SignPredict: A machine learning approach to gesture recognition — AAU mandatory summary for thesis projects

Today, 5% of the world's population has some degree of hearing loss, and it is expected to increase by 100% within the next 30 years. Deaf people, in particular, experience communication barriers when trying to integrate into a hearing society, affecting both their health and social life. This project aimed to explore the possibilities of creating a baseline for a system capable of recognizing Danish sign language, which in turn could be used to minimize these barriers. This paper has explored the domain of recognition of Danish sign language using a machine-learning approach. It has been solely conducted by a single individual, which resulted in certain limitations to be set to finish it within reasonable time due to the hard deadline. Agile project management has been used to structure the workflow, through the utilization of sprints. A total of four different sprints have been conducted, each with their own focus. The first sprint entailed creating the necessary data to be used for training a machine intelligence model, capable of predicting single gestures performed within small, pre-recorded videos. The next phase focussed on training a total of nine models split into three types of three variants each. The first type was solely trained on coordinate data, the second type used linear graphs, and the final type was trained on a collection of both data types. The variants of each type were trained on either 25, 50, or 100 samples of each label. The labels used were "can", "thumb", and "peace". With all models successfully trained, the next sprint entailed the evaluation of each model with the given metrics; accuracy, loss, precision, recall, F1-score, and prediction on a new set of data. The final sprint entailed bundling the models into an API, allowing other systems to freely utilize the prediction prowess of the model. This is accessible through public endpoints.

For data extraction, the open-source library, OpenCV2 has been used. This library provides a collection of different computer vision functionalities, one of which is frame extraction in videos, which has been the only functionality used throughout the development of this project originating from said library. For locating hand landmarks within each frame, the library MediaPipe has been utilized. This is a well-trained module that's easy to use. This module locates the coordinates of each landmark, which can then be extracted and saved for later use. For model training, the open-source library Keras has been used, which provides a collection of different deep learning functionalities, such as training of custom MI models. The API has been configured using FastAPI, and deployed using the web server uvicorn. The API has been structured in a REST architecture. For evaluation of the model, Keras has been used for extraction of accuracy and loss, while another open-source library, SciKit, has been used to extract precision, recall, and F1-score. The API has been evaluated using the open-source tool Locust, allowing for quick and easy configuration in order to test both scalability and response time.

The results gathered from evaluating the model yielded positive results for the models trained only with coordinate data. They proved able to predict and differentiate fairly between the different three gestures. However, in terms of API performance, that evaluation yielded poor results, showing an inability to successfully scale as a higher load was received. All models proved themselves slow when it came to predicting a gesture, taking more than 2 seconds to produce a single label. Overall, the entire system should undergo a major refactor, focusing on improving the architecture and infrastructure, in order to combat these long processing- and response times.

This paper concluded that it is possible to recognize single gestures through machine intelligence, only using publicly available resources. However, it is fairly slow with its current implementation and does therefore not fit into any real-time systems. Optimizing this system could provide the wanted accessibility for deaf people, allowing them to more effectively communicate with the hearing society. The next step would be to improve the system and model, followed by conducting a user study to see if it works in real contexts.

Author's address: [illegible]

SignPredict: A machine learning approach to gesture recognition

Nicholas D. Jørgensen
ndja19@student.aau.dk
Department of computer science
Aalborg, Denmark

ABSTRACT

Around 5% of the world's current population has some sort of hearing loss, which is predicted to double over the next 30 years. Being deaf in a hearing society brings communication barriers within their daily life, and affects them in several ways. This research aims to explore the possibilities within sign language prediction using machine intelligence models, through three different approaches, all bundled into a single system called SignPredict. This solution utilizes a Sequential model and predicts using a series of coordinates, representing 21 landmarks located on a hand. The broader aim of this study is to provide a baseline solution, to be further developed in the future, capable of predicting Danish sign language gestures accurately. To secure a baseline to determine the success of the system, Oracle's Quality of Service criteria have been used throughout development, as fulfilment of these ensures quality within the system. Furthermore, an agile project management approach has been used, splitting the development process into four phases. For the MI models, the Sequential model from Keras has been used, utilizing an LSTM, a type of RNN, to produce a prediction giving a series of data. For data extraction, computer vision and landmark detection have been used. Every model developed has undergone benchmarking, evaluating their accuracy, loss, precision, recall, F1-score, as well as their correctness of predicting gestures on a data set. The models trained solely on coordinate data yielded positive results and adhered to the industry standard. The other models, trained on either linear graphs or a combination of both graph and coordinate data, yielded poor results, entailed both bad metrics and an inability to predict more than one of three labels. Throughout the research and development of this project, it was discovered that through utilizing coordinates as primary data, inputted into a Sequential model, it was possible to predict single gestures and differentiate between similar ones. However, the final bundled API, containing all functionality accessible through public endpoints, proved itself unacceptable in terms of overall response time during higher loads. A major refactor of this API is needed, focusing on optimization.

KEYWORDS

Sign Language Recognition, SLR, MI, evaluation, system development

1 INTRODUCTION

According to WHO, around 430 million (5%) of the world's current population requires some kind of rehabilitation to address their disabling hearing loss[1] [34]. This number is estimated to increase to 700 million (10%) within the next 30 years[34]. As hard-of-hearing individuals often can communicate through spoken language, as well as being able to benefit from external devices such as hearing aids, deaf individuals often rely entirely on communication through sign language [2, 8, 19, 34].

Due to this, many deaf individuals experience communication barriers, which present themselves within a broad spectre of contexts, such as their workplace [19, 21]. This causes desocialization, which in turn can lead to significant long-term health issues [14, 28]. These communication barriers therefore hinder deaf individuals in the integration into a hearing society, causing them to experience a feeling of segregation. Some deaf individuals can read lips, hereby establishing a one-way communication channel, yielding mediocre results in group dialogue [7]. However, this ability only provides them with mediocre results in terms of communication with hearing individuals. As the amount of deaf individuals is set to increase significantly [34], and they of all ages experience some sort of communication barrier, preventing them from successfully integrating into a hearing society [19], systems focusing on providing the necessary accessibility can be seen as a priority. Such systems would strengthen deaf individuals' independence, as it would allow them to communicate more easily with society, thereby minimizing, or completely avoiding, the communication barriers. Advancements in providing sight-impaired individuals with more independence have already been made, whereas deaf individuals still struggle[2].

A handful of software applications exist, containing functionality in aiding deaf individuals communicating with a hearing society, such as Google's *Live Transcribe* [9]. This application acts as a communication medium between deaf and hearing individuals. It provides a deaf individual with the ability to participate in the conversion through text messages. However, deaf individuals often have a lower literary level compared to others, and can therefore not fully express themselves with this format [19]. This lower literary level stems from the difference in syntax between Danish sign language and Danish. This issue can be a result of misunderstandings between the parties involved in the conversation.

To develop and maintain a reliable system to translate sign language, hereby providing deaf individuals with the ability to communicate without barriers, is an elaborate venture and therefore has not been done yet, as such a solution would be very resource-dependent. Google has conducted a study, where they evaluated the usability, preferability, and likeability of different sign language recognition (SLR) approaches in controlling a personal assistant on a smartphone [16]. Furthermore, just as well as there exist different languages, with several dialects, so is the case with sign language. Over 300 different official sign languages exist, where within each language exists up to several *styles* [10, 11]. These *styles* result in gestures being performed differently, hereby adding additional

[1]Disabling hearing loss refers to hearing loss greater than 35 decibels (dB) in the better hearing ear. [34]

[2]www.bemyeyes.com/language/danish

complexity to training and perfecting a functional system for SLR [10].

Therefore, my contribution to this field is an exploration into the ability of gesture recognition, to predict Danish sign language using a machine intelligence (MI) model. Danish sign language has been chosen as the language to use for the proof of concept. This is partly due to the author being Danish himself, but also due to the author having had contact with a local deaf society. The recognition is accomplished utilizing a vast amount of data inputted into an LSTM model, training both pattern and sequence recognition. The process entailing data extraction and processing, in conjunction with prediction, is encapsulated into a publicly available API, requiring only a video containing a single gesture as input. The capabilities of the API and model are shown thorough evaluation, entailing different techniques. This paper shows that it is possible to predict sign language utilizing machine intelligence, thereby opening a wide range of contexts for it to adapt into.

The paper is structured as follows: Next in section 2 we will present the work related to bridging the gap within the communication barrier between hearing and deaf people. section 3 will entail the formal description of the workflow and algorithms utilized, allowing for the reproduction of the system. Following this, in section 4, the approach chosen for testing purposes will be presented. In section 5 the results gathered from testing the different parts of the system will be presented factually. These results will be discussed and compared to the related works in section 6. Finally, the conclusion of the work will be presented in section 7.

2 RELATED WORK

The following section aims to present related work conducted by other entities, that has affected the designing and experimentation of the final model and system. This will include other approaches to sign language recognition, performed both by other studies and companies who have developed technology to solve the problem of the limited communication capabilities of a deaf individual.

2.1 Research

Several contributions, in terms of studies, regarding sign language recognition have been conducted [22, 27, 33]. One thing in common with all of these contributions, is they all involve training an independent machine intelligence model from scratch. Pathan et al. have developed a model capable of recognizing finger spelling gestures. The model was a trained multi-headed convolutional neural network (CNN)[3], and yielded positive results in terms of its ability to accurately predict the gesture performed, with a 98.981% test accuracy [25]. The purpose of this study was to provide a foundation, which may be used by other entities, to develop an accurate and efficient communication channel for deaf individuals. Their model's primary focus was to recognize finger spelling gestures and translate the gesture into the corresponding letters. Their model achieved successful prediction of a total of 24 letters, all characterized by not containing any movement [25].

Another study, on MI-based gesture recognition, was conducted by Mihir Garimella. Their solution, the same as above, was able to predict a static finger spelling gesture. However, instead of benchmarking his solution, he created a functionality that allowed for calculating the confidence in the correctness of the prediction [23]. This approach was tried solely by a single individual, which can be seen in its overall contribution as well, as there is no actual platform or application utilizing this yet. However, as his approach is fully open source, other developers can benefit from his work, and adjust functionalities to their liking to research a related topic [23]. The final approach is yet another attempt at making a sign language prediction model. Srivastava et al. developed a model with the capability of recognizing Indian Sign Language through the utilization of the open-source library TensorFlow [31]. Their approach was limited to only a subset of the entire language, yet yielded positive results. They manage to upkeep a confidence rate of 80-90% of every single letter on average [31].

Furthermore, Google, in collaboration with Gallaudet University, conducted a study where they looked into the preferability and likeability of a system recognizing sign language [16]. This system provided the ability to control a mobile assistant, such as Apple's Siri. To gather their results, they tested three different interfaces, with different controls. The interfaces were tested with people who depend on sign language as their primary communication medium [16]. The purpose of their study was also to mitigate the communication barrier between deaf and hearing individuals, by developing a platform allowing for real-time sign language recognition. This yielded positive results for their tap-to-sign interface, which involves holding down a record button, signing a single gesture, and then the gesture would be translated to text and used as input for the mobile assistant [16]. However, they utilized a Wizard-of-Oz prototyping approach, meaning they had no underlying model performing the prediction, which creates cause for another study to be conducted in the future to validate the results produced.

2.2 Product

This section of the related works will formally present released products, which aid in providing deaf individuals with the ability to minimize the communication barriers that occur during their daily routine. This will entail both a presentation of products utilizing hardware as a primary component and software platforms accessible to the public.

2.2.1 Hardware. CyberGlove Systems[4], a company established in 1990, has developed a physical piece of hardware. This hardware takes the form of a physical wearable glove. These gloves utilize sensor technology to accurately track the movement of hands, thereby providing them with the ability to be used to recognize gestures corresponding to sign language. Another piece of hardware, also in the form of a wearable glove, was developed by a pair of undergraduates at the University of Washington in 2016 [13]. The technology within these gloves involves capturing everything from XYZ coordinates to individual finger- flex and bending, to accurately recognize the gesture performed. These gloves were developed with communication with the hearing world as the primary focus, and specialized in American Sign Language (ASL). However, the gloves only provide the necessary data needed for a computer to process,

[3]A CNN is a type of machine intelligence model, often used when processing static images, due to its filter optimization method.

[4]www.cyberglovesystems.com/

therefore leaving the responsibility of predicting the gestures to the computer [13]. However, due to ASL being more complex than just single gestures, their system was limited to only a subset of the entire language, as proof-of-concept (PoC). These gloves ended up winning the Lemelson-MIT student prize, for their possibilities within the field of providing accessibility and independence to deaf individuals [13].

2.2.2 Software. Hello Monday, a company located in Denmark and the USA, has developed an online web platform, fingerspelling.xyz, freely available, that allows users to enter and try spelling a collection of different words using their fingers through finger spelling [17]. This platform was developed for the American Society for Deaf Children and was developed to help bridge the communication barrier between deaf and hearing children [17]. As it is made with children as the primary focus, gamification elements have been utilized. This presents itself by being awarded a point score upon each successful completion of a gesture. Gamification, when done correctly, aids in motivating the users during their utilization of the system [17]. The author of this paper has been in contact with Hello Monday's Aarhus division, where it was discovered that they are looking into improving the system further, enabling it to predict additional gestures and hereby, hopefully, bridge the gap within the communication barriers present in today's society even further.

Before starting this project, the group had been in contact with a local deaf society[5] located in Aalborg. Through email correspondence, it was presented to us that many deaf people, especially in Denmark, utilize a free mobile application called *Live Transcribe*[6]. This application is developed by Google, in collaboration with Gallaudet University, an institution specializing in providing education for deaf and hard-of-hearing individuals. The application is often used as the primary communication medium between deaf and hearing individuals, as it provides a two-way communication channel between the parties involved [9].

3 METHODOLOGY

The entire process of developing the system has been split up into several phases, each part representing an agile sprint. An agile sprint is a period of time allocated with the sole focus of finishing a subset of the entire project backlog, a log containing all tasks to complete before a project version is achieved [1]. This approach has been chosen as the project management structure to follow throughout this project, as it provides a clear overview of the entire process that must be completed, to reach a minimal viable product (MVP) [1]. This paper entails detailed descriptions of all parts and phases, which collectively make up the MVP. These phases will be presented throughout this section and the next. Each phase consists of 7 parts, where some specific choices have been made. These will be presented when necessary. Phase one, *Produce data set*, consists of the steps taken to develop the entire data set consisting of 300 videos containing gestures. The second phase, *Model Training*, describes the steps taken to develop all nine models developed throughout the production of this paper. The third phase, *Model Evaluation*, will be elaborated in section 4, and contains the steps performed to evaluate the model. The final phase, *Bundle API*, consists of the steps taken to bundle the trained prediction model into a Python FastAPI, and making it accessible through public endpoints.

3.1 Data extraction and preparation

Before any data was produced, the gestures and the corresponding matching labels had to be determined. A label is a classifier used to determine the type of gesture performed, meaning if the Danish sign language gesture for "elephant" is performed, the corresponding label, that will be produced by the prediction model, used for classification is "elephant" as well. This involved gathering inspiration from the Danish Sign Language Dictionary[7]. As the author, who produced the data set, is not fluent in Danish sign language, the gestures performed may not be 100% accurate. Therefore, it being said that inspiration from Danish sign language has been taken, means that the gestures performed may be incorrect, and therefore are not fit for production. However, it does share similarities with the performed gesture found in the dictionary. Furthermore, as one individual had to produce all data, the amount of gestures was limited to three, where they all shared similarities in their performance, and therefore could be used to test whether the produced model would be able to differentiate between them. The gesture and corresponding labels were limited to a total of three and were determined to be "can", "peace" and "thumb", as earlier presented. The limitation of three gestures was made as they each require 100 samples, and with a hard deadline set for the project, three were deemed reasonable to make within the given time frame. These specific gestures were chosen as they were similar to each other, with only a few differences, meaning that they would provide a good baseline for how effective the models developed were in differentiating between small differences. This step and the decisions made are all depicted in Figure 1, and represent the step *Determine labels.*

With this step done, a format for the produced video data was determined. This was done to ensure consistency within the entire data set. As this paper has taken inspiration from the use case of commanding a mobile assistant[16], as presented in section 2, the device chosen to record all data on was determined to be a mobile device, more specifically an iPhone 14 Pro. Furthermore, it was determined to record the videos with HDR disabled, in 720p resolution, 30 fps, and of a duration between 1–2 seconds. These decisions were made in an attempt to limit the file size of each video. Furthermore, it was also decided that all videos had to contain one, and only one, hand, as it was decided that the device should be held with the other hand. Collectively, these decisions made up the entirety of step 2, depicted in Figure 1 as the step *Determine video format.* With this determined, the data was now ready to be recorded. All of this was done on a single device. A total of 300 videos were recorded, 100 for each of the three labels, all consistent with the requirements determined in steps 1 and 2. The process of recording data is represented in Figure 1 as the step *Record training data.*

[5] aalborgdeaf.dk/

[6] https://play.google.com/store/apps/details?id=com.google.audio.hearing.visualization.accessibility.scribe&hl=en_US&pli=1

[7] https://www.tegnsprog.dk/

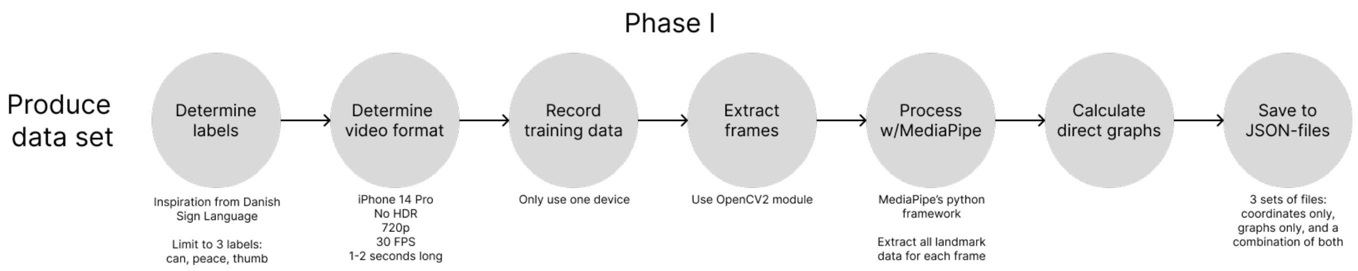


Figure 1: Phase 1, *Produce data set*, and the seven steps it entails. This process constitutes a full sprint, lasting three weeks.

The next step in Figure 1, *Extract frames*, entailed developing a Python script capable of loading each video, extracting each separate frame, and storing them in a list. The pseudocode for this process is presented in Algorithm 1 in the procedure `ExtractFrames(vp)`. This procedure utilizes OpenCV2, a Python library containing functionality allowing others to easily perform computer vision tasks on sets of data, such as images or videos. The procedure `ExtractFrames(vp)` gets a string representation of a path containing a video file as input. This path is then inputted into OpenCV2's function `Capture`, which returns a list of objects containing different attributes, one being all frames. These objects are then looped through, extracting the frame `f` and appending it to the list of all frames `vf`. After extracting all frames, the entire list of frames is returned to the caller of the function, which then can be used for further processing. This will be elaborated in the next step *Process w/MediaPipe*.

With all data having undergone the frame extraction process, the next step, *Process w/MediaPipe*, entails processing all frames using Google's MediaPipe[8] solution. This is done by looping through every frame of every video and inputting them as a parameter to the procedure `ProcessFrames`, presented in Algorithm 1. When the procedure processes a frame, it loops through all the landmarks found. A landmark is one of the fundamentals within the field of computer vision, and describes the coordinates of interest within a picture or frame [4]. A landmark within the hand could for example be the coordinate of the outermost joint on an index finger. The hand recognition model, MediaPipe, used for this implementation locates a total of 21 landmarks on a single hand [3]. A total of 21 landmarks are present on every hand, and with the restriction of having a max of one hand within every frame, this resulted in the procedure's loop in line 18 to loop 21 times. All 21 landmarks were then saved into a custom object, `FrameData`, storing the relative image coordinates. These coordinates will be used as input to calculate the linear graphs between every landmark of every frame in the next step *Calculate direct graphs*.

This step entails calculating every linear graph located between each landmark, within all frames. Linear graphing is a graph between two data points, such as coordinates, containing an x- and y-value. These graphs follow the notation of "ax + b", and are used to show the relationship between two or more quantities [5]. Through the utilization of linear graphs, the ability to perform interpolation is present. Interpolation is heavily used in a wide variety of software, such as games, 3D animation, and image manipulation [6].

[8]developers.google.com/mediapipe/solutions/vision/gesture_recognizer

Algorithm 1 The "Extract Frames" and "Process w/MediaPipe" Algorithms from step 4 and 5 in Figure 1

```
1:  procedure ExtractFrames($vp$)
2:  Description: Given a path to a video $vp$ return a list of all frames
    $vf$ from video $v$. Represents the algorithm used in step 4 from
    Figure 1.
3:      $vf \leftarrow \emptyset$
4:      $v \leftarrow \{\text{CV2.CAPTURE}(vp)\}$
5:      for all $f \in v$ do
6:          append $f$ to $vf$    ▷ Add element $f$ to the end of list $vf$
7:      end for
8:      return $vf$
9:  end procedure

10: procedure ProcessFrames($f$)
11: Description: Given a single frame $f$ return a list of FrameData
    $FD$. Represents the algorithm used in step 5 from Figure 1.
12:     $FD \leftarrow \emptyset$
13:     $mph \leftarrow \{\text{MP.SOLUTIONS.HANDS}\}$    ▷ Initialize MediaPipe
    gestures recognition
14:     $pf \leftarrow \{\text{CV.CVTCOLOR}(f, cv.BGR2RGB)\}$    ▷ Convert frame
    from RBG to BGR
15:     $r \leftarrow \{\text{MPH.PROCESS}(pf)\}$
16:     if $r.multi\_hand\_landmarks$ then
17:         for all $L \in \{(r).multi\_hand\_landmarks\}$ do
18:             for all $l \in L$ do
19:                 $fd \leftarrow \{\text{FRAMEDATA}([l.x, l.y, l.z])\}$
20:                 append $fd$ to $FD$
21:             end for
22:         end for
23:     end if
24:     return $FD$
25: end procedure
```

Interpolation is the ability to find data in between other data [6]. The pseudocode for the procedure used for these calculations can be seen in `CalculateLineEquation`, found in Algorithm 2.

This procedure takes two `FrameData` objects as input. However, the procedure can be used for every object containing an `x` and `y` coordinate. As the coordinates change according to the movement of the landmarks, several security checks are made to not provide inconsistent data or crash the system. The first check occurs in lines

Algorithm 2 The "Calculate direct graphs" Algorithm from step 6 in Figure 1

```
1: procedure CalculateLineEquation($p1, p2$)
2: Description: Given two sets of X and Y coordinates, $p1$ and $p2$,
   return a string $G$ representing the graph equation.
3:     $cp1 \leftarrow p1$
4:     $cp2 \leftarrow p2$
5:     if $cp1.x > cp2.x$ then
6:         $cp1 \leftarrow cp2$
7:         $cp2 \leftarrow cp1$
8:     end if
9:     if $cp2.x - cp1.x = 0$ then
10:        return 0
11:    end if
12:    $S \leftarrow (cp2.y - cp1.y) \div (cp2.x - cp1.x)$
13:    $Y \leftarrow cp1.y - S \times cp1.x$
14:    $G \leftarrow$ "$Sx + Y$"
15:    return $G$
16: end procedure
```

5-7 as if the first coordinate's x-coordinate is higher than the second coordinate, the slope will be negative. The second check occurs in lines 9-10, where, if the subtraction of the second coordinate's x-value with coordinate one is 0, division by zero will be performed, which would cause the system to crash. If these checks are passed successfully, the slope is then calculated using the equation $(cp2.y - cp1.y) \div (cp2.x - cp1.x)$, and the y-intercept of the graph with the equation $cp1.y - S \times cp1.x$. These values get stored in a single string representation and returned to the caller of the function. This process constitutes the entire step *Calculate direct graphs*.

The final step, *save to JSON-files*, entails saving everything extracted and calculated from the previous two steps into JSON-files. This is done, as it makes it faster to reuse when needed, by providing the ability to skip the whole extraction and processing steps. These files are stored locally within the code repository.

3.2 Building the model

This section will entail a detailed description of the process of building all the developed models. It follows the seven steps depicted in Figure 2.

When it comes to deciding on an MI model, it is important to use a model suitable for the needs. This was the purpose of the step *Determine model type*. The Sequential model[9] has been chosen as the model type to be used for all the models produced. This type of model is appropriate to use when the input comes as a list of layers, such as the list of all frames within a video. For this, a Long short-term memory (LSTM) model has been used, which is a type of Recurrent Neural Network (RNN). The LSTM has been chosen as it allows the output of the previous sequence, in this case the previous frame, to be used as input in the current sequence. This allows for the model to predict patterns, such as the pattern of landmarks within each frame. Furthermore, the Sequential model requires the density to be determined. The density informs the model of how many types of results are possible to reach, which in this case would be three, as three different labels can be produced. Finally, it was determined that there would be three kinds of Sequential models to be developed. The first would only consist of sequences containing the coordinates extracted from the frames, the other would utilize the linear graphs calculated, and the final model would use a combination of both. These models will be presented as `coordinates`, `graphs`, and `combined` throughout the rest of this paper. These three kinds of sequential models were determined to be developed to evaluate their performance and accuracy regarding each other, as they all provide different approaches to reaching the same goal.

Furthermore, each of these models will have three variations, correlating to the amount of samples per label used for each model. These variations will be trained on 25, 50, and 100 of each sample. The number of samples per label used will be added to the end of the model name, e.g. `coordinates25`.

With the general model type and configurations, in conjunction with the number of different models and their variations, determined, the process advanced to the next step *Load data*, as depicted in Figure 2. This step entailed loading the data save into JSON files, as presented earlier. The data loaded had to be concurrent with the kind of model, therefore coordinate data would be loaded if the coordinate model was the model in question being trained.

When the data had been successfully loaded, the process progressed to the next step, *Transform to sequences*. This step entailed using the loaded data and transforming it into sequences consistent with the format required by the Sequential model. The pseudocode for this procedure can be seen in Algorithm 3, and is contained in the procedure `TransformToSequences`. When the data is loaded from the JSON files using the `json` python module, they are stored within a dictionary object. This procedure takes this dictionary and converts it to a list of lists. The outermost list represents a video, with each containing a sublist representing a frame. Each of these sublists, representing frames, contains a single list. This innermost list contains all data regarding the coordinates and/or linear equations. In Listing 1, a pseudo-representation of this structure can be seen.

```
[ outmost list, representing a video
 [ inner list, representing a frame in the video
  [ innermost list, containing the coordinate/graph data
    values...
  ]
 ],
 ... repeat for amount of frames in video
]
```

Listing 1: Sequence structure

When the data has been transformed, the list is flattened, to remove all unwanted empty sub-lists occurring during the transformation process. This list is then returned to the caller of the procedure, which will be explained later in the step *Train model*.

With the transformation of all video data, the process moves forward to step *Determine sequence length*. This step entails figuring out the longest possible sequence within the entire set of video data. This length is dependent on the amount of frames within a video. With this length calculated, all other sequences not matching the length, are padded with the value 0. This ensures consistency within the data set and is required for the model to successfully predict a

[9]https://keras.io/guides/sequential_model/

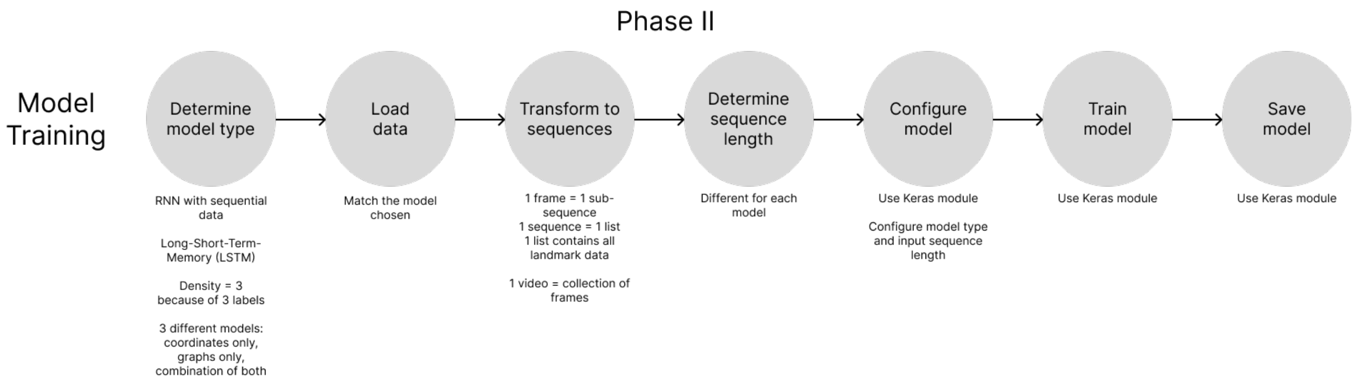


Figure 2: Phase 2, *Model Training*, and the seven steps it entails. This process constitutes a full sprint, lasting three weeks.

single gesture. If the length were of different variation, more than one label could be produced. This process of determining length and padding sequences can be seen in procedure `LoadModelData`, found in Algorithm 3, in lines 14-15. These two operations constitute this entire step.

The next step, *Configure model*, utilizes the configurations determined in step *Determine model type*. This configuration is made in procedure `TrainModel`, found in Algorithm 3, in lines 16-19. This utilizes the methods provided by the python Keras module[10], which allows for easy configurations.

With all data preparations and the model configuration done, the next step, *Train model*, involves training and fitting the model. This is done with a single function call, again utilizing a method from the Keras module, which can be seen in procedure `TrainModel` in Algorithm 3 in line 20. This function uses the padded sequences, the corresponding labels for each sequence, the number of epochs to perform, as well as the number of batches as input. The method used returns a history object, containing the accuracy and loss of the trained model, which will be elaborated in section 4. With the model trained, the last step, *Save model*, entails saving the recently trained model into a local Keras file, allowing it to be loaded and used at another time.

3.3 Bundling the API

To access and use the model to predict gestures, the entire solution has been bundled into an API. The process of initializing the API and bundling the prediction functionality is divided into seven steps, as depicted in Figure 3. This section will contain a detailed description of each step, as well as the decision made.

The first step, *Determine structure*, concerns determining the structure of the API. As the model prediction does not require storing any data to successfully predict a label, the RestAPI architecture has been chosen. As Keras, OpenCV, and MediaPipe all have been implemented using Python as the programming language, it has been decided to use this for the API as well. For the practical implementation of the API, the module `FastAPI`[11] has been used. `FastAPI` is a web framework used for developing Python-based APIs. This framework was chosen due to its native high performance and its ease of use. In conjunction with this, `uvicorn`[12] is used to deploy a web server containing the API. `uvicorn` is best known for being a minimal low-level server/application interface, acting as an ASGI web server for Python-based applications and systems.

With the structure determined, the next step, *Initialize framework*, entailed initializing the `FastAPI` and `uvicorn` module. This can be seen in procedure `RunAPI`, located in Algorithm 4. When the API was initialized, it loaded all nine models trained earlier using the Keras module. In the pseudocode in Algorithm 4, an example of how a single model was loaded is seen in line 1, using the `load_model` method. This method requires a string representation of the model's path to locate the model. This method constitutes the entirety of step *Load model(s)*.

The next step was to determine the public endpoints and develop the video processing procedure within the API, which the user had to communicate with to obtain a prediction for their video. It was determined that the API had a different endpoint for each of the models, allowing public access to all three approaches and their different variations. An example of this, specific to the coordinate models, is located in procedure `Predict`, seen in Algorithm 4. This endpoint requires the request received to contain a video file, which then in turn saves the video as a temporary file on the machine. This temporary file is then processed with the data frame extraction and processing procedures, `ExtractFrames` and `ProcessFrames` from Algorithm 1, presented earlier. Following this, `Predict` utilizes the data transformation functionality from `TransformToSequences`, also presented earlier. Furthermore, it also utilizes the functionality of padding sequences to the length determined during phase II. Finally, with the data transformation completed, it can utilize the loaded model to predict the gestures performed, using the `Predict` method from the Keras module. With the prediction performed, the label is extracted and converted to a string, which is then returned as a `JSONReponse` to the user. When the response has been returned to the user, the temporary video file is deleted, thereby ensuring privacy. The process explained here constitutes step *Determine endpoints* and *Develop video processing* shown in Figure 3.

[10]https://keras.io/
[11]https://fastapi.tiangolo.com/
[12]https://www.uvicorn.org/

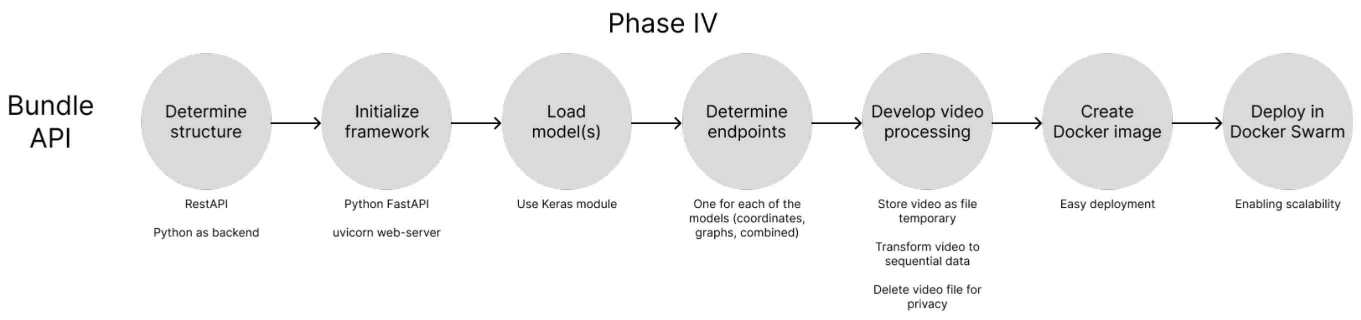


Figure 3: Phase 4, *Bundle API,* and the seven steps it entails. This process constitutes a full sprint, lasting three weeks.

With the API developed, a Docker image was created. This was done during the step *Create Docker Image*. Creating a Docker image for the entire system and bundled API allows for easy deployment, independent of the machine used. Furthermore, through the utilization of the Docker image and the Docker engine, the system could be deployed in a containerized environment using Docker Swarm. Docker Swarm allows developers to deploy more replicas of the same system on the same machine, without changing anything in the implementation. Meaning, that when the usage of the system increases, the number of instances can be increased as well, allowing for the system as a whole to handle higher loads. It therefore opens the door for scalability within the final system, which will be tested in section 4. Deploying the system with Docker Swarm constitutes the last step of this phase.

With phases I, II, and IV explained, one phase remains, the *Model evaluation phase*. This phase will be explained in section 4, which entails an elaborate description of testing the different parts of the system.

4 TESTING

This section will contain a formal description of the test processes conducted to evaluate the system capabilities in terms of Oracle's quality of service (QoS) requirements [24]. In total, 4 experiments have been performed. Within each presentation of the experiment, its relation to Oracle's QoS will be elaborated.

4.1 Test #1: Model Metrics

This experiment entails benchmarking the model, extracting specific metrics, and analysing the results of them through comparison with the suggested standard [15]. This is a common practice to engage in when developing new MI models, as it gives an objective quantification of the model's performance [15]. This entire process describes phase III of developing the system and is depicted in Figure 4.

The metrics of interest in this test are summarized below:

- *Accuracy*: is the metric describing how often predictions equal labels [30].
- *Loss*: measures the difference between the predicted class probabilities and the actual class labels [30].
- *Recall*: is the model's ability to find all positive samples [30].
- *Precision*: is the ability of the model not to label a negative sample as positive [30].
- *F1-score*, also known as the F-measure, is classified as the weighted harmonic mean of the results extracted from *precision* and *recall* [30].

To extract these metrics from the saved trained models acquired in the final step of phase II, Figure 2, a workflow was created. This entire process was repeated for all the nine models created and described in section 3. The first step, *Load model*, entailed loading the saved model through the usage of the Keras function `load_model`. The next step entailed using the fitting function, presented in procedure `LoadModelData` from Algorithm 3 in line 20, to extract the accuracy and loss of each model using the Keras functionality. These metrics were saved to a local .csv file. As the tests do not always provide the same results, this extraction process was conducted 10 times for each of the aforementioned models.

The next step, *Extract precision and recall*, entailed extracting the precision and recall of the loaded model. This was done through utilization of Scikit's `sklearn.metrics`[13] functionality, allowing for easy extraction of the wanted metrics. With both the precision and recall of the model extracted, these values were then used in the following step, *Calculate F1-score*. For this another of Scikit's functionalities was used, however, it was also calculated manually with the following equation: $F1 = 2 * ((precision \times recall) \div (precision + recall))$ These three metrics were when saved and stored in a local .csv file. These two processes were also performed 10 times for each model.

With all of these metrics extracted and saved, the next step was to perform a train-test split test of the model. As the train segment of the model was already performed during fitting, the next step, *Load test data*, involved loading 20 videos not used for training. These videos underwent the same frame extraction and processing procedure, as presented in `LoadModelData` from Algorithm 3, as the training data did. The next step, *Predict test data labels*, was to use the loaded models to predict the gesture and produce the correct label for each video. This procedure is similar to the prediction performed in procedure `Predict` from Algorithm 4. These produced labels were compared to the actual label, with all results saved into .csv files as well. The final step of this phase, *Evaluate results*, involved analysing and evaluating the results gathered from this phase, which will be done in section 5.

[13]https://scikit-learn.org/stable/modules/model_evaluation.html

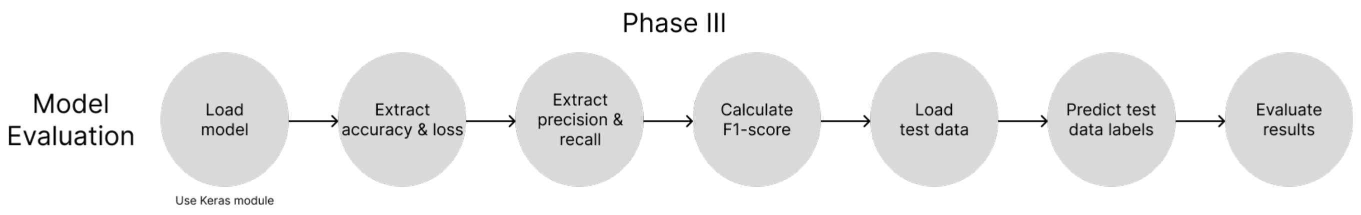


Figure 4: Phase 3, *Model Evaluation*, and the seven steps it entails. This process constitutes a full sprint, lasting three weeks.

4.2 Test #2: Load Testing

This test focuses on the QoS requirements regarding scalability, reliability, and availability of the bundled application within the API. This involves sending varying amounts of requests to the server, asking it to process and predict a gesture performed within a prerecorded video, all while monitoring metrics such as throughput, latency, and response time [12, 29]. Each of the three models, and their variations, will undergo four types of load testing scenarios, as described below:

- *Standard* load testing entails evaluating the API's performance under the expected amount of load. As this is not yet determined, as there is no official user base utilizing the system in its current state, this has been determined to be 10 users [12, 29].
- *Stress* testing entails testing the system's capabilities to perform under unusually high demand [12, 29].
- *Spike* testing entails sending varying amounts of loads towards the server, where some are more moderate and other of higher demand [12, 29].
- *Soak* testing concerns itself with subjecting an API to high request rates over a longer period of time. For this paper, the extended period of time has been determined to be 60 minutes, meaning 4x the standard used for the other tests [12, 29].

The open-source library, Locust[14], has been used to perform these tests. This has been chosen due to its ease of use and fast configurability. This was done by loading the 60 videos used for testing purposes, 20 of each label, into a Python script, followed by sending a post request, containing a video as the data body, to the API and awaiting the response. Each test ran for a total of 15 minutes, with all metrics recorded and dumped into a save-file, provided by locust itself.

Both the API handling the load, and the locust instance, were deployed on the same machine.

4.3 Test #3: Scalability Testing

As presented in section 3, the system is deployed within a containerized environment using docker swarm, allowing for more than one instance of the system running. This, by default, utilizes a load balancer, ensuring the load is as evenly distributed between the worker nodes as possible. The architecture allows for scalability testing, where the performance of the system is evaluated in terms of its ability to handle bigger loads when more resources are allocated between each test [32]. To test the scalability, the tool Locust, presented above, is once again used, and the only configurations in need of alteration are in the environment and not server-sided. The test will seek out to see, if changing the amount of containers in the environment, permits the system to handle higher loads more efficiently. This test tries to answer the question concerning if the amount of containers is scaled, will even higher loads, with reasonable metrics, be possible to successfully handle.

This experiment will run a total of 9 times, each time with different configurations. This amount of tests has been chosen, due to the number of containers within the environment, altering between 1, 3, and 5 containers running simultaneously. This, in conjunction with the simulation of 10, 100, and 1000 users, results in a total of 9 different tests. The amount of users simulated will be configured within the Locust UI, upon execution of the tool. The amount of containers is configured within the Docker-compose specification for the docker swarm cluster. The models, which are accessed through endpoints on the API, will be the ones trained on 100 samples of each label, resulting in 300 total videos used for training each model.

Both the containerized environment and the locust instance will be deployed on the same machine during these tests.

4.4 Test #4: Response Benchmarking

The final test entails testing the response time of the application. Response time is a conjunction of both server latency and processing time [20]. Processing time refers to the time it takes for the application to execute its functionality and compile the result, whereas latency refers to the time it takes to receive and send data from/to the client [20]. The processing time is evaluated through gathered data elaborating on the time spent per function executed within the system. The time variable directly correlates to the response time of the application. To gain the aforementioned metric, the python module `time`[15] has been used. The `time` module has been used to save the start time of each function, and once more when a function has ended. These times have then been subtracted from each other, providing the time taken in nanoseconds, which then later is converted to seconds. This was achieved using a regular function call from within the module and saving the values to a variable.

5 RESULTS

This section will entail the formal representation of the results gathered throughout testing and evaluation of the system. All results have been gathered through the tests described in section 4.

[14] locust.io/

[15] https://docs.python.org/3/library/time.html

Algorithm 3 The "Train model" and "Transform to sequences" Algorithms from step 6 and 3 in Figure 2

```
1:  procedure TrainModel
2:  Description: Loads and transforms model data, configured model, fits and saves model. Represents the algorithm used in step 6 from Figure 2.
3:      $F \leftarrow \{\text{LIST\_FILES}\}$
4:      $S \leftarrow []$
5:      $L \leftarrow []$
6:      $r \leftarrow \emptyset$
7:      for all $f \in F$ do
8:          $d, l \leftarrow \{\text{LoadJson}(f)\}$
9:          append $l$ to $L$
10:         $r \leftarrow \{\text{TransformToSequences}(d)\}$
11:         append $r$ to $S$
12:     end for
13:     $L \leftarrow \{\text{NP.ARRAY}(L)\}$
14:     $ml \leftarrow \{\text{MAX}(S)\}$
15:     $PS \leftarrow \{\text{PAD\_SEQUENCES}(S, ML, "post", "float32")\}$
16:     $lstm \leftarrow \{\text{Sequential}(128, (ml, 84))\}$
17:     $d \leftarrow \{\text{Dense}(3, "softmax")\}$
18:     $M \leftarrow \{\text{Sequential}([lstm, d])\}$
19:     $\{\text{M.COMPILE}([lstm, d])\}$
20:     $h \leftarrow \{\text{M.FIT}(PS, L, 10, 1)\}$
21:     $\{\text{SAVE\_MODEL}(M)\}$
22: end procedure

23: procedure TransformToSequences($d$)
24: Description: Given a dictionary of data $d$, convert and return a list-representation $s$ of the same data. Represents the algorithm used in step 3 from Figure 2.
25:     $s \leftarrow []$
26:     for all $f \in d$ do
27:         $l \leftarrow []$
28:         for all $ld \in \{\text{F.VALUES}\}$ do
29:             $\{\text{L.EXTEND(ld)}\}$
30:         end for
31:         $fl \leftarrow [i$ for $sublist \in l$ for $i \in s]$
32:         append $fl$ to $s$
33:     end for
34:     return $s$
35: end procedure
```

In Figure 5 a table providing an overview of the models trained, their abbreviations, and the amount and type of samples used for training can be seen.

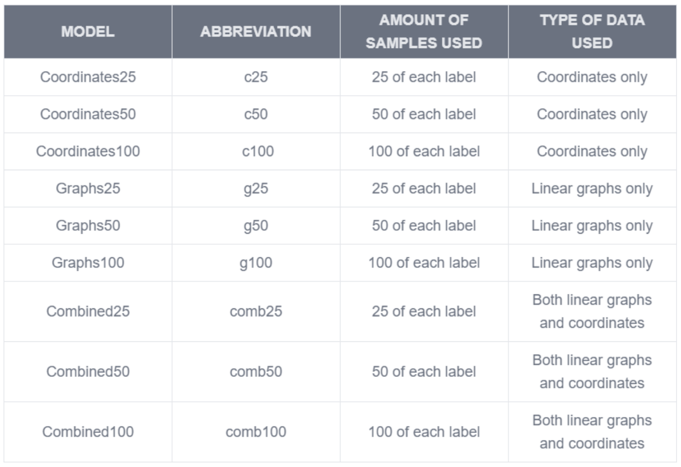
5.1 Model accuracy and loss

The first couple of metrics gathered were the accuracy and loss of each model. These metrics were gathered simultaneously upon training the model, and saved into a variable where they could be extracted. This process of gathering and saving the metrics is depicted in procedure `LoadModelData`, from Algorithm 3, in line 20, where `h` is an object that contains both metrics. This was done

Algorithm 4 The API Algorithm

```
1:  $M \leftarrow \{\text{LOAD\_MODEL}(PATH)\}$
2:  $MR \leftarrow \{\text{APIRouter}("/model")\}$
3:  procedure Predict($V$)
4:  Description: Given a Video file $V$, return a JSON-reponse $R$ containing the prediction.
5:      $v \leftarrow \{\text{V.READ}\}$
6:      $t \leftarrow \{\text{CreateTempFile}(v)\}$
7:      $F \leftarrow \{\text{ExtractFrames}(t)\}$
8:      $D \leftarrow []$
9:      for all $f \in F$ do
10:         append $\{\text{ProcessFrames}(f)\}$ to $d$
11:     end for
12:     $d \leftarrow \{\text{ConvertListToDict}(D)\}$
13:     $r \leftarrow \{\text{TransformToSequences}(d)\}$
14:     $s \leftarrow [r]$
15:     $ml \leftarrow sequence_lengths["c100"]$
16:     $ps \leftarrow \{\text{PAD\_SEQUENCES}(s, ml, "post")\}$
17:     $p \leftarrow \{\text{M.PREDICT}(ps)\}$
18:     $pl \leftarrow \{\text{NP.ARGMAX}(p, 1)\}$
19:     $l \leftarrow pl[0]$
20:     $R \leftarrow \{\text{JSONResponse}(\{"prediction" : l\}, 200)\}$
21:     return $R$
22: end procedure

23: procedure RunAPI
24: Description: Deploy the API, running on port 8000.
25:     $API \leftarrow \{\text{FastAPI}\}$
26:     $\{\text{API.INCLUDE\_ROUTER}(MR)\}$
27:     $\{\text{UVICORN.RUN}(API, 0.0.0.0, 8000)\}$
28: end procedure
```



MODEL	ABBREVIATION	AMOUNT OF SAMPLES USED	TYPE OF DATA USED
Coordinates25	c25	25 of each label	Coordinates only
Coordinates50	c50	50 of each label	Coordinates only
Coordinates100	c100	100 of each label	Coordinates only
Graphs25	g25	25 of each label	Linear graphs only
Graphs50	g50	50 of each label	Linear graphs only
Graphs100	g100	100 of each label	Linear graphs only
Combined25	comb25	25 of each label	Both linear graphs and coordinates
Combined50	comb50	50 of each label	Both linear graphs and coordinates
Combined100	comb100	100 of each label	Both linear graphs and coordinates

Figure 5: All models developed, their respective abbreviations, amount of samples used, and type of data, used to provide an overview.

for each of the models. The accumulated results of ten total tests for each model, were used to provide an average accuracy score for each model and can be seen in Figure 6.

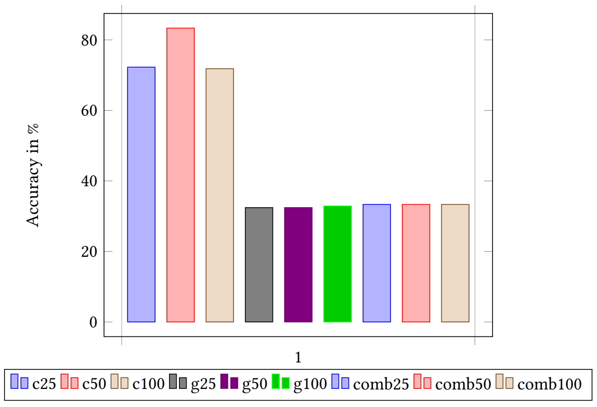


Figure 6: The accumulated average accuracy of all the 9 models, gathered during training and fitting of the models.

These results reveal that all models trained only on coordinate data have performed fairly well. The industry standard for all metrics regarding prediction models is between 70- and 90%[26], meaning the models c25, c50, and c100 all manage to fulfil this standard. The remaining model, however, does not fulfil this standard, by having results around 32-33%.

The same approach of extracting the metrics 10 times for each model was also utilized when extracting the loss of each model. These values were used to provide an average loss score for each model, and can be seen in Figure 7. However, this figure only shows the results for the coordinate models, as the remaining models' loss scores were extracted as a "NaN" value. Why this occurred is discussed in section 6.

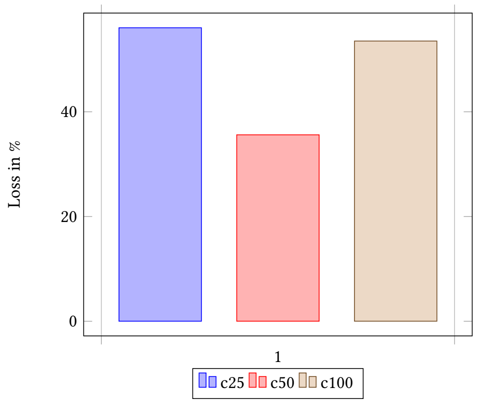


Figure 7: The accumulated average loss of all the 9 models, gathered during training and fitting of the models.

As shown by the results from Figure 7, the coordinate models do not manage to adhere to the industry standard [26]. The loss function used for this metric is *sparse categorical cross-entropy*, and the closer the result is to 0, the better. Therefore, having loss values of 35–56% means the models are underperforming.

5.2 Precision and recall of the models

As mentioned in section 4, the precision and recall were extracted using Scikit's `sklearn.metrics` module. Each metric was extracted 10 times each, to calculate an average value, same as before with accuracy and loss. In Figure 8, the accumulated average precision score of each model can be seen. The closer a value is to 100, the better the score.

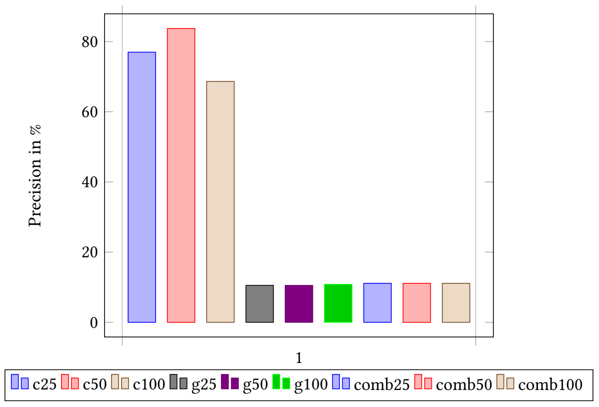


Figure 8: The accumulated average precision of all the 9 models, gathered after training a model, saving it, and loading it. All values are in percent, and the results gathered are multiplied by 100.

The results depicted in Figure 8 show, the values for c25, c50, and c100 manages to fulfil the aforementioned industry standard[26], with scores ranging between 68-84%. The remaining models, once again, underperforms with significantly low values, reaching a max value of 11.11%. This means the last six models fail to adhere to the industry standard.

The same approach, of extracting the metric 10 times for each, was performed for the recall metric. These 10 values were used to create an average. The averages for each model can be seen in Figure 9.

As presented by the results found in Figure 9, the coordinate models adhere to the industry standard, however, the remaining graph- and combined models do not. The coordinate models' values range between 70-85%, whereas the other only manages to score values around 33%.

5.3 F1-scoring

The F1-score, as mentioned in section 4, is calculated using the precision and recall scores of each model. For calculating the F1-score for these models, a total of 10 calculations has been done for each model, where an average of the 10 values has been used as the final result. The results of this is depicted in Figure 10.

As the results in Figure 10 show, the spread between the coordinate models and the rest is large. The coordinate models yielded values around 75%, whereas the remaining models were around 16%. This means that 1/3 of the models, specifically the coordinate models, adhere to the industry standard for MI-model metrics[26].

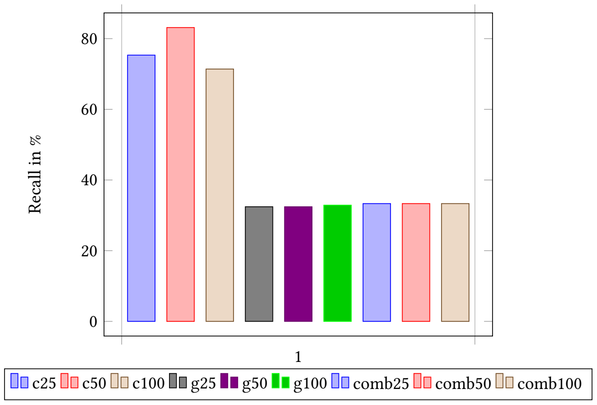


Figure 9: The accumulated average recall of all the 9 models, gathered after training a model, saving it, and loading it. All values are in percent, and the results gathered are multiplied by 100.

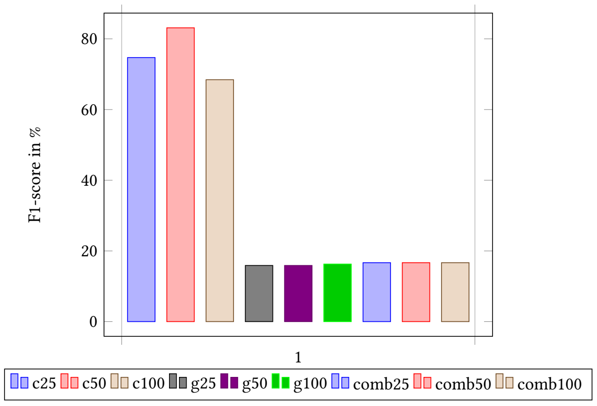


Figure 10: The accumulated average F1-score of all the 9 models, gathered during training and fitting of the models. All values are in percent, and the results gathered are multiplied by 100.

5.4 Test split

This next test involves testing each model's ability to predict a gesture, given a video not used within the training segment of the model. This means these videos represent input that could be sent from a potential user. Each model was tested with the same 60 pre-recorded videos. In Figure 11 the results of the amount of true positive predicted is shown.

As depicted in the graphs from Figure 11, the coordinate models all did fairly well. They all managed to successfully predict 16 of the samples labelled "can" and 90-100% success rate on predicting "thumb". However, c25 did not manage to predict any of the "peace" gestures, but when adding more samples to the training, the model succeeded with 11 true positives. The remaining models all managed to predict the samples representing the gesture "can", however managed to not predict a single one of the others successfully. This could indicate these models predict everything as "can", disregarding the other possible labels.

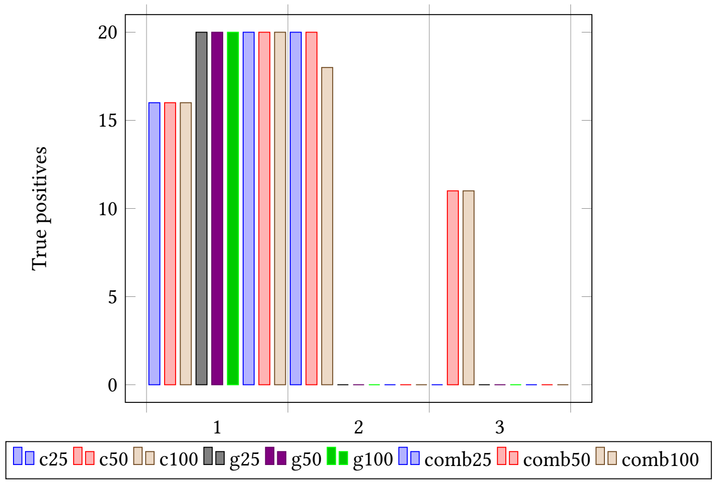


Figure 11: The accumulated amount of true positive prediction. "1" containing the results for the label "can", "2" containing the results for the label "thumb", and finally "3" containing the results for the label "peace". All values are in whole numbers.

Overall, the coordinate models managed to yield fairly positive results in terms of all metrics except loss. However, the high loss values seemed not to have a significant meaning for the general prediction process, as they were still quite successful in producing the correct labels for the test samples, as shown in Figure 11. The graph and combined models, however, did not yield any acceptable results during the entire testing process.

5.5 Response time

A system's response time is split in two different parts; latency and processing time [20]. If we have a look at the response time of the system developed, this metric describes the amount of time used by the system for executing the code and its corresponding functionality. The amount of time it takes to load a video, process it, and predict the gesture has been tracked for each model. The total amount of times this has been tracked is 10. These values were then used to calculate the average processing time. In Figure 12 the results can be seen.

As shown by the results from Figure 12, the average processing time of the functionality is of small variation between the models. The biggest factor that could be causing the difference, is that the graph and combined models need to calculate the linear graphs using the coordinates extracted from the video. Therefore, additional actions need to be performed, involving looping an additional time through the entire data set extracted. However, this variation is shown to only be up to a maximum of .42 seconds.

When calculating the latency of the system, an instance of the API was deployed locally on the machine used for testing. This test was conducted 10 times as well, however, it was only conducted using the models that were trained using 100 samples of each label. The resulting average response time can be seen in Figure 13.

With the average response time calculated, these values, in conjunction with the average processing times of the models, can be

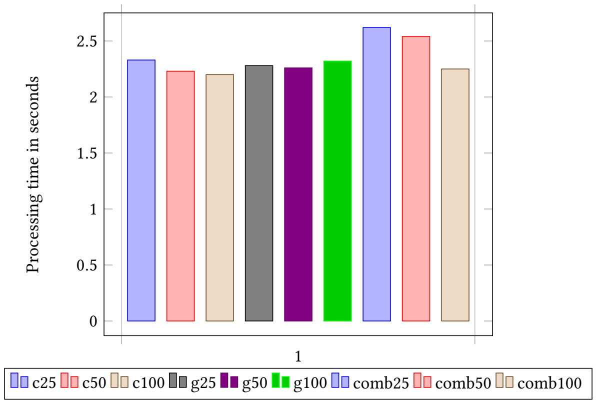


Figure 12: The average processing time of predicting a gesture with a given model. All values are in seconds.

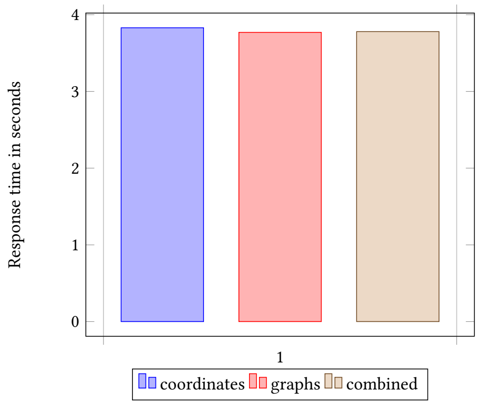


Figure 13: The average response time of predicting a gesture with a given model trained on 100 samples of each label. All values are in seconds.

used to calculate the average latency of the locally hosted instance of the system. The result of this is presented in Figure 14.

From Figure 14 it can be concluded that the average latency for each model is around 1.5 seconds in total. This time indicates that just from providing the video within the request made to the system, and waiting for the response to be received, the system already uses 1.5 seconds. This can come as a result of the size of the video, requiring additional time to have all its bytes successfully uploaded. This latency already exceeds the time that many high-performing APIS manage to adhere to. Furthermore, as the average response time exceeds 2 seconds, it will be noticeable within an application communicating with the system.

5.6 Scalability- and load testing

The results revealed by these tests aims to provide insight into the system's overall ability to scale as the user base increases, as well as to test how well it performs under different loads. Load testing is important, as it can not be expected that the same load will be present at all times, on the contrary, the load will be constantly altering. These tests have been performed to evaluate the system

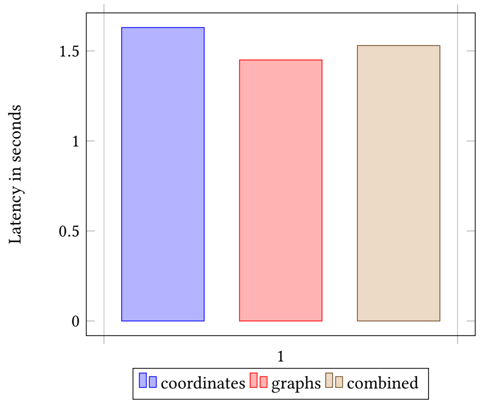


Figure 14: The average latency of predicting a gesture with a given model trained on 100 samples of each label. All values are in seconds.

regarding the QoS requirements concerning scalability and latent capacity [24].

A total of nine scalability tests have been conducted, utilizing 1, 3, and 5 containers within the environment. Each amount of containers has been tested with 10, 100, and 1000 users. All tests ran for 15 minutes.

The median and average response time of all the scalability tests can be seen in Figure 15 and Figure 16.

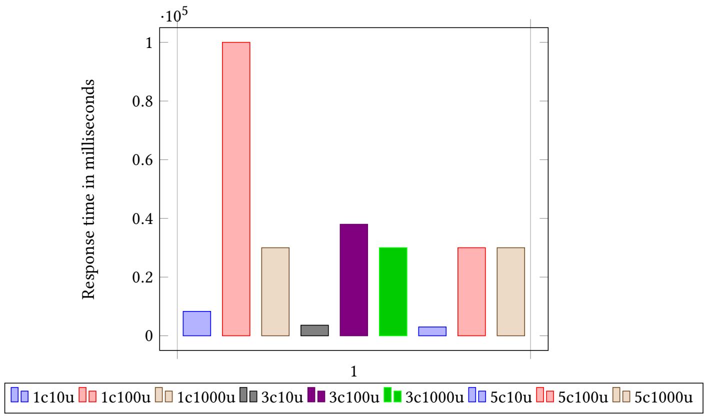


Figure 15: The median response times during each scalability test. All values are in ms.

As presented in Figure 15, all the tests that ran with only 10 concurrent users managed to provide reasonable median response times. These values are close to the response time tracked in Figure 13. However, when the load increases, such as handling 100 or 1000 concurrent users sending requests over a longer period of time, the median response time significantly increases. The lowest median value of any configuration handling 100 or more users is 30 seconds.

A similar conclusion can be drawn regarding the average response time. These results are depicted in Figure 16. As seen in the graph, once the amount of users scale with 1000%, the average response time also scales. This could be caused by either bad optimization within the architecture of the system, or the fact that additional CPU and GPU power did not get allocated as the amount of users increased.

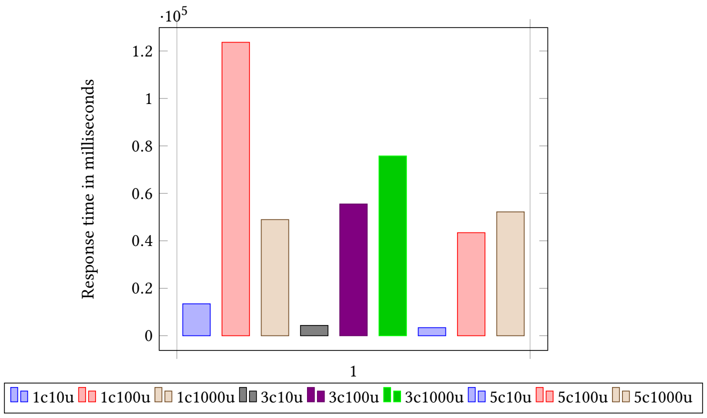


Figure 16: The average response times during each scalability test. All values are in ms.

On every configuration when running with 1000 concurrent users, the failure rate was between 80-99%. The majority of failures during these tests were "ConnectionResetError" or "RemoteDisconnected", due to the server having unacceptable response time and therefore closing the connection itself. With the error rate so high, the system was unable to process almost any response, and therefore the system is unfit for use regarding this amount of load. In Figure 17 the number of requests sent throughout each test configuration can be seen.

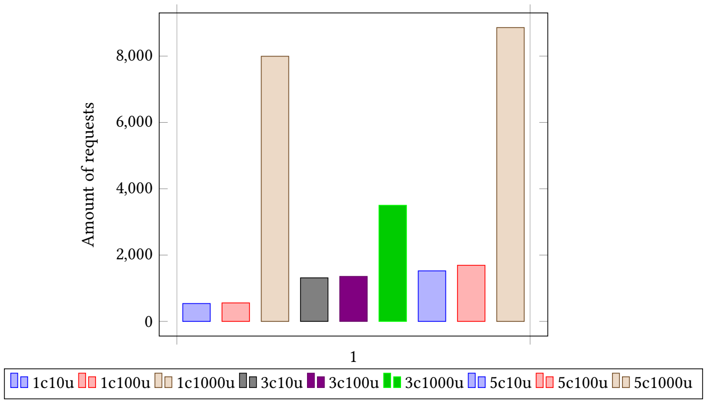


Figure 17: The number of total requests send upon testing each environment configuration. All values are in whole numbers.

As depicted in Figure 17, the number of requests possible to send scales simultaneously as the number of users simulated. However, the amount of requests sent also increases when the amount of containers running in the environment increases. This means that when increasing the amount of containers present in the environment, allows for higher throughput. All the requests sent did, however, not return a response successfully. This can be seen in Figure 18, where the number of requests resulting in an error, presented in percentage, is shown.

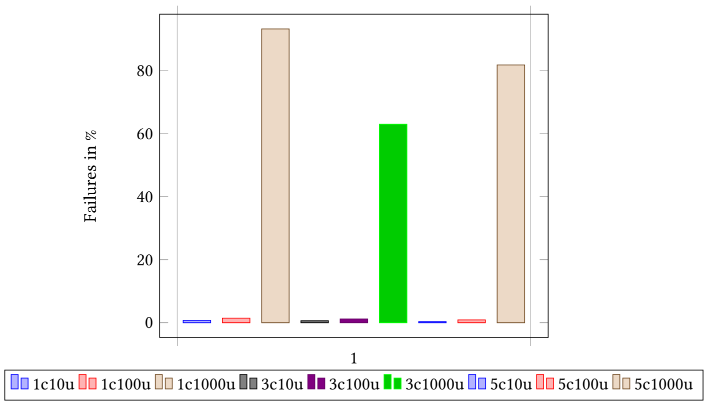


Figure 18: The percentage of failures within each of the tests.

As depicted in Figure 18, when the amount of users simulated is equal to 1000, the amount of failures is a significant part of the entire amount of requests, ranging in values between 63-93%. This means, that even though the system can receive a vast amount of requests, as shown in Figure 17, the amount of these requests that fail is unacceptably high. Therefore, scaling the users to a high amount, even with 5 containers deployed, provides an unstable system.

6 DISCUSSION

This section aims to formally discuss the study in general. This will entail an elaborated discussion about the results presented in the previous section, and comparing them to the related works presented in section 2. Following this, the limitations of this study will be presented and discussed, and it will end with a presentation of the future iteration concerning the system. But before any of this, the system fulfilment of Oracles Quality of Service[24] criteria will be discussed in detail.

6.1 Quality of Service fulfilment

The criteria determined to fully evaluate the usability of the system are Oracle's Quality of Service requirements. Every criterion affects each other. E.g., focusing on improving performance, greatly affects the security of a system. To utilize these criteria, to fully evaluate the system, an objective approach will be used during evaluation. This entails looking at each of the six criteria, evaluating the system's fulfilment of it, and then, in the end, giving a full assessment of the usability of the system, from a user's perspective. The QoS requirements evaluated during this section are performance, availability, scalability, security, latent capacity, and serviceability.

Beginning with **performance**, a good performance is reached when the system has reasonable response time and throughput. The assessment of this can be drawn from the response time recorded and depicted in Figure 13. These results revealed that the system was unable to optimally handle more than 2 users sending requests per container within the environment. This indicates that the architecture and structure of the API are suboptimal, and should be considered alternation, to successfully handle bigger loads. If the user base gradually grew, or spike loads happened, the yielded response times would be too slow, and greatly harm the usability of the application communicating with the system. In terms of the performance criteria, the system has been unable to provide acceptable response times under a reasonable users-to-container ratio greater than two-to-one.

The **availability** of the system, and its underlying docker swarm environment, is, however, a great success. If an instance of the system, running within a container, crashes or malfunctions, the environment would automatically restart and deploy a new container, as well as connect it to the underlying load balancer. This would result in a close to constant uptime of the environment and system, therefore allowing the users to utilize its functionality every day and night of the week. The only time the system would experience downtime is during a scheduled restart or maintenance, or if all containers ended up crashing at the same time, therefore requiring a full environment reboot.

The **scalability** of the system is hard to determine, as the ratio between the amount of users and containers is so low per default. However, the results depicted in Figure 13, also depict that when trying to scale the system to a higher load of concurrent users, deploying additional containers did have an overall positive effect on both the average and median response times. Therefore, with the results gathered throughout the testing and evaluation of the system, the scalability criteria can be assessed as possible, however, it would require a vast amount of resources both in terms of hardware, and potentially a refactor of the entire backend architecture. This can be concluded since, during each test, the amount of RAM allocated by the entire containerized environment was monitored. It reached its max of 14 GB during the test involving 1000 users and 5 containers (5c1000) and reached its lowest amount of RAM allocated (2.4 GB) during the test involving 10 users and 1 container (1c10u).

Security throughout the development of this system has not been of any priority. This is due to the system not requiring any personal information about its users to be stored. The security criterion is therefore of non-importance regarding the system and its usage. Evaluating this criterion objectively does not alter this opinion, and therefore it can be concluded that this criterion is redundant.

The **latent capacity** criterion evaluates how well a system can handle unusual peak loads without requiring any additional resources to be allocated. The keyword here is *unusual peak loads*, and with the results gathered and depicted in Figure 13, it can be determined that the system does not fulfil this criterion. The same scenario is present here as mentioned above, involving a big refactor of the architecture and structure of the developed API. This could lead to a big performance increase and therefore could result in higher general performance and fulfilment of this criterion.

Regarding the final criterion, **serviceability**, this has not been a priority throughout the design and development of the final system. This criterion entails making the deployed system easily maintained through monitoring, repairing problems as well as upgrading hardware and software components. However, objectively, upgrading the hardware within the deployed system is easily done, as it has been configured to deploy into a containerized docker swarm environment. This means the system functions independently of the operating system and hardware components in the physical server. To deploy the system, the docker engine needs to be installed in the machine, thereby allowing for easy deployment and additional configuration. However, a tool lacking within the implementation is a monitoring tool such as Jaeger[16]. Installing Jaeger would allow system administrators and developers to monitor every request made, to gather data on where the system may experience bottlenecks. This could be used to compile a list of possible improvements done for both the software and hardware used to deploy the system.

To conclude, the system has not successfully fulfilled the QoS criteria set for it. However, the next move following this conclusion would be to refactor the system significantly, to increase the API's general performance. The model, and its underlying prediction functionality, yield great results and therefore are fit for use. This means that, if possible, the prediction of gestures could be packed into an application utilizing the model to predict gestures without the involvement of the API. To determine the possibility of this, the number of total resources used should be gathered and analysed, to determine on which platforms such as a solution would be feasible. These platforms could include a mobile application or a webpage.

6.2 Model metrics

The previous section entailed the factual presentation of the results gathered from testing the system. These results will be discussed throughout this section, highlighting both the positive and negative side of their meaning.

Stating with **accuracy**, the graphs depicted in Figure 6 show the mean accuracy for all nine models. Where the models trained only with coordinate data revealed positive results, with stats ranging from values between 71-83%. This adheres to the industry standard of metrics ranging between 70-90%[26], meaning these models are functional in terms of accuracy. However, the remaining six models, all showed results around 32%, which in terms, hurts the model's overall reliability, as this metric described how often the model will return a valid label, given a known gesture. Not only do these results not live up to the industry standard, but it also means, given that the data used for training has a corresponding label, that the model is unfit for use within any application. The reason for this metric being of such low quality could be because of an inconsistency within the data used for training. As the model type used within the testing is an RNN, it can not be compared directly to the CNN developed by Pathan et al. presented in the section 2, as each model has their strength and purpose. However, they managed to receive a 98.981% test accuracy, which led to a successful prediction of all of their data, meaning the models developed in conjunction with this paper severely underperform.

In terms of **loss**, the metric closely associated with accuracy, this metric was not possible to extract for the graph- and combined models, as it provided a NaN result. For the coordinates models, however, this metric was also fairly high. This means, more often than not, these models would provide an incorrect label given a known gesture, with values ranging between 35-56%. However,

[16] www.jaegertracing.io/

comparing this to Pathan et al., their model was trained with 500 images of each gesture, resulting in a five times bigger data set. Their loss was fairly low, and as shown in their results, the loss of their model slowly decreased as the model underwent more epochs. The models presented throughout this paper only used 100 samples of each label and were saved in video format, which in turn resulted in more frames to be processed. This could explain the high loss values gathered through the evaluation, as both the total amount of data was smaller, and videos containing more data to be exactly compared to images. To compare these studies more equally, more data samples for each label should be recorded and processed, to make the models more comparable in terms of sample size.

The **precision** metric for Pathan et al.'s model yielded yet again good results, with every result being close to perfect. Compared to the results gathered from the models developed in conjunction with this paper, depicted in Figure 8, Pathan et al.'s model greatly outperformed these. The best results, extracted from this paper's models, ranged from high 60s to low 80s, gained through testing of the coordinates models. They adhere to the industry standard, where Pathan et al.'s model severely exceeds it, meaning that these models are acceptable to be implemented and used. If this metric is equal to "1", it means that the model is as perfect as can be, meaning that comparing my models with Pathan et al.'s would be redundant, as they are not reasonable in their results, as it is deemed close to impossible to have an advanced model with these results. This same argument can be used for the **recall** metric, where Pathan et al. managed to reach values close to perfection once more, whereas my coordinate models managed to maintain their position within industry standards. This allows the coordinate models to be deemed fit for use. The same goes for the **F1-score**, where the coordinate models managed to maintain their position within the industry standards, whereas Pathan et al.'s model manage the 99-percentile result.

However, these arguments can not be said for the graph- and combined models, which achieved poor results in every test conducted and failed to adhere to the standard [26]. As all models used the same model configurations, this could indicate an error within the data after being processed, likely stemming from the conversion of the linear graph equations to a unique Float32 value. These values had a wide variety, some being over the millions, and when used in conjunction with the coordinate values ranging between 0-1000, this could lead to potential issues within the sequence recognition of the LSTM utilized. This could be combatted by changing the values of the float, from instead being a unique Float32 value of the string, to being the product of the slope and y-intersect values. This approach would make the equations no longer unique, however, it could result in smaller numbers and possibly better results.

6.3 API performance

This section aims to formally compare the results gathered through the testing phase of the bundled API and compare them to the solutions *Live Transcribe* and *Fingerspelling.xyz*, presented in section 2. As perceived by the results given in the previous section, the response time of the application is fairly slow, providing an average response time of around 3.5 seconds when having only to manage a single request. For a real-time system, such as *Fingerspelling.xyz* to teach children a subset of sign language, this solution could be used. However, for solutions such as *Live Transcribe*, where the deaf user is depending on it as their primary source of communication with hearing people, the system would provide the user with bad experiences. Having to wait 3–4 seconds between each gesture performed, also indicates that interacting with *Live Transcribe* through the keyboard provided would be more effective for the users. As depicted in Figure 15 and Figure 16, when the system experiences soak loads of 100 concurrent users, both the average and median response times increase significantly, reaching values up to 48 seconds. These long period between request and response also results in the system reaching an error state, such as timing out the request or closing the connection. This hurts the users' reliability to the system and therefore does not make the system fit for any practical use.

The reason for these slow response times can be caused by a variety of different things. One thing could be how the API itself is structured, another thing could be that the framework used is not the optimal choice for the purpose at hand. It could also be caused by the machine running the containerized environment. However, to determine which case causes the issue at hand, refactoring and monitoring of the system needs to be conducted.

6.4 The system's limitations

The gathered results from testing several aspects of the system performance could have been affected by certain limitations caused by the produced data or utilized hardware. Starting with the data. All video data was produced by a single entity, which could result in either a small or large variation between each sample. Having a large variation between the samples, causes the prediction model to have difficulties predicting the correct gesture, by lowering its confidence in the produced label. This could be the case with the comparison done between Pathan et al.'s model and the coordinate models developed in conjunction with this paper. Having a small variation within the training data could provide better metrics, however, it could increase a user's difficulty to perform a gesture correctly. If the system expects a certain sequence of data as input to successfully predict, having a small altercation between the trained data and new data trying to predict, could cause the wrong label to be produced. Furthermore, as all gestures were recorded by a single entity, it could have an impact on the model's prediction confidence when other users try to utilize its functionality. This is due to the difference in hand size, causing a difference in landmark placement. Another limitation within the data was the number of samples for each label being limited to 100. Pathan et al. used 500 pictures for each of their gestures, and the minimum industry standard is set to be 1000 [18]. This can cause the coordinate models to underperform within certain metrics, as they require at least ten times the amount of data to be representable.

Concerning the results gathered from scalability- and load testing, a limitation here, which may have affected the results, is the fact that both the containerized environment and locust instance were both running on the same machine during the testing. This means that the same machine had the purpose of sending, receiving, processing, and responding to every request, which may have caused an unweary load on the resources available, which in turn then affected the response times, error rate, and amount of requests possible to handle concurrently.

6.5 The deaf perspective

This paper aims to aid deaf individuals in bridging the gap within communication barriers, occurring when trying to integrate into a hearing society, through the development of a system to understand and predict Danish sign language. Their benefit from the developed system will be discussed. As mentioned, when discussing the results, the system, in its current state, is incapable of being integrated into any external application, as it is unreliable in terms of responses. However, when optimized, the system could be integrated into a mobile application, such as *Live Transcribe*, presented in section 2. As this product currently requires the deaf user to communicate through text, the system could be integrated, thereby enabling the ability to communicate through sign language, with the produced text being read aloud. This would accommodate deaf individuals with lower literary levels by allowing them to "speak" in their native language.

Furthermore, advancement and improvement within the developed system would allow it to be integrated into various contexts. The primary context is whenever a deaf individual would be forced to communicate with someone unable to understand sign language. These could be situations such as getting through airport security or doctor appointments. Here their independence could be strengthened, enabling them to participate without the need of others.

6.6 Future works

The process of developing the system and reaching its current state is, as earlier presented, split into four phases and are depicted in Figure 1, Figure 2, Figure 4, and Figure 3. These phases collectively represent a version of the system, where each phase, in the best-case scenario, represents a single sprint. However, a phase can be repeated until the desired results within the phase have been met. This means, that if a new version of the system were to be developed, these phases could be followed chronologically.

As suggested by the results of the scalability- and load testing of the API, a refactoring of the system is necessary to improve its processing time. This could include choosing another language, other than Python, to develop the API in. This could allow for potentially faster processing times, compared to the gathered results from Figure 12. Optimizing the processing time of the application's internal functionalities would have a great impact on the system's overall usability, as it would result in lower response times and make the system more fit for real-time system implementations. Latency could be combatted by getting a web host to deploy the API.

Another way to improve this is to structure the application into a microservice architecture. Through this, it would be possible to deploy additional containers to handle the resource-heavy procedures, thereby allowing for more users to be handled concurrently. For example, the MediaPipe processing was the procedure that took the longest amount of time of all the functionalities, as well as being the most resource-heavy. Deploying additional containers with the sole purpose of processing frames with MediaPipe, would allow more resources to be allocated for this specific purpose, enabling the system to perform this procedure more efficiently.

When the response times, and other results, have improved, a user study is suggested to be performed. As presented in section 2, Google performed a user study looking into the preferability and likeability of systems using sign language recognition, and yielded positive results [16]. However, they utilized Wizard-of-Oz prototypes, which may have had an impact on the result as it was not affected by response times and model performance. By having a functional implementation, the same test could be run to compare the results of both studies and determine if such a system is of interest to its user base. This test should be run with a group of people dependent on sign language as their primary form of communication.

7 CONCLUSSION

Throughout this paper, the objective has been to explore the domain of recognizing Danish sign language through a machine intelligence model. Other approaches within the field of sign language recognition have been conducted, however, they either focus on finger spelling or involve using additional hardware. The system developed, in conjunction with this paper, has focused on small videos containing only a single gesture, performed with one hand, as suggested by Google [16]. The functionality has been bundled into an API, and deployed within a containerized environment, allowing applications of different sorts to utilize the prediction functionality.

The agile approach has been split into four phases/sprints; *Produce data set*, *Model training*, *Model evaluation*, and *Bundle API*. The process begins with creating the data, followed by the necessary extraction and processing tasks to make them compatible with the Sequential model chosen for prediction. The next step involves using the data created to train and fit a machine intelligence model, able to predict gestures performed. This model is then in turn evaluated, benchmarked, and compared to both other studies and industry standards. Functionality to utilize the model is then created and bundled into a python FastAPI, packed into a docker image and finally deployed within a containerized docker swarm environment.

The proposed system contains a total of 3 different kinds of models; one utilizing only coordinates as data, one utilizing linear graphs, and one utilizing a combination of both data types. All data samples were labelled with one of three labels; "can", "thumb", or "peace". Furthermore, each type of model had three variants, representing the number of samples of each label they were trained on. The variations were 25, 50, and 100 of each label. All coordinate model variants yielded positive results and proved able to predict all three labels. The other variants of the two other models proved incapable of predicting successfully, as independent of the input, the label produced as output would always be "can". The API yielded poor results in terms of almost every quality of service requirement, hereby deeming itself unfit for use. A major refactor of the infrastructure within the API would solve these issues.

The novelty of this contribution proposes the usage of a mobile application, which sends small videos as payload to the developed API, and utilizes the prediction received. This attempts to accommodate the deaf individuals wanting to bridge the gap in the communication barrier present between them and a hearing society. Furthermore, the system attempts to accommodate the lower literary level present within the deaf community, to let them communicate more clearly.

REFERENCES

[1] [n. d.]. *The 2020 Scrum Guide.* https://scrumguides.org/scrum-guide.html

[2] [n. d.]. Fakta om Døvhed. https://www.cfd.dk/cgi-bin/uploads/media/pdf/R%C3%A5dgivning/Faktaark/Faktaark-doevhed.pdf

[3] [n. d.]. *Hand landmark model bundle.* https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer#hand_landmark_model_bundle

[4] [n. d.]. *Introduction to Landmark Detection.* https://www.baeldung.com/cs/landmark-detection

[5] [n. d.]. *Linear Graph.* https://www.cuemath.com/data/linear-graph/

[6] [n. d.]. *Linear Interpolation Explained.* https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/linear-interpolation-explained-r5892/

[7] [n. d.]. Reasonable adjustments in the workplace. https://www.ndcs.org.uk/information-and-support/professionals/workplace/reasonable-adjustments/

[8] 2023. https://ddl.dk/dansk-tegnsprog/

[9] 2023. https://www.android.com/accessibility/live-transcribe/#ready-to-start Official webpage for the application on Android.

[10] 2024. Deaf Culture | Sign Language "Accents" or "Styles". https://www.startasl.com/sign-language-accents-or-styles/ Accessed: 2024-05-01.

[11] 2024. Sign Language. https://education.nationalgeographic.org/resource/sign-language/ Accessed: 2024-05-01.

[12] Anna Irwin. 2023. API Performance Testing: Best Practices and Strategies. https://aptori.dev/blog/api-performance-testing-best-practices-and-strategies. Accessed: 2024-04-05.

[13] Daivd Lumb. [n. d.]. These Student Built A Glove That Translates Sign Language Into English. https://www.fastcompany.com/3059616/these-students-built-a-glove-that-translates-sign-language-into-english. Accessed: 2024-05-22.

[14] Alan Emond, Matthew Ridd, Hilary Sutherland, Lorna Allsop, Andrew Alexander, and Jim Kyle. 2015. Access to primary care affects the health of deaf people. *British Journal of General Practice* 65, 631 (2015), 95–96. https://doi.org/10.3399/bjgp15x683629

[15] Fatmanurkutlu. 2024. Model Evaluation Techniques in Machine Learning. https://medium.com/@fatmanurkutlu1/model-evaluation-techniques-in-machine-learning-8cd88deb8655. Accessed: 2024-04-05.

[16] Saad Hassan, Abraham Glasser, Max Shengelia, Thad Starner, Sean Forbes, Nathan Qualls, and Sam S. Sepah. 2023. Tap to Sign: Towards using American Sign Language for Text Entry on Smartphones. *Proc. ACM Hum.-Comput. Interact.* 7, MHCI, Article 227 (sep 2023), 23 pages. https://doi.org/10.1145/3604274

[17] Hello Monday. [n. d.]. Fingerspelling. https://www.hellomonday.com/work/fingerspelling. Accessed: 2024-05-20.

[18] Kili. [n. d.]. Evaluating data: How much training data do you need for machine learning? https://kili-technology.com/training-data/how-much-data-do-you-need-for-machine-learning. Accessed: 2024-04-05.

[19] Lena B Larsen, Steen Bengtsson, and Mette L Sommer. 2014. Døve- og døvblevne mennesker - hverdagsliv og levevilkår. https://viden.sl.dk/artikler/voksne/handicap-samfundsdeltagelse/doeve-og-doevblevne-mennesker-hverdagsliv-og-levevilkaar/

[20] Lazar Nikolov. 2023. What's the difference between API Latency and API Response Time? https://blog.sentry.io/whats-the-difference-between-api-latency-and-api-response-time/. Accessed: 2024-04-05.

[21] Pamela Luft. 2000. Communication barriers for deaf employees: Needs assessment and problem-solving strategies. *Work (Reading, Mass.)* 14 (02 2000), 51–59.

[22] S.A. Mehdi and Y.N. Khan. 2002. Sign language recognition using sensor gloves. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02.*, Vol. 5. 2204–2206 vol.5. https://doi.org/10.1109/ICONIP.2002.1201884

[23] Mihir Garimella. [n. d.]. Sign Language Recognition with Advanced Computer Vision. https://towardsdatascience.com/sign-language-recognition-with-advanced-computer-vision-7b74f20f3442. Accessed: 2024-05-20.

[24] Oracle. [n. d.]. Quality of Service Requirements. https://docs.oracle.com/cd/E19636-01/819-2326/gaxqg/index.html. accessed: 15/05-2023.

[25] Refat Khan Pathan, Munmun Biswas, Suraiya Yasmin, Mayeen Uddin Khandaker, Mohammad Salman, and Ahmed A. F. Youssef. 2023. Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network. *Scientific Reports* 13 (2023). https://doi.org/10.1038/s41598-023-43852-x

[26] Randall Hendricks. [n. d.]. What is a good accuracy score in Machine Learning? https://deepchecks.com/question/what-is-a-good-accuracy-score-in-machine-learning/. Accessed: 2024-04-05.

[27] G. Anantha Rao, K. Syamala, P. V. V. Kishore, and A. S. C. S. Sastry. 2018. Deep convolutional neural networks for sign language recognition. In *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES).* 194–197. https://doi.org/10.1109/SPACES.2018.8316344

[28] Katherine D. Rogers, Emma Ferguson-Coleman, and Alys Young. 2018. Challenges of Realising Patient-Centred Outcomes for Deaf Patients. *The Patient - Patient-Centered Outcomes Research* 11 (2018), 9–16. https://doi.org/10.1007/s40271-017-0260-x

[29] SauceLabs. 2022. API Load Testing Tutorial. https://saucelabs.com/resources/blog/api-load-testing-tutorial. Accessed: 2024-04-05.

[30] Scikit. [n. d.]. 3.3. Metrics and scoring: quantifying the quality of predictions. https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics. Accessed: 2024-04-05.

[31] Sharvani Srivastava, Amisha Gangwar, Richa Mishra, and Sudhakar Singh. 2022. Sign Language Recognition System using TensorFlow Object Detection API. *CoRR* abs/2201.01486 (2022). arXiv:2201.01486 https://arxiv.org/abs/2201.01486

[32] Thomas Hamilton. 2024. What is Scalability Testing? Learn with Example. https://www.guru99.com/scalability-testing.html. Accessed: 2024-04-05.

[33] Ankita Wadhawan and Parteek Kumar. 2020. Deep learning-based sign language recognition system for static signs. *Neural Computing and Applications* 32, 12 (2020), 7957–7968. https://doi.org/10.1007/s00521-019-04691-y

[34] World Health Organization. 2024. Deafness and hearing loss. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss. Accessed: 2024-05-01.