Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Cybernetics

Master's thesis

# Design and Implementation of an End-to-End Speech Assistant

***Felix Staudigl***

Supervisor: Sascha Schade

May 23, 2019

# Acknowledgements

I want to express the most profound appreciation to Sascha Schade for his academic perspective, his technical advice and overall guidance throughout the whole process of this thesis. Also, I would like to thank Jiří Spilka to be my opponent of my master's thesis. Finally, I received generous support from Martin Hlinovský, who has always been available for questions about this thesis and my studies in general.

# Declaration

In Prague on May 23, 2019 . . . . . . . . . . . . . . . . . . . .

## Citation of this thesis

Staudigl, Felix. *Design and Implementation of an End-to-End Speech Assistant.* Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2019.

# Abstrakt

Efektivní komunikace mezi člověkem a strojem představuje důležitou oblast výzkumu. Nedávné pokroky v oblasti strojového učení, konkrétně v oblasti zpracování přirozeného jazyka, umožňují inteligentním robotům využít hlasové rozhraní, které je pro člověka přirozené a efektivní. Nicméně, většina z dostupných systémů pro inteligetní roboty vyžaduje stabilní připojení k internetu a zpracování probíhá z velké části v cloudu. Tento požadavek však naráží například v aplikacích asistenčních robotů, kde internetové spojení není vždy zajištěno. Cílem této práce je (a) poskytnout přehled současných a dostupných open-source frameworků s důrazem na požadavky hotelového asistečního robota; (b) navrhnout a realizovat aplikaci hlasového asistenta; c) implementovat řešení s jinými systémy.

**Klíčová slova** Strojové učení, Zpracování přirozeného jazyka, Asistent řeči.

# Abstract

Efficient communication between robots and humans represent an important field of research. Recent advances in machine learning, specifically in the field of natural language processing, enable smart robots to benefit from voice interaction interfaces. This interface allows customers to communicate with robots naturally and conveniently over spoken utterances by employing a personal component at the same time.
However, most of the available systems ,for example, require a stable internet connection to work, which interferes with standard requirements of service robots.
This thesis, therefore, aims to (a) provide thorough research of current open-source frameworks with an emphasis on clearly defined objectives of a service robot within a hotel environment; (b) design and implement a speech assistant application; (c) evaluate the developed system by comparing it to other commercial state-of-the-art systems.

**Keywords** Machine Learning, Natural Language Processing, Speech Assistant.

# Contents

# List of Figures

# List of Tables

# Introduction

This thesis aims to implement a speech assistant application which is able to work within strictly defined requirements. Initially, the *Introduction* chapter described the objectives, the motivation and the resulting research questions. The *Theory* chapter covers the key topics to comprehend the developed system and its components. Next, the *Result* chapter is separated into two sections. First of all, thorough research of current machine learning frameworks is presented, followed by the system architecture and the description of its internal components. The *Evaluation* chapter aims to compare the developed system to other assistants by comparing both with the previously defined objectives. Finally, the last section discusses the consequences of the findings and the utilisation of the system.

## Background

Machine Learning is an evolving field which already influences our daily lives[45]. The interaction between machines and human beings is one area in which machine learning has gained significant influence. The ability to converse with machines in a natural way has finally become a reality in the form of intelligent assistants. By 2020, between 30 to 50 per cent of all search requests is expected to be powered by, so-called, speech assistants like Amazon Alexa or Apple Siri[36]. Speech Assistants are incorporating three main technologies of natural language processing, a subfield of machine learning:

1. Speech Recognition
2. Intent Matching
3. Speech Synthesis

The former describes the transformation of a spoken utterance in a sequence of characters. Utterances describe the smallest unit of spoken language, which is separated from each other with clear pauses. The intent matcher uses this sequence to infer the purpose of the original user request. The latter transforms the generated response into intelligible speech. Nevertheless, several other algorithms are used to support computational assistants on their way to become more natural. For instance, Hot-Word Recognition is used to parse a continuous stream of speech to recognise the well known "Hey Siri" which invokes the assistant to process the subsequent request. Despite remarkable advances in this field, the usability of those systems is still unsatisfying[55]. Furthermore, the majority of the currently available speech assistants on the market are utilising a cloud backend to compute responses, which may let the doubt arise that privacy and data protection are violated[11]. In general, these platforms represent a security bottleneck of smart home infrastructures by providing potential attackers with the opportunity to control and monitor all connected devices[42]. With this in regard, speech assistants offer a novel way to convey with machines by introducing new weak spots in the form of security and privacy of sensible data.

## Objectives

Robotise, a German robotics startup, is interested in integrating voice assistant capabilities in their service robots. These robots are currently offering various customer services in hotels. The robots include a large touchscreen, microphones and speakers. The following requirements were set to guarantee compatibility with the company standards and the environment in which the robot operates:

1. Security
2. Information Privacy
3. Scalability
4. Offline
5. User Experience
6. Domain tailored

Security represents besides the privacy of customer data, one of the essential priorities of the company. Therefore, an architecture which ensures the security of the communication and the intellectual property must be guaranteed. Robotise is utilising virtualisation such as containers heavily for their development and deployment processes. Considering this premise, the voice assistant ought to use this technology as a baseline to accomplish the required scalability. The system itself should be able to work entirely offline, based on the inadequate internet coverage of some areas within hotels. The last two requirements express the desire to create a unique customer experience tailored for hotel guests. Hence, the voice assistant must be capable to focus on its specific audience in the form of utilising a smaller vocabulary to generate customised responses. With these requirements in mind, the goal of this thesis is to gain knowledge in the theoretical and practical implications using the latest technologies to compose a voice assistant who can communicate with customers within a particular environment. The goal for the company is to gain insight and guidance regarding which extent they should adopt this internally developed solution or to purchase a third party product.

## Research Question

This thesis investigates the primary research question:

*"Can currently available software frameworks be composed to form a voice assistant which outperforms comparable third-party products regarding requirements of a specific environment"*

## Methodology

In order to investigate the defined research question in a scientific fashion, a research methodology was adopted. Peffers *et al.*[52] propose a *Design Science Research Methodology (DSRM)* for system design which involving the following five activities:

1. Problem identification and motivation
2. Objectives definition
3. Design and development
4. Demonstration
5. Communication

Following these five steps, the *Introduction* chapter identifies the problem and motivates to develop an entirely new domain tailored intelligence. The design and implementation of this system are described in the *Result* section. The *Evaluation* chapter tries to assess the developed system in terms of the outlined objectives. Finally, the *Discussion* chapter debates the outcome of this work.

Chapter 1

# Theory

This chapter covers the important topics to understand the thesis. It follows the processing pipeline of speech assistants which consists of the *Speech Recognition*, *Dialogue System* and *Speech Synthesis.* The last section aims to introduce the basic concepts of virtualisations with a specific focus on containers.

## 1.1 Speech Recognition

Speech is the most natural form of human communication, and consequently, speech recognition is one of the most exciting modern applications regarding machine learning[6]. The objective of speech recognition is parsing a given input audio stream to create a textual representation in the form of a sequence of words and sentences[64]. The upcoming section describes the challenges to process such audio streams and various techniques of such algorithms.

### 1.1.1 Challenges

Major advances in machine learning allow applications to use speech recognition modules simplifying the interaction and consequently improving the user experience[6]. Nevertheless, there are still numerous challenges which influence the accuracy of such systems. In general, noise represents one of the most significant difficulties since it overlaps the actual signal, which results in distortion and lastly misinterpretation. The level of noise strongly depends on the surrounding environment as well as the nature of the applied microphones. The microphone transforms the sound wave into a digital representation. This transformation process can add further artefacts like echos and distortions. Besides, unique properties of the speaker like dialect, sex, age and physical state affect the computed output as well. Conclusively, a lot of parameters are influencing the outcome of speech recognition algorithms, which leads to the fact that even the latest publications still suffering from problems. However, speech to text systems reached a crucial mark which allows the usage within machine learning based assistances[6, 64, 21, 48].

### 1.1.2 Techniques

Modern speech recognition systems are consisting out of three processing stages: *Preprocessing*, *Feature Extraction* and *Classification*. Each of these stages fulfils a crucial task to transfer spoken utterances into text. Figure 1.1 shows the pipeline from the audio input on the left, transformed through the microphone into a digital representation. Next, the pipeline itself processes the raw audio information and last but not least generates a set of characters.

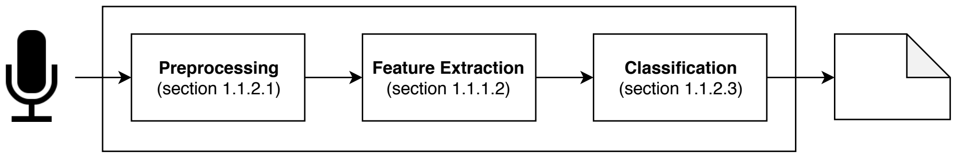


Figure 1.1: Speech Recognition Pipeline

#### 1.1.2.1 Preprocessing

The preprocessing step servers in general as a filter. Interference through surrounding noise leads to insufficient accuracy. Therefore, the preprocessing stage removes all kind of unwanted artefacts based on the zero-crossing rate and the signal energy[28].

#### 1.1.2.2 Feature Extraction

The main goal of the feature extraction stage is to determine a sequence of features in the form of a vector providing a compact representation of the given input signal[43]. Usually, this extraction performs three substeps:

1. The acoustic front-end computes raw features which are describing the envelope of the power spectrum.
2. The second stage calculates static and dynamic features.
3. Converts previous feature vector into more robust and compact representations.

Anusuya et al. [6] mentions various feature extraction algorithms.
The Principal Component Analysis (PCA) represents a non-linear feature extraction method which applies eigenvectors to calculate the required features. In comparison, the cepstral analysis uses static features calculated from the power spectrum of the signal. This approach can be improved by using the

Fourier transformation to determine the power spectrum which is called Mel-Frequency Cepstrum (MFC).
In general, the computed features shall enable the speech recognition module to distinguish diverse yet similar sounding records.

#### 1.1.2.3 Classification

The Classification stage represents the main part of the speech recognition task which learns the relationship between utterances, sentences and words[64]. Three different approaches can be distinguished:

**Acustic Phonetic Approach** The acoustic-phonetic approach assumes that there are finite, distinctive phonetic units in spoken language which are characterised by a set of acoustic properties. A spectral analysis serves as a first step to extract these acoustic properties. On this basis, the input signal can be segmented into stable acoustic regions attached with one or more labels resulting in the so-called phoneme lattice characterisation of the speech. Finally, the attached labels are used to obtain a valid word out of a database which matches the phoneme lattice characterisation[6].

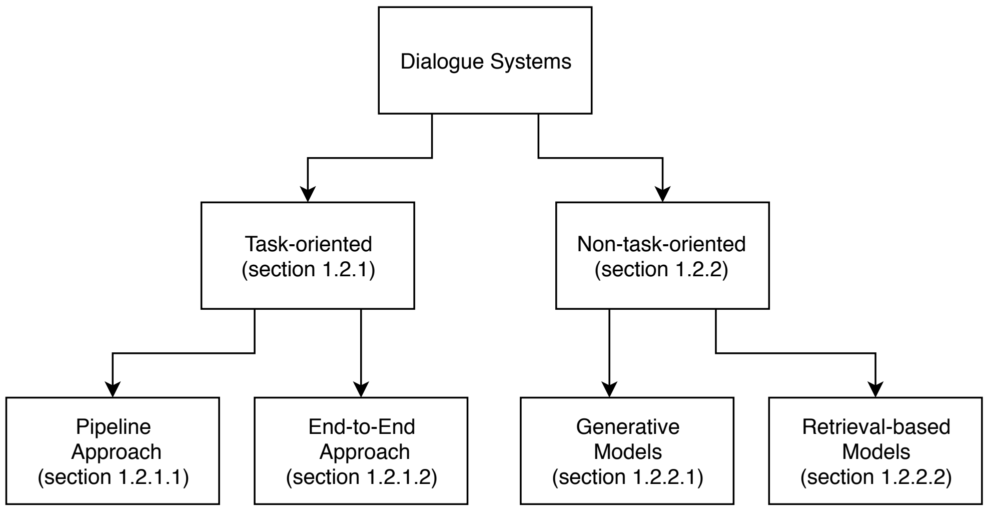
**Pattern Recognition Approach** A well-formulated mathematical framework characterises this approach. The training stage introduces consistent speech pattern representations which enable reliable pattern comparison. These representations can be generated in a statistical or template style. The template approach uses stored pattern references representing the dictionary of possible words. In comparison, the statistical approach utilises probabilistic models, for instance, Hidden Markov models to deal with uncertain or incomplete information. The pattern recognition stage attempts to find a direct match between the input speech and the previously learned patterns to determine a textual transcript[6].

**Artificial Intelligence Approach** The artificial intelligence approach combines the two previous strategies in the form of using the knowledge about linguistic, phonetic and spectrograms as well as occurring patterns. Neuronal Networks enable modern speech recognition systems to learn about the relationship among phonetic events. Besides the immense potential of this relatively new approach, neuronal networks often require hundreds of iteration over large amounts of training data which prolongs the overall time to train such models drastically[6].

## 1.2 Dialogue Systems

Human-machine interaction promises an effective and natural way to exchange information. The machine must, therefore, keep track of various states and parameters such as user speech, conversational context and collected data to be capable of understanding the current phase of the ongoing conversation. This task, so-called dialogue tracking, is one component of a system which enables machines to follow and interact with us in a natural way[33]. In general, dialogue systems can be distinguished in two different categories: task-oriented systems and non-task-oriented systems (see figure 1.2). The latter offers the user a set of actions which can be triggered through a corresponding speech command. As a result of this, the system can guide the user through a whole process and even ask additional questions to limit the intention of the request. That kind of system is commonly structured in the form of a pipeline. Firstly, the system represents the request as an internal state. The second stage selects an action according to the internal state and the implemented policy, which finally generates the response of the system.
Non-task-oriented systems, also termed chatbots, aim to have an open-domain dialogue with their counterpart. Two main approaches are used to realise such systems: generative methods such as sequence-to-sequence models and retrieval-based methods which selecting responses from a database[16].

Figure 1.2: Dialogue System Overview


Dialogue Systems
Task-oriented (section 1.2.1)
Non-task-oriented (section 1.2.2)
Pipeline Approach (section 1.2.1.1)
End-to-End Approach (section 1.2.1.2)
Generative Models (section 1.2.2.1)
Retrieval-based Models (section 1.2.2.2)

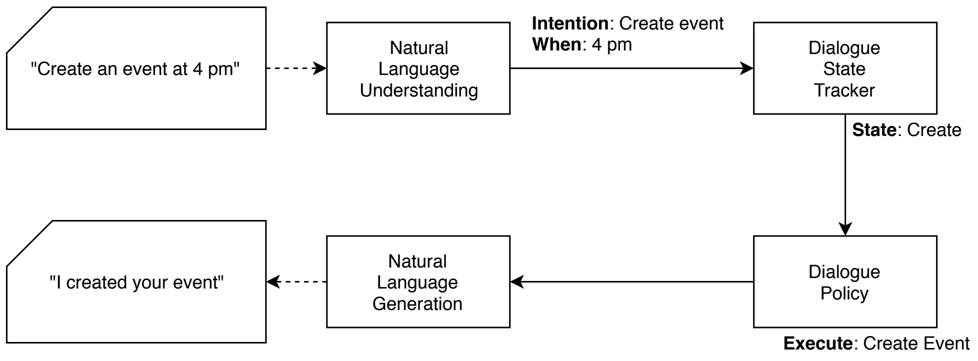
### 1.2.1 Task-Oriented Systems

This chapter introduces two approaches of task-oriented systems: pipelines and the end-to-end method.

#### 1.2.1.1 Pipeline Methods

The pipeline approach of task-oriented systems consists of four separate stages, which are visualised in figure 1.3.

Figure 1.3: Dialogue System Processing Pipeline



**Natural Language Understanding (NLU)** parses the input utterance, which has already been transformed into a string. The NLU attempts to map the given statement into predefined semantic slots. These slots are specific to the scenario of the current conversation. For instance, the user wants to book a flight to New York City, then the required semantic slots could be the destination and the kind of transportation. The same example leads us to the next task of the NLU intent detection. The intent detection identifies the primary reason for the given request and therefore, specifies the individual action later on. The final output of the NLU is the so-called semantic representation, which includes the semantic slots as well as the intention.

**Dialogue State Tracker** manages the current state by using predefined rules to calculate the most likely subsequent state according to the semantic representation of the previous stage. However, these predefined rules are prone to frequent errors as the most likely result is not always the desired state[16].

**Dialogue Policy** chooses the most likely action based on the previously calculated dialogue state. It can consider handcrafted rules or work entirely with a statistical model. For example, a customer wants to book a hotel room. Hence the dialogue policy module could fetch the contingent of available rooms from the hotel database which would represent the most likely action[16].

**Natural Language Generation** represents the last stage of the pipeline and generates out of an abstract action selected by the previous step a natural language utterance. There are two main approaches to implement this module. The first approach consists of a handcrafted collection of predefined responses in comparison to the second approach, which generates the answers entirely by itself. Those two approaches represent the typical trade-off between flexibility and accuracy. The template-based method guarantees grammatically correct responses. However, the second approach offers versatility.

#### 1.2.1.2 End-to-End Methods

The End-to-End method represents a completely new concept of a dialogue system based on deep learning and sequence-to-sequence models. As admitted from[66], the pipeline approach suffers two major limitations. The *Credit Assignment* problem describes the fact that developers, in general, get feedback about the final output of their system. Applying this fact to the given pipeline architecture, tiresome error analysis has to be obeyed to determine the malicious stage. The other dilemma, termed *Process Independence*, occurs if one stage of the pipeline receives an update. In this case, the entire pipeline has to be updated, respectively, because each stage depends on its predecessor. Considering these two drawbacks, a system which only consists out of one single module would be beneficial.[10], and others propose such a module in the form of a network-based end-to-end trainable task-oriented model, which treats the given task as a mapping from the dialogue history to a new response[16].

### 1.2.2 Non-Task-Oriented Systems

In comparison to *Task-Oriented* systems, *Non-Task-Oriented* systems aim to have an open-domain conversation with their counterpart. The upcoming two chapters discuss *Generative* and *Retrieval-based* models which are generally used to implement such procedures.

#### 1.2.2.1 Generative Models

Generative models treat the response creation task as a translation from input requests to corresponding responses. Sequence-to-Sequence (seq2seq) models resulted in great accuracy in translating words from one language in another[63]. Nevertheless, using the same models to solve the given task of generating responses was found to be much harder, considering the fact that a given request fits a large number of various responses. Besides, Sequence-to-Sequence models suffer from a lack of variety and meaning of their generated responses. Both concerns are current research topics and challenges for the generative approach of dialogue systems[46, 16].

#### 1.2.2.2 Retrieval-based Methods

This approach takes advantage of response repositories to select a suitable answer to a given demand. The essential part of this method is the matching algorithm. Single-turn response matching algorithms take into account the current request message of the utterance. The algorithm calculates out of this message a vector representation and tries to maximise the correspondence with precalculated vectors of the repository responses. Multi-turn response matching algorithms work with the same principles but considering the previous massages of the dialogue to enrich the calculated feature vector. This method guarantees grammatically correct and reasonable responses during the conversation, as long as the matching algorithm manages to determine a corresponding reaction within the repository[16].

## 1.3 Text Synthesis

The text synthesis represents the last element in the processing pipeline of a modern voice assistant. In general, text synthesis aims to solve a signal-inversion problem. The information encoded in a string of characters represents a highly compressed signal which must be transformed into speech utterances, so a decompressed signal. Conventional text-to-speech (TTS) systems are composed out of several submodules like acoustic front-ends, duration models, acoustic prediction models and vocoder models. The complexity of such systems is reasonably high, plus the accuracy nor the naturality of the generated speech is comparable to a human utterance. Recent advances in deep learning and particularly of end-to-end models allow a novel approach to creating utterances out of a text passage with only one single module. This new approach utilises audio-text pairs to train a sequence-to-sequence model and decreases, thereby the complexity respectively by improving the quality at the same time[65]. There are two major categories of text-to-speech systems which can be subdivided according to the utilised technique. Fundamentally, systems can generate a speech in the form of composing recorded waveforms or generating the waveform completely by itself. Both, *Sample-based* and *Generative* methods are described in the following chapters as stated by[26]. Furthermore, there are *Hidden Markov Models* and *End-to-End Models* described attempting to benefit from recent advances in machine learning (see figure 1.4).

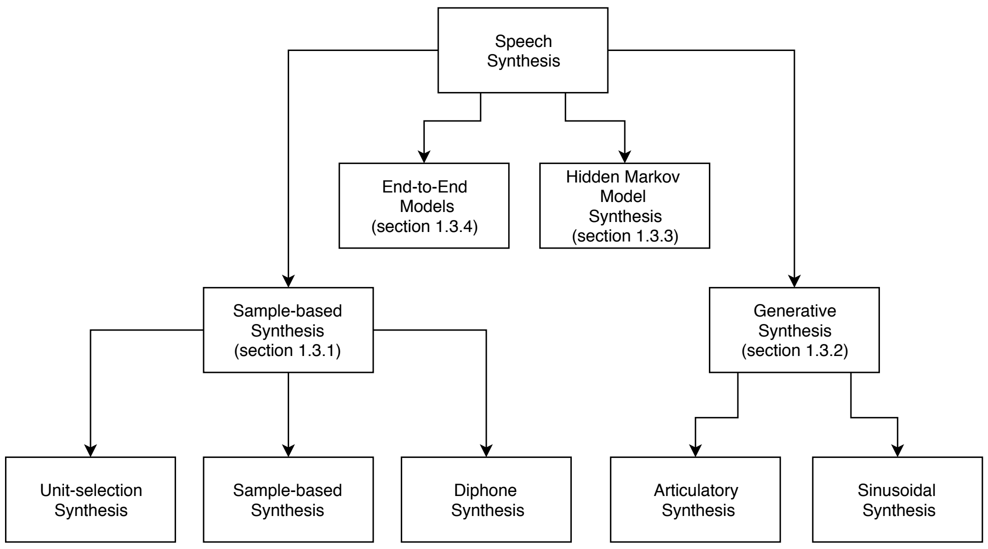

Speech Synthesis
End-to-End Models (section 1.3.4)
Hidden Markov Model Synthesis (section 1.3.3)
Sample-based Synthesis (section 1.3.1)
Generative Synthesis (section 1.3.2)
Unit-selection Synthesis
Sample-based Synthesis
Diphone Synthesis
Articulatory Synthesis
Sinusoidal Synthesis

Figure 1.4: Speech Synthesis Overview

### 1.3.1 Sample-based Synthesis

Sample-based Synthesis based on the fact that human speech represents the most natural possible utterance. Therefore, the simplest possible approach might concatenate recorded real-world utterances to form new speech responses.

**Limited Domain Synthesis** This approach can be applied for use cases which are limited to a specific domain, for instance, train announcements. Following that example, each station name and further required information are recorded in advance. If the train approaches a train station, the synthesiser concatenates the required utterances to inform the passengers about the upcoming stop. Limited domain synthesis is straightforward to implement and can be comfortably extended. Overall, this system can produce excellent natural speech responses. However, it is thereby limited to its narrow domain, which makes it worthless for real text-to-speech use cases.

**Unit Selection Synthesis** This method generalises the previous approach in the form of extending the domain. The system allocates a database to store a vast amount of recorded utterances which takes a considerable effort to provide. Generally, speech synthesis systems must be able to pronounce arbitrary words which our current system is still not capable of. A first solution might be to record and store phones as well as words and sentences in the same database. The system could use the phones to build the required waveforms of arbitrary words. Unfortunately, this approach results in low intelligibility and naturalness according to the fact that phones are differently pronounced depending on their context. This effect can be bypassed in recording phone transitions, so-called diphones. Diphones are far more stable in their pronunciation in comparison to phones itself. However, the resulting database consists of many redundant entries since each phone has to be recorded multiple times with varying intonation, speed, and pitch. All in all, unit selection systems embody the most potent and natural sample-based speech synthesis system. However, it costs an enormous effort to create the required database.

**Diphone Synthesis** The *Unit Selection Synthesis* uses a database containing words, sentences and diphones to form new speech utterances. This database has to manage numerous entries which increase the size tremendously. The *Diphone Synthesis* attempts to solve this problem in storing only diphones in its database. The diminished capacity of the module enables the application to run on embedded systems and uses less computational resources to search through the dataset at the same time. However, the generation of utterances only with diphones introduces new challenges. Speech irregularities occur if diphones are not recorded with the same pitch, speed and pronunciation. Furthermore, the recorded waveforms should start and end at a common point to ensure continuity of the generated response. Generally, *Diphone Synthesis* results in a high level of intelligibility but provides a low level of naturalness. The proposed solution to ensure continuity also leads to waveforms which do not represent the changing pitch and emphasis of a human utterance.

### 1.3.2 Generative Synthesis

Generative Synthesis approaches generating response waveforms purely programmatically by using software models to simulate the physical and acoustic appearance of humans. This leads primarily to models with a meagerer memory footprint. However, generative approaches require more computation resources.

**Articulatory Synthesis** The *Arcticulartory Synthesis* applies knowledge of the human vocal tract in the form of a physical simulation to approximate human utterances. However, the physical simulation requires advanced biomechanical and fluid dynamic models to calculate the air flow through the vocal tract. The major challenge of this synthesis method embodies the model generation. For this purpose, advanced scanning techniques like x-ray cameras are used to reconstruct a virtual clone of the vocal tract of a human being. The reconstruction demands high-level algorithms to build out of two-dimensional data a three-dimensional model. Nevertheless, through changing the internal model parameter, various speaker voices can be produced, which is a massive benefit of this approach[26].

**Sinusoidal Synthesis** In comparison to *Articulatory Synthesis* the *Sinusoidal Synthesis* aims to create directly a waveform produced in ordinary speech. This approach does not need an advanced model of a vocal tract and is therefore much more comfortable to implement. The method superimposes different sinusoidal waves together with noise to create vowels and consonants. This method leads to intelligible utterances with a lack of naturalness[26].

### 1.3.3 Hidden Markov Model Synthesis

Hidden Markov Models (HMM) are statistical models describing the dependency between states of a given system. The central statement of Markov says that the current state does only depend on the previous state of a system. The *Hidden Markov Model Synthesis* aims to benefit from both fundamental approaches: *Sample-based* and *Generative* synthesis. This approach utilises huge datasets to learn an HMM in predicting the next phone depending on the previous one comparable to a sample-based synthesis method. Notwithstanding, the HMM does not concatenate recorded samples to generate waveforms. It generates programmatically different waveforms like a *generative* approach[26].

### 1.3.4 End-to-End Models

End-to-End models describe a novel architecture which applies recent advances in neural networks. Thereby, two different kinds of end-to-end models can be distinguished: *Recurrent* and *Convolutional* Neuronal Networks. Both variants qualify a speech synthesis as a mapping of a given input sequence to a sequence of phones building a waveform. Encoder-Decoder architectures are a preferred way of solving sequence-to-sequence tasks in the field of natural language processing. The encoder network, represented through a convolutional or recurrent neuronal network, calculates out of a string of words an internal state. The decoder network uses this internal representation to determine in a second step the waveform utterance. Recent research papers introduced an additional layer between encoder and decoder so-called attention mechanism, which manages the alignment between the input and the output sequence[29, 65].

## 1.4 Virtualization

A monolith is a software application whose modules cannot be executed independently [24]. Therefore, monoliths suffering various issues like scalability, dependency management, maintainability. The *microservice* architecture has been proposed to cope with these problems. The *microservice* architecture describes a system which splits the application in simple, small and lightweight services. Those services are designed to perform a very cohesive business function by communicating over messages with each other and are therefore completely independ[2]. This innovative software style solves some of the known issues of monolith application. Services enable dynamic and natural load balancing. If a system has to manage more workload regarding user requests, for example, the same module can be relaunched to provide several instances. This modularity also improves the development of other systems, since reusing a given module with its clear interface is an additional advantage of this architecture. Furthermore, the modularity of the microservice architecture allows teams to work entirely distinct from which the testing and verification process benefits equally. However, it also introduces new challenges with its character as a distributed system. Primarily, the communication within the system represents a critical weakness. Distributed systems use conventional web technologies like Representational State Transfer (REST), JavaScript Object Notation (JSON) and Extensible Markup Language (XML) to realise the interaction between services and third-party modules. These protocols require additional overhead to secure the data exchange and ensure the integrity of the whole system. Another challenge represents the system performance. The network latency has a significant influence since it is much slower than data exchange through in-memory processes[24]. The following chapters discuss the realisation of a microservice architecture with Docker, an open-source containerisation tool and focus thereby on its security mechanism (see figure 1.5).

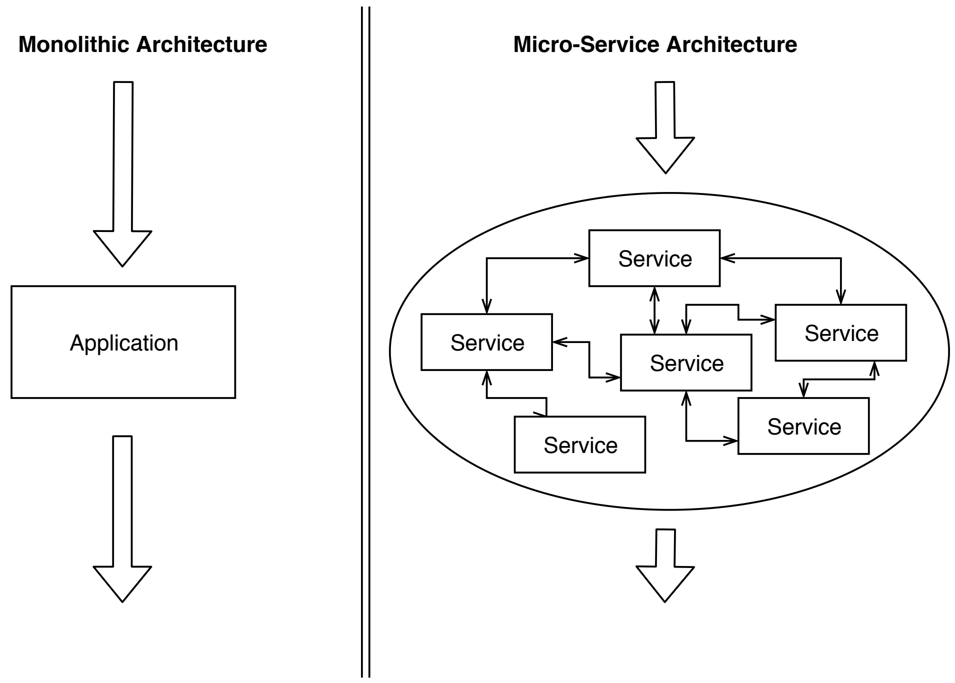


Figure 1.5: Monolithic vs. Micro-Service Architecture

### 1.4.1 Docker Ecosystem

Docker[22] describes a set of open-source products which are maintained and developed by the Docker Incorporation. This toolset is current one of the most successful containerisation application since it offers unique advantages in comparison to his predecessors. First, Docker offers an easy and straightforward interface to run an application within a container. This benefit increases the group of potential users and use-cases. Furthermore, the Docker ecosystem collaborates with various third-party tools like Kubernetes[7], Vagrant[32] and Ansible[58] which improve the management and the deployment of Docker containers or more general microservices. Lastly, a huge community helps to improve and increases the containerisation universe of Docker with new features and particularly base images, which function as blueprints for new containers as well as complete implemented services like databases or web servers[12].

**Docker Engine** Docker Engine is a packaging tool to create and launch a containerised application. Figure 1.6 shows the principal components of the Docker Engine. The Docker Daemon runs on top of the host operating systems and provides the basic virtualisation functionality. The Docker Client is a command-line tool which allows the user to interact with the Docker Daemon and its containers over a RESTful API. In general, the Docker Daemon establishes a secure and isolated environment to run its container. This environment uses two main features of the Linux kernel: namespaces and control groups (cgroups). The kernel namespaces are responsible for altering the view of internal container processes. That means that processes running inside the container do not see processes of other namespaces and vice versa. This mechanism works with six different categories of namespaces: mount, hostname, Inter-Process Communication (IPC), Process Identifiers (PID), and network[37]. Control Groups provide the functionality to limit the access to specific resources which a particular process within a container is allowed to access[12].
Nevertheless, processor security vulnerabilities like Meltdown or Spectre clarifying that Docker with its numerous security layers depends lastly on a secure operating system as well as reliable hardware. In general, the system administrator must ensure that the latest versions of the Linux kernel as well as the Docker engine itself are used to minimize the risk of such vulnerabilities.

**Docker Container** The Docker Engines launches its container from images. These images represent a blueprint which instructs the Docker Engine how to build and start a new container. Images consist of layers which subdivides and structures the build workflow into steps. This mechanism simplifies the update and distribution process of containers over a network significantly. Figure 1.7 shows the structure of an image with examples for each layer. The base image

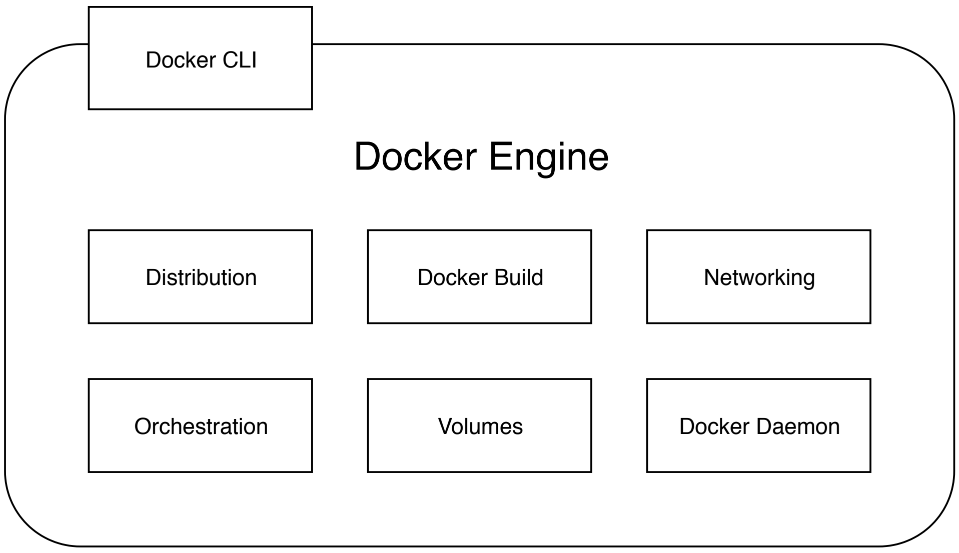


Figure 1.6: Docker Engine Overview

serves as an essential layer which sets up the container environment like the operating system and its basic system configurations. After that, additional libraries and application are installed as well as files which can be added to the container.

**Docker Hub** Docker Hub[23] serves as a platform to exchange customised container images stored in public and private repositories. Furthermore, the platform signs and authenticates all uploaded images so that potential users can verify their integrity[18].

### 1.4.2 Docker Security

This chapter reviews the security mechanism, namely *Isolation*, *Host Hardening* and *Network Security*, discussed by [18] which are enforced by Docker.

**Isolation** As previously mentioned, the Docker Daemon isolates its container by using Linux kernel features exclusively. By default, namespaces are automatically configured in comparison to cgroups which need to be enabled manually. The default configuration of cgroups is fairly strict beside the fact that all containers are sharing the same network bridge, which enables ARP poisoning attacks. The configuration can be changed to deactivate the mutual network bridge. However, this would restrict the whole communication between containers and is therefore really often not applicable.

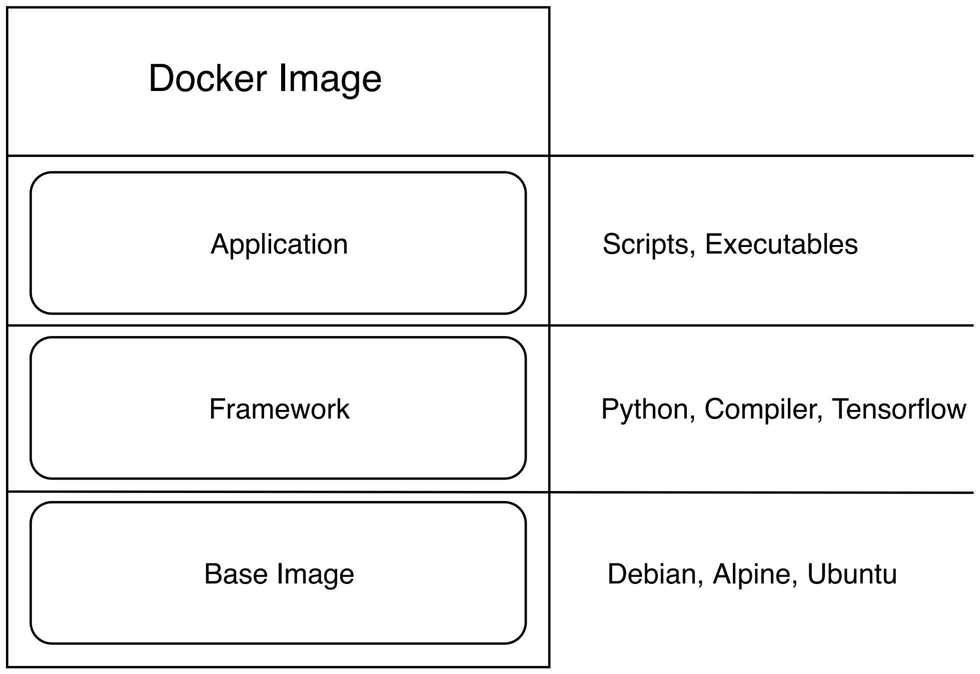


Figure 1.7: Docker Image Structure

**Host Hardening** Host Hardening can be enforced through additional Linux kernel modules like SELinux, Apparmor and Seccomp which are currently supported by Docker. These modules restrict cases such as a container escapes or compromising containers from the host side. Docker offers a default configuration for those modules which hardens the host from the containers. Nevertheless, further settings, for example, restricting the access from the host to the containers, must be configured individually.

**Network Security** The Docker Daemon is controllable over the network, which enables the usage of Docker containers in cloud structures. This connection is by default, encrypted with TLS, and the images itself are verified with a hash. Docker Content Trust is another security mechanism introduced by Docker to address package manager flaws. This mechanism implements a whole PKI process to ensure a certain level of security in the process of managing images over the network.

Chapter 2

# Result

The result chapter presents the findings and proposals of this thesis. Initially, a model defines the objectives regarding the DSRM to evaluate the designed solution (see *Methodology* chapter). Next, chapter 2.2 presents the findings of a thorough comparison of various frameworks and justifies the final choice respectively for each stage of the processing pipeline. Chapter 2.3 discusses the system architecture, the micro-services and the defined communication schema.

## 2.1 Solution Objectives Model

Following the chosen methodology, an objectives model has to be defined. The objectives model represents the baseline with which the developed system can be compared to evaluate its performance. Regarding the defined requirements in chapter , the model focuses on three main aspects.

1. **Offline**: The whole system must be capable of working at least partial offline, which means that the system must be able to process a user request without having a stable internet connection.

2. **User Experience**: The user must be able to have a smooth, understandable and rational communication with the system.

3. **Security**: The system must be secure in the matter of information privacy and general attacks from outside.

## 2.2 Framework Selection

This chapter examines the outcome of thorough research, aiming to find frameworks in the domain of natural language processing. Both open and closed source projects are considered, as well as the latest introduced algorithms based on scientific papers. The following three sections evaluating frameworks distinct for the three different stages of a voice assistant: speech recognition, intent matching and speech synthesis. Thereby, each section defines an evaluation metric to ensure a systematic comparison of pre-selected frameworks.

### 2.2.1 Speech Recognition

Speech Recognition represents the first stage of a recent speech assistant application. This stage influences the accuracy of the following stages directly, which leads to misinterpretation of utterances. A designated metric evaluates a subset of speech recognition frameworks based on six different characteristics which rely on the previously defined objectives in chapter . The following enumeration discusses the specified characteristics and their purpose.

1. **Information Privacy** discusses the collection and dissemination of user data.
2. **Adaptability** reviews the capacity to adapt to a specific domain.
3. **Offline Capability** reflects the ability to operate offline.
4. **Language Support** inspects the number of out-of-the-box supported languages.
5. **Support/Documenation** reflects the given customer support and documentation.
6. **Maintainability** surveys the effort of maintaining this framework.

#### 2.2.1.1 CMU Sphinx

CMUSphinx[13] represents a set of application focusing on language processing developed by the Carnegie Mellon University (CMU). Some of the possible use-cases are speech transcription, closed captioning, speech translation, voice search and language learning[15]. The CMUSphinx toolkit offers four major applications, which are all together speaker independent. Pocketsphinx, together with Sphinxbase, serves as a lightweight recogniser tool with a small vocabulary which is implemented in the C programming language. Sphinxtrain includes all the necessary tools to train models for the CMUSphinx toolkit. Sphinx4[41] represents a Java implementation of the same algorithm developed to run on servers allowing to search through large vocabularies. Furthermore, the project offers pre-trained models in different languages as a download for free.
The CMUSphinx family uses a similar approach like the acoustic-phonetic approach by utilising Hidden Markov models to recognise speech utterances[4]. Furthermore, various feature selection methods are implemented like MFC,

PCA, and Linear Discriminant Analysis (LDA) as well as additional language models[39].
Table 2.1 shows the evaluation results for the Pocketsphinx package. The CMUSphinx application family is developed as an open-source project[14] with a specific focus on a lightweight implementation which enables the usage on embedded systems. This fact implicates that the framework is offline usable since the source code can be compiled for every platform.
Furthermore, the system architecture allows through its modular design to fork from the latest development branch developing individual features and improvements. The Sphinxtrain toolset empowers the opportunity to adapt the framework to various languages and particularly to create domain-specific vocabularies which enhance the accuracy of the system. The system itself does not store any individual data, and given the fact of its open-source character, we can review the entire code base to justify this assumption. This suggests that the privacy of user input data is warranted.
In contrast, the maintainability of this framework rates rather weak in comparison to closed-source solutions. A thorough foundation in the field of natural language processing, as well as software architecture, including several programming languages, is required to maintain this project. However, CMUSphinx engages an active community which develops new features and bugfixes.

| **Characteristic** | **Result** |
|---|---|
| Information Privacy | High |
| Adaptability | High |
| Offline Capability | Yes |
| Language Support | English, Chinese, French, Spanish, German, Russian |
| Support/Documentation | Active community, brief documentation |
| Maintainability | Medium |

Table 2.1: CMUSphinx evaluation

#### 2.2.1.2 DeepSpeech 2

DeepSpeech 2[50] represents an implementation of the end-to-end speech recognition algorithm proposed by Amodei et al.[3]. The project takes advantage of PaddlePaddle[51], which is an efficient deep learning platform from Baidu. In comparison to the CMUSphinx framework in chapter 2.2.1.1, this algorithm adopts a radically different approach. Figure 2.1 shows the architecture of the model, which combines several convolutional input layers followed by multiple recurrent layers together with a finally softmax layer. This architecture allows training the model with simple audio-text pairs.

On the contrary side, this class of model requires a vast amount of compu-

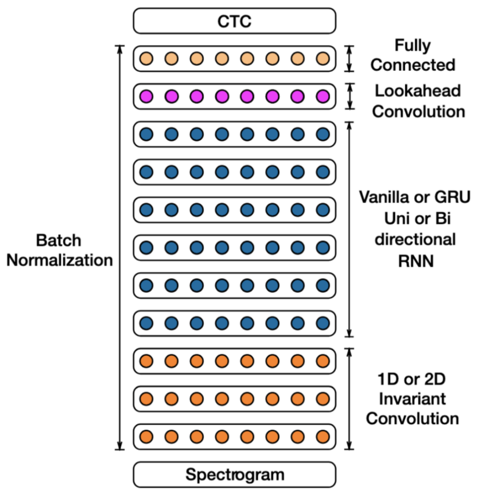

CTC
Fully Connected
Lookahead Convolution
Vanilla or GRU Uni or Bi directional RNN
Batch Normalization
1D or 2D Invariant Convolution
Spectrogram

Figure 2.1: Architecture of the DeepSpeech2 model (Retrieved from[3])

tational resources to determine the up to 52 million parameters of the Deep-Speech 2 model[20]. These facts influenced the evaluation of this framework, which is presented in the following table 2.2. The considerable effort to train new models and therefore lowers the adaptability to new languages lowers the rating for the maintainability and the adaptability. Though, the project offers two pre-trained models in English and Mandarin.
On the other hand, the project supports offline usage and does not store any user data which can be again guaranteed by reviewing the source code. Furthermore, the project has got an active community. However, the documentation consists only out of the research paper and sparse yet complete instructions on the project page.

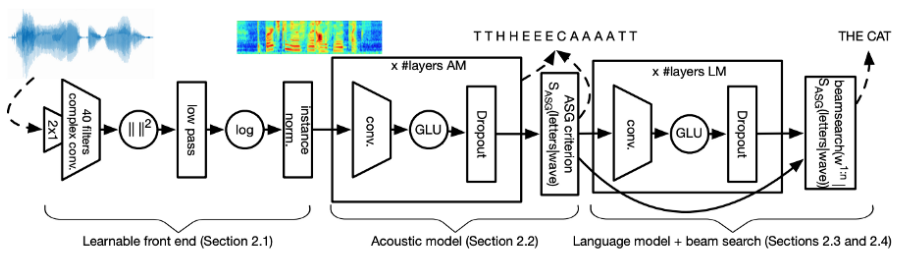
| Characteristic | Result |
|---|---|
| Information Privacy | High |
| Adaptability | Medium |
| Offline Capability | Yes |
| Language Support | English, Chinese |
| Support/Documenation | Low/Good |
| Maintainability | Low |

Table 2.2: DeepSpeech 2 evaluation

#### 2.2.1.3 wav2letter++

The Facebook AI research team develops an open-source speech recognition toolkit based on an end-to-end architecture. The toolkit focuses on efficiency, which makes wav2letter++ two times faster in training compared to other equivalent frameworks[54]. Figure 2.2 shows the different modules of the wav2letter++ framework. The learnable front-end chooses features out of the raw waveform based on Mel-Frequency Cepstral Coefficients (MFCC) or the power spectrum. Next, the acoustic model uses 1-dimensional convolutional neural networks which offer the benefit to see a huger context without increasing the number of parameters. Lastly, the language model adopts a similar architecture of convolutional neuronal networks and connects it to the segmentation criteria. The segmentation criteria represent a simpler form of the CTC criteria of DeepSpeech2 algorithm[17].

Figure 2.2: Architecture of the wav2letter++ model (Retrieved from[25])



The following table 2.3 shows the evaluation of this framework. The framework is published as an open-source project allowing us to run the code offline on any device which offers enough computation resources. Furthermore, the source code does not imply that the framework stores user data during the inference process. The facebook AI research team used for their framework an end-to-end architecture, which represents one of the latest approaches in the field of speech recognition. As already mentioned in chapter 2.2.1.2, end-to-end models differ significantly in their structure of the training dataset. This means that training data can be generated much more comfortable in the form of using audiobooks, for example, since the required dataset consists out of audio-text pairs. This simplified dataset generation increases the adaptability significantly in comparison to the approach used within chapter 2.2.1.1. Unfortunately, the wave2letter++ framework does not offer pre-trained models which would involve a model training before the framework can be used in an application. Furthermore, to use wav2letter++ a thorough foundation of machine learning and language processing knowledge is required, which evaluates the maintainability as relatively low in comparison to other frameworks.

#### 2.2.1.4 Summary

This chapter evaluated three pre-selected speech recognition frameworks with a metric described in chapter 2.2.1. The first framework, CMUSphinx, implements a traditional architecture of a speech recognition task which involves

| Characteristic | Result |
|---|---|
| Information Privacy | High |
| Adaptability | High |
| Offline Capability | Yes |
| Language Support | No pre-trained models available |
| Support/Documenation | Low/Good |
| Maintainability | Low |

Table 2.3: wav2letter++ evaluation

a Hidden Markov Model. Besides being good documented, the framework supports various languages out-of-the-box. DeepSpeech, the other evaluated framework, follows a more recent approach in applying an end-to-end architecture. Therefore, the framework benefits from much easier training dataset structures. However, neuronal networks require substantial computational resources and data to be able to outperform the traditional approach with HMMs. Last but not least, wave2letter++ presents a similar end-to-end architecture but stands out with its efficient implementation, which decreases the training time by a factor of two. In conclusion, CMUSphinx convicted with its various supported languages and the HMM-based implementation. Nevertheless, the speech recognition stage fulfils a crucial task and must, therefore, operate with the highest accuracy possible. That in mind, the CMUSphinx framework should be exchanged in the long-term with modern end-to-end models.

### 2.2.2 Intent Matching

The second stage of our processing pipeline is the intent matching. Intent matcher trying to extract the intent out of spoken utterances as described in chapter 1.2. As well as in the previous chapter, a metric supports the comparison of three pre-selected frameworks discussed in the following enumeration.

1. **Information Privacy** discusses the collection and dissemination of user data.
2. **Endpoint Support** reviews the capacity to communicate with third-party endpoints.
3. **Offline Capability** reflects the ability to operate offline.
4. **Language Support** inspects the number of out-of-the-box supported languages.
5. **Documenation** reflects the given customer support and documentation.
6. **Maintainability** surveys the effort of maintaining this framework.

#### 2.2.2.1 Rasa

Rasa[57] is one of the leading open-source machine learning toolkits providing algorithms to extract and manage conversations between humans and computers[56]. The toolkit consists of two major application: Rasa NLU and Rasa Core. The former describes the natural language unit of Rasa, which is responsible for translating text utterances in an understandable computer format. This translation task involves to fill in entities and slots representing the memory of the assistant as described in paragraph 1.2.1.1. The figure 2.3 shows the general components of the overall rasa system. The NLU is realised through the first step.

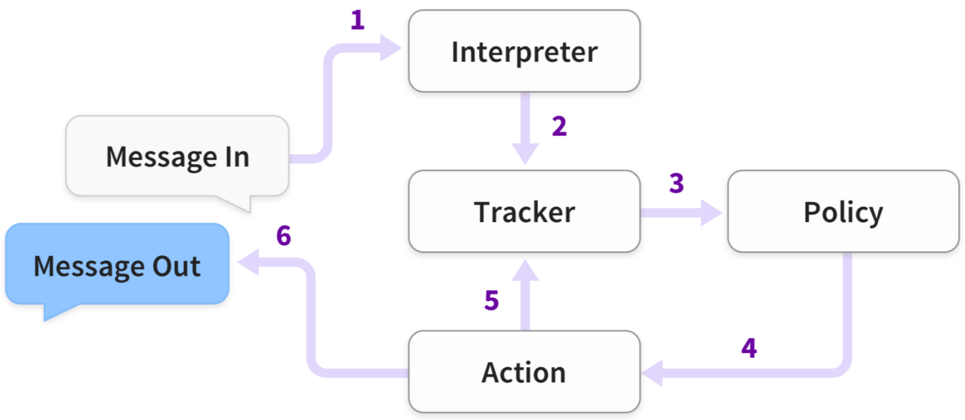


Figure 2.3: Architecture of the Rasa system (Retrieved from[9])

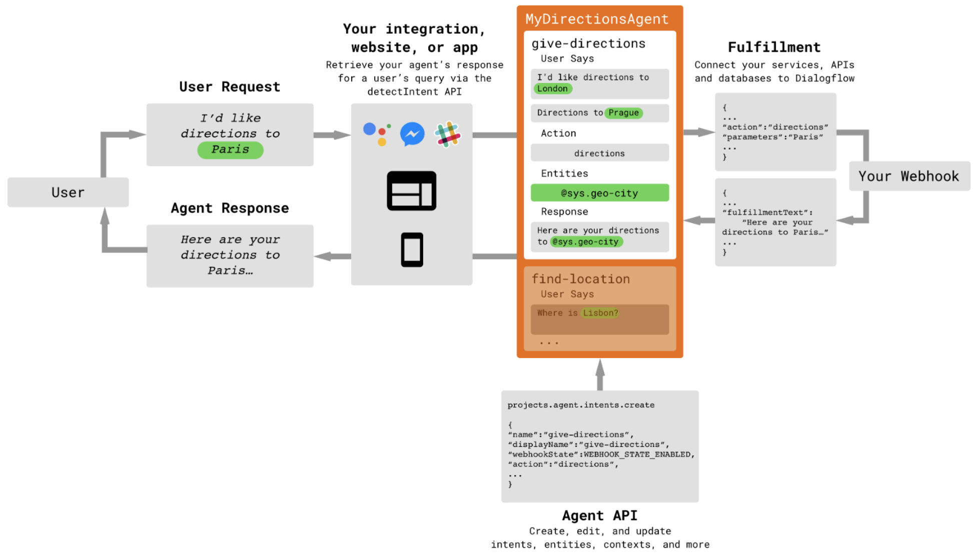
Rasa Core represented through step two to six in figure 2.3 is responsible for tracking and executing of actions according to the given output of the NLU. The framework offers through its modular architecture based on RESTful APIs the possibility to separate the implementation of actions. This can be particularly beneficial if the application backend runs on a different device. Furthermore, Rasa Core allows integrating various third-party platforms like Facebook Messenger, Telegram, SocketIO. Table 2.4 shows the various language which the rasa platform supports and highlights the excellent documentation of this framework. The offline capability ensures information privacy, though Rasa allows outsourcing the tracker to an external database. The configuration of the rasa framework works primarily through files encoded in markdown or JSON format. Third-party projects started to implement graphical user interfaces to simplify the process of generating the required training datasets utilising the RESTful API of Rasa Core and NLU[9].

| Characteristic | Result |
|---|---|
| Information Privacy | High |
| Endpoint Support | Yes |
| Offline Capability | Yes |
| Language Support | English, Spanish, Portuguese, Italian, French, Chinese |
| Support/Documenation | Good/Very Good |
| Maintainability | Medium |

Table 2.4: Rasa evaluation

#### 2.2.2.2 Dialogflow

Dialogflow serves as an online provider of chatbots based on the Google Cloud Platform. It offers a simple and powerful way to create assistants which are fully integrated within the Google universe as well as several third-party providers like Kik, Telegram, Twitter and Skype. The following figure 2.4 explains the overall structure of the Dialogflow platform. Dialogflow works with agents describing a custom chatbot application. The end-user device and Dialogflow communicating with the provided SDK available for various programming languages. The agent extracts the intention together with pre-defined entities. The Fulfilment component connects with the company internal server or backend, which implements the required actions. The agent itself is configurable through the web interface, which serves as documentation and guide at the same time.


User
User Request
I'd like directions to Paris
Agent Response
Here are your directions to Paris…
Your integration, website, or app
Retrieve your agent's response for a user's query via the detectIntent API
MyDirectionsAgent
give-directions
User Says
I'd like directions to London
Directions to Prague
Action
directions
Entities
@sys.geo-city
Response
Here are your directions to @sys.geo-city
find-location
User Says
Where is Lisbon?
...
Fulfillment
Connect your services, APIs and databases to Dialogflow
Your Webhook
Agent API
Create, edit, and update intents, entities, contexts, and more

```
{
...
"action":"directions"
"parameters":"Paris"
...
}
```

```
{
...
"fulfillmentText":
    "Here are your
directions to Paris…"
...
}
```

```
projects.agent.intents.create

{
"name":"give-directions",
"displayName":"give-directions",
"webhookState":WEBHOOK_STATE_ENABLED,
"action":"directions",
...
}
```

Figure 2.4: Dialogflow SDK structure (Retrieved from[30])

The evaluation of Dialogflow reports proper documentation, including an

| Characteristic | Result |
|---|---|
| Information Privacy | N/A |
| Endpoint Support | Yes |
| Offline Capability | No |
| Language Support | up to 20 languages |
| Support/Documenation | Very Good/Very Good |
| Maintainability | Very Good |

Table 2.5: Dialogflow evaluation

extensive selection of examples, which allows everybody without specific knowledge in machine learning to create a chatbot agent. Furthermore, Dialogflow benefits from the Google service backend, which allows it to support a vast amount of languages as well as various other components like text-to-speech and phone call capabilities. This advanced feature set requires a considerable amount of resources which forces Dialogflow agents to run online. Information privacy cannot be guaranteed considering the fact that the service runs on Google servers as well as the source code is closed-source. Last but not least, the maintainability outperforms the previous framework based on the graphical user interface and the provided support. Nevertheless, Dialogflow charges their customer with fees if a limited amount of requests may be exceeded.

#### 2.2.2.3 Snips NLU

Coucke et al. describe in their paper[19] Snips as a machine learning based voice platform which is able to perform inference on embedded systems. Furthermore, the platform does not store any user data which supports the so-called "privacy by design" paradigm. This paradigm implies that the platform is offline capable as well as published in the form of an open-source project. Snips supports currently seven languages, including English, German, French.

In general, Snips NLU follows the architecture described in figure 2.5. The system applies two different intent matching algorithms successively. The deterministic intent matching component uses regular expressions to fill intents and slots, which results in perfect matches as long as the input has been part of the training dataset. If the deterministic matcher is not able to fill the entities probably, the probabilistic intent matcher takes over. This intent matcher works with a machine learning approach and is, therefore, able to outperform the deterministic intent matcher in such cases. Table 2.6 presents the evaluation results of the Snips NLU. The framework strikes with its "private by design" premise, which ensures information privacy as well as the possibility to change and adapt the framework. Currently, the framework supports six languages[62]. Besides the published scientific paper, several examples are documenting the NLU. However, the framework does not provide any out-of-

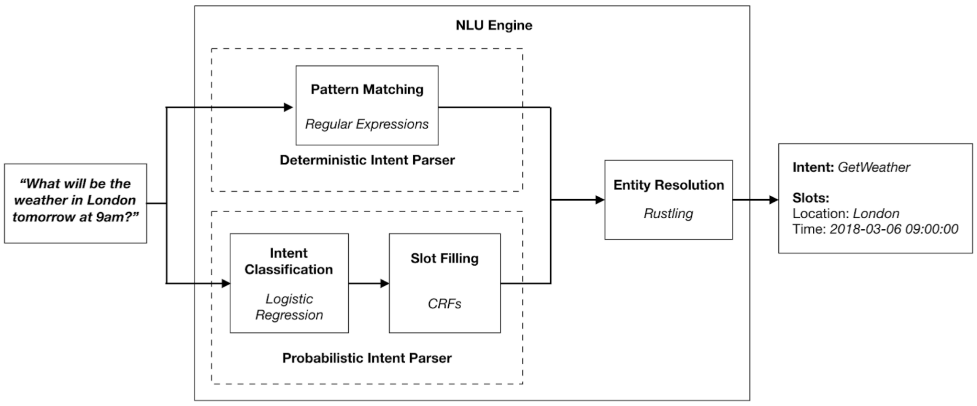


Figure 2.5: Snips NLU architecture (Retrieved from [8])

| Characteristic | Result |
|---|---|
| Information Privacy | High |
| Endpoint Support | No |
| Offline Capability | Yes |
| Language Support | English, French, German, Spanish, Korean, Italian |
| Support/Documentation | Medium/Good |
| Maintainability | Good |

Table 2.6: Snips NLU evaluation

the-box support for third-party applications.

#### 2.2.2.4 Summary

This chapter intends to examine intent matching frameworks based on the developed metric. This metric focuses on the key aspects of modern intent matching mechanism like adaptability, information privacy, language support and the accessibility of third-party services. Rasa convinces with its completely modular architecture and its abundant accessibility. The framework supports the most common languages as well as offers the opportunity to run entirely offline. On the other hand, Rasa lacks in usability and maintainability based on a missing unified graphical user interface. Dialogflow represents an application powered by the Google infrastructure. Regarding this fact, the service offers a vast amount of supported languages and a unique user experience throughout the whole lifecycle. However, Dialogflow does not support to run their intent matching algorithms locally, which violates a fundamental requirement. Snips NLU presents a similar feature set like Rasa by supporting various languages and offering offline capabilities. The small memory footprint predestinates this framework to run on embedded machines with a limited amount of resources.

However, Snips NLU does not support third-party accessibility yet which outlines as a disadvantage for later adaptations. Overall, Rasa conveniences with its modular architecture, which supports a smooth integration into existing infrastructure.

### 2.2.3 Speech Synthesis

Speech synthesis represents the last module in our processing pipeline. The module aims to synthesis waveforms out of the response provided by the intent matcher. This problem can be seen as the decompression of highly compressed information, as described in chapter 1.3. The intelligibility and naturality are the two highest weighted parameters during the evaluation process since these parameters significantly influence the user experience of the robot. In this context and together with the baseline requirements equivalent to the previous evaluations, the following metric resulted.

1. **Information Privacy** discusses the collection and dissemination of user data.
2. **Intelligibility** reviews the intelligibility of the synthesised utterances.
3. **Naturality** reviews the naturality of the synthesised utterances.
4. **Offline Capability** reflects the ability to operate offline.
5. **Language Support** inspects the number of out-of-the-box supported languages.
6. **Documenation** reflects the provided customer support and documentation.
7. **Maintainability** surveys the effort of maintaining this framework.

#### 2.2.3.1 Tacotron

Tacotron describes a speech synthesis algorithm proposed by Wang et al. [65]. This algorithm works in an end-to-end fashion and outperforms traditional approaches in naturality and computation time. Figure 2.6 shows the overall architecture of the model and its underlying components. The model consists of three major parts: Encoder, Decoder and Audio Reconstruction[44]. The overall goal of the Decoder is to find a feature representation of the input. Therefore the Decoder takes character embeddings as an input and applies a set of non-linear transformations, so-called pre-net, followed by a CBHG module. The abbreviation CBHG describes a module of a one-dimensional convolutional network followed by a highway network and a bidirectional Gated Recurrent Unit (GRU).

The Tacotron model utilises an attention-based decoder with a stateful recurrent layer providing the attention query at each time step. The Decoder aims to generate a mel-scale spectrogram based on the internal state representation computed by the encoder. A mel-scale spectrogram expresses an acoustic time-frequency representation of sound. As shown in figure 2.6, the Decoder feeds the last predictions through a pre-net module to the next prediction stage. Based on the usage of the Griffin-Lim algorithm to finally synthesise the audio sequence, the post-processing network attempts to predict a spectral magnitude spectrogram sampled on a linear frequency scale. Besides that, the authors arguing that the post-processing network provides another significant benefit. The post-processing network has the ability to see the full decoded sequence, which offers the opportunity to correct the prediction error of each frame.

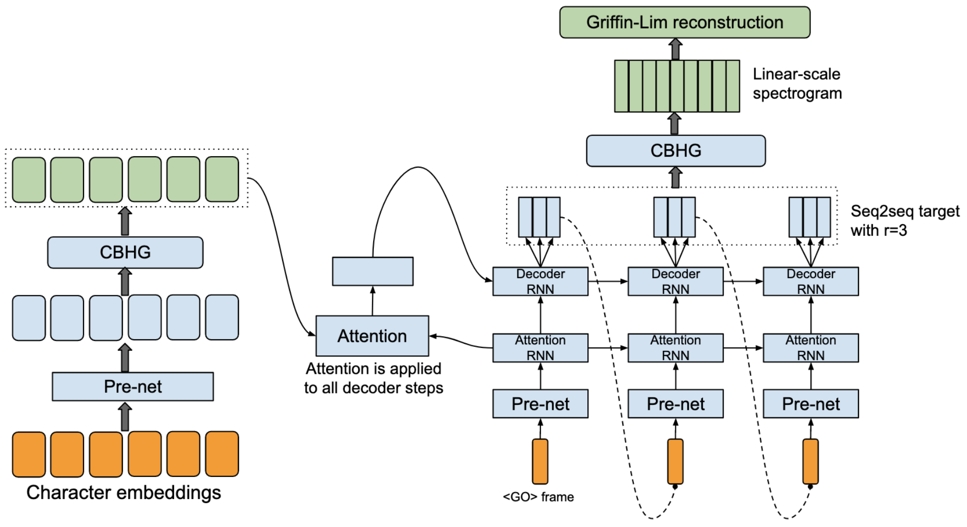


Figure 2.6: Tacotron Model Architecture (Retrieved from[65])

Unfortunately, Wang et al. did not provide their implementation of the Tacotron model neither their internal training dataset. This matter of fact resulted in numerous open-source implementation of the model with varying quality in comparison to the published audio samples of the authors. Keith Ito[38] implemented the Tacotron algorithm[35] by using Tensorflow, an open-source machine learning framework[31]. His implementation seems to be the most promising one besides the version of Alex Barron[1] and Kyubyong Park[40]. Therefore, the evaluation of the Tacotron model based on the implementation of Keith Ito, who also provides a pre-trained model.

Tacotron outperforms its predecessors in intelligibility by decreasing the computation time. Nevertheless, the open-source implementations were not able to reach the same accuracy and quality as the results provided by the authors. At the same time, the model itself still needs sufficient infrastruc-

| Characteristic | Result |
|---|---|
| Information Privacy | High |
| Intelligibility | Good |
| Naturality | Medium |
| Offline Capability | Yes |
| Language Support | English |
| Support/Documenation | Medium |
| Maintainability | Medium |

Table 2.7: Tacotron evaluation

| Characteristic | Result |
|---|---|
| Information Privacy | High |
| Intelligibility | Good |
| Naturality | Medium |
| Offline Capability | Yes |
| Language Support | No pre-trained model available |
| Support/Documenation | Good |
| Maintainability | Good |

Table 2.8: Mimic 2 evaluation

ture and computation resource to train new models to provide support for additional languages. Nevertheless, the previously mentioned projects allow to customise the source code and run the models offline. Overall, Tacotron represents a state-of-the-art speech synthesis algorithm, but the usability and the maintainability suffer from the fact that neither the source code nor the training dataset was published (see 2.7).

#### 2.2.3.2 Mimic 2

MyCroft AI claims to develop the first open-source voice assistant[49]. The company uses an improved version of the Tacotron algorithm outlined in the previous chapter. It utilises the implementation of Keith Ito[35] and applied various improvements to his code base. Mimic 2 improves through its support and maintenance of MyCroft AI the evaluation slightly in comparison to the implementation of Keith Ito. However, MyCroft AI does not offer any pre-trained models which increase the effort to use Mimic 2 significantly. Table 2.8 discusses the results of the evaluation of the algorithm and shows the mentioned improvements.

#### 2.2.3.3 DeepVoice 3

The team of Ping et al. proposes Deep Voice 3, a fully convolutional text-to-speech model[53]. The model utilises a sequence-to-sequence approach together with a dot-product attention mechanism. The architecture follows the common structure by consisting of a *Encoder*, and *Decoder* and a *Converter*. The *Encoder* converts the characters or phonemes in vector representation. After resizing this representation, the Decoder extracts time-depended features through various convolutional blocks. The very same features are used to form the attention key vector by projecting them back to the embedded dimension. The attention key vector together with the original input embeddings resulting finally the context vector, which represents local as well as long term context information.

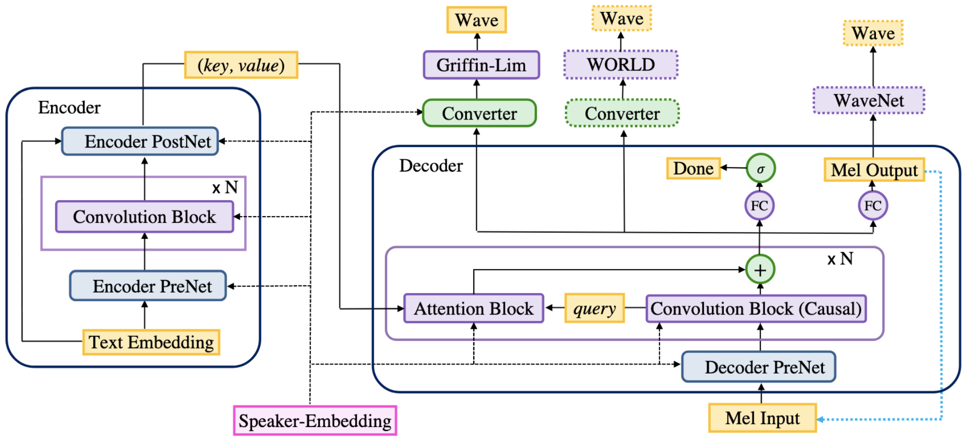


Figure 2.7: DeepVoice 3 Model Architecture (Retrieved from[53])

As shown in figure 2.7, the *Decoder* uses causal convolutional blocks together with the computed attention vector of the *Decoder* to generate in an autoregressive fashion audio. Thereby, the authors chose mel-band log-magnitude spectrogram as the audio frame representation. Based on the autoregressive fashion, the *Decoder* does exclusively consider past audio frames by employing causal computation blocks. The attention mechanism uses a dot-product of a query vector which represents the hidden states of the Decoder and per-timestep key vector to compute the required weights. Finally, the *Converter* computes the parameters required by the synthesiser. The framework supports various backends to synthesis the computed parameters. Ping et al. concluded that the WaveNet Vocoder produces the most natural audio output.

Table 2.9 presents the evaluation of the DeepSpeech 3 model. Similar to the previous text-to-speech frameworks, the source code is published in the form of an open-source project[60] and a scientific paper documents the architecture of the model, respectively. Several sound samples of pre-trained models, as

| Characteristic | Result |
|---|---|
| Information Privacy | Yes |
| Intelligibility | Good |
| Naturality | Medium |
| Offline Capability | Yes |
| Language Support | English |
| Support/Documenation | Medium |
| Maintainability | Good |

Table 2.9: DeepSpeech 3 evaluation

well as the models itself, are available for download[59]. The intelligibility and naturality of the resulting audio sequences are comparable with the results from Tacotron 2.2.3.1. Furthermore, information privacy is guaranteed based on the fact that the source code is accessible and executable offline.

#### 2.2.3.4 Summary

This chapter evaluates speech-to-text algorithms with a particular focus on intelligibility and naturality of the audio output. Tacotron describes a cutting-edge end-to-end trainable model based on a sequence-to-sequence architecture. Several implementations of this algorithm are available in the form of open-source projects. However, neither of them were able to reproduce the original results. Mimic 2, an open-source project from MyCroft AI, uses the Tacotron implementation of Keith Ito to apply several improvements. This project is under current development and profits of the maintaining and usage of the MyCraft AI community. Last but not least, Deep Voice 3 represents a fully convolutional speech synthesis model which computes intelligible and natural speech sequences. The authors are offering various models as downloads which can be directly used or adapted for further training. Overall, the evaluation shows that the given requirements are massively limiting the set of possible frameworks. Various providers like Google, Amazon and Microsoft are offering online speech synthesis services outperforming the presented models in both intelligibility and naturality. Considering the Tacotron model, multiple developers tried to reproduce the results given by audio samples but failed because the authors did not describe every parameter as well as used an internal dataset. This leads to the conclusion that currently, no offline capable speech-to-text framework satisfied the defined requirements adequately. However, projects like Mimic 2 positioning themselves as promising future solution. Considering the fact, that the first version of the speech assistant implements only a subset of the features, a Limited Domain Synthesis as described in chapter 1.3.1 embodies the best trade-off between development effort and quality of the speech sequences. A more sophisticated solution can substitute the

Limited Domain Synthesis during the further process of this project.

## 2.3 System Architecture

This chapter presents the design and implementation of the system. The *Services* section describes the implemented services and their relationships with each other. Chapter 2.3.2 explains the deployment mechanism of the project.

### 2.3.1 Services

Figure 2.8 presents the overall system architecture deployed on the robot. Starting from the bottom, the *Board Support Package* describes a set of drivers and scripts, allowing the micro-services to communicate with the peripherals of the robot itself. The *Speech-to-Text* service offers over a defined interface the possibility to transform a wave encoded sound-file into a sequence of characters. Following the processing pipeline, the *Rasa NLU* uses the output from the previous service to fill in the entities and determine the intention of the request. The service forwards this collection of information to the *Rasa Core* module. *Rasa Core* decides based on its input and the current conversation state which is stored in the *Conversation Tracker Database* which action shall be triggered. The service can trigger action by calling the *Rasa Action Server*. Last but not least, the *Text-to-Speech* service synthesises the given response and send the created sound-file to the *Board Support Package*. The following paragraphs describe the implementation of each micro-service.

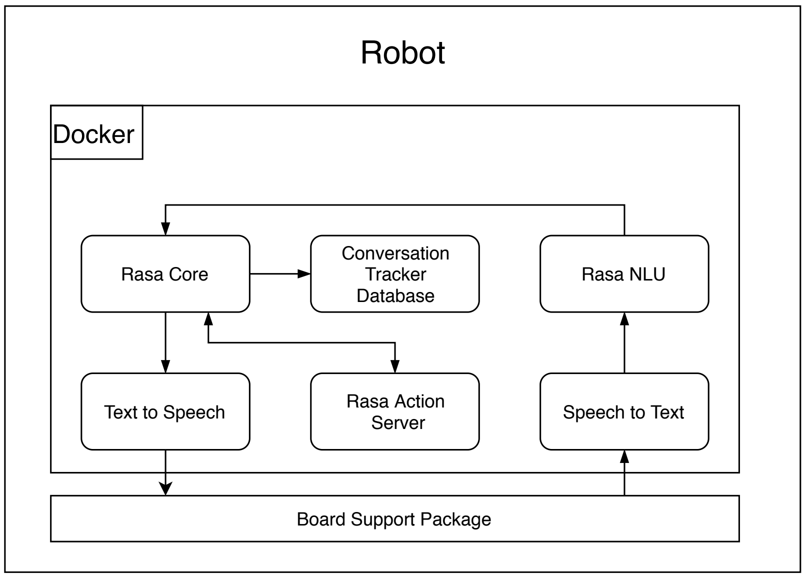


Figure 2.8: Micro-Service Architecture Overview

**Board Support Package** This module provides an abstraction layer between the docker micro-services and the robots peripherals. Currently, the speech assistant requires a microphone and speakers to interact with the user. The internal computation unit of the robot communicates with those two devices over a USB bus. The computation unit itself runs a Linux distribution which supports general audio drivers and devices. PulseAudio[27], a sound system for POSIX OSes, serves as middleware to stream the audio data from the host system to the docker containers. The middleware must be installed on both, host and client and offers there a native representation of the audio device within the Linux environment which is beneficial because most frameworks require a native Linux audio device source as an input. In general, the abstraction layer offers a clean yet straightforward opportunity to map other devices in the docker environment.

**Text-to-Speech Service** This service implements the text-to-speech functionality by using the python package *SpeechRecognition*[5]. This package offers a unified interface to various speech recognition frameworks which simplify the effort to use those algorithms as well as replace them later on. The package accesses directly over the *PyAudio*[34] package the Linux standard audio input device which represents in our case the *PulseAudio* streaming device. The *SpeechRecognition* package determines an environment noise threshold automatically, which is used to filter the noise and recognise the beginning of an utterance. The service employs multiple threads to allow ongoing speech recognition. The first thread runs the audio collection routine of the *SpeechRecognition* package. Each time a novel utterance is recognised, the package calls a callback function which starts a new thread. The thread transforms the given audio chunks by calling the underlying *CMUSphinx* framework. The service sends this sequence of characters within an HTTP message to its consumers.

**Rasa Core Service** The *Rasa Core* service functions as the central control unit of the whole *Rasa* framework. Figure 2.9 discusses the sequence of calls originated from a single request issues by a user request. The *Speech-to-Text* service sends the transformed utterance to the *Rasa Core* service, which forwards the call to the *Rasa NLU* module. Based on the response of the NLU, the *Rasa Core* service emits the correspondent actions by calling the *Rasa Action Server* service. Finally, the response is generated and send to the *Text-to-Speech* service. Regarding this order, the primary function of this service consists of predicting the next state of the current conversation and hence triggering the individual actions. The training data, so-called stories, are the essential input to train the machine learning model in predicting the required conversation states. Listing 2.1 shows an example of such a story.

```
## take indirect order
* mood_confirm
    - action_enumerate_products
* choose_product
    - action_check_availability
    - utter_anything_else
> chose_product
```

Listing 2.1: Rasa Core - Story File

The story file aims to represent a possible conversation between the user and the assistant. Each line which starts with either an asterisk or a hyphen describes an utterance from the user or the assistant respectively. The *Rasa Core* service recognises based on the response from the *Rasa NLU* which of the learnt stories might fit the best. Based on this, the service executes the actions below the recognised intents to response to the user request.

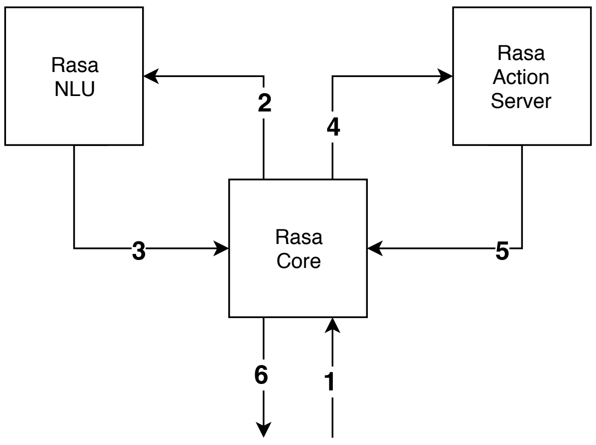


Figure 2.9: Rasa Framework - Call Sequence

**Rasa NLU Service** This service provides the natural language understanding functionality of the robot. The framework offers various algorithms to perform the intent matching and the entity parsing, respectively. The training process of the model requires two substantial information. The configuration file defines the components of the processing pipeline, and a set of files serves as training data. An example of such a training data entry is shown in listing 2.2.

```
{
  "text": "show me german restaurants",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 8,
      "end": 15,
      "value": "german",
      "entity": "cuisine"
    }
  ]
}
```

Listing 2.2: Rasa NLU - Training Dataset

The JSON encoded data structure provides three different information. Firstly, the *text* field provides the given input text. The *intent* field described the overall intention of this request and the *entities* field documents a set of entities. The latter contains the start and end position within the given input text string as well as the value and name of the entity itself. The NLU Service provides an HTTP interface which accepts requests of JSON encoded messages to interact with other services.

**Rasa Action Server** This service serves as an abstraction layer which provides the implementation of customised actions. Generally, actions can be implemented in every programming language and do not need to run on the machine itself. The *Action Server* uses an HTTP interface which makes it completely independent from the other services.
A callback function realises an action executed through an incoming HTTP request. This request consists of the name of the action, the current domain and a reference to the dispatcher and tracker object (see listing 2.3). The dispatcher reference enables the action to interact with the user in the form of generating dynamic responses, on the other hand, the tracker allows the action server to fetch information about the history of the conversation. The following listing 2.3 shows an exemplary implementation of a custom action which checks the product availability by fetching the information from a database (in this case, a simple python array).

```
class ActionCheckAvailability(Action):
    def name(self):
        return "action_check_availability"

    def run(self, dispatcher, tracker, domain):
        intent = tracker.get_latest_entity_values('product')
        if not intent:
            logger.info("No intent recognised")
            dispatcher.utter_template('utter_default', tracker)
            return []
        logger.info("Chosen Product: {}".format(intent))
        products = ['water', 'nuts', 'grapes', 'wine', 'apple']
        if intent in products:
            dispatcher.utter_message(
                "Sure, I added a {} to your cart!".format(intent))
        else:
            dispatcher.utter_template('utter_not_available', tracker)
        return []
```

Listing 2.3: Rasa Action Server - Action Implementation

**Conversation Tracker Database Service** The *Rasa* framework supports numerous types of databases to maintain conversation tracking. The robot itself employs already MongoDB[47] instances which is known to be lightweight, reliable and easy to maintain.
The *Rasa Core* conversation tracker is configurable through a file which is shown below in listing 2.4. The configuration file includes information about the URL and the credentials to access the database. Once configured, the *Rasa* framework uses the employed database as the central conversation tracking module.

```
action_endpoint:
  url: http://action_server:5055/webhook
nlu:
  url: http://rasa_nlu:5000
tracker_store:
  store_type: mongod
  url: mongodb://mongo:27017
  username: trackerUser
  password: trackerPassword
```

Listing 2.4: Rasa Core - Configuration

**Text-to-Speech Service** This service implements the *Text-to-Speech* module which employes *Swagger*[61], a standardized schema to implement and

document RESTful APIs. This schema does only provide the structure of the interface which needs to be deployed by a web server application. In general, the service offers a webhook, which starts the text-to-speech algorithm. As stated in section 2.2.3.4, a simple *Limited Domain Synthesis* embodies the first implementation of this module. Therefore, prepared sound utterances are stored within the module. The triggered webhook compares the generated response from the *Rasa Core* call with its internal database and records the chosen file. Thereby, the service accesses the internal audio speaker device, which is a virtual representation of the *PulseAudio* streaming device.
Besides, the *Speech-to-Text* module grants three different policies to be configured. These policies manage the behaviour of concurrent requests if the user interaction is faster than the duration of the last response. For instance, the user may interrupt the robot based on a misunderstanding. Following policies are supported:

1. **Purge**: The module finishes the current response, purges the queued response and finally plays the latest correspondent sound file.

2. **Add**: The system adds the response to the queue.

3. **ASAP**: The module interrupts the current playback, clears the queue and plays the latest response.

### 2.3.2 Deployment

A modern software development process requires an *Continuous Delivery* deployment process. This paradigm ensures through implemented automation pipeline that the project is continuously built, tested and deployed, which results in three major advantages. The pipeline implements a standardised routine which automates repetitive and especially time-consuming work. Furthermore, the pipeline implementation serves as documentation of the overall building process. This leads to improved productivity and simplifies additional testing of commits and features.
In the context of the developed system, a *Continuous Delivery* pipeline is used to build and run tests on each of the described micro-services. Docker, the chosen platform virtualisation, serves as an essential component by providing a clean and pre-defined environment to build the micro-services. In general, the pipeline consists of three distinct stages.

1. **Lint**: The linting stage guarantees that the structure and the source code itself meets the quality standards of the company.

2. **Build**: The building stage utilised the *Docker Engine* to build the micro-services by using the defined *Dockerfiles*.

3. **Test**: This stage aims to run various tests to ensure the functionality as well as the integrability of the services.

4. **Release**: This stage pushes the built and tested *Docker Containers* to the internal *Docker Registry* of the company from which the containers are distributed to the robots.

The implemented pipeline offers another feature which is responsible for training the machine learning models automatically. This functionality is part of the *Build* stage and is triggered only in the cases that the training dataset of the corresponding machine learning model has been changed. This ensures that developers without a deeper understanding of the training process are able to update the models of the system. Furthermore, the pipeline assures that each new model is versioned together with the dataset and the corresponding *Docker Container*.

Chapter 3

# Evaluation

This chapter covers the evaluation of the implemented system. Section 3.1, Satisfaction of Objectives, compares the implementation with the objectives defined in section . Section 3.2, Comparison with Other Systems, evaluate how the implemented system performs in comparison to a state-of-the-art device.

## 3.1 Satisfaction of Objectives

This chapter attempts to evaluate the designed and implemented system with the previously defined objectives in chapter .
The developed speech assistant is part of a robot which provides hospitality in hotels. One of the greatest challenges of the robot is the insufficient wireless internet connection throughout the hotel. Therefore, the offline capability of the whole system must be ensured. In chapter 2.2, the capability to work entirely offline led among others to the choice of CMUSphinx as the speech recognition framework, Rasa as the intent matcher platform and especially to the limited domain synthesis as the text-to-speech module.
The ability to work offline automatically provides a higher standard of privacy and security of the system. This fact also implies that there is no exchange with third-parties of sensible user-specific data nor anonymous generated performance data.
The selected frameworks are characterized by being open-source developed, which encourages to customize and adapt the frameworks to the specific use case. This confirms the ability of the system to be able to tailor its behaviour and generated responses to fit perfectly into its later environment. For instance, CMUSphinx offers the possibility to train its model with a customized vocabulary, which significantly improves the accuracy of the model.
Furthermore, the chosen micro-service architecture underlines this paradigm of a highly flexible and customizable system. This architecture ensures another level of security by implementing the "security through separation" concept, which also outperforms monolithic systems in scalability and complexity.

The ability to adapt to a specific environment together with other parameters like the accuracy of the modules and the overall system latency contribute to the overall user experience of the system.
Conclusively, the developed system provides a solution which fits the previously defined objectives. Nevertheless, the strict requirements prevent the system from using the latest algorithms, which is noticeable in the user experience of the system.

## 3.2 Comparison with Other Systems

This chapter compares the implemented system with a state-of-the-art device of a third-party. This measurement is important to justify whether the development of a novel system is worth it in comparison to use an already existing framework.
Several companies offer devices which are including speech assistant capabilities. One of the most successful assistants is called Alexa, which is developed by Amazon Incorporation. Alexa is a cloud-based virtual speech assistant which can be included in many devices like smart home controls, mobile phones and speakers. The greatest advantages of Alexa in the context of this evaluation is the so-called Alexa Skills Kit. This development kit allows developers to implement custom action for Alexa, which can be executed over a speech command.
The following subsection 3.2.1 evaluates the Alexa speech assistant based on the same objectives as used in chapter 3.1. The final comparison between the two systems is discussed in chapter 3.2.2.

### 3.2.1 Amazon Alexa

Amazon presents with Alexa, a cloud-based speech assistant. This assistant requires a constant connection to the internet without it is not able to respond to user requests. On the downside, the assistant violates one of the essential objectives.
Nevertheless, this constant connection enables the assistant to employ more sophisticated algorithms to process the request. This, overall, leads to the fact that Alexa offers a great user experience based on its cloud backend as well as the feedback of millions of users.
Amazon offers with its broad server portfolio a unique opportunity to extend and scale a customized Alexa application. However, Amazon uses this requirement to obtain and gather statistical data to improve and monitor their algorithms. For this reason, the security, as well as the privacy of the system, may suffer under these circumstances.

### 3.2.2 Summary

As stated in chapter 3.1, the developed system fulfils the defined objectives. However, the user experience suffers because more sophisticated models would require additional resources in the form of a stable and constant internet connection.
In comparison, Amazon Alexa uses a cloud-backend which improves the user experience with the speech assistant. Nevertheless, this introduces new challenges, like data privacy and security issues. Conclusively, the developed system represents a hand-tailored solution for a particular use case which would not be reproducible with Amazon Alexa without reconsidering the objectives of the application.

Chapter 4

# Discussion

This chapter discusses the outcome and results of this thesis and outlines future work opportunities for this thesis.

## 4.1 Research Question Satisfaction

The fundamental research question of this thesis attempts to justify the development of a hand-tailored speech assistant application. The origin of this question based on the fact that the majority of applications requires a stable and constant internet connection to be able to generate responses. Furthermore, those assistants are developed to work for a broad target group.
Nevertheless, the usage of such systems in products requires the ability to adapt to a specific audience in the form of a particular vocabulary, phrases and behaviour in standard situations.
This leads finally to the conclusion concerning the result of this thesis that it is possible to develop a hand-tailored application which outperforms its competition within a particular, defined use-case. Nonetheless, as soon as the use-case changes in the form of loosening the requirements, the developed system cannot compete with its competitors.

## 4.2 Hardware Specification

This section yields to discuss an opportunity to improve the overall performance of the system without altering the system itself. As mentioned in chapter 1.1.1, one of the challenges of speech recognition is the signal distortion through undesired noise. The first stage of a speech recognition pipelines aims to minimise the influence of such artefacts on the accuracy of the output.
Nevertheless, if the distortion reaches a specific level, the module itself cannot manage to produce reasonable text sequences anymore, which interrupts the whole user experience. Numerous vendors of speech assistant capable devices

started to install multiple microphones to overcome this issue. By sampling multiple microphones with a synchronous clock signal, it is possible to filter distortions like noise much better than a system with a single audio input. Furthermore, a whole matrix of microphones can be used to determine the direction and the distance of the speaker, which might be a beneficial input for the assistant.

## 4.3 Deployment

The current deployment setup consists of a pipeline which builds, tests and releases new versions of the system. This setup guarantees a minimum quality of builds by automatically running software test. The current first version of this pipeline implements only basic testing mechanism like an overall system startup. This may be extended in future work to ensure the functionality of each service and its defined communication interface. Therefore, various mockup containers would work as a counterpart to test the communication modules of a specific service. The mockup container simulates the environment of a given application to test its behaviour. This could ensure the integration of each micro-service in the system and further increase the quality of the source code.

## 4.4 Chatbot Ability

The implemented system uses a micro-service architecture which enables the system to be highly adaptable and scalable. This advantage offers the possibility to run the system on every kind of platform, which is an essential feature of the chosen virtualisation.
Chatbots are representing a growing development within the field of customer service, which automatically responds to a standard user request. The developed system offers the ability to be adapted as a chatbot application, which increases its value by almost zero migration effort. The internally used intent matching algorithms allows the interaction with third-party messaging services like Slack and Facebook Messenger.
This example shows the ability of the system to adapt to new use-cases by minimising the effort of changes are adaptations as well as outlines a future work opportunity.

# Conclusion

This thesis attempts to answer the given research question whether a system can be developed which fulfils the strict requirements of service robots by outperforming commercial applications. Therefore, the thesis introduces the theoretical foundation in the three subfields (speech recognition, intent matching, speech synthesis), which compose a modern speech assistant. Various frameworks of each discipline are evaluated, which serve as fundamental components of the application. The evaluation uses for each subfield a custom-tailored metric to compare the corresponding frameworks. The thesis proposes a micro-service architecture which proves to be scalable and highly adaptable. Furthermore, the architecture enables the system to be used in various use-cases because of the underlying virtualisation which can be deployed on robots as well as on server clusters. Finally, the developed system outperforms within its precise defined environment, a popular commercial speech assistant. Nevertheless, the general performance of the system must be improved to guarantee a sufficient level of user experience. This may require an adaption of the requirements which restricts the selection of framework and therefore, the quality of the system dramatically.

# Bibliography

[1] Alex Barron. *Barron Alex - Tacotron Implementation.* URL: `https://github.com/barronalex/Tacotron` (visited on 04/29/2019).

[2] Nuha Alshuqayran, Nour Ali, and Roger Evans. "A systematic mapping study in microservice architecture". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA).* IEEE. 2016, pp. 44–51.

[3] Dario Amodei et al. "Deep speech 2: End-to-end speech recognition in english and mandarin". In: *International conference on machine learning.* 2016, pp. 173–182.

[4] Dimitra Anastasiou. "Survey on Speech, Machine Translation and Gestures in Ambient Assisted Living." In: *2011-Paris* (2011).

[5] Anthony Zhang. *Speech recognition module for Python, supporting several engines and APIs, online and offline.* URL: `https://github.com/Uberi/speech_recognition` (visited on 05/14/2019).

[6] M. A. Anusuya and S. K. Katti. "Speech Recognition by Machine, A Review". In: *International Journal of Computer Science and Information Security, IJCSIS, Vol. 6, No. 3, pp. 181-205, December 2009, USA* (Jan. 13, 2010). arXiv: `http://arxiv.org/abs/1001.2267v1 [cs.CL]`.

[7] The Kubernetes Authors. *Kubernetes.* URL: `https://kubernetes.io/` (visited on 04/15/2019).

[8] Adrien Ball. *An Introduction to Snips NLU, the Open Source Library behind Snips Embedded Voice Platform.* URL: `https://medium.com/snips-ai/an-introduction-to-snips-nlu-the-open-source-library-behind-snips-embedded-voice-platform-b12b1a60a41a` (visited on 04/22/2019).

[9] Tom Bocklisch et al. "Rasa: Open Source Language Understanding and Dialogue Management". In: (Dec. 14, 2017). arXiv: `http://arxiv.org/abs/1712.05181v2 [cs.CL]`.

[10] Antoine Bordes, Y-Lan Boureau, and Jason Weston. “Learning end-to-end goal-oriented dialog”. In: *arXiv preprint arXiv:1605.07683* (2016).

[11] Martin Brinkmann. *Amazon blunder leaks Alexa data to other customer.* URL: `https://www.ghacks.net/2018/12/21/amazon-blunder-leaks-alexa-data-to-other-customer/` (visited on 05/10/2019).

[12] Thanh Bui. “Analysis of docker security”. In: *arXiv preprint arXiv:1501.02967* (2015).

[13] Carnegie Mellon University. *CMUSphinx.* URL: `https://cmusphinx.github.io/` (visited on 04/18/2019).

[14] Carnegie Mellon University. *CMUSphinx Github Repositories.* URL: `https://github.com/cmusphinx` (visited on 04/19/2019).

[15] Carnegie Mellon University. *CMUSphinx Tutorial For Developers.* URL: `https://cmusphinx.github.io/wiki/tutorial/` (visited on 04/18/2019).

[16] Hongshen Chen et al. “A Survey on Dialogue Systems: Recent Advances and New Frontiers”. In: (Nov. 6, 2017). arXiv: `http://arxiv.org/abs/1711.01731v3 [cs.CL]`.

[17] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. “Wav2letter: an end-to-end convnet-based speech recognition system”. In: *arXiv preprint arXiv:1609.03193* (2016).

[18] Theo Combe, Antony Martin, and Roberto Di Pietro. “To Docker or Not to Docker: A Security Perspective.” In: *IEEE Cloud Computing* 3.5 (2016), pp. 54–62.

[19] Alice Coucke et al. “Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces”. In: (May 25, 2018). arXiv: `http://arxiv.org/abs/1805.10190v3 [cs.CL]`.

[20] *Deep Speech 2 Trained on Baidu English Data.* URL: `https://resources.wolframcloud.com/NeuralNetRepository/resources/Deep-Speech-2-Trained-on-Baidu-English-Data` (visited on 04/19/2019).

[21] Ratnadeep Deshmukh and Abdulmalik Alasadi. “Automatic Speech Recognition Techniques: A Review”. In: Feb. 2018.

[22] Docker Inc. *Docker.* URL: `https://www.docker.com` (visited on 04/15/2019).

[23] Docker Inc. *Docker Hub.* URL: `https://hub.docker.com` (visited on 04/15/2019).

[24] N Dragoni et al. “Microservices: yesterday, today and tomorrow.(2017)”. In: *arXiv preprint arXiv:1606.04036* (2017).

[25] Facebook Inc. *Open sourcing wav2letter++, the fastest state-of-the-art speech system, and flashlight, an ML library going native.* URL: `https://code.fb.com/ai-research/wav2letter/` (visited on 04/20/2019).

[26] David Ferris. “Techniques and Challenges in Speech Synthesis”. In: (Sept. 22, 2017). arXiv: `http://arxiv.org/abs/1709.07552v1 [cs.SD]`.

[27] freedesktop.org. *PulseAudio.* URL: `https://www.freedesktop.org/wiki/Software/PulseAudio/` (visited on 05/14/2019).

[28] Mayur R Gamit, Kinnal Dhameliya, and Ninad S Bhatt. “Classification techniques for speech recognition: A review”. In: *International Journal of Emerging Technology and Advanced Engineering* 5.2 (2015), pp. 58–63.

[29] Albert Gatt and Emiel Krahmer. “Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation”. In: *Journal of AI Research, volume 60, 2017* (Mar. 29, 2017). arXiv: `http://arxiv.org/abs/1703.09902v4 [cs.CL]`.

[30] Google Inc. *Dialogflow SDK Documentation.* URL: `https://dialogflow.com/docs/sdks` (visited on 04/22/2019).

[31] Google Inc. *Tensorflow - An end-to-end open source machine learning platform.* URL: `https://www.tensorflow.org/` (visited on 04/29/2019).

[32] HashiCorp. *Vagrant.* URL: `https://www.vagrantup.com` (visited on 04/15/2019).

[33] Matthew Henderson. “Machine Learning for Dialog State Tracking: A Review”. In: *Proceedings of The First International Workshop on Machine Learning in Spoken Language Processing.* 2015.

[34] Hubert Pham. *PyAudio.* URL: `https://people.csail.mit.edu/hubert/pyaudio/` (visited on 05/14/2019).

[35] Keith Ito. *A TensorFlow implementation of Google’s Tacotron speech synthesis with pre-trained model (unofficial).* URL: `https://github.com/keithito/tacotron` (visited on 04/28/2019).

[36] Mike Jeffs. *OK Google, Siri, Alexa, Cortana; Can you tell me some stats on voice search?* 2018. URL: `https://edit.co.uk/blog/google-voice-search-stats-growth-trends/` (visited on 03/30/2019).

[37] Ann Mary Joy. “Performance comparison between linux containers and virtual machines”. In: *2015 International Conference on Advances in Computer Engineering and Applications.* IEEE. 2015, pp. 342–346.

[38] Keith Ito. *Keith Ito - Github Profile.* URL: `https://github.com/keithito` (visited on 04/29/2019).

[39] Veton Këpuska and Gamal Bohouta. “Comparing speech recognition systems (Microsoft API, Google API and CMU Sphinx)”. In: *Int. J. Eng. Res. Appl* 7.03 (2017), pp. 20–24.

[40] Kyubyong Park. *Kyubyong - Tacotron Implementation.* URL: `https://github.com/Kyubyong/tacotron` (visited on 04/29/2019).

[41] Paul Lamere et al. “Sphinx-4: A flexible open source framework for speech recognition”. In: *Sun Microsystems, Report Number: TR-2004-139* (2004).

[42] Xinyu Lei et al. “The Insecurity of Home Digital Voice Assistants - Amazon Alexa as a Case Study”. In: (Dec. 9, 2017). arXiv: `http://arxiv.org/abs/1712.03327v2 [cs.CR]`.

[43] Akansha Madan and Divya Gupta. “Speech feature extraction and classification: A comparative review”. In: *International Journal of computer applications* 90.9 (2014).

[44] Michael Nguyen. *Mimic2 is LIVE!* URL: `https://mycroft.ai/blog/mimic-2-is-live/` (visited on 04/29/2019).

[45] Terence Mills. *The Impact Of Artificial Intelligence In The Everyday Lives Of Consumers.* 2018. URL: `https://www.forbes.com/sites/forbestechcouncil/2018/03/07/the-impact-of-artificial-intelligence-in-the-everyday-lives-of-consumers` (visited on 03/30/2019).

[46] Maali Mnasri. “Recent advances in conversational NLP : Towards the standardization of Chatbot building”. In: (Mar. 21, 2019). arXiv: `http://arxiv.org/abs/1903.09025v1 [cs.CL]`.

[47] MongoDB Inc. *Mongo Database - Webpage.* URL: `https://www.mongodb.com/` (visited on 05/15/2019).

[48] Nicolas Morales, Zhenyu Tang, and Dinesh Manocha. “Receiver Placement for Speech Enhancement using Sound Propagation Optimization”. In: (May 29, 2018). arXiv: `http://arxiv.org/abs/1805.11533v3 [cs.SD]`.

[49] Mycroft AI Inc. *Who is Mycroft?* URL: `https://mycroft.ai/about-mycroft/` (visited on 04/22/2019).

[50] PaddlePaddle. *DeepSpeech2 PaddlaPaddle Implementation.* URL: `https://github.com/PaddlePaddle/DeepSpeech` (visited on 04/19/2019).

[51] PaddlePaddle. *PaddlePaddle (Parallel Distributed Deep Learning).* URL: `https://github.com/PaddlePaddle/Paddle` (visited on 04/19/2019).

[52] Ken Peffers et al. “A design science research methodology for information systems research”. In: *Journal of management information systems* 24.3 (2007), pp. 45–77.

[53] Wei Ping et al. “Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning”. In: (Oct. 20, 2017). arXiv: `http://arxiv.org/abs/1710.07654v3 [cs.SD]`.

[54] Vineel Pratap et al. “wav2letter++: The Fastest Open-source Speech Recognition System”. In: *arXiv preprint arXiv:1812.07625* (2018).

[55] Page Laubheimer Raluca Budiu. *Intelligent Assistants Have Poor Usability: A User Study of Alexa, Google Assistant, and Siri.* 2018. URL: `https://www.nngroup.com/articles/intelligent-assistant-usability/` (visited on 03/30/2019).

[56] Rasa Technologies GmbH. *Build contextual chatbots and AI assistants with our open source conversational AI framework.* URL: `https://rasa.com/docs/` (visited on 04/22/2019).

[57] Rasa Technologies GmbH. *Rasa Webpage.* URL: `https://rasa.com/` (visited on 04/22/2019).

[58] Red Hat Inc. *Ansible.* URL: `https://www.ansible.com` (visited on 04/15/2019).

[59] Ryuichi Yamamoto. *An open source implementation of Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning.* URL: `https://r9y9.github.io/deepvoice3_pytorch/` (visited on 04/30/2019).

[60] Ryuichi Yamamoto. *PyTorch implementation of convolutional neural networks-based text-to-speech synthesis models.* URL: `https://github.com/r9y9/deepvoice3_pytorch` (visited on 04/30/2019).

[61] SmartBear Software. *Swagger - Webpage.* URL: `https://swagger.io/` (visited on 05/15/2019). (accessed: 2019-05-15).

[62] Snips. *Which languages are supported?* URL: `https://docs.snips.ai/additional-resources/faq/general-faq` (visited on 04/22/2019).

[63] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems.* 2014, pp. 3104–3112.

[64] Ayushi Y Vadwala et al. "Survey paper on Different Speech Recognition Algorithm: Challenges and Techniques". In: *Int. J. Comput. Appl.* 175.1 (2017), pp. 31–36.

[65] Gary Wang. "Deep Text-to-Speech System with Seq2Seq Model". In: (Mar. 11, 2019).

[66] Tiancheng Zhao and Maxine Eskenazi. "Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning". In: (June 8, 2016). arXiv: `http://arxiv.org/abs/1606.02560v2 [cs.AI]`.