of the recording. To use Apple Speech the consent of the user must be given. The consent allows the transfer of the audio over the internet. Since 2019, the great majority of the process has taken place on the device. The consequence is lacking progress in improving the learning of Siri. (N. Agarwal 2019) To request permission, the documentation provides useful code in Swift or Objective-C (Apple Inc. 2022b).

The languages offered by Apple Speech will work properly. In contrast, to not provided languages. In the announcement of 2019, over 50 languages were promoted. The already mentioned Core ML provides machine learning and the ability to apply pre-trained models in an app. To create a new model, the framework Create ML can be utilised. As by Speech, there is a detailed

API description for creating a new model. With this API all different kinds of models can be developed, as can be seen in the documentation. The model does not rely on the internet, thus the model can be used on an offline device. For training CPU and GPU can be utilised. (Apple Inc. 2022c)

The core of Apple Speech is formed by a Deep Neural Network (DNN) displayed in figure 2.7. For feature extraction, MFCC is used. The extracted features are displayed as filled blue rectangles in the DNN. This DNN consists of five layers, in which each containing 192 hidden units. As a result, the phonetic probability scores are transferred from the acoustic model. The probability score, which is comparable to a language model, combined with HMM is used to determine the word. Figure 2.7 describes the small DNN used for the wake word detection. Regarding the article of Apple Machine Learning Research, the large DNN uses a similar structure, except for the size of the layer and hidden units as previously mentioned. The Trigger Score of the figure is in this case the HMM. (Apple Machine Learning Research 2017)

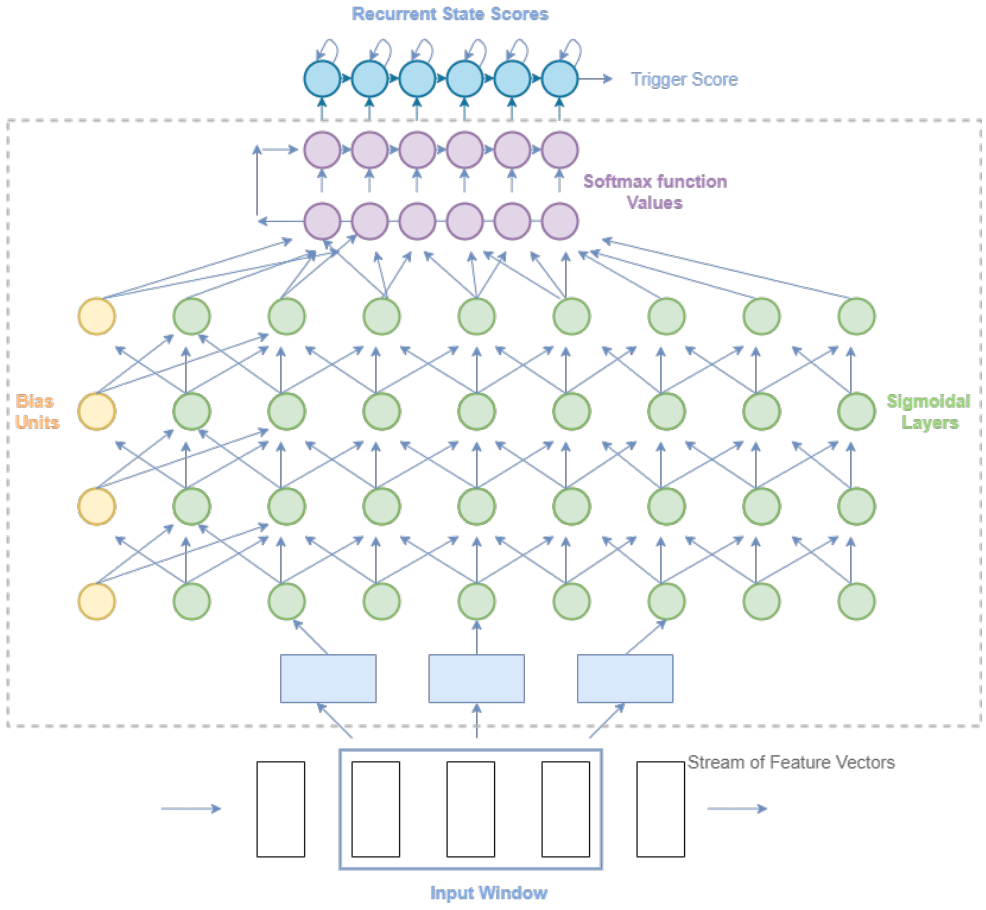

Figure 2.7: Structure of the DNN behind Apple Speech.

Source: Based on: `https://machinelearning.apple.com/research/hey-siri`, adapted by the author

Apple Speech is under an Apple Inc. License.

The GitHub Repository for Apple Speech can be found at: `https://github.com/apple`

2.1.2.3 Microsoft Azure Cognitives Service Speech to Text

The present version of Microsoft Azure Cognitive Services Speech SDK is 1.20.0. The beginning of the productivity at the SDK repository was in April 2018. Since then, a greater part of the code is in C#. (Microsoft Azure GitHub Repository 2022) This Speech SDK is used for the STT task for Microsoft Cortana (Perez 2017). The price for STT is based on packages. The basic package is available for free. It contains one Web Container with one concurrent request of five audio hours per month. The improvement of a standard package is a standard Web Container that allows 100 concurrent requests. This container costs € 0.900/h of audio. These costs only apply if the resources are used. These costs depend on the chosen region, respective of the currency. (Microsoft 2022)

Hands-on examples are provided on GitHub with either a file or a stream as possible inputs. These were tested on Windows 10 and on some Linux distributions, Android equipment equally or above to Android 6.0, Mac x64 with OS equal or higher version than 10.14 and Mac M1 arm64 with OS version 11.0 or above and lastly on iOS 11.4 gadgets. There are tutorials for each supported programming language combined in each case with the corresponding platform. Supported programming languages are C++, C#, Java, JavaScript, Python, Swift and Objective-C. (Microsoft Azure GitHub Repository 2022) The documentation itself gives a brief overview of the functionality. Additionally, the Speech Command Line Interface (CLI) is presented which does not require writing code. Generally, there are two different ways to achieve STT conversion: either via Representational State Transfer Application Programming Interface (REST API) or the Speech SDK (Urban 2022g). For common languages, the baseline model of Microsoft will satisfy (Urban 2022d). An account is required to access the STT functionality and to obtain a subscription token (Urban 2022e). For simplicity reasons, the focus lies on the Speech SDK. Practical code examples for the use of a microphone in combination with the API are given in C#. The same applies to the example with an audio file. (Urban 2022a) At the moment, there are twelve different locales offered (Urban 2022b).

It is feasible to create a new model, also called a custom model. It requires prepared sample data. The GitHub repository of the SDK provides speech samples ready to use. Various audio samples with transcripts are required for the development of a new model. (Urban 2022c) An important fact is that the custom model, as well as the baseline model, has an expiration date (Rahmel 2022). The baseline model is accessible for for the duration of a year. The custom model can be used for two years from the point of creation. The deployment and all instructions are explained under practical circumstances (Urban 2022f).

The core of the Microsoft Azure STT functionality consists of acoustic and language models. Firstly, the acoustic model contains a Residual Network (ResNet) and Layer-wise Context Expansion (LACE). Both are CNN models. Figure 2.8 below pictures the structure and components of LACE. LACE is not just a CNN. It is a more accurate version of a TDNN. A further element of the acoustic model is a Bidirectional Long Short Term Memory (BLSTM), with 512 hidden units for each layer and for inputs and outputs, which are deployed. The last element of the acoustic model is a CNN-BLSTM with three convolutional layers plus six BLSTM layers. Secondly, every acoustic model processes a language model and develops a small matrix, which is subsequently rescored with the top-500-table. Further details are provided in the paper by Xiong et al. (2018).

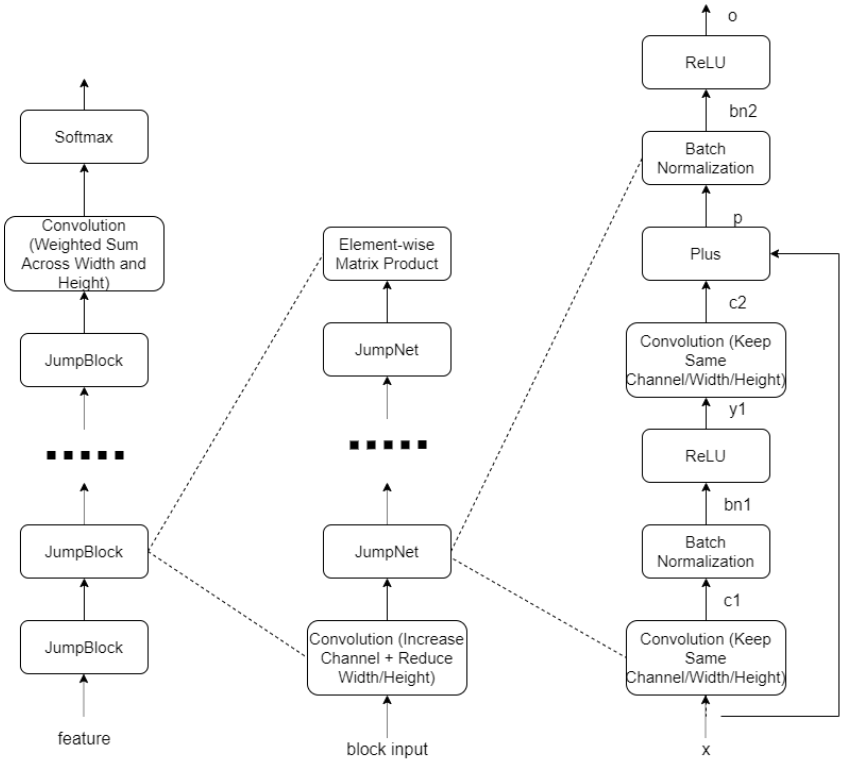

Figure 2.8: Componentes of a LACE CNN.

Source: Based on: `https://arxiv.org/pdf/1708.06073.pdf`, adapted by the author

Azure Cognitive Services Speech SDK is under an MIT License.

The GitHub Repository for Microsoft Azure Cognitives Service-Speech to Text can be found at: `https://github.com/Azure-Samples/cognitive-services-speech-sdk`

2.1.2.4 Google Cloud Speech-to-Text

The first initial movement in the Cloud Speech API for a Node.js Client GitHub Repository was in 2016. There is a still continual activity in the repository. The present stable version of the Google STT API for Node.js is v4.10.0. Almost the whole repository consists of TypeScript, JavaScript and Python code. The name Cloud Speech is indicating that to access the API an internet connection is required (Google Cloud 2018b). The Google Cloud STT offers a wide range of supported languages (Google Cloud 2018d). New customers receive 0-60 minutes of free access and after the duration of one-hour, payment is required (Google Cloud 2022).

The documentation on the Google website advertises itself with a listing of the benefits for using this API. The state-of-the-art, easy model adaption and customizable deployment is emphasized. The API can be tested with an audio file uploaded to the website (Google Cloud 2018e). The documentation provides examples in the programming languages, such as Go, Java, Node.js, PHP, Python, C++ and Ruby (Google Cloud 2018a). Each mentioned programming language has a repository. The repository contains references to the Google documentation and the API reference for the Node.js client. To use the API in a Cloud Platform project, an activated speech API and authentification are necessary. The installation of the library can be done via NPM. A practical example is given in JavaScript (Google Cloud Repository 2022).

To improve the results of Google STT, it is possible to name the origin of the audio. Thus, Google can use a better fitting model to transcribe the audio. Audio origins such as phone calls, medical dictation or conversations are predefined. Additional JavaScript Object Notation (JSON) example configurations for the model adaption are given (Google Cloud 2018f). An API request with the corresponding word in audio format and a booster value are utilised to fine-tune a model (Google Cloud 2018c).

In 2014, the tendency of speech recognition was directed towards a singular neural network. As a result, Google uses an "attention-based" and "listen-attend-spell" Listen Attend Spell (LAS) model, as can be seen from the Google AI Blog from 2019. An attention-based model continuously extracts relevant content (Chorowski, Bahdanau, Serdyuk, Cho, & Bengio 2015). Typically, LAS performs the same task except for the difference that it transcribes words beginning with every single character (Chan, Jaitly, Le, & Vinyals 2015). An Recurrent Neural Network Transducer (RNN-T) does not follow an attention-based approach directly, But it uses a sort of sequence-to-sequence procedure. It is fed the characters of the alphabet as inputs whilst speaking (Schalkwyk, Fellow, & Team 2019). Figure 2.9 below shows the structure of the RNN-T. The features of the audio wave are extracted via a log-mel filterbank. The prediction network consists of two LSTM layers and 2048 hidden nodes, displayed in purple in the figure below. Furthermore, each layer contains 640-dimensional projections. Afterwards, eight layers of unidirectional LSTM nodes are used as the encoder part. Every LSTM layer contains 2048 hidden nodes and a 640-dimensional projection layer. To improve the consistency of the hidden states a normalisation layer is applied per LSTM. Additionally, a time-reducer layer is integrated to reduce time effort. Next, the result of the encoder combined with the result of the prediction

network is fed to a feed-forward joint network with 640 hidden nodes, pictured in yellow. The encoder is the red rectangle in figure 2.9. This output is fed to the softmax layer to calculate the probability distribution. The softmax layer is illustrated as a blue rectangle. (He et al. 2018).

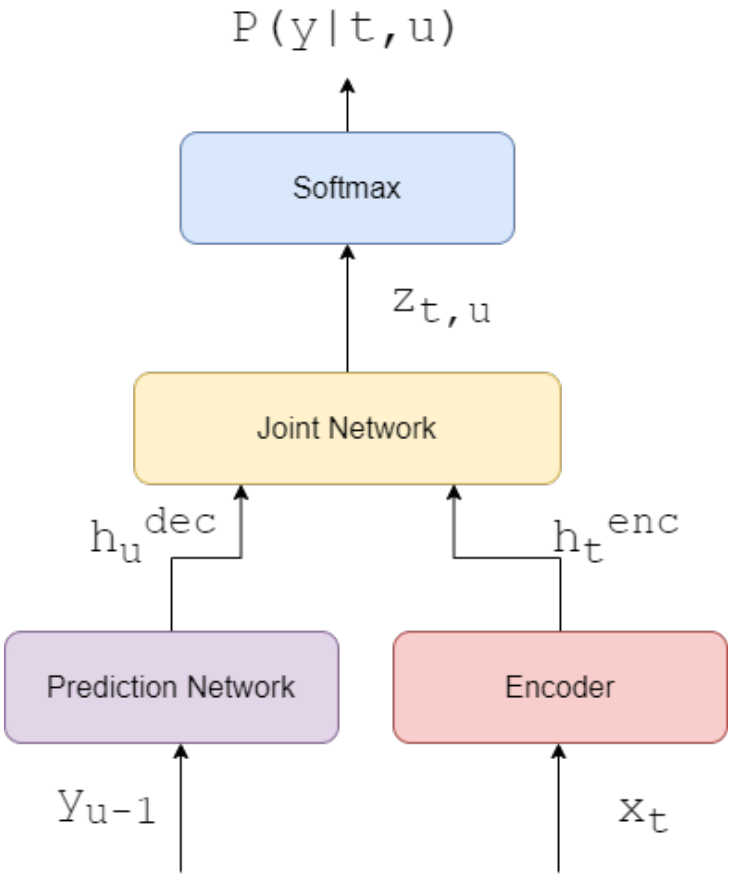

Figure 2.9: Componentes of a RNN-T with a sequence-to-sequence process.

Source: Based on: `https://opensource.com/article/20/6/mycroft`, adapted by the author

Nodejs Speech from Google is under an Apache-2.0 License.

The GitHub Repository for Nodejs Speech can be found at: `https://github.com/googleapis/nodejs-speech`

2.2 Speech Assistants

STT engines can be applied for transforming speech into text or for transcribing some audio recordings and other audio. Another use case for an SST engine is the integration into a speech assistant. In the case of a speech assistant, the STT engine forms the foundation for a speech assistant. Depending on the speech assistant, the STT engine is exchangeable. However, each assistant offers different skills to a user. Some of these skills are extendable with individually developed abilities. Each assistant was researched regarding offline usage, extendability of provided functionalities and hardware resources. Before the comparison, a brief overview of the used technologies will be given.

2.2.1 Open-Source Tools

Due to privacy concerns and the financial aspect, open-source speech assistants are prefered for the exhibit. The selected representatives of the open-source speech assistants are Mycroft, Jasper and Rhasspy Voice Assistant. This choice is based on the top seven open-source speech assistants (yourtechdietAdmin 2021).

2.2.1.1 Mycroft

In May 2016, the development of Mycroft began. Almost all of the code is in Python. The latest version of Mycroft is v21.2.2 (Mycroft AI GitHub Repository 2022). To use Mycroft, registration on the so-called Mycroft Homesite is necessary. This Homesite eases the configuration of Mycroft. On the mentioned site, all devices need to be registered to use and manage their skills and to configure personal preferences, regarding the accent and voice of the assistant (Mycroft AI Inc. 2021b). To execute a skill, Mycroft requires an internet connection. Consequently, it cannot be used offline. To use Mycroft offline the complete selene-backend[9] needs to be hosted on a server. This requires that the server and network are maintained and configured. For a low amount of connected devices, a server with 4 CPUs, 8 GB RAM and 100 GB of disk space is suggested (Mycroft Selene GitHub Repository 2022).

Mycroft is advertised with open-source, privacy, community-driven and a variety of hardware support (Mycroft AI Inc. 2020). The documentation provides all relevant information regarding special vocabulary and customisations for Mycroft, such as language settings and other criteria. Mycroft is designed with skill extendable stacks. Several practical examples of skill development are provided in the documentation (Mycroft AI Inc. 2021e). Concerning the hardware, there is a the possibility to buy a gadget that is plug-and-play. Without buying a gadget, Mycroft can be installed on Raspberry Pi, called Picroft, on Linux, on Windows, on Mac, on Docker via Image and Android. For the setup on Windows or Mac, a virtual environment is required (Mycroft AI Inc. 2021a). In the case of the Picroft Image, earlier versions of Raspberry Pis as model 2 are not supported. A Micro SD card with at least 8 GB is suggested, power supply, ethernet cable, speakers and a USB microphone are assumed.

[9] https://github.com/MycroftAI/selene-backend

Optional purchases are a monitor or a USB keyboard. This depends on how the interaction with Picroft will take place, for example via Secure Shell (SSH). (Mycroft AI Inc. 2021c)
Apart from the hardware costs, Mycroft is free to use and download. Concerning privacy, Mycroft can convert speech to text locally, if an offline STT engine is integrated. Only the text gets forwarded and processed by the Mycroft servers. To exchange the STT engine, practical examples are given. Currently, Google's STT engine is used as the STT engine, but as soon as Mozilla's Deepspeech is applicable this will be exchanged (Mycroft AI Inc. 2021d).

As mentioned before, Mycroft's skills can be enhanced with individually developed skills. For installation, either a voice command or the command line can be used. The same applies for the uninstallation of a skill. To develop a skill, the programming language Python, a GitHub account and an instance of Mycroft are necessary. The process of creating a skill is demonstrated using an example. A skill can be input via speech or by typing it into the command line (Mycroft AI Inc. 2021c). The main part of the skill displayed in figure 2.10 looks like the code fragment 2.1. To demonstrate how Mycroft works, the example of a timer skill[10] is used. A small extract of the timer skill delegates the timer function to the timer class.

```
def get_timers_matching_utterance(
    utterance: str, timers: List[Countdown], regex_path: str
) -> List[Countdown]:
    """Match timers to an utterance that matched a timer intent."""
    matcher = TimerMatcher(utterance, timers, regex_path)
    matcher.match()

    return matcher.matches
```

Code Fragment 2.1: A fraction of the Timer Skill of Mycroft, Based on: `https://github.com/MycroftAI/mycroft-timer/blob/21.02/skill/match.py`, adapted by the author

To address a skill, either the command line or the language can be used. To communicate with Mycroft via speech, the wake-word engine must be triggered, followed by the skill which should be carried out by Mycroft. For the wake-word functionality, either Pocket Sphinx or Precise can be used. The voice command, which is in this case "set a timer for 10 minutes.", is recorded by the microphone of the device. Afterwards, an STT engine converts it into text. Currently, Google's STT engine is used due to accuracy reasons. Then, keywords are extracted from the transcribed voice command. Keywords are then extracted from the transcripted voice command. The keywords, in combination with the intent parser, map to the fitting mycroft-timer skill. As an intent parser either Adapt[11] or Padatious[12] can be used. To transport the information verbally to the user, a TTS engine is deployed. Currently, Mimic[13] is used as a TTS engine. (Ovens 2020) Figure 2.10 pictures the process described above. (Ovens 2020)

[10] https://github.com/MycroftAI/mycroft-timer/blob/21.02/skill/match.py
[11] https://mycroft-ai.gitbook.io/docs/mycroft-technologies/adapt
[12] https://mycroft-ai.gitbook.io/docs/mycroft-technologies/padatious
[13] https://mycroft-ai.gitbook.io/docs/mycroft-technologies/mimic-overview

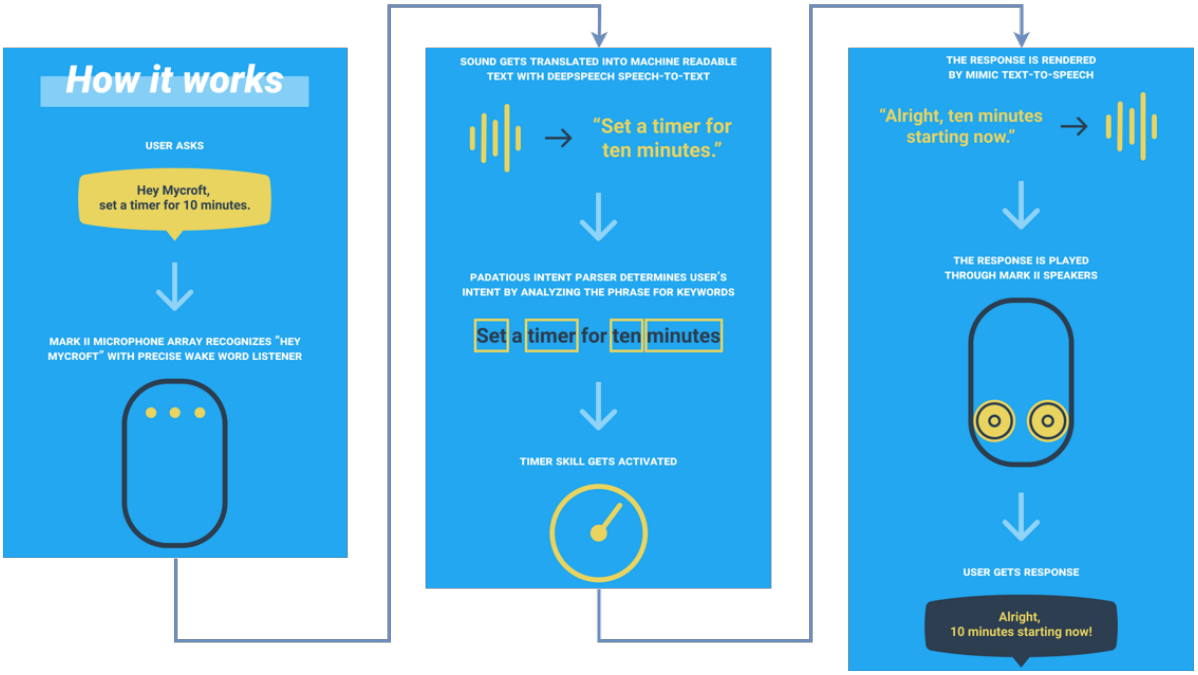

Figure 2.10: Workflow of a Mycroft skill

Source: Based on: `https://opensource.com/article/20/6/mycroft`, adapted by the author

Mycroft-core is under an Apache-2.0 License.

The GitHub Repository for Mycroft can be found at: `https://github.com/MycroftAI`

2.2.1.2 Jasper

In 2014, the development of Jasper started as can be seen in the GitHub repository. Further information taken from the repository shows that almost 100% of the code is written in Python (Jasper Client GitHub Repository 2022). This assistant is promoted with a low financial commitment, as it is plug-and-play and is easy to expand with user code. Jasper requires either an ethernet or WiFi connection. (Saha & Marsh 2014a).

The documentation covers the basics for starting to use Jasper. Concerning the hardware requirements, a Raspberry Pi Model B, USB microphone, 4 GB SD card, internet connection, power supply, micro USB cable and speakers are needed. After building all previous components on the Raspberry Pi, the installation of Jasper follows. (Saha & Marsh 2014a)
There are three ways for installation; the quick start, the installation via the package manager and lastly the manual installation. The first and recommended quick start installation provides a confirmed disk image for model B. After putting the image into operation, the installation instructions for Jasper are four command line inputs. The second installation guide gives instructions for Jasper on ArchLinux[14]. The third and last option guides through the manual installation (Saha & Marsh 2014b). Additionally, the STT and TTS engines need to be configured. Currently, there are five supported STT and eight TTS engines for Jasper.

[14] https://archlinux.org/

Pocketsphinx[15], Google STT [16], AT&T STT [17], Wit.ai STT[18] and Julius [19]can be used as an STT engine. Julius and Pocketsphinx are the only two engines that can process the speech locally on the machine. ESpeak[20], Festival[21], Flite[22], SVOC Pico TTS[23], Google TTS[24], Ivona TTS[25], Mary TTS[26] and Mac OS X TTS[27] are the options for the TTS engine. (Saha & Marsh 2014c)

To start interaction with Jasper, the trigger word is used to activate the wake-word engine. Afterwards, audio feedback will be given by Jasper. (Saha & Marsh 2014d) Some basic skills offered by Jasper are time, weather, news, Gmail, Facebook notifications, jokes and some more (Saha & Marsh 2014e). For some skills, credentials are required. It is possible to add a new skill to Jasper. There are two different developer APIs defined called Standard and Notification Module (Saha & Marsh 2014f). If Jasper needs to play an interactive role, the Standard Module. For other cases, the Notification Module is used. A code fragment 2.2 of a notification skill [28] is provided. It shows the email skill of Jasper:

```
def handleGMXEmailNotifications(self, lastDate):
    # grab gmx emails
    emails = Gmxmail.fetchUnreadEmails(self.profile, since=lastDate)
    if emails:
        lastDate = Gmxmail.getMostRecentDate(emails)
    # notifications read to user as provided
    def styleGmxEmail(e):
        return "New email from \%s." \% Gmxmail.getSender(e)
    # put notifications in queue
    for e in emails:
        self.q.put(styleEmail(e))
    # return timestamp of most recent gmx email
    return lastDate
```

Code Fragment 2.2: Code of a notification skill of Jasper ,Based on: `https://jasperproject.github.io/documentation/api/notification/`, adapted by the author

[15]https://sourceforge.net/projects/cmusphinx/files/pocketsphinx/5prealpha/
[16]https://cloud.google.com/speech-to-text?hl=de
[17]https://www.att.com/gen/sites/ipsales?pid=17755
[18]https://wit.ai/
[19]https://github.com/julius-speech/julius
[20]https://github.com/espeak-ng/espeak-ng
[21]https://www.cstr.ed.ac.uk/projects/festival/
[22]https://github.com/festvox/flite
[23]https://github.com/naggety/picotts
[24]https://cloud.google.com/text-to-speech?hl=de
[25]http://www.tts-systeme.de/ivona-home/index.html
[26]http://mary.dfki.de/
[27]https://www.cereproc.com/de/products/Mac_OSX_voices
[28]https://jasperproject.github.io/documentation/api/notification/

Practical straightforward examples are given for both APIs. It is advisable to look at the provided modules of other developers before developing a skill. How to add a new skill to Jasper depends on the instructions of the third-party developer (Saha & Marsh 2014e).

The main function is placed in a file called jasper.py. It coordinates everything, as can be seen in figure 2.11. This main function initialises the microphone, conversation and profile displayed, as red rectangles in the figure below. The representative of the conversation, pictured in green, can use the microphone and the information from the profiles file. This results in building a brain and a notifier. The brain and the notifier are displayed as the purple rectangles in the figure. The main passes the microphone and profile to the brain and brings the components into the ready-to-use state. The task of the brain is the communication between the self-created modules/skills and the basic offered functionalities. A prerequisite for a usable module is that the functions isValid(), handle() and WORDS = [...] are implemented.(Saha & Marsh 2014c)

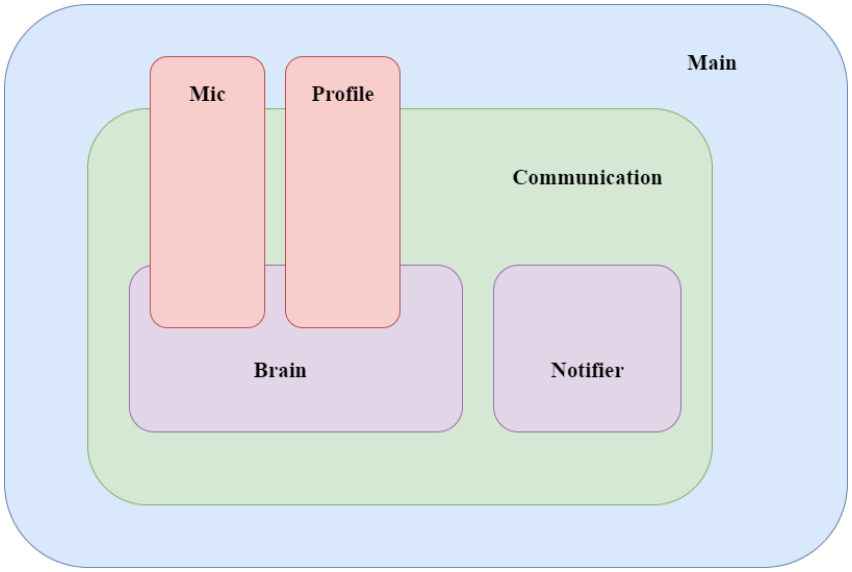

Figure 2.11: Programm organisation of Jasper

Source: Based on: `https://jasperproject.github.io/documentation/configuration/`, adapted by the author

Jasper is under an MIT License.

The GitHub Repository for Jasper can be found at: `https://github.com/jasperproject`

2.2.1.3 Rhasspy Voice Assistant

Michael Hansen is the creator and maintainer of project Rhasspy. Since November 2018, he has been working on Rhasspy. The latest release v2.5.11 of Rhasspy supports a wide range of languages, although it is designed to be completely offline in a local network. The German model is provided by Kaldi, Pocketsphinx or Deepspeech as the STT engine. (Hansen 2022d) The target group for Rhasspy was initially home assistants or smart home assistants, as can be seen in the old Github repository (Hansen 2016). In the new repository, the target group was expanded with experienced users. As can be seen on the website, the assistant is compatible with Home Assistant, Hermes Protocol[29], Hass.io[30], Node-RED, Jeedom[31] and OpenHAB[32] (Hansen 2022c). No costs arise with the usage of Rhasspy. (Hansen 2022d)

The documentation gives a brief overview of the installation steps and the supported hardware. Rhasspy is supported on Raspberry Pi 2 model B/B+ and Raspberry Pi 3 model B/B+ or Raspberry Pi Zero. An SD Card of 32 GB is recommended. Further requirements are a laptop or a server, as well as a microphone. A monitor is also advantageous. The installation deals with the supported platforms and the corresponding installation steps. To install Rhasspy a Docker image, Debian-based Linux distribution, Virtual Environment, Hass.io or Windows Subsystem for Linux (WSL) can be used. The Docker installation is recommended to start with Rhasspy (Hansen 2022a). In the Tutorial chapter, detailed instructions on customisation using the web interface are given. The web interface has a clear design and is easy to understand. Despite other functionalities of the web interface, it allows managing tools like the STT engine, the audio recording library, the TTS engine and further configurations. (Hansen 2022e)

Another benefit of the web interface is the option to add new sentences. These sentences are used to find the right skill, by entering the intent name and the sentences that should be recognized. It only requires the saving this file and retraining. To acquire speech-like feedback of the Rhasspy via TTS, which is called a service, a second interface called Node-RED flow is used, as can be seen in the figure 2.12 below. The API for TTS, audio output and other services need to be connected with the corresponding intent. This is achvied by dragging and dropping the components into the canvas and connecting boxes. The first node on the left side is the connection to the Rhasspy. This node receives all incoming requests. The green nodes are debug nodes. These are used to check the message content. The task of the intent switch is to filter the requests. If this node is missing, all skills are executed. The orange boxes are the intents containing logic. The output of these intents is set into the message body with the nodes on the right side. The TTS node performs the audio feedback. On the internet, video tutorials on customisation and intent creation can be found and are useful supplements to the description found in the documentation. To add a new functionality, the choice of programming languages is restricted to the ones which provide access to Message Queuing Telemetry Transport (MQTT), WebSockets and other offered protocols of Rhasspy. (Hansen 2022e)

[29] https://hermes-protocol.com/
[30] https://www.home-assistant.io/
[31] https://www.jeedom.com/en/
[32] https://www.openhab.org/

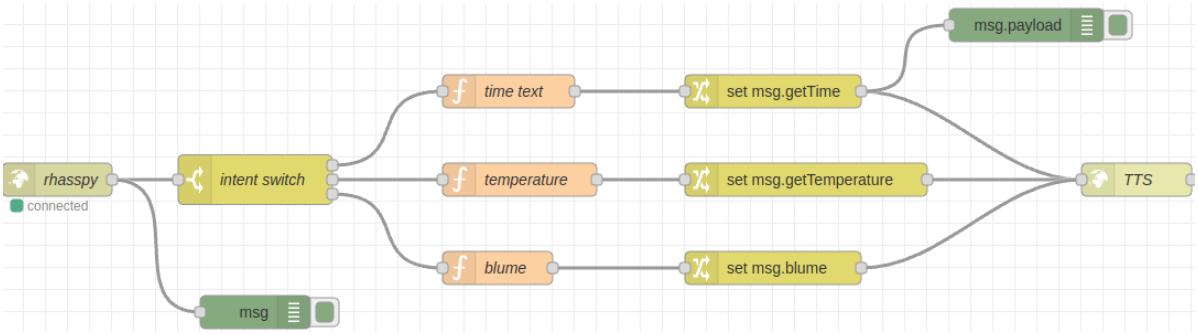

Figure 2.12: Three example intents of Rhasspy created in Node-RED.

Source: Created by the author

In contrast to other speech assistants, Rhasspy can run completely offline. This means that no further cloud is needed to work with Rhasspy. As can be seen in the figure 2.13 below, the basic structure of Rhasspy provides the microphone, speakers and the core with the functionality. The core functionality is illustrated as a brain. The speech is transcribed locally and parsed with the intent parser. Subsequently, with the then known intent, services like Home Assistant or Hass.io can be addressed. To address one of these services an API call over the local network is used. After the task is completed, often the TTS engine gives feedback to communicate that the task has been performed. The basic structure of Rhasspy can be configured to have more than one microphone and speakers. These additional microphones and speakers are called satellites. In this case, the brain is decoupled from the rest, such as the services. (Hansen 2022f)

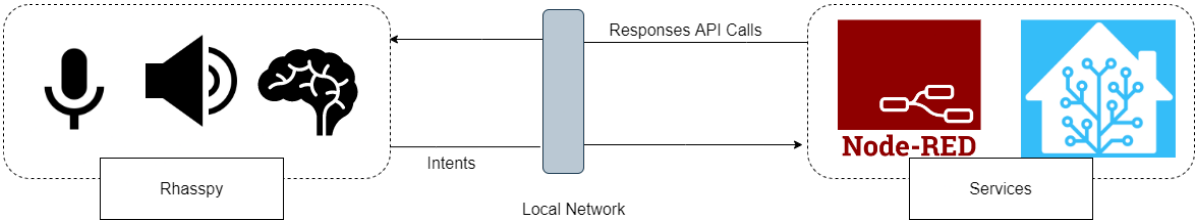

Figure 2.13: Communication of Rhasspy with Hass.io and Node-RED

Source: Based on: `https://rhasspy.readthedocs.io/en/latest/why-rhasspy/`, adapted by the author

Rhasspy is under an MIT License.

The GitHub Repository for Rhasspy can be found at: `https://github.com/rhasspy/rhasspy`

2.2.2 Proprietary Tools

In contrast to open-source software, proprietary software can be expensive in terms of usage or have hardware restrictions. There is also a downside in terms of privacy, as some users are not aware of the fact that their audio recordings are being saved and can be used for training. However, before the audio becomes useful a person has to firstly transcribe it. (Thakur 2021)

2.2.2.1 Amazon Alexa

The first Amazon Echo was released in 2014 (Etherington 2014). A statistic by Kunst (2021) in the year 2021 has shown that almost three-quarters of the smart-speaker owners in the US have owned an Amazon Alexa product (Kunst 2021). Further research carried out by Laricchia (2022) has pointed out that Amazon was the international ruling supplier in 2021 with a market share of 26.5% followed by Google. Since audio recordings are transmitted to an Alexa Service in order to perform a task, this demonstrates that Alexa relies on an internet connection. (Amazon.com Inc. 2010b) The STT engine of Alexa can be configured to local voice control (Amazon.com Inc. 2022a). The ongoing version of the Alexa Skills Kit (ASK), which can be used for skill development, is in the latest version v2.10.0. (Amazon.com Inc. 2010b)

A main part of the documentation focuses on the skills. It touches upon how the user can use a skill and how the user receives the result of a skill. There is a listing of different skill types combined with existing skills for demonstration purposes. APIs, such as the Smart Home API or Custom Skill Interface (Custom Interface), focus on the feature of the explicit skill. (Amazon.com Inc. 2022f). To use Alexa, either an Amazon Alexa device can be purchased or an Alexa integrated device can be used (Amazon.com Inc. 2022g) and (Amazon.com Inc. 2022b).

Each skill includes a voice interaction model. This model specifies the expressions that need to be said by the user in order to trigger a skill (Amazon.com Inc. 2022j). In contrast to the custom voice interaction model, the pre-built interaction model uses pre-formed requests (Alexa GitHub Repository 2022). For skill development, some requirements should be met. Firstly, an Amazon developer account is necessary. Secondly, depending on the skill, an Amazon Web Services (AWS) account is mandatory, with either the Alexa developer console or ASK CLI, an Alexa Skill Kit SDK or an IDE. Lastly, the non-mandatory Alexa Presentation Language (APL) authoring tool is required. The ASK SDK supports JavaScript, Java and Python. A skill created with an existing voice interaction model provides benefits, such as other programming languages than those with a custom voice interaction model (Amazon.com Inc. 2022c). To offer a skill as cloud-based, the skill should be hosted for instance on AWS Lambda which is a service provided by AWS (Amazon.com Inc. 2022d). An extract of a skill example for a Hello World[33] is displayed in listing 2.3.

[33] https://developer.amazon.com/en-US/docs/alexa/alexa-skills-kit-sdk-for-python/sample-skills.html

```
class SayHiIntentHandler(AbstractRequestHandler):
    """Handler for Say Hi Intent."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return is_intent_name("SayHiIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speech_text = "Hi, from Classes!"

        handler_input.response_builder.speak(speech_text).set_card(
            SimpleCard("Hi", speech_text)).set_should_end_session(
            True)
        return handler_input.response_builder.response
```

Code Fragment 2.3: Say Hi Intent of Alexa, Based on: https://developer.amazon.com/en-US/docs/alexa/alexa-skills-kit-sdk-for-python/sample-skills.html, adapted by the author

How Alexa works is shown with the Smart Home Skill example in figure 2.14. A prerequisite is that the user has enabled the Smart Home Skill. Consequently, the user can address the skill by the usage of the wake-word that Alexa combines with the respective utterance. Alternatively, an app can be utilised to trigger the skill, as pictured as a smartphone in figure 2.14. Alexa extracts the information and the endpoint to operate. Afterwards, Alexa transmits the message, also called the directive, to a lightbulb for example. The directive needs to contain all necessary information in order to fulfill the task, as well as the authentication of the client, the endpoint and other criteria. In addition, the directive is transmitted to the skill code in AWS Lambda. At this point, the directive is checked whether the directive has been authenticated or not. At this point, the directive is checked whether the directive has been authenticated or not. The skill then exchanges information with the device that has triggered the skill, illustrated as the smartphone or Echo. As a result, the skill transmits an event to Alexa with feedback when the task has been performed. This event can be asynchronous, in the case of the device cloud, and synchronous, in the case of Lambda. In the example of a Smart Home Lightbulb, Alexa would also receive notice if the light state has been changed manually. (Amazon.com Inc. 2022i)

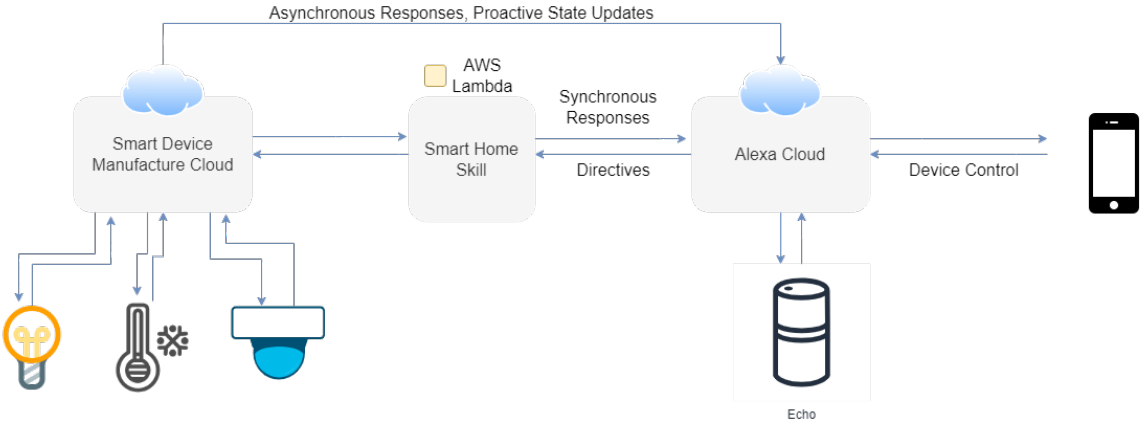

Figure 2.14: Communication between Alexa, smart home devices and the Alexa Coud.

Source: Based on: `https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html`, adapted by the author

Alexa Skills Kit SDK is under an Apache-2.0 License.

The GitHub Repository for Alexa Skills Kit can be found at: `https://github.com/alexa/alexa-skills-kit-sdk-for-nodejs`

2.2.2.2 Apple Siri

In 2011, Apple first revealed Siri with the iPhone 4S. Since the reveal, much has changed. Presently, in 2022, Siri is one of the most well-known speech assistants in the world. Almost all iPhone users have tried Siri at a minimum of once. Siri is used for online purchases by a quarter of the users. (Wardini 2022)
The first versions of Siri, the speech assistant, were dependent on the internet. However, this has also changed. Siri is now capable of processing the majority of the learning process offline, but not entirely. (Apple Inc. 2022e). Siri can speak more than 30 languages (Apple Inc. 2022f).

Siri can be utilised by users as a Smart Home Assistant, in order to manage the ambience at home. Siri is comparable to a personal assistant that receives dictated texts and then sends these onwards. Of course, Siri can perform other useful delegable tasks (Apple Inc. 2022h). The present version of Xcode for the SiriKit, the API provided by Apple, is 13.0-13.3. The SiriKit provides the functionality which is known to Siri users. The basic functionality of Siri originates from the Intents and IntentsUI framework tied to the actions of the users. The documentation offers topics such as User Interface (UI) components for interactivity, instructions for a new extension and how to design an Intents UI extension. The Intents UI extension displays more information on the screen after the task is completed. However, SiriKit is supported on iOS and iPadOS at 10.0+, macOS 12.+, tvOS 14.0+ and finally, watchOS 3.2+. Generally, there are seven different intents to handle a given Intent. The "Standard Intents" are the default intents of Siri. They are grouped by fields such as Car Commands and Messaging. The Shortcuts and Donations focus on the synergy between usability and Siri. The "Vocabulary" allows addressing functionality via customised voice commands. The Custom UIs display additional information

on the screen of the device. For each subcategory in the previously mentioned subjects, interface descriptions are provided in Objective-C and Swift. (Apple Inc. 2022i) Usage of the API itself is free, but access to the API requires a paid account (Apple Inc. 2022k).

For creating a skill or intent there are two options. Option A is to create a completely new skill or intent. Option B is to adapt a standard or default skill or intent, if there is already one existing. Each individual intent should have three phases, which are called the resolve, confirm and handle phases (Apple Inc. 2022g). To process the intent a corresponding handler needs to be implemented, as well as the resolving and handling of intents. The intent handler addresses the app and causes the loading of the corresponding app to the intent. A brief extract of an intent of Siri, in listing 2.4 shows what occurs within the shortcuts. A practical example of a Custom Intent[34] is provided. The previously mentioned resolve and handle phases of an intent are used at this point, as confirmation needs to take place before the intent can be completed. User interaction is necessary to check and confirm the information. Code examples are provided in the documentation for resolving and confirming the intent. (Apple Inc. 2020)
A new intent can be added to Siri's functionality via shortcuts. That means that the created intent needs to offer a shortcut in order to be accessible by Siri. To address the intent, the trigger word followed by the name of the shortcut name is used. To develop the intent, Xcode with an API is account is a prerequisite, such as the programming languages Swift and Objective-C. (Apple Inc. 2022k) and (Apple Inc. 2022a)

[34]https://developer.apple.com/documentation/sirikit/soup_chef_accelerating_app_interactions_with_shortcuts?language=obj

```
/// The system calls this method when continuing a user activity
// through the restoration handler
/// in `UISceneDelegate scene(_:continue:)`.
override func restoreUserActivityState(_ activity: NSUserActivity) {
    super.restoreUserActivityState(activity)

    if activity.activityType == NSUserActivity.viewMenuActivityType {
        // This order came from the "View Menu" shortcut that is based
        // on NSUserActivity.
        prepareForUserActivityRestoration() {
            self.performSegue(withIdentifier: SegueIdentifiers.mySoupMenu.rawValue
                                        , sender: nil)
        }

    } else if activity.activityType == NSStringFromClass(OrderSoupIntent.self) {

        // This order started as a shortcut, but isn't complete because the user
        //  tapped on the SoupChef Intent UI
        // during the order process. Let the user finish customizing their order.
        prepareForUserActivityRestoration() {
            self.performSegue(withIdentifier: SegueIdentifiers.mySoupMenu.rawValue,
                                        sender: activity)
        }

    } else if activity.activityType == NSUserActivity.orderCompleteActivityType,
             let orderID = activity.userInfo?[NSUserActivity
                                        .ActivityKeys.orderID.rawValue]
                                                    as? UUID,
             let order = mySoupOrderManager.order(matching: orderID) {

        // This order was just created outside of the main app through an intent.
        // Display the order in the order history.
        prepareForUserActivityRestoration() {
            self.displayOrderDetail(order)
        }
    }
}
```

Code Fragment 2.4: MySoup intent of Siri, Based on: https://developer.apple.com/documentation/sirikit/soup_chef_accelerating_app_interactions_with_shortcuts?language=objc, adapted by the author

Figure 2.15 shows how the SiriKit works. At first, Siri will extract the intent from the speech, which is shown as the first purple circle in the figure below. This happens with the use of the vocabulary specified as according to the intent. The intent is illustrated as the second purple circle next to Speech. The vocabulary is shown as the first green rectangle. This is where the resolve and subsequently the confirm phases take place. In the confirm phase, the intent and all parameters need to be validated. Depending on the intent certain actions will be performed. Some of the actions use the app logic in order to process the task given by the operator. Lastly, either audio or optical feedback will be given by Siri, which is illustrated as the purple response circle. This is the last phase and is called the handle phase. (Ortinau 2021)

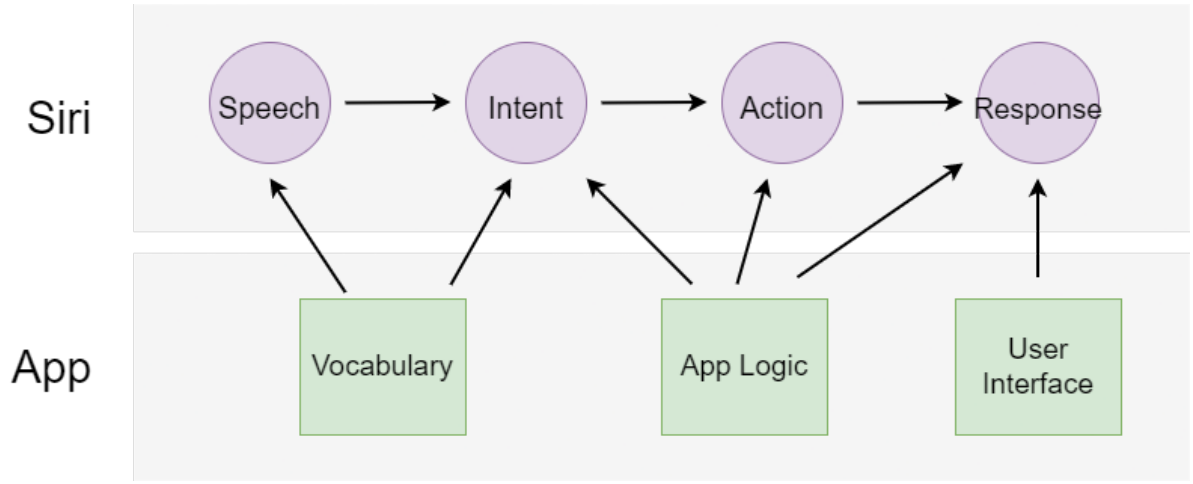

Figure 2.15: Communication between apps and Siri

Source: Based on: `https://docs.microsoft.com/en-us/xamarin/ios/platform/sirikit/understanding-sirikit/`, adapted by the author

SiriKit is under an Apple Inc. iOS Software License.

The GitHub Repository for SiriKit can be found at: `https://github.com/apple`

2.2.2.3 Microsoft Cortana

According to the GitHub repository of Microsoft (MS), the development of Cortana started in 2017. The predominant programming language is C#. (Microsoft GitHub Repository 2022) Regarding the Microsoft documentation, Cortana is no longer supported on Windows 10 since May 2020. For this reason, the developer stage of Cortana Skills Kit was also halted. (Bridge 2021) Since then, Cortana has been adapted and is now available with Microsoft 365 in order to improve the efficiency of users (Microsoft 2021). In an article in 2021, further information regarding Cortana for Microsoft 365 was published (Bridge 2021). Cortana can now be accessed as a cloud-based assistant using a Microsoft 365 work or school account. (Microsoft 2021).

As expected from such an assistant, Cortana in MS 365 helps to connect everyday life with more hand-free experiences. This happens through the integration of MS teams and when combined with Outlook (Zawideh 2022). Functionality, such as briefing emails and reading emails out aloud, is currently also provided. Additionally, Cortana helps users to stay updated in regards to appointments or information. Further, Cortana can help creating new appointments. Only the English language is currently supported, but further language support is planned. However, Cortana is still in use with Windows 10 and newly with Outlook and MS Teams on the mobile app for iOS and Android. To improve the learning continuously, MS uses the text version of the command and not the audio itself. For machine learning in the Office 365 cloud, there is no Human-in-the-loop (HITL). This means that there is no human interaction with the data. (Microsoft 2021)

A brief retrospective of the time before Cortana was integrated into MS 365. The GitHub repository provides task-based samples divided into Consumer and Enterprise categories. The difference between these is that consumer samples are from third parties and the Enterprise samples are provided by Azure Active Directory. The samples are written in C# and Node.js (Microsoft GitHub Repository 2022). As Cortana is outdated, the documentation is not available anymore.

To create a skill for Cortana, the Bot Framework is recommended. For the development, an Azure account is essential. At first, there is a possibility of a trial free of charge, however further costs can be applied. (Microsoft GitHub Repository 2022). For the Bot Framework SDK, the version of .NET Framework 4.6 or more recent is necessary. Explicit examples are provided (Standefer & Fingold 2017). The Bot Framework SDK offers practical examples for C#, Java, JavaScript and Python. The requirements differ depending on the programming language. For the creation of a code template, the usage of either Visual Studio or the VS Code/CLI is possible. In the case of the language C#, an ASP.NET Core Runtime 3.1 and a Bot Framework Emulator are essential. Additional knowledge of ASP.NET Core and asynchronous programming is a benefit. Afterwards, the Bot needs to be installed in Azure (Fingold 2021). An extract of the example code from the Roller Skill sample[35] of the Bot Framework is shown in the listing 2.5. This code is used to display the cards on the canvas. (Microsoft BotBuilder GitHub Repository 2022)

[35]https://github.com/microsoft/BotBuilder-Samples/tree/releases/v3-sdk-samples/Node/demo-RollerSkill

```
bot.dialog('HelpDialog', function (session) {
    var card = new builder.HeroCard(session)
        .title('help_title')
        .buttons([
            builder.CardAction.imBack(session, 'play roll some dice', 'Roll Dice'),
            builder.CardAction.imBack(session, 'play craps', 'Play Craps')
        ]);
    var message = new builder.Message(session)
        .speak(speak(session, 'help_ssml'))
        .addAttachment(card)
        .inputHint(builder.InputHint.acceptingInput);
    session.send(message).endDialog();
}).triggerAction({ matches: /help/i });

/** Helper function to wrap SSML stored in the prompts file with <speak/> tag. */
function speak(session, prompt) {
    var localized = session.gettext(prompt);
    return ssml.speak(localized);
}
```

Code Fragment 2.5: Extract of the Roller Skill sample of a Bot for Cortana, Based on: `https://github.com/microsoft/BotBuilder-Samples/tree/releases/v3-sdk-samples/Node/demo-RollerSkill`, adapted by the author

To interact with Cortana, it firstly needs to be activated. Further on, the concern needs to be addressed. Figure 2.16 illustrates an example of a cancelled Las Vegas trip using Cortana. For this concern, the "Adventure Works" app is active in the background. The interaction with the app takes place via canvas and the voice interaction via Cortana. The addressed concern is displayed on the canvas, where user interaction such as confirming or cancelling is required. Several possible results fit the user's concern. All possible concerns are displayed on the canvas. Cortana tries to observe which of the elements is correct by initiating interaction. The user designates the name of the element. The functionality of cancelling an appointment does not provide a rollback action. For this reason, verification is required to avoid errors, after which the completed canvas is presented. (Bridge 2021)

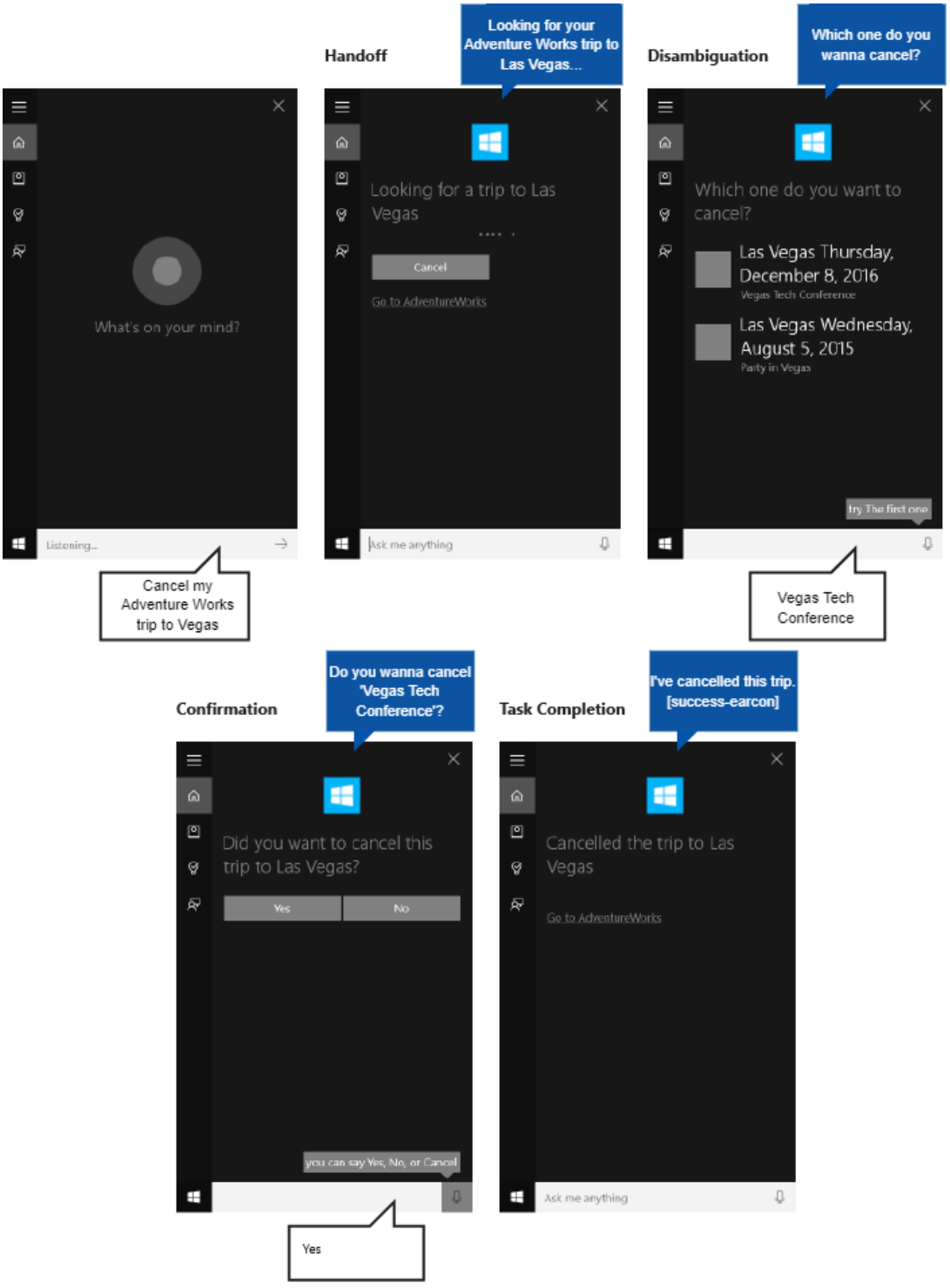

Figure 2.16: Interaction example of Cortana

Source: Based on: `https://docs.microsoft.com/en-us/windows/apps/design/input/cortana-design-guidelines`, adapted by the author

Cortana is under an Creative Commons Attribution 4.0 International Public License.

The GitHub Repository for Microsoft Cortana can be found at: `https://github.com/microsoft/cortana-skills-samples`

2.2.2.4 Google Assistant

In 2017, the first Google Assistant SDK was released on GitHub. The SDK is used to integrate voice-driven control into individual projects by communicating with the Google Assistant. In Google's GitHub repository, Assistant SDKs for Node.js, Python and C++ are offered. The latest version of the Assistant SDK for Python is 0.6.0. (Google Developers 2022c)
Since the beginning of 2017, Google and Amazon have been competitors. A statistic, by Laricchia (2022) in 2022, has shown that Amazon is the market leader in the international smart speaker market, followed by Google. Google has a share of 20.5% (Laricchia 2022). Features such as Google Maps, with downloaded maps, and playing music can be used offline. However, some configurations are required beforehand. (Muelaner 2021) Google Assistant can be connected to smart home devices (Google 2022b). Depending on the Google device, different languages are supported. All devices, except the Google Nest Hub Max, support German (Google 2022a). In terms of the costs, the usage of the Google Assistant is free (Stegner 2018).

The documentation presents the Google Assistant as a faster alternative for interacting with an Android app, a possibility for natural conversations, a way to improve internet presence and as a tool to operate with smart home devices (Google Developers 2022b). Each of the previously mentioned topics are discussed in the documentation. For the sake of simplicity and understandability, the focus is on Google Assistant embedded into a Raspberry Pi, which uses the Google Assistant SDK. Google Assistant SDK on Raspberry Pi counts as an experimental device. It needs to be emphasised that it allows only experimental and not commercial purposes (Google Developers 2020). The SDK is used to address Google Assistant Service. The feature of the so-called Hands-free Activation is not supported on experimental devices. Therefore, push-to-talk is instead. This service consists of an API that enables direct access to audio bytes of the requests and the responses of the assistant. The API provides connections for Node.js, Go, C++, Java and other languages that are capable of Google's Remote Procedure Calls (RPC), also called GRPC (Google Developers 2020). Hardware prerequisites are a running device with an internet connection, a microphone and a speaker (Google Developers 2022g). For the installation, a step-by-step guide for C++, Node.js, Python and Android is offered (Google Developers 2022f). Additional information on configurations for audio, developer projects, account settings, registration of the model and installation instructions is given (Google Developers 2022e). To use a device with Google Assistant, it needs to be registered. This can be done via UI (Google Developers 2022g). After completing all the steps, the practical examples can be used for testing (Google Developers 2022h).

It is possible to extend the abilities of the Google Assistant. To address your device with a voice command, Device Action is necessary. In the documentation, an example use case of a light is used for demonstration purposes. For this example, the following hardware requirements need to be met: a breadboard, an LED, a series resistor and two jumper wires female and male connectors. Instructions to mount these components on the Raspberry Pi Model B can be found in the documentation. Furthermore, detailed information on how to register a skill or also call trait, and handle commands is given (Google Developers 2022d). The code listing 2.6 shows an example for a trait[36] which handles all devices.

```
device_handler = device_helpers.DeviceRequestHandler(device_id)

@device_handler.command('action.devices.commands.OnOff')
def onoff(on):
    if on:
        logging.info('Turning device on.')
    else:
        logging.info('Turning device off.')

@device_handler.command('com.example.commands.BlinkingLight')
def blink(speed, number):
    logging.info('Blinking device \%s times.' \% number)
    delay = 1
    if speed == "SLOW":
        delay = 2
    elif speed == "QUICK":
        delay = 0.5
    for i in range(int(number)):
        logging.info('Device is now blinking.')
        time.sleep(delay)
```

Code Fragment 2.6: Extract of the blinky light code example of a Google Assistant skill, Based on: `https://github.com/googlesamples/assistant-sdk-python/blob/master/google-assistant-sdk/googlesamples/assistant/grpc/pushtotalk.py`, adapted by the author

[36] assistant-sdk-python/pushtotalk.py at master · googlesamples/assistant-sdk-python · GitHub

The example with the Raspberry Pi and the light demonstrates the workflow displayed in figure 2.17. In this case, the device (Raspberry PI) has installed the Google Assistant SDK and holds the credentials to access the API of the Google Assistant Service. The assistant is activated via a button, followed by the intent. Afterwards, the device model and its identifiers are sent to the service. Then the corresponding response actions are identified. The service then will perform speech to text conversion. Natural Language Processing (NLP) is used to figure out the intent of the user. The context specifies the supported device actions. Depending on the context, actions are chosen to accomplish the request. To verify the request, device matching takes place. Subsequently, the service transmits the text for the output linked with the supported command for the device. (Google Developers 2022a)

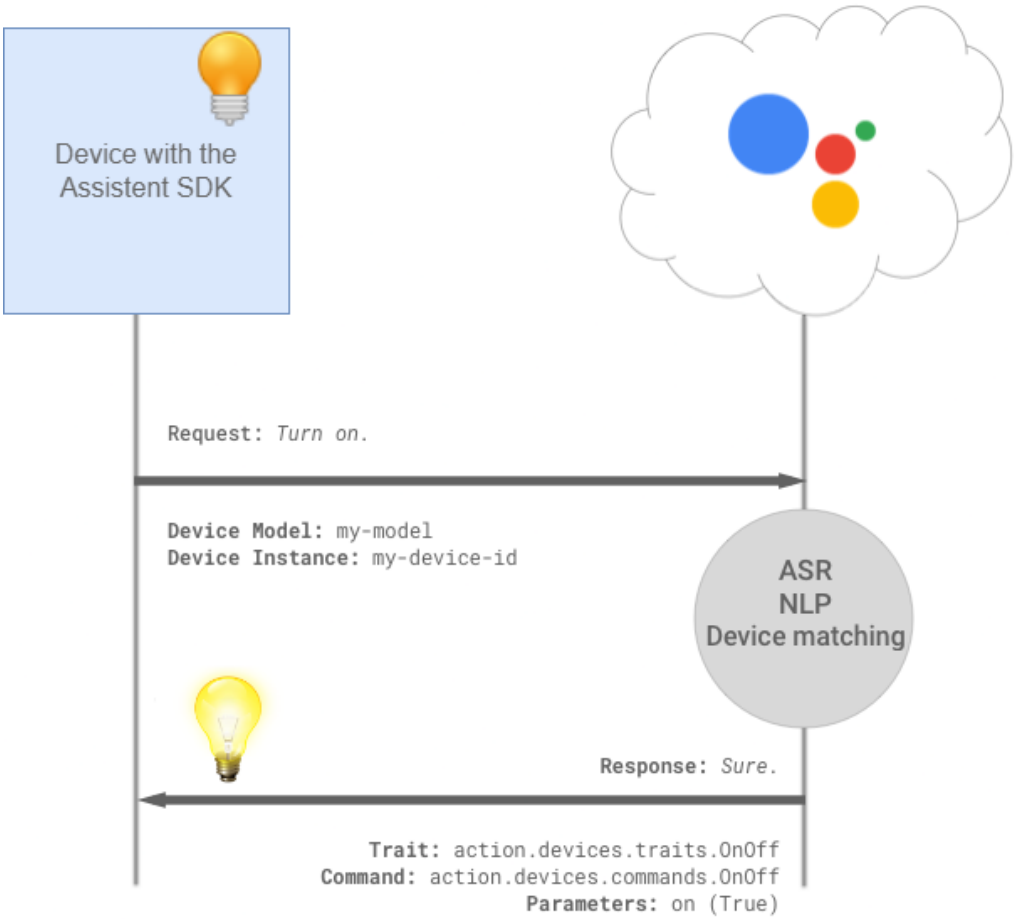

Figure 2.17: Communication of the Google Assistant to turn on a light

Source: Based on:
`https://developers.google.com/assistant/sdk/device-actions-overview/`, adapted by the author

Google Assistant SDK is under an Apache-2.0 License.

The GitHub Repository for Google Assistant SDK can be found at: `https://github.com/googlesamples/assistant-sdk-python`

2.3 Technology Decision

Due to the amount of diverse audio recordings, the proprietary STT engines and assistants perform well. The proprietary STT engines and assistants are excluded from the choice of technology for the exhibit because the transmission of the audio files to external servers cannot be reconciled with data protection directive. Further reasons for exclusion are the reliance on an internet connection and the costs for API calls.

In the following section, the open-source technology decision is visualised in the table 2.1. The evaluation of the technologies is based on a scale of points. The point score ranges from 0 to 6 points, whereas 0 - means it does not apply, and 6 - means it does fully apply to this technology.

Initially, Deepspeech from Mozilla was considered, but the German model is not yet ready for usage (Mycroft AI Inc. 2021d). Consequently, the performance of Deepspeech was not convincing and training resources would be necessary in order to increase the performance. For this reason, other STT engines were researched.
The rating of the STT reveals a score of 33 points for Vosk, followed by CMUSphinx and Coqui. Vosk provides good documentation for installation and different models. The documentation, as well as the models, are continuously updated. With the provided scripts from the documentation of Vosk, it is possible to transcribe the audio in real-time. Regarding adaptability, all mentioned STT technologies require a certain amount of data, transcription and technical resources. Another reason in favour of Vosk is: that it can, or is, already integrated into some speech assistants such as Mycroft and Rhasspy.

	DeepSpeech	**Flashlight**	**Kaldi**	**Coqui**	**Sphinx**	**Vosk**
German model	6	6	6	6	6	6
WER	4 [37]	4 [38]	5 [39]	4 [40]	5 [41]	6 [42]
Documentation	5	3	3	4	5	5
Speed	4	5	5	5	5	6
Adaptability	3	4	3	4	4	4
Offline-capable	6	6	6	6	6	6
Sum	**28**	**28**	**28**	**29**	**31**	**33**

	Mycroft	**Jasper**	**Rhasspy**
German model	6	6	6
Documentation	6	4	5
Speed	5	5	5
Skill maintenance	5	4	5
Offline-capable	0	4	6
Sum	**22**	**23**	**27**

Table 2.1: Technology decision matrix of the open-source tools.

The rating of the speech assistants yields the score of 27 points for Rhasspy followed by Jasper with 23 points and 22 points in the case of Mycroft. Mycroft got excluded due to the incapability to run offline without further ado. Additionally, the creator of Rhasspy mentioned that Rhasspy is the user-friendliest speech assistant for users with privacy concerns (Hansen 2022f). Due to the documentation and the userfriendly UI, Rhasspy was chosen as a possible speech assistant for the exhibit.

[37] https://github.com/AASHISHAG/deepspeech-german
[38] https://goofy.zamia.org/zamia-speech/asr-models/
[39] https://github.com/german-asr/kaldi-german
[40] https://coqui.ai/german/AASHISHAG/v0.9.0
[41] https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/German/
[42] https://alphacephei.com/vosk/models

3 Model Adaption

Since this exhibit is going to be a permanent part of the exhibition at the Inatura[1], it is beneficial to the curator that the exhibit can be modified. Due to the fact that not all questions can be processed after the initial installation, modifications, such as additions of excluded questions or changes in vocabulary, are desirable. The focus of this chapter is on the customisation of the STT engine models. The first option is to enhance the model with new words. The second option is to remove words from the model. (Alpha Cephei Inc. 2016b)
For reproducibility purposes, the used hardware and software are disclosed. A conclusion of the process is also given.

3.1 Hardware

The following hardware was used for this section:

- Microsoft Windows 10 Pro, x64-based PC
- Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz, 1190 Mhz, 4 Core(s), 8 Logical Processor(s)
- Installed Physical RAM 16,0 GB
- Intel(R) UHD Graphics

3.2 Software Prerequisite

In this section, the software requirements for the model adaption are introduced. In order to adapt the model of Vosk, some software needs to be installed before starting.

3.2.1 Vosk

Vosk is one of the previously presented open-source STT engines. One of the reasons that Vosk was selected was due to the uncomplicated documentation. Vosk needs to be installed before it can be used. It can be installed via pip3 (Alpha Cephei Inc. 2016a). Another reason in favour of Vosk was that there is an explicit German model offered for model adaption (Alpha Cephei Inc. 2016b). The large model was used in this chapter, although the loading time on the mentioned system and the high memory consumption (Alpha Cephei Inc. 2016c).

[1] https://www.inatura.at/

3.2.2 SRILM

To enhance the vocabulary of a Vosk model, SRILM should be pre-installed. SRILM is a language modelling toolkit, next to OpenGRM[2], Kenlm[3] and MITLM[4]. Only SRILM and OpenGRM support the essential characteristics. These are interpolation, Witten-Bell discounting, LM pruning and training with a dataset size of one Tb. SRILM can be used for free with a "Research Community License" for non-profit organizations and projects, without independent financial support. Otherwise, a commercial license is required. (SRI International SRILM 2021) (Alpha Cephei Inc. 2016b)

3.2.3 Kaldi

Kaldi was used to perform a RNNLM modification of the Vosk model. One advantage is that Kaldi provides the best accuracy. The disadvantage is that it is slow and memory intense. However, Kaldis RNNLM is capable of adding new words to an existing RNNLM. Kaldis RNNLM needed to be installed on the server. (Alpha Cephei Inc. 2016b)

3.2.4 Phonetisaurus

The Vosk model adaption section suggests Phonetisaurus for creating phoneme representations of the new words. There are several options to get a phoneme representation for the dictionary. The first option is that it is extracted from an existing dictionary. The second option is that a library creates it. Phonetisaurus[5] and G2P[6] are representants of potential libraries. For this reason, Phonetisaurus was installed on the system. Phonetisaurus was already integrated into the models and was used for prediction. (Alpha Cephei Inc. 2016b)

3.2.5 Docker for Windows

To enhance the model of Vosk, a Linux environment is recommened (Alpha Cephei Inc. 2016b). For this reason, Docker for Windows was used for virtualisation. Docker Desktop 4.5.1 (74721) is the most recent available version and was used in this scenario. In addition, Docker was connected with the WSL 2. Therefore, it was possible to adjust the memory, the CPU and the size of the swap in a separate file.

[2] https://www.opengrm.org/twiki/bin/view/GRM/WebHome
[3] https://github.com/kpu/kenlm
[4] https://github.com/mitlm/mitlm
[5] https://pypi.org/project/phonetisaurus/
[6] https://github.com/cmusphinx/g2p-seq2seq

3.3 Setup

Firstly, the Docker Image was downloaded. Subsequently, a container was created using an Ubuntu 20.04 image. This container was mapped to port 2200:22 Transmission Control Protocol (TCP). After connecting to the container, the Ubunutu was updated. In addition, tools such as Automake, sox[7], gfortran[8], libtoolize[9] and some further requirements for Kaldi were installed which took approximately an hour. The next step was the installation of Kaldi and corresponding packages. Onwards SRILM got installed. This installation was time-consuming in comparison to the rest of the installations. After consenting SRILM license, it was downloaded and installed on the docker container. Lastly, Phonetisaurus was installed via pip3 package manager.
To begin with the model modification, an adaptable model was required. This adaptable model was downloaded from the Vosk website[10] with wget[11]. This model contained an RNNLM. (Alpha Cephei Inc. 2016c)

3.4 Conclusion

In theory, it is possible to adapt an RNNLM by adding a text file, followed by a bash script for compiling the graph, as stated in the guide.
To summarise this chapter, the aim was to firstly enhance the model. The extension of the model failed. The only successful model adaption was achieved by removing words from the dictionary. For this reason, the model was censored by removing inappropriate words.
The extension failed because of a lack of information. The chapter Language Model Adaption shared basic information and was difficult to understand. However, the documentation was updated after these attempts. Hardware requirements and additional instructions were supplemented. Precise instructions and data preparations are still missing. Due to the hardware prerequisites and lack of time, further attempts were postponed.

[7] http://sox.sourceforge.net/
[8] https://gcc.gnu.org/wiki/GFortran
[9] http://manpages.ubuntu.com/manpages/trusty/man1/libtoolize.1.html
[10] https://alphacephei.com/vosk/lm
[11] https://www.gnu.org/software/wget/

4 Development

The core parts of the implementation of the exhibit will be discussed in this chapter. The core parts consist of the STT engine and the matching strategy of the transcript to the path to video answer. Furthermore, considered solution proposals and solutions are introduced. Subsequently, the implemented solutions and their technologies are presented and justified.

4.1 Solution Proposals

There are two possible solutions for the realisation of the exhibit. The first one is a speech assistant. The second is an individual implementation, where the mapping and matching from question to answer was inspired by the previously mentioned speech assistants.

4.1.1 Solution 1 - Speech Assistant

This solution utilises a speech assistant to handle speech input and process the transcribed output of the STT engine. At first, Mycroft was suggested as the used speech assistant. Following the setup and testing, it was apparent that Mycroft can not be run entirely offline, except if the backend is hosted on a server (Mycroft Selene GitHub Repository 2022). For this reason, Mycroft was excluded. After further research, Rhasspy was envisaged for usage. Rhasspy can run entirely offline. Additionally, it can be configured to use different STT engines. With Rhasspy, it is possible to use various questions to trigger the same answer. In other words, questions with a different text but the same answer could be cumulated into one skill, which improves the manageability. (Hansen 2022e) Furthermore, Rhasspy provides a userfriendly UI for skill maintenance.

4.1.2 Solution 2 - Individual Implementation

This solution was individually implemented and can be used with different STT engines. For the case of the exhibit, Vosk was chosen as STT. For this reason, the implementation will use Vosk to transcribe the input questions. This individual solution provides two different strategies to match an input question to a known question catalogue and therefore, to the path of the answer video. The two matching strategies which can be selected are called "Text Matching" and "Text and Keyword Matching". Both strategies have in common that they access an JSON file with the known questions which provide a mapping of the path to the answer video. This JSON file represents all questions which can be answered by the system. It also contains keywords and synonyms for each question in the case of word deviations.

The difference between the Text and Keyword Matching and the Text Matching algorithm lies in the way they compare the input question with the known questions of the catalogue to find the correct answer.

The Text Matching algorithm compares the input question to each known question in the JSON file using Jaro Similarity[1]. The Jaro Similarity checks for the similarity of two strings. This algorithm returns a floating value between zero and one, whereas zero is a complete mismatch and one an exact match (T. James 2021). The Text Matching algorithm uses the Jaro Similarity to compare the input question to a known question. If the comparison results in a value of at least 0.75, this question is used to match the corresponding answer. If the value is below 0.75, the response will be a random alternative answer. The text matching strategy approach attempts to match questions with an exact, or at least a close wording to the known questions. In the case of a question where the best match is below 75 %, the text deviates more than 25 % and it is presumed that there is no known question which matches this input question. Therefore, the answer is unknown. As a result, a random answer will be given. This approach can be adapted to use the keyword comparison as well.

The Text and Keyword Matching algorithm compares all known questions of the JSON file with the input question using Damerau-Levenshtein Distance[2]. The comparison considers the number of character modifications from the text of the input question and the known questions (T. James 2021). The questions with the least number of character modifications are saved for later comparisons with the keywords. If the least number of changes is lower than 5, it is assumed that this question is the input question. Afterwards, the corresponding answer is given. The keywords are used if the number of modifications is in an interval of 5-20. The keywords should improve the confidence of the question to match the input question. If the number of changes is above the value of 20, a random alternative answer is given. This text and keyword approach attempts to match more input questions to context-related questions, rather than give an alternative answer right away when the context becomes lost. For example, it allows the matching of an input question with a specific context regarding the fear of a scythe to a similar known question regarding fear in general, instead of giving an alternative random answer. Therefore, the user has an impression of being slightly understood.

Those approaches were developed to reduce the problem of the exact wording of the question in order to match them to a known question. This procress of matching an input to an action is called intent parsing in the domain of speech assistants. It provides the possibility of slight deviations in the syntax or the wording of the questions and errors of the transcription of the STT engine. Additionally, these approaches allow the usage of some dialect questions as input in contrast to the speech assistants.

[1] https://rosettacode.org/wiki/Jaro_similarity
[2] https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

4.2 Justification of the Solutions

There are several reasons why Solution 1 was excluded. It was planned that a video would be displayed when the system is idle. A further video would be played during the question matching process and another during the answer. It is possible to implement these video requirements, but an individual solution is associated with less effort. Another argument in favour of individual solutions was the maintenance of the video configuration.
The UI of Rhasspy is an advantage, in regards to managing the question repertoire. For the functionality of a new question, the drag and drop interface of Node-RED was used. In consideration of the number of questions which should be understood, the Node-RED canvas can become complex to maintain.
A further desired functionality was the push-to-talk recording. The push to talk can be implemented with a Python script. To find out which input questions are asked by the users, the audio recordings should be logged. The saving of these files can also be implemented in the push-to-talk script.
It is possible to configure Rhasspy so that it does not need exact wording to match an input question to skill correctly. (Hansen 2022b) However, it is still necessary to announce all textual variations to Rhasspy that should be understood. To preserve the clear structure, questions with the same answer can be aggregated to the same skill and then to the correct answer. As a result, the number of textual deviations, for example for one question, can be massive. In particular, if the dialect or synonyms are taken into account. These cases can complicate when matching the corresponding skill and diminishes the conciseness of the question repertoire.
Another reason, against the usage of a speech assistant in combination with the exhibit, was the event of an input question which cannot be matched to a known question of Rhasspy. For the alternative answer strategy, a fallback skill should be implemented. This can be done in Node-RED (Hansen 2021). To respond with a contextual close answers phonetic string matching can be configured, but requires more effort to be tested and customised. (Hansen 2022b)
Solution 1 provides a list of predefined STT engines, in contrast to Solution 2 where the STT engine can be exchanged arbitrary.

The reason in favour of Solutions 2 is the flexibility regarding customisations, such as the desired videos for the idle, question matching and answer process. Additionally, push-to-talk can be realized with a Python script comparable to the script of Solution 1. To save the audio recordings minor changes in the push-to-talk script can be carried out.
A further benefit is that all requested functionalities can be implemented with the same programming language, in this case, Python. The STT engine can be exchanged, but for testing reasons, the STT engine of Vosk was used. The code example of the Vosk API is written in Python, which is a benefit. This example uses an audio file as input and can be adjusted for unit tests of the implementation of the text matching and text and keyword matching algorithm.
Both algorithms use the same JSON file. To add a new question, only the JSON file needs to be adjusted. Supplementary keywords are required for the text and keyword matching approach.
There are two main differences between these two algorithms. One difference lies in the matching procedure. The text matching algorithm tries to match only the questions which are almost

identical to the known questions. For this reason, the text matching approach is not as robust as the text and keyword matching algorithm. The text and keyword matching approach considers not only the known question but also the keywords. This feature enables the text and keyword matching algorithm to match more questions correctly, even though there are textual deviations. However, this is only if the textual deviations are announced in the keywords.
Another difference between both algorithms lies in the strategy for not correctly matching questions. The text matching algorithm follows a straightforward approach. If it cannot successfully match an input question, a random answer will be the response. In contrast to the text matching algorithm, the text and keyword matching algorithm tries to find an answer which has at least some context in common with the input question. If the variation between the input question and the known question is too high, the same approach as in the text matching algorithm will become effective.

As a result of the previous argumentations, Solutions 2 with both algorithms was implemented and tested. The better performing solution, in terms of matching the input question to the corresponding question, will be used for the exhibit.

4.3 Used Technologies

Python was used to develop both solutions. Furthermore, WSL 2 for Windows was used to develop with Python and Visual Studio Code. Two different functions of the phonetic string matching library jellyfish have been used for the matching process. For testing purposes, the example which receives an audio file from the GitHub Repository of the STT engine Vosk was adjusted and integrated. It is possible to exchange the STT engine, because there are no dependencies between the implementations.

4.3.1 Python

Python version 3.8.10 was installed for development. Python was chosen because of the code examples of Python in combination with a Raspberry Pi of Kofler, Kühnast, and Scherbeck (2019), which will be useful for the exhibit. In addition, Vosk provided examples and connections for Python. As a result, the exhibit consists solely of one programming language that eases maintenance. The phonetic matching library Jellyfish is offered via the Python package installer pip. Another advantage of Python is the possibility of cross-platform development. The unit test package of Python was used for unit testing the implementations.

4.3.2 Jellyfish

Both implemented solutions utilise the phonetic matching library Jellyfish version 0.9.0. Solution 2 uses the Jaro Similarity to compare the input question to the known questions. Solution 3 is implemented with the Damerau-Levenshtein Distance. Both used functions of Jellyfish are case sensitive and consider punctuations (T. James 2021). Due to this, these factors were removed before comparing the questions and keywords.

4.3.3 Vosk

Initially, Deepspeech was intended for the exhibit. After testing, it became apparent that Deepspeech currently does not perform as well as expected. This was proven in the documentation of Mycroft. (Mycroft AI Inc. 2021d) During the research for suitable STT engines, more emphasis was focused on Vosk. It is the new STT engine of CMUSphinx (Shmyrev 2019f). As previously mentioned, it is an open-source offline STT engine. There are two different German models available for it. Both models were tested and evaluated for the hypotheses, of whether the open-source offline STT engine performs as well as a proprietary cloud-based STT engine. The models of Vosk were compared to Google's Cloud Speech to Text. For the development and testing of the mapping process, the small model version 0.15 and Vosk 0.3.32 were used. The small model is designed for Raspberry Pi's, which will be used for the exhibit. The STT engine itself can be exchanged.

4.3.4 WSL 2 for Windows

WSL 2 was used for development with Python on a Windows machine. It provides Intellisense, linting and debugging support in combination with Visual Studio Code. Additionally, it allows running a script in a terminal (Wojciakowski, Junker, Patel, Davis, & Coulter 2022).

4.3.5 Hardware

The following hardware was used for this section:

- Microsoft Windows 10 Pro, x64-based PC
- Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz, 1190 Mhz, 4 Core(s), 8 Logical Processor(s)
- Installed Physical RAM 16,0 GB
- Intel(R) UHD Graphics

4.4 Implementation

The intended workflow of the exhibit is displayed in the sequence diagram of figure 4.1. If the system is idle, a video for this state will be shown on the screen. To interact with the exhibit, a visitor uses the push-to-talk functionality of the exhibit. This causes the recording of the question using "arecord". As soon as the button is released, the recording stops. The recorded audio is then transmitted to the STT engines and is transcribed. The transcribed question is used to find the corresponding answer. For this reason, the transcribed question is passed to the matching algorithm to find the corresponding answer video. Depending on the applied matching strategy, either the text matching or the text and keyword matching is used. Regardless of which solution is used, this software part responds with the path to the corresponding answer video. This path is used as an input to the "omxplayer" to playback the answer video, in order to communicate the answer to the question asked by the visitor.

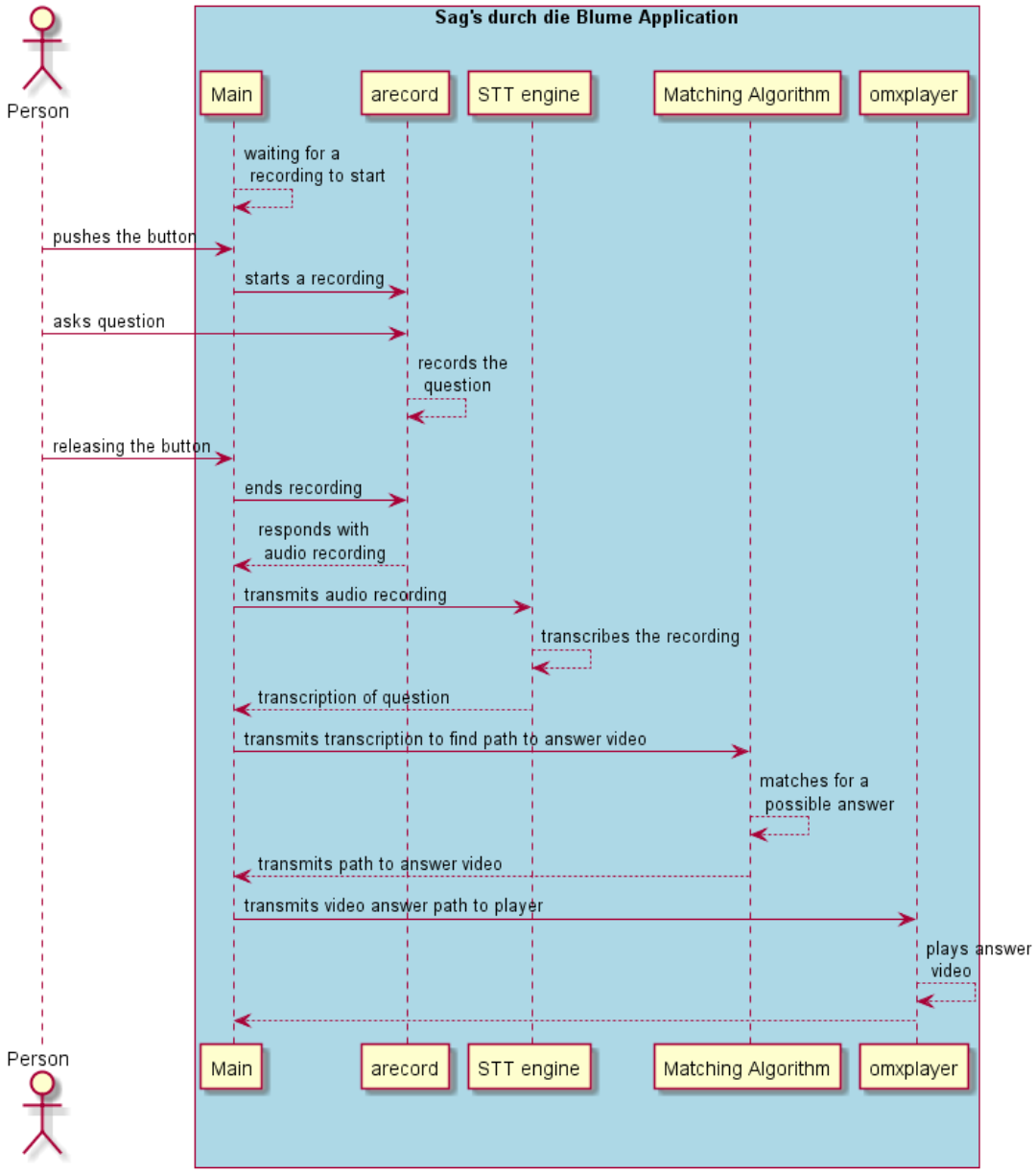

Figure 4.1: Sequence diagram of the exhibit communitation parts

Source: Prepared by the author

Due to automation testing purposes test function of Vosk was designed as a higher-order function. This higher-order function receives the matching algorithm as an input parameter and the path to the audio file. The concept of a higher-order function results in decoupling and therefore allows the exchange of the matching algorithm. Additionally, both matching algorithms follow the same interface, which is the transcribed text of the input question, the path to the JSON file with the known questions and the optional parameter to exchange the phonetic string matching library. Both matching algorithms return the path to the answer video. Furthermore, all code parts are implemented in such a way that all components can be independently tested.

5 Evaluation

Firstly, the quality of the developed matching algorithms was examined. Secondly, an offline open-source STT engine was compared to a proprietary cloud-based STT engine to evaluate the performance. The comparison was based on the robustness of the engines. Additionally, the examination of whether an offline STT engine can be a reliable alternative to a cloud-based solution was discussed.

5.1 Google Cloud Speech to Text vs Vosk

For the comparison between an open-source and a proprietary variant, an adequate representative was chosen. Google Cloud STT was selected as a surrogate of a proprietary engine. This proprietary engine was used as a benchmark to evaluate the performance of a not proprietary service. Vosk represents the open-source solution. For this comparison, two hypotheses were established:

H0: The open-source offline speech to text engine performs as well as a proprietary cloud-based speech to text engine.

H1: The proprietary cloud-based speech to text engine performs better than the open-source offline speech to text engine.

5.1.1 Method

A requirement for a fair comparison is the same preconditions. Therefore, audio recordings were used for testing. Each engine was tested with the same audio recordings. For this reason, audio recordings of 30 test subjects were collected. In total, the number of participants were 15 female and 15 male test subjects. Five of the male and female participants were between the ages of four to eighteen. The female adults were between the ages of 26-42 years old. The male adult subjects were between the ages of 26-40 years old. The lowest age boundary of the test subjects was the age of four. One subject had a speech impediment. Each participant was given seven questions, in which six questions were context-related. One question was context-related, but not within the known questions catalogue. Each subject was instructed to record themselves asking all seven questions in standard German and also in their common dialect. Some of the younger test subjects did not speak dialect at all, which is the reason why the total amount of audio recordings totalled 392.

For the comparison of the previously mentioned engines, their robustness was consulted. Each audio recording was manually transcribed beforehand. Afterwards, each audio was used with both engines. The manual transcript was compared with the transcript of the engine. Each correctly transcribed word was counted. Therefore, the number of correctly understood words was compared to the number of misunderstood words. Subsequently, the percentage of the correctly transcribed words per sentence was calculated.

5.1.2 Results

Table 5.1 presents a first overview of the spoken and correctly transcribed words. The data from big and small models, from Vosk and Google Cloud STT, were compared. In total, 1835 words were included in this test scenario, which is illustrated on the right-hand side of table 5.1. This number varied depending on the test subject group. This was caused by the exact wording not being used and due to the differences in dialect. In total, Google Cloud STT transcribed 86.81 % of the words correctly. The big model of Vosk achieved 82.83 % and the small model 78.91 %.

As can be extracted from table 5.1, 383 words from a total amount of 534 words, spoken by children between the age of four to eighteen, were correctly transcribed by the big model of Vosk. These 383 words are 71.72 % and also included dialect words. The number of correctly transcribed words by the small model is reduced by 35 words, in comparison to the big model. The accuracy of the small model totals 65.17 %. The best performing engine was Google Cloud STT with 419 words and an accuracy of 78.46 %.

Overview 30 Participants

	Vosk Big	Vosk Small	Google STT	Total words
Nr. of recognised words	1520	1448	1593	1835
Nr. of recognised words in %	82,83	78,91	86,81	
Nr. of not recognised words	315	387	242	1835
Nr. of not recognised words in %	17,17	21,09	13,19	

10 Children - 5 Female, 5 Male (aged between 4-18)

	Vosk Big	Vosk Small	Google STT	Total words
Nr. of recognised words	383	348	419	534
Nr. of recognised words in %	71,72	65,17	78,46	
Nr. of not recognised words	151	186	115	534
Nr. of not recognised words in %	28,28	34,83	21,54	

10 Females (aged between 26-42)

	Vosk Big	Vosk Small	Google STT	Total words
Nr. of recognised words	554	541	592	660
Nr. of recognised words in %	83,94	81,97	89,70	
Nr. of not recognised words	106	119	68	660
Nr. of not recognised words in %	16,06	18,03	10,30	

10 Males (aged between 26-40)

	Vosk Big	Vosk Small	Google STT	Total words
Nr. of recognised words	583	559	582	641
Nr. of recognised words in %	90,95	87,21	90,80	
Nr. of not recognised words	58	82	59	641
Nr. of not recognised words in %	9,05	12,79	9,20	

Table 5.1: Table of correctly transcripted words by Vosk and Google Cloud Speech-to-Text

The accuracy increased with the adult test subjects significantly. An amount of 592 from 660 words, from the female test participants, were correctly transcribed by the Google Cloud STT engine. The big model transcribed 554 from 660 words, which are 38 words less than Google. The small model transcribed 541 from 660 words, which was are 51 words less than Google. The percentage of successfully transcripted words of the female subjects was 89.70 % from Google, 83.94 % from the big Vosk model and 81.97 % from the small Vosk model.

Regarding the male test subjects, both Vosk models' performances were similar to Google Cloud STT. The small model transcribed 559 from 641 word correctly, which was 23 words less than Google, as can be extracted from table 5.1. The percentage of successfully transcripted words from the male subjects was 90.80 % from Google, 90.95 % from the big Vosk model and 87.21 % from the small Vosk model.

Percentual Arithmetic Mean of all 30 Participants

	Vosk Big	Vosk Small	Google STT
Mean of all understood words	81,45	77,98	86,01
Children (age between 4-18)	71,61	66,77	79,51
Children speaking dialect	58,93	50,54	69,09
Children speaking standard german	80,75	77,27	86,62
Females (age between 26-42)	82,78	80,84	88,78
Females speaking dialect	71,02	69,44	80,80
Females speaking standard german	94,54	92,24	96,76
Males (age between 26-40)	89,60	86,33	89,74
Males speaking dialect	80,74	75,19	80,40
Males speaking standard german	98,46	97,46	99,43

Table 5.2: Table of the percentual arithmetic means per group

In the table 5.2, the arithmetic mean of all participants is presented. Overall, Google transcribed on average 86.01 % of the spoken words correctly in this user testing, whereas the big Vosk model transcribed 81.45 % and the small Vosk model transcribed 77.98 %. These averages also took dialect into account. The big Vosk model performed 21.82 % better, when a child was speaking standard German. The mean of the small Vosk model was reduced by 26.73 %, when a child was speaking dialect instead of standard German. The difference by Google on the children's data was 17.53 % from dialect to standard German. All of the data from the female and male speakers scored at least 69.44 %. This percentage was achieved by the small Vosk model with the dialect data of the female participants.

Speaker Deviations in Percentage

	Vosk Big	Vosk Small	Google STT
All participants	13,76	16,79	11,94
Children	18,40	21,51	16,88
Females	7,15	11,91	7,57
Males	4,81	6,64	6,78

Table 5.3: Table of the speaker depending deviation per group

Table 5.3 shows the percentual speaker deviations by engines. The standard deviation of the children's data was 18.40 % by the big model, 21.51 % by the small model and 16.88 % by Google Cloud STT. The standard deviation of the female adult data was 7.15 % by the big model, 11.91 % by the small model and 7.57 % by Google. The smallest deviation was reached in the male adult data. The big model reached 4.81 %, 6.64 % by the small model and 6.78 % by Google Cloud STT.

The Kolmogorov-Smirnov[1] test was used to examine whether the data was normally distributed. The result of the Kolmogorov-Smirnov test was that there was no normal distribution. As a result, the data was further compared using the Wilcoxon[2] test. No statistically significant differences in accuracy could be detected by a Wilcoxon Test between the models. The result of the Wilcoxon test considered the big model from Vosk and Google Cloud STT entailing indications of equality (z value of -11.791). The same applied to the Wilcoxon test, in which Google was compared to the small Vosk model (z value of -13.504). Furthermore, both models of Vosk were compared with each other, whereby equality was also indicated (z value of -11.929).

5.1.3 Interpretation

The results of table 5.2 demonstrates that all three engines performed best when the input was in standard German. The lowest performance in standard German was 92.24 % with the female speaker data, taken from the small Vosk model. The accuracy levels shown in table 5.3 indicate that the quality of a test subject's speech affects transcription.

Since the Wilcoxon test did not confirm significant differences between the big model of Vosk and Google Cloud STT, the hypothesis H0 was accepted. Thus, the assumption is made that an offline open-source STT engine performs just as well as a proprietary cloud-based STT engine.

The Wilcoxon result of the comparison between the small Vosk model and Google also did not confirm significant differences. The same applies to the comparison between the big model and the small model of Vosk.

5.2 Text Matching and Text and Keyword Matching Implementation

The basis of both strategies was the selected STT engine. In the best case, each word is correctly interpreted by the engine and matches exactly a known question. But this depends on the accuracy of the engine and the speech of the user. Some words can be misinterpreted, thus the worst case is that a completely incorrect word or a similar word will be transcribed by the engine. The goal of both matching strategies is to improve the matching quote from an input question to a valid question, in the known question catalogue.

[1] https://de.wikipedia.org/wiki/Kolmogorow-Smirnow-Test
[2] https://de.wikipedia.org/wiki/Wilcoxon-Vorzeichen-Rang-Test

In order to identify possible matching questions, the input question should be exact or close to the correct question. To mitigate the problem of an exact matching question, these strategies were developed and implemented. Both strategies can weaken this problem to a certain degree. The concrete flow of the text matching approach can be seen in figure 5.1. This algorithm matches a question with a compliance of at least 75 %. If the threshold of 75 % is reduced, this algorithm will match more questions incorrectly without considering the context of the input question.

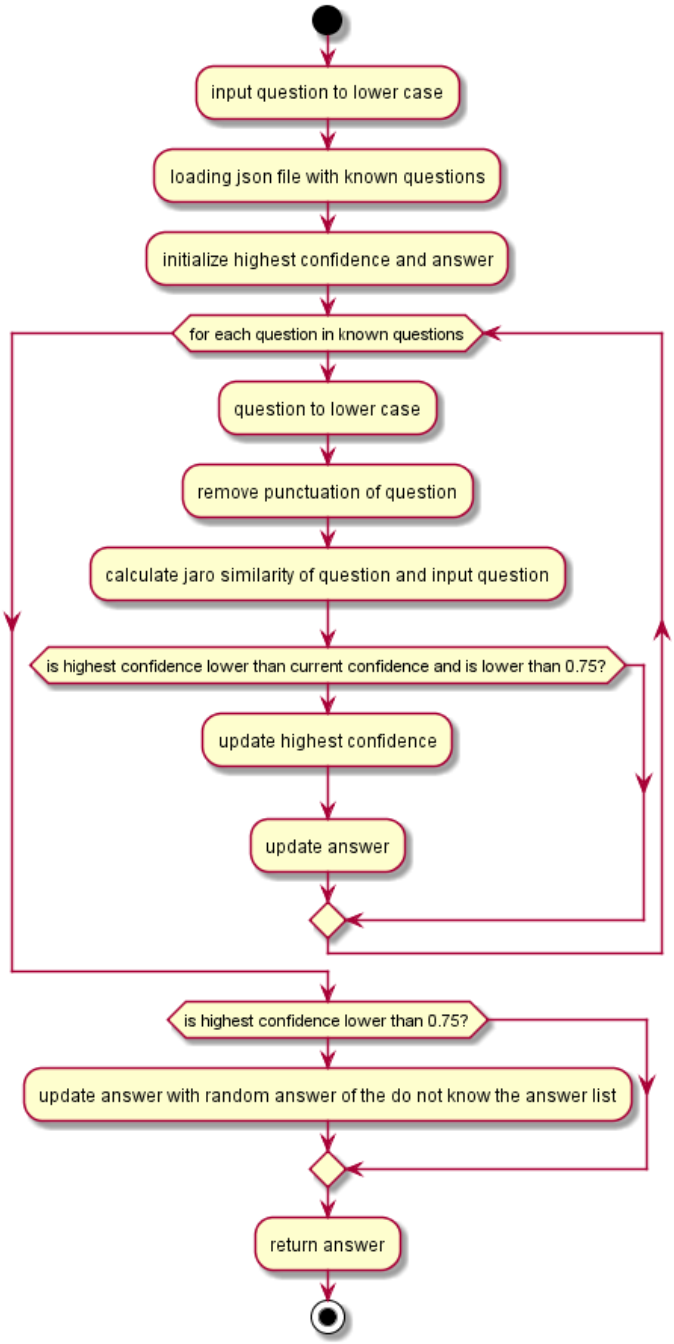

Figure 5.1: Flow diagram of the text matching algorithm.

Source: Prepared by the author

In contrast to the threshold of the text matching algorithm, the thresholds of text and keyword matching are more flexible. It is illustrated in figure 5.2, which assumes that the worst case of similar wording is more likely to happen. For this reason, the number of character modifications, such as deletions or insertions, were taken into account for comparison. To achieve more success when synonyms are used instead of the originated words, the keywords are considered. This solution allows the matching of input questions to known questions with similar wording and thus assumes context-related meaning. As a result, more context-related questions will be matched instead of giving an alternative answer.

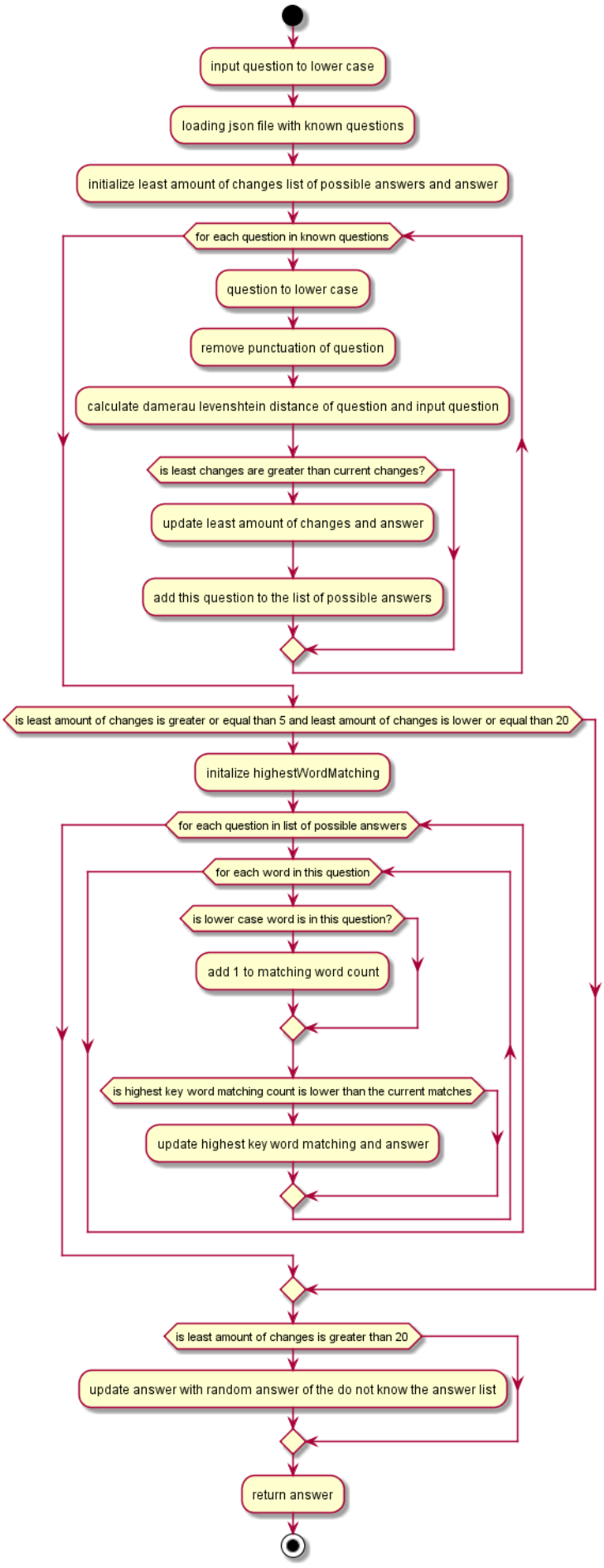

Figure 5.2: Flow diagram of the text and keyword matching algorithm.

Source: Prepared by the author

5.2.1 Method

The matching implementations were tested with the unit test library of Python. Each implementation had two files of unit tests. The first test involved questions and recordings of the STT user-testing. The second test included the remaining questions of the JSON file with all known questions. Therefore, all questions from the known questions catalogue were covered at least once. The first test included 392 test cases for each implementation. 182 from these 392 test cases were in dialect and the rest were in standard German. The second involved 232 test cases with standard german recordings for each implementation. These tests, in combination with the report of the unit test library, were used to evaluate the algorithms. As mentioned in chapter 4, these tests were automated by feeding the audio recordings to the STT engine of Vosk. The audio was transcribed using the small Vosk model. The output of the engine is the text of the recording. This text is the input to the function, which uses the text and keyword matching or the text matching algorithm. Both algorithms return the path to the answer video.

5.2.2 Results

In the first test, the user-testing recordings were used to test the text matching algorithm that matched 270 from 392 recordings correctly to the corresponding questions and answers. This had a success rate of 68.88 %. 111 successfully matched questions were dialect recordings (60.99 %). Four of the 122 incorrectly matched questions were not matched to the alternative answers. However, three of these four questions were matched to context-related questions. The highest rate of correctly matched questions was 94.64 % and the lowest rate was 0 %, as displayed in the figure 5.3. The reason for the question with the lowest success rate of 0 % was because it deviated textually far from the original question, within the known question catalogue. On average 60.99 % of the questions were correctly matched. The time taken to process all 392 tests was 1159.475 seconds.

The test with the text and keyword algorithm, in the user-testing data, shows a successful matching rate of 329 from 392 (83.92 %). Of these 329 correctly matched questions, 136 were spoken in dialect (74.73 %). As can be extracted from figure 5.3, the lowest success rate of a question was 55.36 %.
The highest score of a question has an accuracy of 96.43 %. The quantity of incorrectly matched questions is 63. Four of the 63 incorrectly matched questions were matched to the alternative answers. 38 of the 63 results were matched to context-related answers. 16 incorrectly matched answers were neither context-close nor alternative answers. On average 83.76 % of the questions were correctly matched. The time taken to run all 392 tests was 1309,521 seconds.

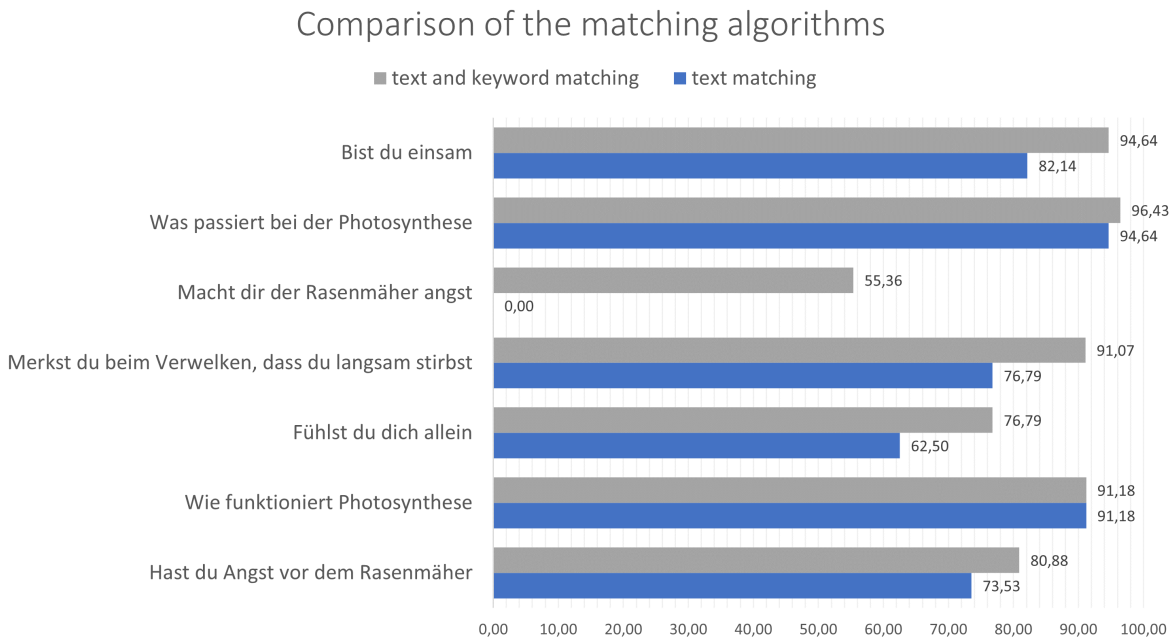

Figure 5.3: Percentual comparison of both matching algorithms.

Source: Prepared by the author

Each of the presented algorithms were tested with the remaining 232 questions of the JSON file containing the known questions. Both the text and keyword matching and text matching algorithms matched all questions correctly without any error. The only difference was in the time it took to complete these tests. The text matching variant needed 609.602 seconds, whereas the text and keywords variant needed 741.097 seconds.

Figure 5.4 shows the finished result of the exhibit. It is activated via the push-to-talk button. After the question is asked, the button needs to be released. Following this, Vosk as the STT engine transcribes the question into text. This text is used to find the path to the answer video which will be displayed on the screen.

Figure 5.4: A photograph of the finished exhibit at the inatura in Dornbirn.

Source: Prepared by the author

5.2.3 Interpretation

As can be extracted from the previous section, the text and keyword matching algorithm performed better on average when matching the questions. Furthermore, the text and keyword matching algorithm was able to match all questions, including the one with textual deviations with an accuracy of at least 62.50 %. In addition, the text and keyword matching variant had a higher matching score regarding the dialect recordings. Due to the ability to match context-related questions, the text and keyword matching variant complied more with the requirements of the exhibit. One disadvantage of the text and keyword matching algorithm is that more questions, which are not context-related, are matched incorrectly. Also, the time aspect shows that the text matching method is faster than the text and keyword matching method. However, the time aspect is not that important, as there is no noticeable difference when an individual question is asked.

6 Conclusion

This chapter contains a discussion on possible improvements to the algorithms, a reflection of the thesis process and a presentation of possible future extensions for the exhibit.

6.1 Discussion

The continuous learning of the proprietary STT and speech assistants can be compared with the enhancement of the known questions catalogue. Adding new words to the known questions catalogue allows the exhibit to learn and understand more questions than before. The only difference is that the proprietary tools learn new words and enhance the model, whereas the solution for the exhibit learns new questions, which can be recognised and answered. (Microsoft 2021)

Due to the fact that some STT utilise the same models, the WER is also identical. The WER in the case of Coqui or Deepspeech was insufficient for the use case of the exhibit. A potential reason for the insufficient model can be the data on which the model was trained, or due to the context of the exhibit. (A. Agarwal 2022b) and (A. Agarwal 2022a)

The workflow and components of the individually implemented solution are similar to the workflow and components of a speech assistant such as Mycroft, except for the fact that it works entirely offline. An STT engine is used to transcribe the speech and an intent parser is used to identify the intent. In the case of the exhibit, the intent parsing took place using the developed algorithms, whereas Mycroft uses Adapt as default. After the intent parsing, the given command or question is performed or answered. (Ovens 2020)

As there is not much of a significant difference between an offline open-source STT, a proprietary cloud-based STT and the WER of Vosk, there would currently be no difference if a cloud-based STT engine is used (Alpha Cephei Inc. 2016c). If the offline models do not improve in comparison to the cloud-based STT engines and models, then it could be advantageous to use a cloud-based solution. (Microsoft 2021) and (Marr 2021)
The issue of data protection and privacy is still present if a cloud-based solution is used for an application, such as with an exhibition. It is not realistic to ask each visitor for consent, so that the data can be transmitted to external servers.

This offline speech assistant-like implementation could be used as an interactive guide for exhibitions in museums. It could be integrated into a smartphone-like device, which is handed out to the visitors. The visitors would walk through the exhibition and could ask questions about the

exhibits or exhibition. Depending on the exhibition and its necessary vocabulary, an offline STT could be unsatisfying. If the vocabulary is specific, an offline STT engine might not perform as well as a cloud-based STT, due to continuous learning. (Marr 2021) The exception is if a model could be developed especially for this use case.

As mentioned in the chapter related work, Rhasspy provides libraries for intent recognition, such as Fsticuffs[1] and Fuzzywuzzy[2]. The Fsticuffs library performs excellent recognition with a large number of sentences. The approach of Fsticuffs is comparable to the text and keyword matching algorithm, which performs better with a large amount of known questions and keywords.
The Fuzzywuzzy approach performs best if the number of sentences is small. This strategy is similar to the method of the text matching algorithm. (Hansen 2022b)

After some adaptation, the developed algorithms could be used for intent recognition in speech assistants or as a phonetic search working in the background of a search machine, such as Google. (Cardillo, Clements, & Price 2008)

Another sector of application of the algorithms, without the STT engines, could be as a code completion[3] tool for IDEs. The written code of the software developer could be compared to the existing keywords and syntax of a programming language using the text and keyword matching algorithm.

6.2 Reflection

The imposed requirements are essentially fulfilled. For the given context, the researched offline STT engine Vosk performs comparable to Google Cloud STT and is integrated into the exhibit in order to transcribe the asked question. To increase the matching rate and to diminish the issue of the need for the exact wording, two algorithms for intent parsing were implemented. The better performing algorithm, which was the text and keyword matching algorithm, was recommended for the exhibit.

The time spent on researching proprietary STT and speech assistants could have been halved. It was still important to research them, in order to learn how proprietary tools work, but the other half of the time could have been used to research algorithms for the string comparison.

The decision of choosing Vosk as the STT engine for the exhibit turned out to be a good choice, due to the results of the Wilcoxon test. As the results showed not much of a significant difference, this only reinforces the decision.
The only negative aspect of this STT engine was the documentation section, concerning the model adaption.

[1] https://rhasspy.readthedocs.io/en/latest/intent-recognition/#fsticuffs
[2] https://rhasspy.readthedocs.io/en/latest/intent-recognition/#fuzzywuzzy
[3] https://en.wikipedia.org/wiki/Intelligent_code_completion#: :text=Intelligent%20code%20completion%20is%20a,typos%20and

For the test of the algorithms, the small model of Vosk was applied. The accuracy would probably have improved, if the big model would have been selected. The main reason in favour of using the small model of Vosk for tests was that there is no significant difference between the big and the small model. However, due to the loading time and memory consumption of the big model, it was not selected for the tests.
Even though it is possible to change the vocabulary of the big model by removing words, the loading time of the model is not an advantage here, in contrast to the memory consumption. Furthermore, the accuracy of the big model remains the same, despite the deletion of possible misguiding words.

Due to the fact that Python connections were provided by Vosk, the decision to use Python for the implementation of the algorithms turned out to be a good choice. This is because the codebase consists of only one programming language, which eases the maintenance and adaptations.

Regarding the string comparison, it may have been a better approach to use a function of Phonetics[4], such as "metaphone", for each string and to use the resulting phonetic key for the comparison, such as using the Jaro Similarity of Jellyfish.

The algorithms used a known questions catalogue of 484 questions. This amount of questions is the foundation of the performance of the algorithms. In retrospect, it would have been better to test the algorithms with the number of questions in which the exhibit would be going into operation.

If the number of the known questions catalogue doubles, it could be possible that the performance of a speech assistant such as Rhasspy could have been a better-suited choice than the individual implementations.

The software architecture could have been designed to receive more dependencies as parameters, such as the JSON file, or the comparison function of another library, such as Jellyfish. But due to the fact that the algorithm contains knowledge of the structure of the JSON file, and because these algorithms are developed to fit this explicit case, the decision of refactoring was postponed. Another reason against the refactoring was the lack of time.

[4]https://pypi.org/project/phonetics/#usage

6.3 Outlook

In regards to the algorithms, the phonetic matching library Jellyfish was applied. This library could have been exchanged and tested, in order to examine the differences in accuracy or performance. Another change could have been to use Soundex[5], Metaphones[6] or the Match Rating Approach of Jellyfish[7] for comparison.

Regarding the text matching algorithm, some improvements would be to consider the keywords. Potentially, this could be matched more accurately. After changing the consideration in the text matching algorithm, both algorithms need to be tested and compared again. To improve the validity of the algorithm tests, the amount of data could be increased.

To maintain the quality of the matching algorithms, further tests should be written. Also, more tests with a focus on dialect recordings would be in favour of quality assurance.

Further optimisation options could be tested and used with the exhibit. To increase the rate of correctly matched dialect questions, the transcriptions of dialect words could be added as keywords.
Another approach would be to investigate whether a model created specifically for this delimited context works better than a proprietary STT engine. Resources such as sufficient audio recordings and better hardware are necessary for this task.

To adapt the known questions catalogue of the exhibit, knowledge of JSON is necessary. A potential upgrade would be to create a User Interface (UI), so to improve this situation. With UI, errors could be avoided when adding new questions to the JSON file.

A potential improvement for the exhibit could also be to display three or five similar questions relating to the input questions on the screen. In this case, a child could select one of these suggested questions. These question proposals could reduce the frustration of the user, especially if the question is not correctly transcribed, due to quality issues of the audio recording or if a user has a speech impediment.

Further user-testing could be carried out, in order to check whether the known questions are the common ones that users are asking. To examine if the known question catalogue covers asked questions, a Wizard-Of-Oz-Experiment[8] could be carried out. Another test could be to put the exhibit under real test conditions within inatura and to record the questions asked, so that these could be examined at a later date.

[5] https://pypi.org/project/phonetics/
[6] https://pypi.org/project/phonetics/
[7] https://jamesturk.github.io/jellyfish/functions/
[8] https://de.wikipedia.org/wiki/Wizard-of-Oz-Experiment

List of Figures

List of Tables

List of Source Codes

References

Adobe. (2019, March). *State of Voice Technology for Brands.* Retrieved 2022-04-01, from https://www.slideshare.net/adobe/state-of-voice-technology-for-brands-145863065

Agarwal, A. (2022a, March). *Automatic Speech Recognition (ASR) - DeepSpeech German.* Retrieved 2022-04-01, from https://github.com/AASHISHAG/deepspeech-german (original-date: 2019-07-01T18:25:53Z)

Agarwal, A. (2022b). *Coqui.* Retrieved 2022-06-30, from https://coqui.ai/german/AASHISHAG/v0.9.0

Agarwal, N. (2019). *Advances in Speech Recognition - WWDC19 - Videos.* Retrieved 2022-04-03, from https://developer.apple.com/videos/play/wwdc2019/256/

Alexa AVS Device GitHub Repository. (2022, May). *alexa/avs-device-sdk.* Alexa. Retrieved 2022-05-29, from https://github.com/alexa/avs-device-sdk (original-date: 2017-02-09T18:57:26Z)

Alexa GitHub Repository. (2022, April). *Alexa Skills Kit SDK for Node.js.* Alexa. Retrieved 2022-04-04, from https://github.com/alexa/alexa-skills-kit-sdk-for-nodejs (original-date: 2016-06-24T06:26:05Z)

Alpha Cephei Inc. (2016a). *Vosk Installation.* Retrieved 2022-04-02, from https://alphacephei.com/vosk/install

Alpha Cephei Inc. (2016b). *VOSK language model adaptation.* Retrieved 2022-04-02, from https://alphacephei.com/vosk/lm

Alpha Cephei Inc. (2016c). *VOSK Models.* Retrieved 2022-04-02, from https://alphacephei.com/vosk/models

Alpha Cephei Inc. (2016d). *VOSK Offline Speech Recognition API.* Retrieved 2022-04-02, from https://alphacephei.com/vosk/

Alphacep GitHub Repository. (2022, April). *Vosk Speech Recognition Toolkit.* Alpha Cephei. Retrieved 2022-04-02, from https://github.com/alphacep/vosk-api (original-date: 2019-09-03T17:48:42Z)

Amazon.com Inc. (2010a). *Get Started with the Alexa Skills Kit | Amazon Alexa Developer.* Retrieved 2022-04-01, from https://developer.amazon.com/en-US/alexa/alexa-skills-kit/start.html

Amazon.com Inc. (2010b). *Overview of the Alexa Voice Service (AVS) Device SDK | Alexa Voice Service.* Retrieved 2022-04-03, from https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/overview.html

Amazon.com Inc. (2010c). *Set Up the AVS Device SDK on Ubuntu | Alexa Voice Service.* Retrieved 2022-04-03, from https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/ubuntu.html

Amazon.com Inc. (2010d). *What is the Alexa Voice Service? | Alexa Voice Service.* Retrieved 2022-04-03, from https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/get-started-with-alexa-voice-service.html

Amazon.com Inc. (2022a). *Amazon Customer Service Help.* Retrieved 2022-05-29, from https://www.amazon.com/gp/help/customer/display.html?nodeId=GCC6XV9DX58VW5YW

Amazon.com Inc. (2022b). *Amazon.de: Alexa kennenlernen - Übersicht: Stores.* Retrieved 2022-05-26, from https://www.amazon.de/b?ie=UTF8&node=12775495031

Amazon.com Inc. (2022c). *Build Your Skill | Alexa Skills Kit.* Retrieved 2022-04-04, from https://developer.amazon.com/en-US/docs/alexa/build/build-your-skill-overview.html

Amazon.com Inc. (2022d). *Host a Custom Skill as an AWS Lambda Function | Alexa Skills Kit.* Retrieved 2022-04-04, from https://developer.amazon.com/en-US/docs/alexa/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html

Amazon.com Inc. (2022e). *Learn to add Alexa to a Speaker, Sound bar, or AVR | Alexa Voice Service.* Retrieved 2022-05-29, from https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/get-started-with-alexa-voice-service-speakers.html

Amazon.com Inc. (2022f). *List of Alexa Interfaces and Supported Languages | Alexa Skills Kit.* Retrieved 2022-04-04, from https://developer.amazon.com/en-US/docs/alexa/device-apis/list-of-interfaces.html

Amazon.com Inc. (2022g). *Set Up the AVS Device SDK on Raspberry Pi with a Script | Alexa Voice Service.* Retrieved 2022-04-04, from https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/raspberry-pi-script.html

Amazon.com Inc. (2022h). *Set Up the AVS Device SDK on Ubuntu | Alexa Voice Service.* Retrieved 2022-05-29, from https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/ubuntu.html

Amazon.com Inc. (2022i). *Understand Smart Home Skills | Alexa Skills Kit.* Retrieved 2022-04-04, from https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html

Amazon.com Inc. (2022j). *What is the Alexa Skills Kit? | Alexa Skills Kit.* Retrieved 2022-04-04, from https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html

Apple Inc. (2017). *Technical Q&A QA1951: How many calls can I make to the Speech Framework API?* Retrieved 2022-04-03, from https://developer.apple.com/library/archive/qa/qa1951/_index.html

Apple Inc. (2020). *Empower your intents - WWDC20 - Videos.* Retrieved 2022-04-05, from https://developer.apple.com/videos/play/wwdc2020/10073/

Apple Inc. (2022a). *Adding User Interactivity with Siri Shortcuts and the Shortcuts App | Apple Developer Documentation.* Retrieved 2022-05-25, from https://developer.apple.com/documentation/sirikit/adding_user_interactivity_with_siri_shortcuts_and_the_shortcuts_app?changes=latest_minor&language=objc

Apple Inc. (2022b). *Asking Permission to Use Speech Recognition | Apple Developer Docu-*

mentation. Retrieved 2022-05-26, from https://developer.apple.com/documentation/speech/asking_permission_to_use_speech_recognition?language=objc

Apple Inc. (2022c). *Core ML | Apple Developer Documentation.* Retrieved 2022-04-03, from https://developer.apple.com/documentation/coreml?changes=latest_minor&language=objc

Apple Inc. (2022d). *Create ML | Apple Developer Documentation.* Retrieved 2022-04-03, from https://developer.apple.com/documentation/createml

Apple Inc. (2022e). *iOS and iPadOS - Feature Availability.* Retrieved 2022-05-26, from https://www.apple.com/ios/feature-availability/#siri-on-device-speech

Apple Inc. (2022f). *iOS and iPadOS - Feature Availability.* Retrieved 2022-04-05, from https://www.apple.com/ios/feature-availability/#siri

Apple Inc. (2022g). *Resolving and Handling Intents | Apple Developer Documentation.* Retrieved 2022-04-05, from https://developer.apple.com/documentation/sirikit/resolving_and_handling_intents?language=objc

Apple Inc. (2022h). *Siri.* Retrieved 2022-04-05, from https://www.apple.com/siri/

Apple Inc. (2022i). *SiriKit | Apple Developer Documentation.* Retrieved 2022-04-05, from https://developer.apple.com/documentation/sirikit?language=objc

Apple Inc. (2022j). *Speech | Apple Developer Documentation.* Retrieved 2022-05-29, from https://developer.apple.com/documentation/speech?language=objc

Apple Inc. (2022k). *Supported capabilities (iOS) - Developer Account Help.* Retrieved 2022-04-05, from https://help.apple.com/developer-account/#/dev21218dfd6

Apple Inc. (2022l). *Xcode.* Retrieved 2022-04-03, from https://developer.apple.com/xcode/

Apple Machine Learning Research. (2017). *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant.* Retrieved 2022-04-03, from https://machinelearning.apple.com/research/hey-siri

Bridge, K. (2021, June). *Cortana design guidelines - Cortana UWP design and development - Windows apps.* Retrieved 2022-04-05, from https://docs.microsoft.com/en-us/windows/apps/design/input/cortana-design-guidelines

Cardillo, P. S., Clements, M. A., & Price, W. E. (2008, July). Mark A. Clements, Lilburn, GA (US);. , 10.

Carman, A., Dzieza, J., & Zelenko, M. (2016, June). *The 13 biggest announcements from Apple WWDC 2016.* Retrieved 2022-04-03, from https://www.theverge.com/2016/6/13/11906654/apple-wwdc-2016-news-highlights-recap-imessage-siri

Chan, W., Jaitly, N., Le, Q. V., & Vinyals, O. (2015, August). Listen, Attend and Spell. *arXiv:1508.01211 [cs, stat]*. Retrieved 2022-04-03, from http://arxiv.org/abs/1508.01211 (arXiv: 1508.01211)

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015, June). Attention-Based Models for Speech Recognition. *arXiv:1506.07503 [cs, stat]*. Retrieved 2022-04-03, from http://arxiv.org/abs/1506.07503 (arXiv: 1506.07503)

Coqui. (2022). *Coqui.* Retrieved 2022-04-02, from https://coqui.ai/blog/stt/a-journey-to-10-word-error-rate

Coqui GitHub Repository. (2022, February). *coqui.* Retrieved 2022-04-02, from https://

github.com/coqui-ai/STT

Coqui GmbH. (2021a). *Coqui.* Retrieved 2022-04-02, from https://coqui.ai/about

Coqui GmbH. (2021b). *Home - Coqui STT 1.3.0 documentation.* Retrieved 2022-04-02, from https://stt.readthedocs.io/en/latest/index.html

Coqui GmbH. (2021c). *Training: Quickstart - Coqui STT 1.3.0 documentation.* Retrieved 2022-04-02, from https://stt.readthedocs.io/en/latest/TRAINING_INTRO.html

Coqui GmbH. (2022). *Coqui.* Retrieved 2022-04-02, from https://coqui.ai/models

DeepSpeech. (2020a). *DeepSpeech Model — Mozilla DeepSpeech 0.9.3 documentation.* Retrieved 2022-04-01, from https://deepspeech.readthedocs.io/en/r0.9/DeepSpeech.html

DeepSpeech. (2020b). *External scorer scripts — DeepSpeech 0.9.3 documentation.* Retrieved 2022-05-26, from https://deepspeech.readthedocs.io/en/v0.9.3/Scorer.html?highlight=data

DeepSpeech. (2020c). *Training Your Own Model — DeepSpeech 0.9.3 documentation.* Retrieved 2022-05-26, from https://deepspeech.readthedocs.io/en/v0.9.3/TRAINING.html?highlight=data

DeepSpeech. (2020d). *User contributed examples — Mozilla DeepSpeech 0.9.3 documentation.* Retrieved 2022-05-26, from https://deepspeech.readthedocs.io/en/r0.9/Contributed-Examples.html

DeepSpeech. (2020e). *Welcome to DeepSpeech's documentation! — Mozilla DeepSpeech 0.9.3 documentation.* Retrieved 2022-05-26, from https://deepspeech.readthedocs.io/en/r0.9/index.html

Etherington, D. (2014, June). *Amazon Echo Is A $199 Connected Speaker Packing An Always-On Siri-Style Assistant | TechCrunch.* Retrieved 2022-04-01, from https://techcrunch.com/2014/11/06/amazon-echo/?guccounter=1&guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAABine5cpC67EZAvjco5qq58xjLKMWaoKMvp_4PIyxOpSoDjTAuIIHOG5oMuieuFidWJvtTW7tSqmREuEQvkU1PWDLfr3h_LOiQOQdnk09R8_T0-lH1GLxIfWEzImuimI8N5MxjasfVuWhzcaDXTE2ZH4gkXUKTo1xUdS6HE7F0kp

Fingold, J. (2021, December). *Create a bot with the Bot Framework SDK in C#, Java, JavaScript, or Python - Azure Bot Service - Bot Service.* Retrieved 2022-04-05, from https://docs.microsoft.com/en-us/azure/bot-service/bot-service-quickstart-create-bot

Flashlight GitHub Repository. (2022a, March). *flashlight/flashlight.* flashlight. Retrieved 2022-04-01, from https://github.com/flashlight/flashlight (original-date: 2018-12-11T18:28:47Z)

Flashlight GitHub Repository. (2022b). *flashlight/flashlight/app/asr at main · flashlight/flashlight.* Retrieved 2022-04-01, from https://github.com/flashlight/flashlight

Flashlight GitHub Repository. (2022c). *wav2letter/recipes/mls at main · flashlight/wav2letter.* Retrieved 2022-05-24, from https://github.com/flashlight/wav2letter

Foster, K. (2021, October). *The Top Free Speech-to-Text APIs and Open Source Engines.* Retrieved 2022-04-01, from https://www.assemblyai.com/blog/the-top-free-speech-to-text-apis-and-open-source-engines/

Google. (2022a). *Change the language of Google Assistant - Android - Google Nest Help.* Retrieved 2022-04-05, from https://support.google.com/googlenest/answer/7550584?co=GENIE.Platform%3DAndroid&hl=en#zippy=%2Cgoogle-home%2Cgoogle-nest-hub

Google. (2022b). *Control smart home devices with Google Assistant - Google Assistant Help.* Retrieved 2022-04-05, from https://support.google.com/assistant/answer/7314909?hl=en

Google Cloud. (2018a). *All Speech-to-Text code samples | Cloud Speech-to-Text Documentation.* Retrieved 2022-04-03, from https://cloud.google.com/speech-to-text/docs/samples

Google Cloud. (2018b). *googleapis/nodejs-speech: Node.js client for Google Cloud Speech: Speech to text conversion powered by machine learning.* Retrieved 2022-04-03, from https://github.com/googleapis/nodejs-speech

Google Cloud. (2018c). *Improve transcription results with model adaptation | Cloud Speech-to-Text Documentation.* Retrieved 2022-04-03, from https://cloud.google.com/speech-to-text/docs/adaptation-model

Google Cloud. (2018d). *Language support | Cloud Speech-to-Text Documentation.* Retrieved 2022-04-03, from https://cloud.google.com/speech-to-text/docs/languages

Google Cloud. (2018e). *Speech-to-Text: Automatic Speech Recognition.* Retrieved 2022-04-03, from https://cloud.google.com/speech-to-text

Google Cloud. (2018f). *Speech-to-Text basics | Cloud Speech-to-Text Documentation | Google Cloud.* Retrieved 2022-04-03, from https://cloud.google.com/speech-to-text/docs/basics?hl=en#select-model

Google Cloud. (2022). *Pricing | Cloud Speech-to-Text.* Retrieved 2022-04-01, from https://cloud.google.com/speech-to-text/pricing

Google Cloud Repository. (2022, March). *Cloud Speech: Node.js Client.* Google APIs. Retrieved 2022-04-03, from https://github.com/googleapis/nodejs-speech (original-date: 2017-07-26T20:52:39Z)

Google Developers. (2020, October). *Overview | Google Assistant SDK.* Retrieved 2022-04-01, from https://developers.google.com/assistant/sdk/overview

Google Developers. (2022a). *Device Actions Overview | Google Assistant SDK.* Retrieved 2022-04-05, from https://developers.google.com/assistant/sdk/device-actions-overview

Google Developers. (2022b). *Google Assistant.* Retrieved 2022-04-05, from https://developers.google.com/assistant

Google Developers. (2022c, March). *Google Assistant SDK for devices - Python.* Google Samples. Retrieved 2022-04-05, from https://github.com/googlesamples/assistant-sdk-python (original-date: 2017-04-25T22:06:02Z)

Google Developers. (2022d). *Install Hardware (Optional) | Google Assistant SDK.* Retrieved 2022-04-05, from https://developers.google.com/assistant/sdk/guides/service/python/extend/install-hardware

Google Developers. (2022e). *Integrate the Assistant into Your Project (Other Languages) | Google Assistant SDK.* Retrieved 2022-04-05, from https://developers.google.com/

assistant/sdk/guides/service/integrate

Google Developers. (2022f). *Introduction to the Google Assistant Service | Google Assistant SDK.* Retrieved 2022-04-05, from https://developers.google.com/assistant/sdk/guides/service/python

Google Developers. (2022g). *Register the Device Model | Google Assistant SDK.* Retrieved 2022-04-05, from https://developers.google.com/assistant/sdk/guides/service/python/embed/register-device

Google Developers. (2022h). *Run the Sample Code | Google Assistant SDK.* Retrieved 2022-05-26, from https://developers.google.com/assistant/sdk/guides/service/python/embed/run-sample

Grabianowski, E. (2006, November). *How Speech Recognition Works.* Retrieved 2022-04-01, from https://electronics.howstuffworks.com/gadgets/high-tech-gadgets/speech-recognition.htm

Han, K. J., Pan, J., Tadala, V. K. N., Ma, T., & Povey, D. (2020). Multistream CNN for Robust Acoustic Modeling. , 5.

Hansen, M. (2016). *Issues · synesthesiam/rhasspy.* Retrieved 2022-04-04, from https://github.com/synesthesiam/rhasspy

Hansen, M. (2021, May). *Is there a way to setup a 'default' or 'fallback' intent? - Help.* Retrieved 2022-05-06, from https://community.rhasspy.org/t/is-there-a-way-to-setup-a-default-or-fallback-intent/1198?page=2

Hansen, M. (2022a). *Installation - Rhasspy.* Retrieved 2022-04-04, from https://rhasspy.readthedocs.io/en/latest/installation/

Hansen, M. (2022b). *Intent Recognition - Rhasspy.* Retrieved 2022-05-29, from https://rhasspy.readthedocs.io/en/latest/intent-recognition/

Hansen, M. (2022c). *Rhasspy.* Retrieved 2022-05-29, from https://rhasspy.readthedocs.io/en/latest/

Hansen, M. (2022d, April). *Rhasspy Voice Assistant.* Rhasspy. Retrieved 2022-04-04, from https://github.com/rhasspy/rhasspy (original-date: 2020-01-08T20:58:48Z)

Hansen, M. (2022e). *Tutorials - Rhasspy.* Retrieved 2022-04-04, from https://rhasspy.readthedocs.io/en/latest/tutorials/

Hansen, M. (2022f). *Why Rhasspy? - Rhasspy.* Retrieved 2022-05-25, from https://rhasspy.readthedocs.io/en/latest/why-rhasspy/

He, Y., Sainath, T. N., Prabhavalkar, R., McGraw, I., Alvarez, R., Zhao, D., ... Gruenstein, A. (2018, November). Streaming End-to-end Speech Recognition For Mobile Devices. *arXiv:1811.06621 [cs].* Retrieved 2022-04-03, from http://arxiv.org/abs/1811.06621 (arXiv: 1811.06621)

James, S. (2020, August). *Top 10 Open Source Speech Recognition Systems [2022].* Retrieved 2022-04-01, from https://fosspost.org/open-source-speech-recognition/

James, T. (2021). *Functions - jellyfish.* Retrieved 2022-05-03, from https://jamesturk.github.io/jellyfish/functions/

Jasper Client GitHub Repository. (2022, April). *jasper-client.* Jasper Project. Retrieved 2022-04-04, from https://github.com/jasperproject/jasper-client (original-date:

2014-03-30T20:02:44Z)

Kofler, M., Kühnast, C., & Scherbeck, C. (2019). *Raspberry Pi: Das umfassende Handbuch* (6th ed.). Rheinwerk Computing.

Kunst, A. (2021). • *Smart speaker ownership by brand in the United States 2021 | Statista.* Retrieved 2022-04-04, from https://www.statista.com/forecasts/997149/smart-speaker-ownership-by-brand-in-the-us

Laricchia, F. (2022, May). *Global smart speaker market share 2021.* Retrieved 2022-04-05, from https://www.statista.com/statistics/792604/worldwide-smart-speaker-market-share/

Manson, H. (2016). *Speech Recognition API - WWDC16 - Videos.* Retrieved 2022-04-03, from https://developer.apple.com/videos/play/wwdc2016/509/

Marr, B. (2021, July). *Machine Learning In Practice: How Does Amazon's Alexa Really Work?* Retrieved 2022-04-03, from https://bernardmarr.com/machine-learning-in-practice-how-does-amazons-alexa-really-work/

Meta. (2018, December). *Open sourcing wav2letter++, the fastest state-of-the-art speech system, and flashlight, an ML library going native.* Retrieved 2022-04-01, from https://engineering.fb.com/2018/12/21/ai-research/wav2letter/

Meta AI. (2021, April). *Flashlight: Fast and flexible machine learning in C++.* Retrieved 2022-04-01, from https://ai.facebook.com/blog/flashlight-fast-and-flexible-machine-learning-in-c-plus-plus/

Microsoft. (2021, October). *Cortana in Microsoft 365 - Microsoft 365 admin.* Retrieved 2022-04-05, from https://docs.microsoft.com/en-us/microsoft-365/admin/misc/cortana-integration

Microsoft. (2022). *Cognitive Speech Services Pricing | Microsoft Azure.* Retrieved 2022-04-03, from https://azure.microsoft.com/en-us/pricing/details/cognitive-services/speech-services/

Microsoft Azure GitHub Repository. (2022, May). *Sample Repository for the Microsoft Cognitive Services Speech SDK.* Azure Samples. Retrieved 2022-05-29, from https://github.com/Azure-Samples/cognitive-services-speech-sdk (original-date: 2018-04-26T11:28:25Z)

Microsoft BotBuilder GitHub Repository. (2022). *BotBuilder-Samples/Node/demo-RollerSkill at releases/v3-sdk-samples · microsoft/BotBuilder-Samples.* Retrieved 2022-05-29, from https://github.com/microsoft/BotBuilder-Samples

Microsoft GitHub Repository. (2022, January). *Cortana Skills Kit.* Microsoft. Retrieved 2022-04-05, from https://github.com/microsoft/cortana-skills-samples (original-date: 2017-09-14T21:05:00Z)

Milde, B., & Köhn, A. (2018, July). Open Source Automatic Speech Recognition for German. *arXiv:1807.10311 [cs].* Retrieved 2022-04-01, from http://arxiv.org/abs/1807.10311 (arXiv: 1807.10311)

Mlopatka. (2021, April). *DeepSpeech update, grant and playbook.* Retrieved 2022-04-01, from https://discourse.mozilla.org/t/deepspeech-update-grant-and-playbook/78569

Morais, R. (2020, August). *Future of DeepSpeech / STT after recent changes at Mozilla.*

Retrieved 2022-04-01, from https://discourse.mozilla.org/t/future-of-deepspeech-stt-after-recent-changes-at-mozilla/66191

Mozilla. (2019, August). *Mozilla Common Voice.* Retrieved 2022-04-01, from https://commonvoice.mozilla.org/

Muelaner, J. E. (2021, April). *How to Use 'Ok, Google' Offline.* Retrieved 2022-04-05, from https://www.lifewire.com/use-ok-google-offline-4589595 (Section: Lifewire)

Mycroft AI GitHub Repository. (2022, April). *Mycroft.* Mycroft. Retrieved 2022-04-04, from https://github.com/MycroftAI/mycroft-core (original-date: 2016-05-20T14:11:07Z)

Mycroft AI Inc. (2020). *Why use Mycroft AI?* Retrieved 2022-04-04, from https://mycroft-ai.gitbook.io/docs/about-mycroft-ai/why-use-mycroft

Mycroft AI Inc. (2021a). *Get Mycroft.* Retrieved 2022-04-04, from https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/get-mycroft#mark-ii

Mycroft AI Inc. (2021b). *Pairing Your Device.* Retrieved 2022-04-04, from https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/pairing-your-device

Mycroft AI Inc. (2021c). *Picroft.* Retrieved 2022-04-04, from https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/get-mycroft/picroft

Mycroft AI Inc. (2021d). *Speech-To-Text.* Retrieved 2022-04-01, from https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/customizations/stt-engine

Mycroft AI Inc. (2021e). *Your First Skill.* Retrieved 2022-04-04, from https://mycroft-ai.gitbook.io/docs/skill-development/introduction/your-first-skill

Mycroft Selene GitHub Repository. (2022, March). *Selene – Mycroft's Server Backend.* Mycroft. Retrieved 2022-04-04, from https://github.com/MycroftAI/selene-backend (original-date: 2018-08-01T02:03:39Z)

Ortinau, D. (2021, July). *Understanding SiriKit Concepts - Xamarin.* Retrieved 2022-04-05, from https://docs.microsoft.com/en-us/xamarin/ios/platform/sirikit/understanding-sirikit

Ovens, S. (2020, September). *Get started with open source voice assistant software | Opensource.com.* Retrieved 2022-04-04, from https://opensource.com/article/20/6/mycroft

Perez, S. (2017, June). *Microsoft's Dictate uses Cortana's speech recognition to enable dictation in Office.* Retrieved 2022-05-29, from https://social.techcrunch.com/2017/06/20/microsofts-dictate-uses-cortanas-speech-recognition-to-enable-dictation-in-office/

Povey, D. (2011a). *Kaldi: History of the Kaldi project.* Retrieved 2022-05-29, from https://kaldi-asr.org/doc/history.html

Povey, D. (2011b). *Kaldi: Kaldi.* Retrieved 2022-04-01, from https://kaldi-asr.org/doc/

Povey, D. (2011c). *Kaldi: Kaldi tutorial: Getting started (15 minutes).* Retrieved 2022-05-29, from https://kaldi-asr.org/doc/tutorial_setup.html

Povey, D. (2011d). *Kaldi: Kaldi tutorial: Prerequisites.* Retrieved 2022-05-29, from https://kaldi-asr.org/doc/tutorial_prereqs.html

Povey, D. (2022, May). *Kaldi Speech Recognition Toolkit.* Kaldi. Retrieved 2022-05-29, from https://github.com/kaldi-asr/kaldi (original-date: 2015-04-20T17:23:16Z)

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., ... Vesely, K. (2011). The Kaldi Speech Recognition Toolkit. , 4.

Rahmel, H. (2022). *Model and Endpoint Lifecycle of Custom Speech - Speech service - Azure Cognitive Services.* Retrieved 2022-04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-speech-model-and-endpoint-lifecycle

Saha, S., & Marsh, C. (2014a). *Jasper | Documentation.* Retrieved 2022-04-04, from https://jasperproject.github.io/documentation/hardware/

Saha, S., & Marsh, C. (2014b). *Jasper | Documentation.* Retrieved 2022-04-04, from https://jasperproject.github.io/documentation/installation/

Saha, S., & Marsh, C. (2014c). *Jasper | Documentation.* Retrieved 2022-04-04, from https://jasperproject.github.io/documentation/configuration/

Saha, S., & Marsh, C. (2014d). *Jasper | Documentation.* Retrieved 2022-04-04, from https://jasperproject.github.io/documentation/usage/

Saha, S., & Marsh, C. (2014e). *Jasper | Documentation.* Retrieved 2022-04-04, from https://jasperproject.github.io/documentation/modules/

Saha, S., & Marsh, C. (2014f). *Jasper | Documentation.* Retrieved 2022-04-04, from https://jasperproject.github.io/documentation/api/

Schalkwyk, J., Fellow, G., & Team, S. (2019, December). *An All-Neural On-Device Speech Recognizer.* Retrieved 2022-04-03, from http://ai.googleblog.com/2019/03/an-all-neural-on-device-speech.html

Shmyrev, N. (2018, October). *CMU Sphinx - Browse /Acoustic and Language Models at SourceForge.net.* Retrieved 2022-04-03, from https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/

Shmyrev, N. (2019a). *About CMUSphinx.* Retrieved 2022-04-03, from http://cmusphinx.github.io/wiki/about/

Shmyrev, N. (2019b). *Before you start.* Retrieved 2022-04-03, from http://cmusphinx.github.io/wiki/tutorialbeforestart/

Shmyrev, N. (2019c). *Building a language model.* Retrieved 2022-04-03, from http://cmusphinx.github.io/wiki/tutoriallm/

Shmyrev, N. (2019d). *Building an application with PocketSphinx.* Retrieved 2022-04-03, from http://cmusphinx.github.io/wiki/tutorialpocketsphinx/

Shmyrev, N. (2019e). *Building a phonetic dictionary.* Retrieved 2022-04-03, from http://cmusphinx.github.io/wiki/tutorialdict/

Shmyrev, N. (2019f, October). *CMUSphinx Open Source Speech Recognition.* Retrieved 2022-04-02, from http://cmusphinx.github.io/

Shmyrev, N. (2019g). *Large scale language model.* Retrieved 2022-04-02, from http://cmusphinx.github.io/wiki/tutoriallmadvanced/

Shmyrev, N. (2019h). *Overview of the CMUSphinx toolkit.* Retrieved 2022-05-29, from http://cmusphinx.github.io/wiki/tutorialoverview/

Shmyrev, N. (2019i). *Training an acoustic model for CMUSphinx.* Retrieved 2022-04-03, from http://cmusphinx.github.io/wiki/tutorialam/

Shmyrev, N. (2020, August). *Vosk/Kaldi German acoustic model for callcenter and broacast transcription.* Retrieved 2022-04-02, from https://alphacephei.com/nsh/2020/08/09/german.html

Shmyrev, N. (2021, July). *Multistream TDNN and new Vosk model.* Retrieved 2022-04-02, from https://alphacephei.com/nsh/2021/07/16/multstream-tdnn.html

SRI International SRILM. (2021). *STAR Laboratory: SRI Language Modeling Toolkit.* Retrieved 2022-04-05, from http://www.speech.sri.com/projects/srilm/

Standefer, R., & Fingold, J. (2017, December). *Bot Framework SDK for .NET - Bot Service.* Retrieved 2022-04-05, from https://docs.microsoft.com/en-us/previous-versions/azure/bot-service/dotnet/bot-builder-dotnet-overview

Stegner, B. (2018, March). *What Is Google Assistant? How to Use It to Full Potential.* Retrieved 2022-05-26, from https://www.makeuseof.com/tag/what-is-google-assistant/ (Section: Android)

Swarup, P., Maas, R., Garimella, S., Mallidi, S. H., & Hoffmeister, B. (2019, September). Improving ASR Confidence Scores for Alexa Using Acoustic and Hypothesis Embeddings. In *Interspeech 2019* (pp. 2175–2179). ISCA. Retrieved 2022-04-03, from https://www.isca-speech.org/archive/interspeech_2019/swarup19_interspeech.html doi: 10.21437/Interspeech.2019-1241

Thakur, A. (2021, November). *How to listen, delete, and ask Amazon not to save your Alexa voice recordings.* Retrieved 2022-04-04, from https://www.idownloadblog.com/2021/11/12/how-to-listen-delete-stop-use-alexa-voice-recordings/

Tošović, e. O.-P. D. B. (2022, April). *Phonem.* Retrieved 2022-04-01, from https://www-gewi.uni-graz.at/gralis/Linguistikarium/Phonetik/Phonem_Wikipedia.html

Urban, E. (2022a, April). *How to recognize speech - Speech service - Azure Cognitive Services.* Retrieved 2022-05-24, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-recognize-speech

Urban, E. (2022b). *Language support - Speech service - Azure Cognitive Services.* Retrieved 2022-04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support

Urban, E. (2022c). *Prepare data for Custom Speech - Speech service - Azure Cognitive Services.* Retrieved 2022-04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-speech-test-and-train

Urban, E. (2022d). *Speech-to-text overview - Speech service - Azure Cognitive Services.* Retrieved 2022-04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/speech-to-text

Urban, E. (2022e). *Speech-to-text quickstart - Speech service - Azure Cognitive Services.* Retrieved 2022-04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-speech-to-text

Urban, E. (2022f). *Train and deploy a Custom Speech model - Speech service - Azure Cognitive Services.* Retrieved 2022-04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-speech-train-model

Urban, E. (2022g). *What is the Speech service? - Azure Cognitive Services.* Retrieved 2022-

04-03, from https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/overview

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., ... Wölfel, J. (2004, December). Sphinx-4: A flexible open source framework for speech recognition. *Sun Microsystems*.

Wardini. (2022, January). *Voice Search Statistics 2022: Smart Speakers, VA, and Users | SerpWatch.* Retrieved 2022-04-05, from https://serpwatch.io/blog/voice-search-statistics/

wav2letter GitHub Repository. (2022, April). *wav2letter++.* flashlight. Retrieved 2022-04-01, from https://github.com/flashlight/wav2letter (original-date: 2017-11-20T17:39:41Z)

Wojciakowski, M., Junker, A., Patel, M., Davis, J., & Coulter, D. (2022, January). *Python on Windows for beginners.* Retrieved 2022-05-09, from https://docs.microsoft.com/en-us/windows/python/beginners

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., & Stolcke, A. (2018, April). The Microsoft 2017 Conversational Speech Recognition System. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5934–5938. Retrieved 2022-04-03, from http://arxiv.org/abs/1708.06073 (arXiv: 1708.06073) doi: 10.1109/ICASSP.2018.8461870

yourtechdietAdmin. (2021). *7 Best Open Source Voice Assistants - 2020.* Retrieved 2022-04-01, from https://yourtechdiet.com/blogs/open-source-voice-assistants/

Zawideh, C. (2022, March). *Microsoft Teams displays - Microsoft Teams.* Retrieved 2022-04-05, from https://docs.microsoft.com/en-us/microsoftteams/devices/teams-displays

Zeghidour, N., Xu, Q., Liptchinsky, V., Usunier, N., Synnaeve, G., & Collobert, R. (2019, April). Fully Convolutional Speech Recognition. *arXiv:1812.06864 [cs]*. Retrieved 2022-04-01, from http://arxiv.org/abs/1812.06864 (arXiv: 1812.06864)