Multi Modal Digital Assistant using Text, Speech and Video Interaction

Pie de Boer, Tom Bakker, Natalia Czaban, Aarón Mulvey Izquierdo, Loris Podevyn, Vitaly Tičkovs, Dejan Vlasakov

Maastricht University
Department of Advanced Computing Sciences
Data Science and Artificial Intelligence

CONTENTS

Abstract—This paper investigates how to build a multi-modal digital assistant. It explores tasks, such as, natural language processing techniques, audio and image processing techniques. A basic skill parsing algorithm was used to create a 'simple' chat system, after which more complex algorithms and parsing techniques were used. One of them being Context Free Grammar (CFG), which is parsed using a recursive algorithm, transforming semi-structured data into a knowledge base. This study demonstrates that our custom tailored tree based CFG parsing algorithm performs better than the traditional Cocke–Younger–Kasami (CYK) with respect to memory. In addition, Rasa framework was explored in order to see, if a chat system could be improved asking additional questions to fill the slots. Training upon given skills and gaining natural language understanding of the inputs. Image processing techniques like Haar cascade classifiers and local binary pattern histogram (LBPH) were implemented, in order to detect faces and eventually 'identify' multiple different users. The parameters for these algorithms, were optimized using grid search. The results showed that Haar cascade outperform the LBPH for our application. Siamese Neural Networks, eigenfaces and Facenet for face detection were also investigated. This study demonstrates that FaceNet embedding showcases the highest accuracy, whereas eigenfaces with a relatively low accuracy proved to be the fastest and most consistent algorithm with respect to runtime. The Siamese neural network showed to be a valid option for classifying faces, as it sits in between FaceNet and eigenfaces. In this study, we evaluated the performance of the Mozilla DeepSpeech and Meta Wav2Vec-2.0 speech recognition models under different noise conditions. Additionally, speaker identification using Gaussian Mixture Models with Mel Frequency Cepstral Coefficient embedding was implemented. We applied audio processing using the TarsosDSP framework and implemented noise gate methods to minimize background noise. Our evaluation, based on metrics such as Word Error Rate (WER) and Rouge-L F1 Score, found no significant difference in the transcription accuracy of both models, confirmed by Wilcoxon rank tests. However, we noted a higher performance variance in DeepSpeech. Furthermore, we found that both models' performance was most negatively impacted by noise that closely resembled white noise. The findings from our speaker identification model were consistent with these results. Our study underlines the need for further work on speaker identification and speech recognition models to increase their robustness to various noise environments.

1 INTRODUCTION

Context

The following study demonstrates the use of various Natural Language Processing (NLP) techniques in combination with image, audio and video processing approaches in order to create a multi-modal digital assistant. The assistants' functionality is in line with similar services, such as; Amazon Alexa, Google Home and Apple Siri. In contrast to being aimed at the general population, it is specifically designed for Maastricht University Department of Advanced Computing Sciences (DACS) students.

Information Retrieval and Context Free Grammars

Furthermore, the assistant will have the ability to perform a range of tasks other than responding to questions. The user will find it possible:

To enable the comprehension of a given sentence as the assistant utilises Information Retrieval Techniques, along with Context Free Grammar (CFG) [7].

Information retrieval techniques allow for the retrieval of relevant information from a large data source, to answer a query given by the user. This is analogous to a person accessing their memory in order to answer a certain question. To enable this ability, we had to structure our semi-structured data in such a way it allows for this process.

CFG allows the assistant to understand the underlying grammatical features and structure of a language. It consists of a set of rules that state how to generate valid sentences [9]. Specifically, the assistant uses CFG Parsing [1]. Parsing

algorithms are used in the context of CFG to determine the syntactic structure of a sentence.

State-of-the-art research has been combined the traditional Cocke–Younger–Kasami (CYK) algorithm with neural networks to optimize the parsing space's search process [47]. While conventional parsing algorithms handle complex grammars, the approach in this study adopts a compact design with a limited set of production rules. Due to the compactness of the grammar used, the Top-Down parsing method used in this project, despite being recursive and potentially time-consuming, is practically effective for the circumstances of this project.

Face Detection, Speech Recognition and Speaker Identification

As a multi-modal assistant, audio, image and video processing techniques have been implemented. The assistant has the ability to access the user's webcam and detect if a user is sitting in front of the webcam. The assistant can furthermore recognize who that person is, given the classifier is trained for this task.

In the field of image and video processing, current developments moved away from traditional classifier techniques, with the seminal paper from 2001 about Haar cascade classifiers, to more 'advanced' deep learning techniques using convolution neural networks (CNN). Recent literature showed that the CNN approach allows for the highest performance of all techniques [32].

On the audio front, the user has the ability to interact with the assistant on a speech basis. A speech recognition model of Meta (Wav2Vec-2.0) has been implemented for real-time transcription of speech provided by the user using the microphone of there device. Recent developments in speech recognition using deep learning models have shown considerable improvements in lowering the word error rates (WER) and therefore making speech recognition usable for practical applications [34]. Furthermore, we trained and implemented an extra model that allows to identify the speaker based on text independent speech. It is important to note that this model was solely trained for a subset of writers of this report, therefore a small group of 5 people. This provided a very personalized experience for the user. This is not only an exciting technique, but serves great functionality for bio-metric identification, where it could be used to access office spaces, monitor employee attendance or help people with dementia identify speakers [17].

Research Questions For our paper, we investigated the following research questions:

- Which algorithm is the most time efficient in order to parse context free grammars?
- Can multi-turn dialogue system improve the information retrival from the user?
- Which classifiers between Haar and LBPH has the highest accuracy?
- Do machine learning strategies for face recognition outperform traditional classifiers like Haar and LPBH?
- Which face recognition algorithm proofs to be most accurate out of: Eigenfaces, Siamese neural network and FaceNet?
- Which face recognition algorithm should be chosen for real-time applications?
- Which open-source speech recognition (Mozilla vs Meta) is most accurate, fast and robust?
- Which types of noise are most detrimental for speech recognition and speaker identification models?

2 Methods and Implementation

2.1 Graphical User Interface

In this project the Graphical User Interface (GUI) was build using the JavaFX library. Throughout the project, several classes use the above mentioned library namely: ChatWindow, DisplaySkills, HelpWindow, LoginScreen, SkillEditor, Skill Detalis, TestSkills and StageManager. Throughout the whole application user will be able to see its camera on screen.

2.1.1 Chat Window

Upon starting the application, the user is met with a login screen (See Appendix A Figure 21) where they can input predefined login details. The user will only be able to log in if there is a person sitting in front of the camera. After signing up, they get redirected to the chat screen (See Appendix A Figure 22) where the process of interacting with the agent can begin. To interact with the agent simply type the text in the input text field and then press the submit button. Similarly to the login page, the user will be able to interact with the chat only if they are sitting in front of the camera. On the left-hand side, a person can choose a desired action, as in opening the help window if they need help with application navigation, logging out, terminating the application or opening the skill editor.

2.1.2 Skill Editor

The skill editor is divided into two main sections: defining a new skill (See Appendix A Figure 23) and editing existing ones. To create a new skill template, first define your question as follows: **Which [TRAIN] did you get to get to [COUNTRY]**, submit by pressing the button 'submit question', then define slots which are associated with key words (words surrounded by pointy brackets). To define a slot for above example type as follows: [**TRAIN**] **InterCity** [**COUNTRY**] **Poland**. To finish the user has to create desired actions which can be done in following way: **[TRAIN] Inter City [COUNTRY] Poland To get to Poland I took Intercity train.** Moreover to access the defined skill on the side menu select the Display Skills button that will redirect to a new frame with all defined skills (See Appendix A Figure 24). Upon selecting a skill, a new frame appears where the user is able to add/delete already defined actions, define new slots and actions. After editing the window the user has to press the save button.

2.1.3 CFG Editor

Similarly to the skill editor user is able to define a new CFG as well as edit the existing one. CFG editing is limited to changing/adding/deleting actions and changing/adding/deleting non-terminal terms.

2.2 Information Retrieval Processes

In order to give an output to user's input, several Natural Language Processing (NLP) techniques were implemented and tested, to see which would be the best suitable variation for this project. The main goal was to retrieve an answer to a user's input by matching it with the unique question specified in multiple 'Skill' files. The focus for this project was to make a robust system, which would account for user error in the input, for example, a typo, using synonyms for a non slot word, etc. But also the system would need to have threshold, for which everything less than the threshold would be identified as not suitable to give an answer, because the input is too obscured, does not contain enough information.

The assistant interfaces with the user on a typing basis or audio basis. The audio features include speech recognition and speaker identificaiton. The assistant has the ability to identify a user through the webcam or through means of speech identification.

For the assistant to function, a user needs to define the question they intend on asking, and the variables that can change each time. A file containing the information is called a 'Skill', and consists of three main parts:

- **Question(s):** The user sets the structure of the question, and defines 'slots'. An example question would look as follows: **Which lectures are there on "DAY" at "TIME", with "DAY"?**, "TIME" being the slots.
- **Slot(s):** The slot indicates a variable that can change whenever the questions is asked. The user is required to define different slots, which will be used in the actions to define a response. A slot will look like this: **"DAY" Monday**.
- **Action(s)** An action specifies a response to a given slot. An example of an action would be as follows: **"DAY" Saturday "TIME" 9 There are no lectures on Saturday at 9**

2.2.1 Perfect Matching

In natural language processing, perfect matching [41] has a wide variety of uses. Ranging from information retrieval to text classification, text summarizing, and question answering. In the case of our assistant, we use perfect matching for information retrieval. Once the user prompts the assistant, the perfect matching algorithm will be used to compare the given sentence to our existing database of 'skills'. If a match is found, the assistant will reply with the specified information. To perform perfect matching on a piece of text, we follow these steps. **Pre-processing:** Text pre-processing augments a given sentence in a way that is efficient for the following steps to work on. In our case, two techniques are utilised. White space normalization[30], standardizes the spacing characters in a sentence. Makes it so all empty spaces are consistent, allowing perfect matching to only consider relevant information. The second technique is Lowercase conversion[45], used for eliminating distinctions between lowercase and uppercase words.

Tokenization: This step breaks down full sentences into individual words that are relevant in the "skills" class.

Distance Measure: Tokens acquired from a prompt are matches against existing words in the database, to identify any matches. As a quantitative measure of a match, the Levenshtein distance[19] is used. Levenshtein distance tells us how many edits of a word are needed, for it to be transformed to a different word. The lower the score/distance, the closer the two words are to each other.

Threshold: As a way to specify how dissimilar two words can be, we use the following formula to set a cutoff value:

$$Threshold = \|WORD\| * 0.8$$

If a given Levenshtein values is higher than the threshold value, no match will be considered.

Output: After all relevant words are compared, a ranking list will be generated, after which the word or phrase at the top of the list will be used for further information retrieval.

2.2.2 Keywords

The Keywords technique in NLP focuses on extracting important keywords and phrases from a given prompt. In our case, the important information could be the 'SLOT' words in a sentence. Identifying those can help us return the correct reply for a given prompt. The keywords approach was implemented in the following way:

Pre-processing: Firstly, stop words are removed from the prompt. Stop words consist of common words that hold no actual meaning. Examples of stop words include "the," "is," "are," "and," "or," and so on. By removing stop words, the algorithm can return more accurate results, due to the more relevant content in the new sentence. The second preprocessing technique is lemmatization. Lemmatization is the process of reducing words to their root. An example of lemmatization in action would be: Transforming "walking", "walks", "walked" to "walk". Doing this helps us capture the core conept represented by a certain word.

Tokenization: This step breaks down full sentences into individual words that are relevant in the 'skills' class.

Distance Measure: The Jaccard distance [33] was used as a distance measure for the keywords approach. The Jaccard distance is used to asses the similarity between sets of keywords. This is done by first calculating the dissimilarity between sets of words or phrases depending on the relative size of their intersection and union. An intersection represents common keywords in both sets, while the union includes all unique keywords from the sets. The Jaccard coefficient J is then calculated by dividing the size of the intersection by the size of the union. Size referring to the number of elements in the set. Finally, we arrive to the Jaccard distance D, which is given by:

$$D = J - 1$$

J is Jaccard similarity coefficient:

$$J = (|a \cap b|/|a \cup b|)$$

The resulting value will be in the range from 0 to 1. With numbers closer to 1 indicating low similarity, and numbers closer to 0 indicating high similarity.

Threshold: By testing different values, we found out that 0.25 works best for our goals. This means that a rank list will only consists of words with a distance of 0.25 or lower.

Output: The keyword with the lowest rank/Jaccard distance will be returned by the algorithm.

2.2.3 Vectors & Vector Sequences

By converting strings to numerical vectors, more complex and efficient algorithms can be utilized. In the field of NLP, "Vectors" also known as word embeddings allow us to easily do this. This representation captures semantic and syntactic relationship between words and sentences. For this approach, the following steps were adapted for our research:

Pre-processing: Firstly, remove stop words as was done for keywords extraction, in order to only focus on the most important words in the given sentence. Secondly, use lemmatization to reduce the word's form to its root.

Vector vocabulary: Vocabulary, in the context of text analysis, refers to a collection of unique words or terms extracted from a corpus of documents. It represents the vocabulary or lexicon of words that are relevant to the specific domain or context of the text data. For our approach we constructed our own vocabulary, based on the given skills, for example, English vocabulary does not contain DACS abbreviation, but our vocabulary would.

Vectorization of a sentence: After pre-processing the sentence, the next step is to represent the pre-processed text as numerical vectors. One of our approaches was to use bag-of-words model[35]. This method assigns a numerical value to each word in the sentence. For our other approach, Vector Sequences, Word2Vec model made by Google was used[26]. It uses the combination of continuous bag of words (CBOW) and skip-gram:

- **CBOW:** The model aims to predict the target word based on the context words surrounding it. The context words are used as input to the model, and the target word is predicted as the output.
- **Skip-gram:** Skip-gram is the reverse of CBOW[26]. It aims to predict the context words given a target word. The target word is used as input, and the model predicts the context words.

During training, Word2Vec adjusts the word vectors in such a way that the vectors of similar words are close to each other in the vector space. This property allows the model to capture semantic relationships between words. For example, words with similar meanings or those that often appear together will have similar vector representations.

Distance measure: To assess the similarity within the "Vectors" approach, cosine distance measure was employed. It calculates the cosine of the angle between the vectors and provides a value ranging from 0 to 2. The formula to calculate cosine distance is as follows:

$$cosine\_distance = 1 - cosine\_similarity$$

$$cosine\_similarity = (a \cdot b)/(||a|| * ||b||)$$

Cosine similarity is computed by taking the dot product of the two vectors and dividing it by the product of their magnitudes.

For "Vector Sequences" approach, a combination of Word Movers D. + Sinkhorn algorithm[44]. Word Mover's distance (WMD) is a distance metric that measures the dissimilarity between two text documents based on the concept of earth mover's distance. It quantifies the minimum amount of "work" needed to transform the word distribution of one document into the word distribution of another. The WMD algorithm considers each word in the documents as a "word vector" and calculates the distance between them by finding the optimal pairing of words that minimizes the total "transportation cost" required to move the word vectors from one document to another[38]. The transportation cost for this project was defined as the euclidean distance or cosine distance. As for the Sinkhorn algorithm, it is a method used to efficiently compute the optimal transport plan between two sets of points, reducing the computational complexity of calculating the transportation cost[8]. The Sinkhorn algorithm is employed to solve the transportation problem efficiently and provide an approximation of the optimal transport plan. In the context of out approach, Sinkhorn's algorithm was used in combination with WMD distance measure to allow for efficient and scalable calculation of the semantic similarity between text documents[44]. It provides a balance between accuracy and computational efficiency, making it suitable for large-scale applications where computational resources are limited.

2.3 Context Free Grammar (CFG)

Context-free grammar is of vital importance in compiler design, since the implementation of a parser is achieved through context free grammars [6]. Grammar inference in general serves a very wide array of applications, think speech recognition and robotics [27]. Besides, it can be used in advanced settings, where it is combined with deep learning, in order to generate dialogues for video games [37]. This project employs a compact form of context-free grammar (CFG) known as "CFG-based skills". In this grammar, each rule is structured with a non-terminal symbol on the left-hand side and a combination of terminal and/or non-terminal symbols on the right-hand side. This arrangement sets the grammar apart from the typical format, as the rules come bundled with a set of actions. These "Actions" represent specific outputs that correspond to certain combinations of symbols in the rules.

2.3.1 CFG Parser

The parser presented in this paper is tailored to handle compact grammar. Using a top-down strategy , it generates all feasible sentences through a recursive depth-first search from the start symbol. It expands non-terminal symbols based on predefined production rules and ensures sentence validity by checking for non-terminal symbols and excluded words or placeholders. The algorithm maps each generated sentence to its corresponding action, utilizing a hash map for efficient input-output mapping. These mappings are stored in a database, enabling prompt responses to recognized inputs. The parser's overall design underscores its potential for time efficiency.

2.3.2 Chromsky Normal Form and CYK-algorithm

Besides, our custom approach, we also implemented the Cocke-Younger-Kasami (CYK) algorithm popular in literature. It allows us to determine whether a word belongs to a certain language which is formed by our grammar of interest [14]. This does induce an additional difficulty, since our 'skill-based' CFG files have to be put into Chromsky

Normal Form. This was done according a procuder with serval steps, as described in literature [9]. This is an overview of the procedure, which served as template for our exact software implementation which is tailed to work with the 'CFG based skills', see appendix X.

1) Eliminate the start symbol from right-hand sides and introduce a new start variable S_0
2) Eliminate ϵ rules
3) Eliminate rules with non-solitary terminals
4) Eliminate right-hand sides with more than 2 non-terminals
5) Eliminate unit rules

2.4 Multi-turn dialog system

Rasa is an open-source framework for developing a NLP based chat system[4]. Being open-source, it gave us a lot of opportunities to learn its applications and implement specific parts for our project. The aspects that we chose to implement were:

- Natural language understanding(NLU), it allows for accurate intent classification and entity extraction. The ability to train the NLU model on annotated data allows us to fine-tune its performance and adapt it to our specific domain of skill based chat system.
- Slot filling support, which is essential for our implementation, giving more freedom to the user of inputting a question/query. It allows to define slots, as well as automatically extract them. This ability to incorporate additional questions for slot filling enhances the accuracy and context-awareness.

In order to identify whether to ask additional questions to extract and fill additional slots we incorporated following strategy:

```
function GETRESPONSE(input: String) : String
    rasaResponse = GETRASARESPONSE(input)
    if rasaResponse != Null then
        return rasaResponse
    else
        return GETSKILLRESPONSE(filledQuestion)
    end if
end function
```

The function getReponse() is the main method used to display the outcome of an input from a user. getRasaResponse method will return either a null object if there are no additional question, or a string question that will be needed in order to get more information about the entry. If there are is an additional question the system will display it, awaiting a response. In the other case, when no further information is needed for specification, the system will retrieve the output for expanded input from the database system(DB). It is a hash-map, where the keys are skill's questions filled with specific slots and the values are the corresponding actions.

2.5 Face detection

Face detection and recognition is used in a wide range of applications, it is well known to be part of surveillance systems [10]. Landmark detection on faces allows filters, like the popular ones used on Snapchat and Instagram, to make users look more beautiful or funny [13].

2.5.1 Haar Cascade Classifiers

Different techniques for detecting faces were investigated. Firstly, the classical Haar cascade classifier, a machine-learning technique for object detection [46]. The object detection framework is based on three main contributions: integral image, adaboost and cascading [46]. Here we present a brief overview of the importance for each of these elements.

- integral image: image representation technique that allows for fast feature evaluation
- adaboost: makes sure that to ensure fast classification, large number of available features are ignored and a small set of critical features is considered
- cascading: combining increasingly more complex classifiers, therefore reserving more complex processing only for the most promising regions

The Haar classifier was implemented in two ways. Firstly, we used both pre-trained Haar classifiers from OpenCV and our own trained classifiers. We used the UTK-Face [48] dataset images to train two classifiers. For each classifier, 10 different training stages were performed [29]. The two classifiers were trained on images of faces with size 32x32 and 50x50 respectively. As mentioned before we also used 2 pre-trained classifiers, these are the *haarcascadefrontalface.xml* and *haarcascadefrontalfacealt.xml*, which can be found on the *OpenCV* GitHub page [28].

In the context of the assistant, the classifier was deployed in a real-time setting, where the webcam was accessed and on detection, the user was greeted. Frames were caped from the webcam at 30 fps. Resolution was in standard 720p. Training for the cascade classifier was done through the *OpenCV* utilities.

2.5.2 Local Binary Pattern Histogram

Local Binary Pattern (LBP) features have performed very well in various applications, including texture classification and segmentation, image retrieval and surface inspection[5]. This operator works by thresholding the neighbourhood of size P around a pixel value, which then creates a binary pattern[5]. This binary pattern allows us to capture the local spatial patterns, which are good landmarks for faces[15]. The training of the LBP classifier is the same as the Haar classifier, however instead of using the Haar feature the LBP features were used. For this classifier, we used 2 pre-trained models from *OpenCV*. The *lbpcascadefrontalface.xml* and the *lbcascadefrontalcatface.xml* were used. Both can be found on the Java OpenCV GitHub page.

2.5.3 Grid search

In order to test the classifier against one another we made sure all classifiers had optimal parameters using grid search. Grid search works by performing a complete search over a given subset of the hyperparameters space.[21] We use the true positive rate (TPR) and false positive rate (FPR) in our objective function:

$$accuracy = (1 - FPR) \cdot TPR$$

This formula will compute the area under the curve, which is the measure of quality in ROC. The hyperparameter space, was defined of openCV parameters, which are the

minimum neighbours, specifying how many neighbors each candidate rectangle should have to retain it, the minimum possible object size, where objects smaller than that are ignored. [28] and, the scale factor specifying how much the image size is reduced at each image scale[28].

2.6 Face Recognition

The methods discussed before can be used to detect a face in an image. In order to recognize a person in an image, more advanced models are needed. Whilst the detection algorithms are not optimal for face recognition they are critical in the task to do so. They help us to cut out the face or faces from the image which are used in the recognition of a face. The following 3 methods were used:

1) Eigenfaces
2) Siamese neural network (SNN)
3) FaceNet embeddings

These methods are arranged in ascending order of complexity. Eigenfaces being the least advanced and FaceNet being the most advanced one. The SNN and eigenfaces were implemented by ourselves, however for the FaceNet embeddings we used a pretrained version. Training a model of this magnitude ourselves will take hundreds up to thousands of hours[39], for this reason we used a pretrained model.

2.6.1 Eigenfaces

Eigenfaces is a widely used dimensionality reduction technique to represent faces[25]. Eigenfaces are designed for face recognition and allowss us to find the dimensions in multi-dimensional data which contain the most information. The dimensions which contain the most information about our dataset are identified as principle components. As variance tells us how spread out the data is we can use this property in the following way: The dimensions with the highest variance are most important to identify the axis with the most information. Eigenfaces works by computing a covariance matrix from the facedataset and using the eigenvalues and eigenvectors from this matrix to identify the principle components[25]. The eigenvalues tell us the amount of variance in the direction of the eigenvector[25]. The eigenvector tells us in which direction the variance is captured[25]. Once we have these eigenvectors/eigenfaces, we can use these to reconstruct the original face using a weighted sum of these eigenvectors and the mean face[24]. The mean face is an average of all the faces in the data set. These weights are obtained by projecting the face image onto the eigenvectors[24]. This array of weights form an embedding, allowing us to have a vector to represents a face. This vector allows the use of a distance measure between vectors to determine if 2 faces are the similar[24].

2.6.2 Siamese neural network

Siamese Neural network is a neural network architecture that contain two or more identical sub-networks (for project purposes it contains only two). For face recognition it will process two input images and will determine if they are similar enough to be considered verified. This project's neural network was built following the Nicholas Renotte approach [36] and by following a paper Siamese Neural Network for One-shot Image Recognition[11].

In the implementation two images get passed to the Siamese Neural network, where they go through an embedding layer and determines the similarity at L1 Siamese distance.

The purpose of an embedding layer is to convert raw image to a data representation, that represent the extracted features of the input images. Two images of shape 100x100 are passed (ref paper). It performs the following operations: In the first block we define a convolutional layer (c1) with 64 filters of size 10x10 and ReLU activation. The output of c1 is then passed through a max pooling layer (m1) with a pooling window of size 2x2 and padding set to 'same'. The purpose of these operations is to capture local features from the input images. We continue to perform these operations 3 more times with different parameters. On the last layer we process the flattened vector into a dense layer (d1) with 4096 units and sigmoid activation function. d1 is a 1D vector of size 4096, which represents the transformed representation of the input image.
The L1 distance layer, also known as the Manhattan distance, is used to compare two streams of images: an anchor image and a positive/negative image. Its purpose is to determine whether the embeddings derived from these images are similar enough to be considered verified. It computes the absolute difference between two images, which represents dissimilarity between them. By comparing the L1 distance with a predefined threshold or applying further decision rules, the system can determine if the embeddings are similar.
To make the Siamese model we perform the steps mentioned above. Then we combine the received two distances into a final fully connected layer with a sigmoid activation which outputs a value between 1 and 0 (making the ultimate decision of whether the images are similar or dissimilar with the use of defined threshold, for out project we set it to 0.8).

2.6.3 FaceNet

FaceNet is a state of the art embedding model for face recognition and clustering from Google[3]. The FaceNet model is a deep convolutional neural network with a special loss function: triplett loss. This triplett loss function directly reflects what we want to achieve in face verification, recognition and clustering[39]. The triplet loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity[39]. A good visualisation can be found in the following figure:

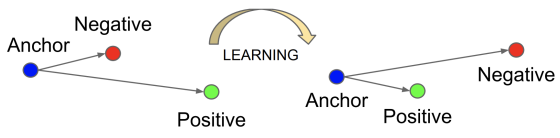

Fig. 1: Triplet loss visualisation [39]

We select the positive sample which has the highest distance to the anchor and the negative sample with the

closest distance to the anchor point at each iteration of training[39]. The anchor point is picked arbitrarily[39]. This embedding allows us to construct a vector in euclidean space from a face. The euclidean space allows us to use distance measurements to cluster and recognize faces. We used euclidean distance with a threshold to recognize persons in our database.

2.7 Audio Processing

2.7.1 Working with audio in Java

To facilitate user interaction with the assistant through speech, our implementation incorporated several key features. These features encompass real-time monitoring and recording of the user's microphone input on their device. We opted for the TarsosDSP framework for Java for this purpose as it provided seamless and efficient audio processing, monitoring, and recording capabilities [40]. Notably, compared to the JavaSound API, the TarsosDSP framework offered enhanced control over audio streams. The basic structure of how TarsosDSP handles audio in Java is visualized below in Figure X. The following list demonstrates how it relates to our implementation:

- **Audio:** the source of audio, specifically, the microphone input stream.
- **Dispatcher:** used to divide incoming audio into blocks of 1024 samples with overlap window size 0, and then scaling into floats between [-1,1].
- **AudioEvent:** encapsulates the blocks, with a pointer to the audio, represented as a float array/float buffer.
- **Processor:** the part where we performed sound analysis and alterations to extract features of interest. Here, we calculated the sound pressure level and applied dynamic range gating (described later).

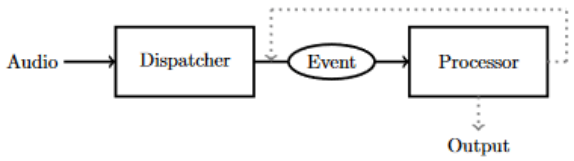

Fig. 2: Schematic representation of TarsosDSP processing pipeline [40]

2.7.2 Processing Input Audio

Our speech recognition and speaker identification models should be fed with meaningful information. Essentially, we want to send speech (signal) and minimize noise. We investigated two approaches for this.

Noise Gate Firstly, a simple noise gate based on a threshold could be used. Only, when the audio-stream sound presser level (e.g. measured in decibel loudness full scale) exceeds a certain threshold, the audio stream is let through; else the audio buffer consists of 0-values, and therefore silence [16].

The sound pressure level was calculated using the formula in figure 3.

$$20 \log_{10} \frac{\sqrt{\sum_{i=0}^{n} b[i]}}{n}$$

Fig. 3: Formula to calculate decibel sound pressure level (dBSPL) on audio buffer 'b'

This was done on the float buffer audio stream.

The threshold **T** was set in an experimental manner to ensure low-level background noise did not prematurely start recording the input stream.

The noise-gate parameters were chosen to have an *attack* of $0ms$, which was possible by using 0 samples, which means, instantaneously letting through the buffered audio once the threshold was exceeded. To take into account pauses in speech, and only stop whenever the end of a sentence was reached, we chose a release-time of $1000ms$.

Speech Energy based Voice Activity Detection Secondly, a more advanced approach was investigated. In voice-activity detection the goal is to detect the presence or absence of human speech given a signal. A straightforward approach here, would be to compute the ratio of energy between the energy of the frequency range of speech (*300 Hz - 3000 Hz* and the energy over the full spectrum. Based on this ratio, set a threshold **T**, from which we allow to decided whether speech is present or not. This processed is named spectrum energy based voice activity detection [31]. However, the simple approach using a noise gate proved to be sufficiently good for our implementation.

2.7.3 Encoding Input Stream

Our speech-recognition and speaker identification models worked with Waveform Audio Files (WAV). Thus, the stream of processed microphone audio needed to be converted to .WAV files. The target sample rate for our model was 16 kHz and the channel configuration MONO. The encoding from float buffers to .WAV files was done in Java (from scratch), using Pulse Code Modulation (PCM) encoding.

2.7.4 Speech Recognition

Our primary research objectives were to implement speech recognition and speaker identification. We focused solely on open-source speech recognition models, and did not consider at any moment in time the use of paid models. This left us with two potential models: Wav2Vec-2.0 model by Facebook/Meta [2] and the Mozilla DeepSpeech model [12]. Wav2Vec-2.0 effectively transcribes speech from raw audio without the need for laborious feature extraction, thanks to its novel self-supervised learning mechanism. This approach greatly enhances system effectiveness. Wav2Vec-2.0 also demonstrates high accuracy in tasks such as English transcription and robustness in various audio environments [2].

The Deepspeech models converts audio streams into sequence of characters by employing two key steps. Firstly, using a deep neural network, the audio get transformed into a series of character probabilities. In the second step, these probabilities get converted into actual characters using a N-gram language model. The neural network is trained on

audio and corresponding transcripts, while the language model is trained on a separate text corpus. In this two-step process, the acoustic model functions as a phonetic transcriber, while the language model acts as a spelling and grammar checker.

We chose the Wav2Vec-2.0 model, supported by audio processing using TarsosDSP and methods written from scratch (noise-gate). The implementation of the Wav2Vec-2.0 model was slightly easier, and the final application had the Wav2Vec model implemented, but could, of course, be changed to include the DeepSpeech Model.

2.7.5 Speaker Identification

The speaker identification approach utilizes Mel-frequency Cepstral Coefficients (MFCC) and Gaussian Mixture Models (GMM). MFCCs extract speaker-specific acoustic characteristics from audio by capturing power in various frequency bands and taking into account non-linear loudness and pitch perceptions [23]. These MFCC features are then statistically modeled by GMMs, creating a probabilistic representation of each speaker's voice [23]. The evaluation of likelihood scores against these models when new speech samples are presented results in successful speaker identification. The user experience is improved by this integration, which also improves the assistant's speech recognition capabilities and makes it easier to provide personalized responses. Our implementation focused on text independent speaker identification, which means that the user is identified irrespective of what he/she/they is saying [23].

Our speaker identification model was trained for 5 speakers, since our purpose was to correctly identify the different speakers in the group. We recorded 15 audio clips per subject, with each clip being approximately 5-10 seconds long. All recordings were done using the same Apple MacBook Pro 2018 default microphone in Ableton Live 10. All .WAV files were subsequently brought into the correct format for the model, with a sample rate of 16kHz and mono channel configuration. For each subject, 10 recordings were kept as training data and 5 recordings for test data. The main packages used for the implementation of our speaker identification model were : *Python Speech Features* to extract the MFCC and *Scikit Learn* to train, test, store and deploy the GMM models for each subject.

3 Experiments

3.1 Context Free Grammars

We only performed algorithmic analysis based on theory. It seemed obvious to not run explicit experiments input data. Verifying the correct working of the algorithm was done through extensive unit testing. The brief complexity analysis can be found in the introduction.

3.2 Multi-turn dialog system

To evaluate the effectiveness of incorporating additional questions using the Rasa framework, we conducted experiments using a dataset of questions that were not present in the database system, for example, "What is the weather?", where no slots are defined. The sample size for our experiments was set to 50 queries, which were randomly selected from our dataset. The Rasa model was trained on 100 epochs, 30 iterations per each epoch, which is the default setting of the model, as well as training 300 epochs on the NLU part of the model, in order to recognize the specific intent of the input, and to be able to classify it to a specific skill. Rasa framework has a ready to use experimental system, which will simulate the dialogues with the given inputs and compare how the dialogues evolved, comparing with the test data.

Metric	Desctiption
Accuracy	Ratio of correctly predicted instances to the total number of instances in the dataset.
F1-score	Combines precision and recall to provide a balanced evaluation of model's performance.

TABLE 1: Metrics used to evaluate multi-turn dialogue system

Accuracy:
$Accuracy = \frac{\text{Number of correctly classified instances}}{\text{Total number of instances}} \times 100\%$
F1-score: $F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

- Precision represents the ratio of true positive predictions to the total number of positive predictions. It measures the model's ability to correctly identify positive instances without falsely labeling negatives.
- Recall, also known as sensitivity or true positive rate, represents the ratio of true positive predictions to the total number of actual positive instances. Recall quantifies the model's ability to correctly identify all positive instances without missing any.

3.3 Face Detection

As mentioned before we used grid search to find the optimal parameters for each of the classifier for both pre-trained models and our own trained models. After we found the optimal parameter for each of the classifiers we examined the accuracy using a method called receiver operating characteristic (ROC). By using this method, we keep track off the of True Positive (TP), False Positive(FP), True Negative(TN) and False Negative(FN) detection rate when applying the classifiers to a face data set (UTKFace [48]) and an object data set (101_ObjectCategories[20]).

- TP: The classifier correctly identified a face when there was indeed a face.
- FP: The classifier incorrectly identified a face when there was no face.
- TN: The classifier correctly identified no face when there was indeed no face.
- FN: The classifier incorrectly identified no face when there was a face.

We used the the cascade classifiers as discussed in the methods and implementation to construct the ROC curve.

3.4 Face recognition

As mentioned in the methods and implementation we made use of 3 face recognition algorithms:

1) Eigenfaces
2) Siamese neural network
3) FaceNet

In order to test the accuracy of each of these algorithms we used the celebrity databse from celebA [22]. This dataset contains 200.00 images of celebrities, which we used to construct a database of our own. We took 100 celebrities with each 4 pictures and then used a clustering approach to measure the accuracy. We used the mean vector generated by FaceNet or eigenfaces of these 4 pictures as the key to our database. In order to measure the accuracy we would go over all of the 400 celebrity images and map them to the closest mean vector. If the id matched the database id we would mark it correct, else we would mark it incorrect. For the siamese network we did slightly different approach, as this network works by comparing 2 images and gives a probability output. For this our mean vector is substituted with the celebrity image which has the least standard deviation to the cluster of images. We would than map the celebrity to the image wich has the highest probability of being the same person. Finally we kept track of the time it took for the classifier to embed a face. This allows us to see if we can actually use the model in realtime.

3.5 Speech Recognition

3.5.1 Metrics

Besides the fact that one model for speech recognition might be more user-friendly to integrate with our application, we were also interested in real-word performance of the model. Several features deemed to be important to us in regards to having a good speech recognition model. The metrics listed in table 1 were used to evaluate the performance of the model.

Metric	Description
RealtimeFactor	Inverse of the ratio between clip duration and inference time
WordErrorCount	Ratio of errors in the transcript to total words
ROUGE-L F1 Score	The F1 score of the longest common subsequence between transcript and hypothesis

TABLE 2: Metrics used to evaluate the speech recognition model

Presented below is a comprehensive overview of the formulas utilized for computing the values of each metric along with their corresponding descriptions.

RealTimeFactor:

$$\text{RTF} = \frac{1}{\left(\frac{\text{InferenceTime}}{\text{ClipDuration}}\right)}$$

- InferenceTime – the time it takes for the model to return the transcription of the audio clip.
- ClipDuration – the duration of the audio clip.

WordErrorRate:

$$\text{WER} = \frac{S + D + I}{N}$$

- S – the number of substitutions.
- D – the number of deletions.
- I – the number of insertions.
- C – the number of correct words.
- N – the number of words in the reference ($N = S + D + C$).

Rouge-L F1 Score:

$$\text{ROUGE-L F1} = \frac{2 \cdot (\text{precision} \cdot \text{recall})}{\text{precision} + \text{recall}}$$

- Precision –the ratio of the length of the longest common subsequence (LCS) and the total number of unigrams in C (Candidate).
- Recall – the ratio of the length of the LCS and the total number of unigrams in R (Reference).

3.5.2 Dataset

To examine the models, we constructed a dataset comprising 120 audio clips featuring English speech. These clips were sourced from the clean LibriSpeech dataset [18], which conveniently provided annotations for each clip. The dataset encompassed approximately 10 minutes of audio, featuring 40 speakers, with each speaker contributing 3 audio samples. To ensure dataset diversity and speech audio length, as well as to allow us to conduct a wide range of experiments within reasonable time constraints, we decided to include 120 samples.

3.5.3 Noise Generator

Since real-world performance is of great importance to any application incorporating speech recognition, we build an extensive noise generator. This approach allowed us to investigate the robustness of each individual model. The noise generator is able to add different types of noise at different loudness levels to our audio clips containing speech. We used three types of noise that related to realistic day-to-day environment, namely *babble*, *street* and *cafe* noise. To determine the loudness of the voice clip and noise clip, we used Loudness Units Full Scale (LUFS), a standard loudness measurement unit for broadcasting, radio, TV, and podcasts [43].

The procedure outlined below illustrates the addition of noise to each individual sample, which can subsequently be applied to the entire dataset.

1) Load the voice clip.
2) Determine the noise type (babble, street, or cafe).
3) Measure the LUFS (Loudness Units Full Scale) of the voice clip.
4) Calculate the target LUFS of the noise clip using formula **NoiseLoudness**.
5) Crop the noise clip to match the length of the audio clip.
6) Add the audio clip and noise clip together to obtain the final output.

7) Save the final output as a new audio file.

NoiseLoudness

$$\text{LUFS}_{\text{noise}} = \alpha \times \text{LUFS}_{\text{voice}}$$

- α is a scaling factor that inversely affects loudness, with larger values resulting in more negative loudness and, consequently, lower volume.

It is helpful to look at the spectrograms and frequency spectra of each type of noise. These visuals show which frequency range contains the most energy and can help us understand why different types of noise have varying effects on the model's performance, e.g. think how each frequency range possible masks human speech. This was also the motivation why we picked different noise types.

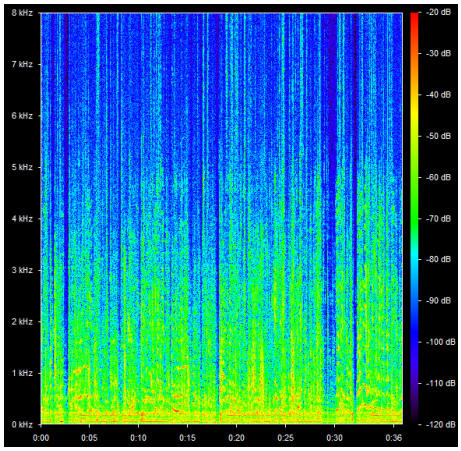

Fig. 4: Spectrogram of babble noise – generated using Spek [42]

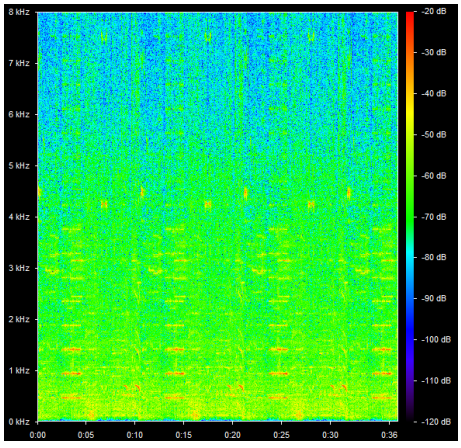

Fig. 5: Spectrogram of street noise – generated using Spek [42]

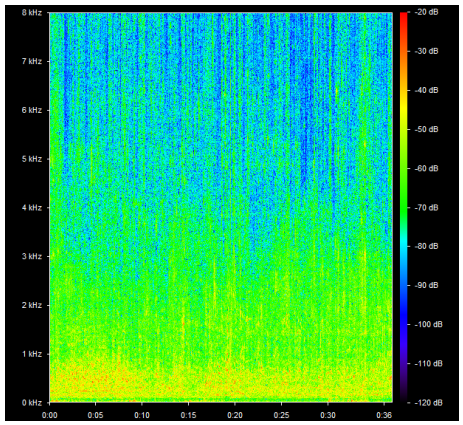

Fig. 6: Spectrogram of cafe noise – generated using Spek [42]

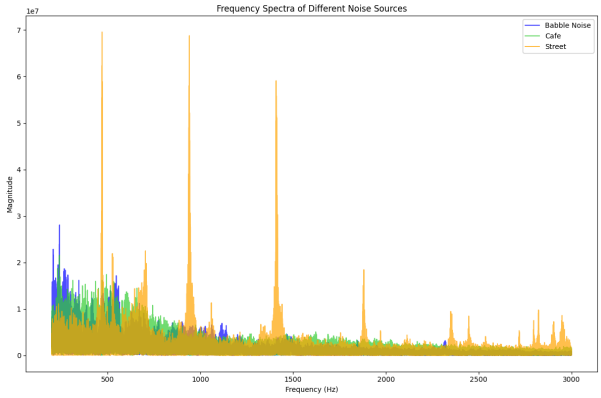

Fig. 7: Frequency spectrum off all noise sources combined

Both models were investigated on the datasets described in table 2, that incorporated the different types of noise, on different loudness levels.

Dataset	Samples	Noise Type	Loudness
LibriSpeech	120	None	-
LibriSpeech	120	Babble	$\alpha = 1.5$
LibriSpeech	120	Babble	$\alpha = 2.0$
LibriSpeech	120	Cafe	$\alpha = 1.5$
LibriSpeech	120	Cafe	$\alpha = 2.0$
LibriSpeech	120	Street	$\alpha = 1.5$
LibriSpeech	120	Street	$\alpha = 2.0$

TABLE 3: Datasets with sample size, noise type, and loudness. $\alpha = 2.0$ corresponds to audible noise, and $\alpha = 1.5$ corresponds to very audible noise.

3.6 Speaker Identification

In order to evaluate the performance of our speaker identification model, and especially the robustness, we used our noise generator to add the different types of noise at varying levels to our test set and evaluated the model accuracy.

The accuracy was computed using the formula in figure 8.

See the results section for table X , containing full description of the noise types and levels applicable, including accuracy results.

$$\text{Accuracy} = \left(\frac{\text{CI}}{\text{NS}}\right) \times 100$$

Fig. 8: Calculation of Accuracy (CI: Correct Identifications, NS: Number of Subjects).

4 Results

4.1 Multi-turn dialog system

Metric	Results
Accuracy	0.91
F1-score	0.86

4.2 Classifier results

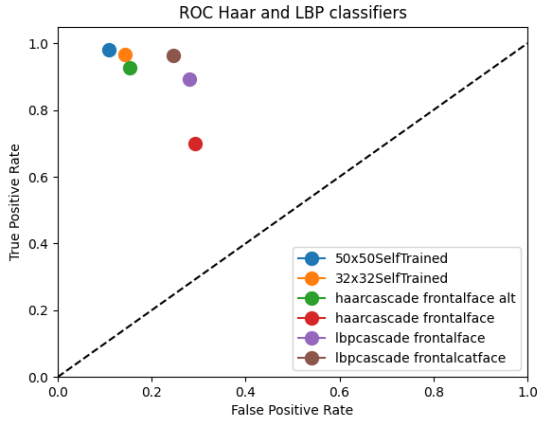

Fig. 9: Classifiers ROC score

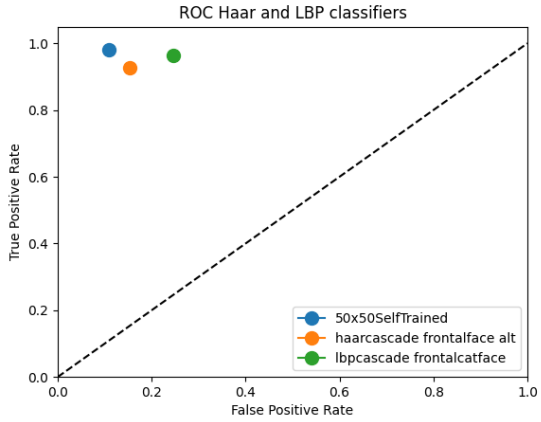

Fig. 10: Optimal classifiers in their category

4.3 Face recognition results

Algorithm	Correct	False	Accuracy
FaceNet	383	17	0.9575
SNN	312	88	0.7800
Eigenfaces	245	155	0.6125

TABLE 4: Face recognition accuracy comparision

Statistic	FaceNet	Eigenfaces
Mean	0.0816	0.0205
Median	0.0790	0.0201
Variance	6.0659e-05	3.1580e-06
Standard Deviation	0.00778	0.00177

TABLE 5: Statistics for FaceNet and Eigenfaces

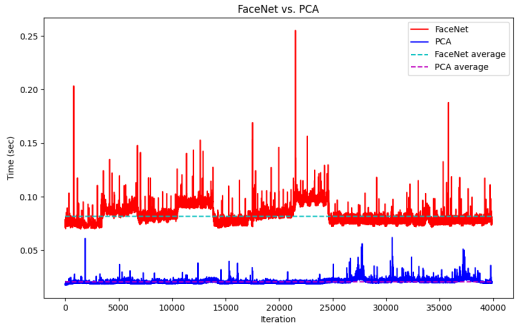

Fig. 11: FaceNet vs Eigenfaces runtime comparision

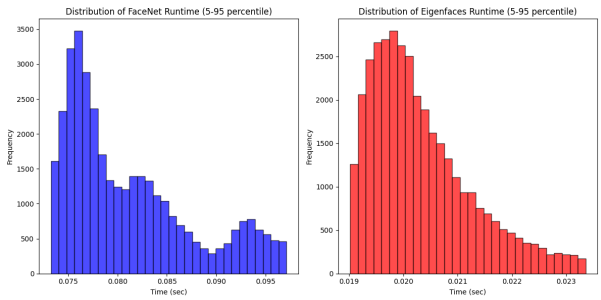

Fig. 12: FaceNet vs Eigenfaces runtime distribution

4.4 Speech Recognition

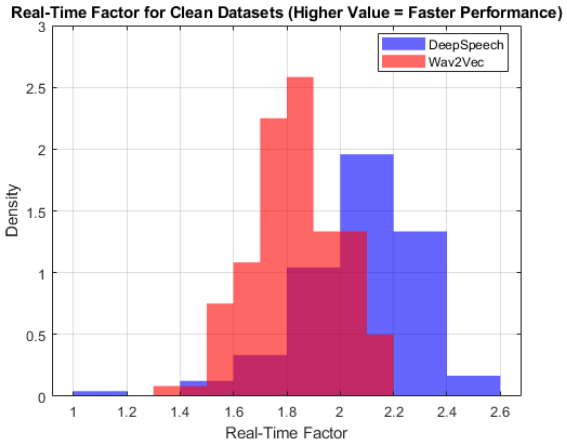

Fig. 13: Comparison of Speech Recognition Models on Clean Librispeech Dataset (120 samples) for the real time factor

Model	Mean	Median	Variance
Mozilla DeepSpeech	2.0703	2.0850	0.0463
Wav2Vec	1.8324	1.8300	0.0267

TABLE 6: Comparison of Speech Recognition Models on Clean Librispeech Dataset (120 samples) for the real time factor

Note: Higher values indicate better performance. No statistical test was conducted between the two models due to the difference in measurement methods: Mozilla utilized a built-in inference time method, while Wav2Vec-2.0 was measured using a Python timer.

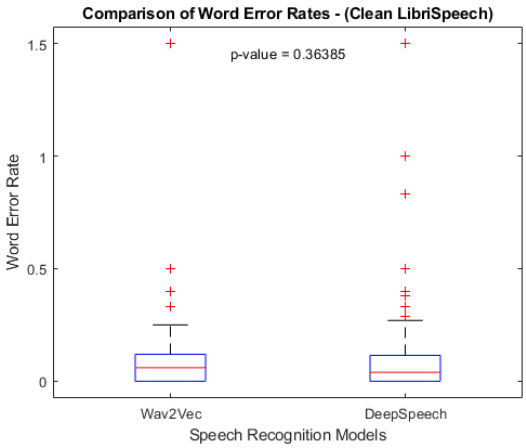

Fig. 14: Wilcoxon Rank Test of Speech Recognition Models on Clean Librispeech Dataset (120 samples) for the Word Error Rate

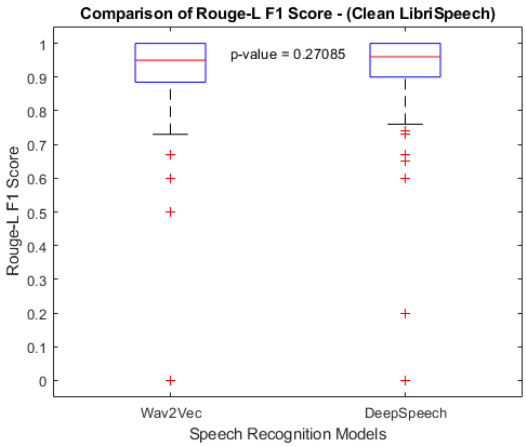

Fig. 15: Wilcoxon Rank Test of Speech Recognition Models on Clean Librispeech Dataset (120 samples) for the Rouge-L F1 Score

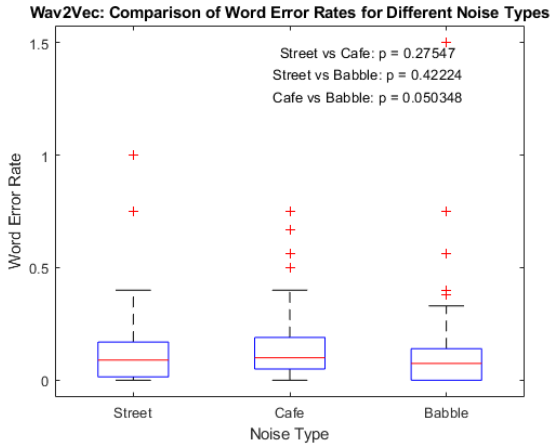

Fig. 16: Word Error Rate - Influence of different noise types on Wav2Vec-2.0 Model

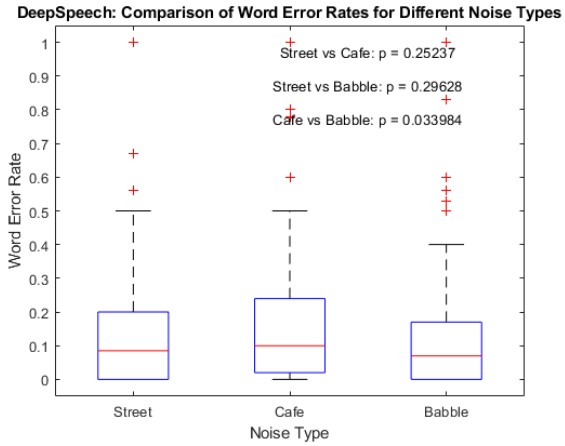

Fig. 17: Word Error Rate - Influence of different noise types on DeepSpeech Model

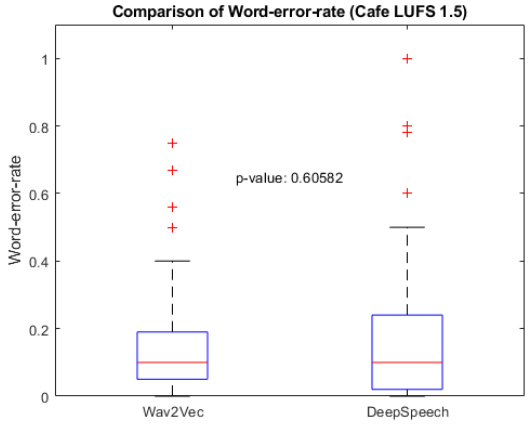

Fig. 18: Word Error Rate - Comparison of models under the influence of noise

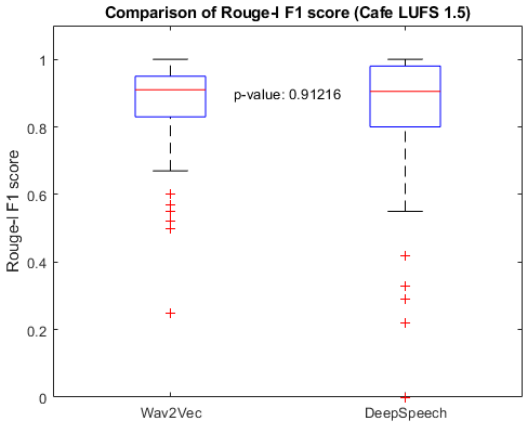

Fig. 19: Rouge L F1 Score - Comparison of models under the influence of noise

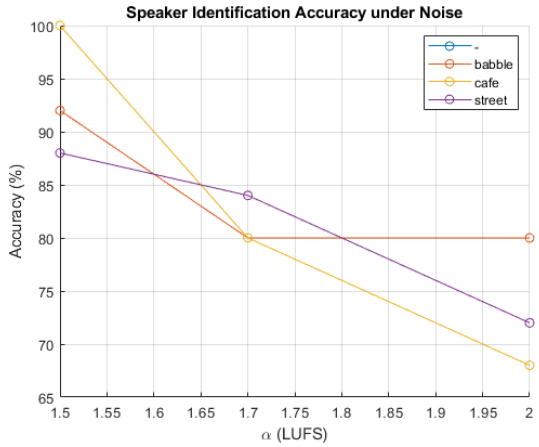

Fig. 20: Accuracy of Speaker Identification Model under Varying Types and Levels of Noise. A lower value of α (LUFS) indicates a higher presence of audible noise.

Model	Mean	Median	Variance
Word Error Rate			
Wav2Vec-2.0	0.0890	0.0600	0.0245
DeepSpeech	0.1012	0.0400	0.0388
F1 Score			
Wav2Vec-2.0	0.9174	0.9500	0.0143
DeepSpeech	0.9117	0.9600	0.0261

TABLE 7: Comparison of models under the influence of cafe noise ($\alpha = 1.5$)

4.5 Speaker Identification

Samples	Noise Type	α (LUFS)	Accuracy (%)
25	-	-	100.0
25	babble	2.0	92.0
25	babble	1.7	80.0
25	babble	1.5	80.0
25	cafe	2.0	100.0
25	cafe	1.7	80.0
25	cafe	1.5	68.0
25	street	2.0	88.0
25	street	1.7	84.0
25	street	1.5	72.0

TABLE 8: Accuracy of the speaker identification model under the influence of different types of noise at varying levels on the testing set. A lower value of α (LUFS) relates to more audible noise.

5 Discussion

From the experiments of our multi-turn dialog system showcased 91% of accuracy. This demonstrates that incorporating additional questions significantly improved the chatbot's ability to gather missing information and fill the slots accurately. The Rasa classifier demonstrated strong performance, achieving F1-score of 0.86. This indicates the classifier's ability to accurately classify user intents and extract entities from user queries. The high F1-score signifies a balance between precision and recall, implying that the classifier effectively identified relevant intents and entities while minimising false positives and false negatives. The achieved F1-score highlights the robustness and effectiveness of the Rasa framework's natural language understanding component in accurately understanding the user inputs and asking additional questions where needed. The results of our experiments highlight the effectiveness of using the Rasa framework to ask additional questions for slot filling in a QA chatbot.

In figure 9, it is noticeable that the performance of all classifiers, except for the LBP frontal catface, are relatively similar. However, our custom-trained classifiers demonstrate a slight advantage over the others. There could be various explanations for this, but the most plausible one seems to be related to the choice of our training dataset. We opted to train these classifiers using the UTKFace dataset[48], which also served as our validation set for the ROC analysis. The fact that this dataset consists solely of images with a single face aligns better with our project's focus on authentication mechanisms, which requires to be handful of people in front of the camera at most. The relation between the validation and training set could have contributed to our classifiers' marginally superior performance.

We selected the FaceNet and eigenfaces as representatives of the best and worst performing algorithms. As FaceNet is a state-of-the-art approach and eigenfaces is a more traditional approach we will use these for the comparision. The SNN is like FaceNet a neural network approach to the face recognition problem, as want to present the comparision that highlights the trade-off between accuracy and computational efficiency between the approaches we found

that these 2 are the most suitable pair. When comparing the accuracy between the face recognition algorithms (table 4 we can see that FaceNet is the clear winner with almost 96% accuracy and Eigenfaces being the worst with an accuracy of 61%. The runtime experiments suggest that FaceNet performs the worst in runtime. With eigenfaces being able to run at $\approx 50Hz$ and FaceNet at $\approx 12.5Hz$. If we look at table 5 we can see that when compared to the eigenfaces both the mean and variance is higher, indicating FaceNet is less consistent and not as fast as eigenfaces. When looking at figures 11 and 12 we can visually confirm this, as the FaceNet shows more deviations and less evenly distributed peaks than eigenfaces. This makes sense as FaceNet is a very large neural network when compared to eigenfaces which is mainly matrix multiplication.

Evaluation of Mozilla DeepSpeech and Wav2Vec-2.0 in the area of speech recognition uncovered some interesting characteristics. Wav2Vec-2.0 seemed to offer a lower Real Time Factor (RTF), resulting in slower operation, as suggested by figure 13, while Mozilla DeepSpeech displayed a higher variance, indicating variable performance times. However, it is crucial to remember that no statistical test was carried out to establish the significance of this difference. Since this test would likely not have any meaning, since the way the inference time for both models could not be measured in the same way, as mentioned in the note in the results section. Word Error Rate (WER) from figure 14 and Rouge-L F1 Score comparison from figure 15 did not reveal any appreciable differences, indicating that both models produced transcriptions with a similar level of accuracy. The addition of noise, however, had a negative impact on both models' performance. This was especially clear in scenarios with Cafe noise, where both models showed an increase in WER. Notably, DeepSpeech demonstrated a higher level of variance, suggesting potential irregularities in its performance under various circumstances. Moving on to the speaker identification, the results showed a decrease in accuracy with increasing levels of noise. The degree of this impact, interestingly, varied depending on the noise type, indicating that noise poses a significant challenge for speaker identification. This emphasizes the importance of robust models and advanced noise-reduction techniques for achieving reliable results in practical applications.

6 Conclusion

Context Free Grammar In conclusion, for CFG, our project employed a custom approach alongside the Cocke-Younger-Kasami algorithm, a widely recognized technique in the field, to determine the membership of words in a specific language defined by "skill-based" grammar. To utilize the CYK algorithm effectively, we had to transform our "skill-based" context-free grammar files into Chomsky Normal Form, following a procedure documented in relevant literature. This procedure involved several essential steps, including the elimination of the start symbol, epsilon rules, non-solitary terminals, right-hand sides with more than two non-terminals, and unit rules. The overview of this procedure served as a foundation for our software implementation, which was specifically tailored to handle the "CFG-based skills" mentioned.

Multi-turn dialogue system The utilisation of the Rasa framework for incorporating additional questions and enhancing slot filling for out project has demonstrated promising results. By leveraging the capabilities of Rasa's NLU component, we improved the accuracy of understanding user queries and provided more context-aware responses. However, certain considerations should be taken into account. One important aspect to note is that while our experiments showcased the effectiveness of additional questions for slot filling, it is essential to emphasise that incorporating new skills would require retraining the model. Also, it is worth mentioning that the process of training the model and incorporating new skills is an iterative one. Fine-tuning the model parameters, refining the training dataset, and continuously evaluating the performance are crucial for achieving optimal results.

Face Detection Based on our results, we can conclude that the LBP and Haar classifier are both suitable for face detection, with both scoring high on the ROC curve (figure 9), but Haar classifier beating the LBP classifier by a small margin. Besides, through the means of highly effective grid-search we managed to find better parameters for pre-trained models and existing models. As for the face recognition algorithms we can conclude that FaceNet is the most accurate however this comes at a computational cost, with FaceNet being over 4 times slower then eigenfaces. If you need to sacrifice accuracy over runtime due to limited computation resources, eigenfaces proofs to be most optimal.

Speech Recognition and Speaker Identification The speech recognition of both Meta and Mozilla, both showcased great performance. Even under the influence of noise, using Wilcoxon Rank tests, we could not show any significant difference between the two models based on the WER metric and Rouge-L F1 Scores. Since the measuring of inference time was build-in with Mozilla and measured through Python timer for Wav2Vec-2.0, we were not able to make any valuable conclusions base don this. The ease of implementation and support of community therefore, could be one key factors to choice one model of the other. We observed that most models however did suffer most under the influence of cafe noise, the noise that that was most close to white noise of noise types we investigated. For example, we saw a significant difference between babble noise and cafe noise, on the WER for the Deepspeech model. Finally, our speaker identification, high level overview, in correspondence to the speech recognition model, decreased its performance most drastically under the influence of cafe noise.

APPENDIX A

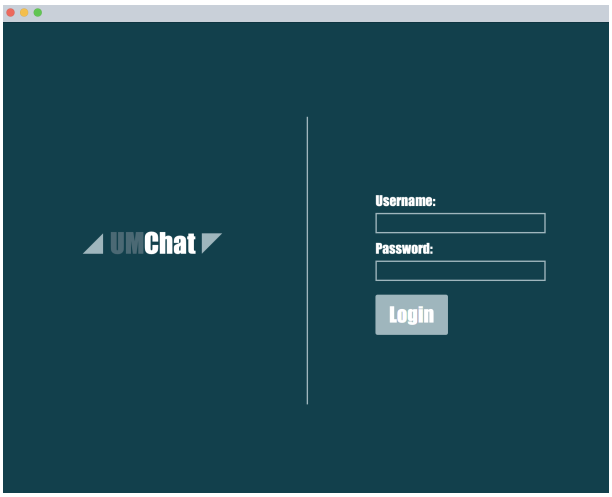

Fig. 21: GUI - Login Screen

Fig. 22: GUI - Chat Screen

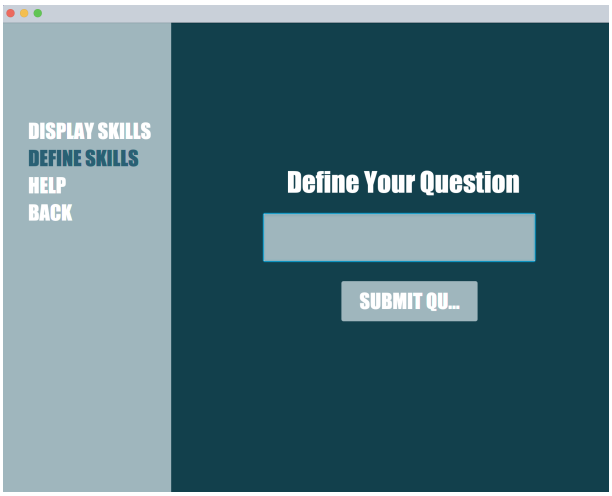

Fig. 23: GUI - Define Skill

Fig. 24: GUI - List Available Skills

Fig. 25: GUI - Edit Skill

REFERENCES

[1] Author(s) if available. CFG Parsing. http://www1.cs.columbia.edu/~kathy/NLP/ClassSlides/Class7-Parsing09/cfg-parsing.pdf.

[2] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

[3] Stephen Balaban. Deep learning and face recognition: the state of the art. In Ioannis A. Kakadiaris, Ajay Kumar, and Walter J. Scheirer, editors, *Biometric and Surveillance Technology for Human and Activity Identification XII*, volume 9457, page 94570B. International Society for Optics and Photonics, SPIE, 2015.

[4] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management, 2017.

[5] Jo Chang-Yeon. Face detection using lbp features. *Final Project Report*, 77:1–4, 2008.

[6] Pallavi Vijay Chavan and Ashish Jadhav. 4 - context-free grammar. In Pallavi Vijay Chavan and Ashish Jadhav, editors, *Automata Theory and Formal Languages*, pages 105–131. Academic Press, 2023.

[7] Noam Chomsky. *Syntactic Structures*. Mouton, 1957. Discusses context-free grammar (CFG).

[8] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances, 2013.

[9] W. Goddard. *Introducing the Theory of Computation*. Jones & Bartlett Learning, 2008.

[10] Joshila Grace.L.K and K. Reshmi. Face recognition in surveillance system. In *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–5, 2015.

[11] Ruslan Salakhutdinov Gregory Koch, Richard Zemel. Siamese neural networks for one-shot image recognition. Technical report.

[12] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition, 2014.

[13] Pontus Hedman, Vasilios Skepetzis, Kevin Hernandez-Diaz, Josef Bigun, and Fernando Alonso-Fernandez. On the effect of selfie beautification filters on face detection and recognition. *Pattern Recognition Letters*, 163:104–111, nov 2022.

[14] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[15] Kushsairy Kadir, Mohd Khairi Kamaruddin, Haidawati Nasir, Sairul I Safie, and Zulkifli Abdul Kadir Bakti. A comparative study between lbp and haar-like features for face detection using opencv. In *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*, pages 335–339, 2014.

[16] Bob Katz. *Mastering Audio: The Art and the Science*. Focal Press, 2 edition, 2007.

[17] Saritha Kinkiri and Simeon Keates. Applications of speaker identification for universal access. In Margherita Antona and Constantine Stephanidis, editors, *Universal Access in Human-Computer Interaction. Applications and Practice*, pages 557–567, Cham, 2020. Springer International Publishing.

[18] Matěj Korvas, Ondřej Plátek, Ondřej Dušek, Lukáš Žilka, and Filip Jurčíček. Free English and Czech telephone speech corpus shared under the CC-BY-SA 3.0 license. In *Proceedings of the Eigth International Conference on Language Resources and Evaluation (LREC 2014)*, page To Appear, 2014.

[19] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 1966.

[20] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022.

[21] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR*, abs/1912.06059, 2019.

[22] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[23] Tahira Mahboob, Memoona Khanam, Malik Khiyal, and Ruqia Bibi. Speaker identification using gmm with mfcc. *International Journal of Computer Science Issues*, 12:126–135, 03 2015.

[24] Satya Mallick. Face reconstruction using eigenfaces, January 2018.

[25] Satya Mallick. Principal component analysis, January 2018.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, Sep 2013.

[27] Amlan Nag. Applications of context free grammars-cfg in sonic system. 01 2022.

[28] OpenCV. CascadeClassifier. OpenCV Documentation, 2019.

[29] OpenCV. Cascade Classifier Training, 2023.

[30] Oracle Corporation. Whitespace normalization. https://www.oracle.com/webfolder/technetwork/data-quality/edqhelp/content/processor_library/transformation/normalize_whitespace.htm, Accessed 2023.

[31] Jing Pang. Spectrum energy based voice activity detection. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–5, 2017.

[32] Sanmoy Paul and Sameer Acharya. A comparative study on facial recognition algorithms. *International Journal of Data Science and Big Data Analytics*, 1:39, 05 2021.

[33] J.M. Phillips. *Mathematical Foundations for Data Analysis*. Springer Series in the Data Sciences. Springer International Publishing, 2021.

[34] Rohit Prabhavalkar, Takaaki Hori, Tara N. Sainath, Ralf Schlüter, and Shinji Watanabe. End-to-end speech recognition: A survey, 2023.

[35] Wisam Qader, Musa M. Ameen, and Bilal Ahmed. An overview of bag of words;importance, implementation, applications, and challenges. pages 200–204, 06 2019.

[36] Nicholas Renotte. Nicholas renotte github. Available at https://github.com/nicknochnack.

[37] James Ryan, Adam Summerville, Michael Mateas, and Noah Wardrip-Fruin. Translating player dialogue into meaning representations using lstms. volume 10011, 09 2016.

[38] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Re-evaluating word mover's distance. *CoRR*, abs/2105.14403, 2021.

[39] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.

[40] Joren Six, Olmo Cornelis, and Marc Leman. TarsosDSP, a Real-Time Audio Processing Framework in Java. In *Proceedings of the 53rd AES Conference (AES 53rd)*, 2014.

[41] John A. Smith. *Perfect Matching in Natural Language Processing*. XYZ Publishing, 2022.

[42] Spek. Spek - acoustic spectrum analyzer. https://www.spek.cc/about, Accessed 2023.

[43] Robert Taylor. Towards standardised loudness normalisation in music streaming. 11 2017.

[44] Jesmin Jahan Tithi and Fabrizio Petrini. A new parallel algorithm for sinkhorn word-movers distance and its performance on PI-UMA and xeon CPU. *CoRR*, abs/2107.06433, 2021.

[45] Towards Data Science. Text preprocessing in natural language processing using python. https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8 Accessed 2023.

[46] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001.

[47] Fabio Massimo Zanzotto, Giorgio Satta, and Giordano Cristini. Cyk parsing over distributed representations. *Algorithms*, 13:262, 10 2020.

[48] Zhifei Zhang. UTKFace Dataset, 2017.