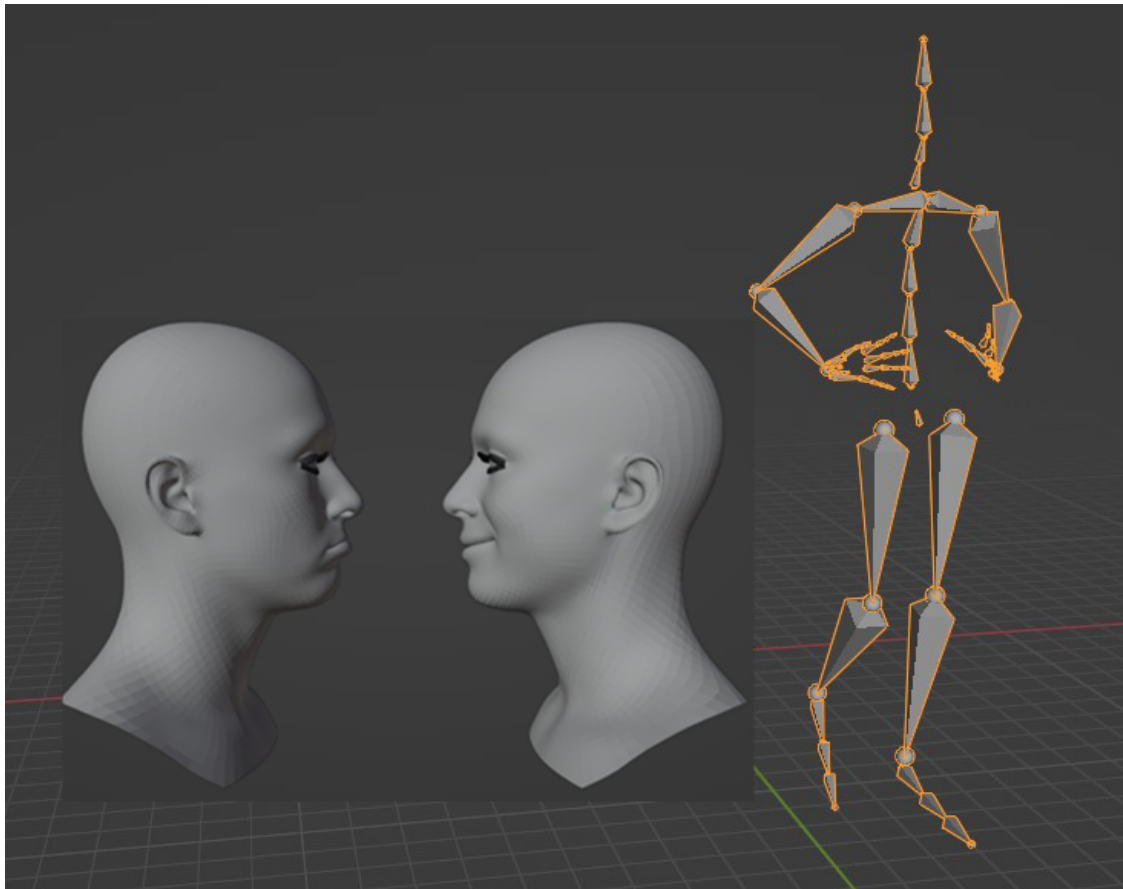




CHALMERS
UNIVERSITY OF TECHNOLOGY



Evaluation of Neural-Network and Large-Language Model Approaches for Generating Instructions for Animations

How machine learning can be used to animate talking avatars for educational purposes

Master's thesis in Complex Adaptive Systems

ELISABETTA BARLETTARO, EMMA ERIKSSON

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2024

www.chalmers.se

MASTER'S THESIS 2024

Evaluation of Neural-Network and Large-Language Model Approaches for Generating Instructions for Animations

How machine learning can be used to animate talking avatars for
educational purposes

ELISABETTA BARLETTARO
EMMA ERIKSSON



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Division of Technical Physics and Technical Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Evaluation of Neural-Network and Large-Language Model Approaches for Generating Instructions for Animations

How machine learning can be used to animate talking avatars for educational purposes

ELISABETTA BARLETTARO, EMMA ERIKSSON

© ELISABETTA BARLETTARO & EMMA ERIKSSON, 2024.

Supervisor: Martin Fröjd, Fictive Reality

Examiner: Bernhard Mehlig, Department of Physics, University of Gothenburg

Master's Thesis 2024

Department of Physics

Division of Technical Physics and Technical Mathematics

Chalmers University of Technology

SE-412 96 Gothenburg

Cover: Face blendshapes applied on an avatar and animated body armature shown in Blender.

Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Evaluation of Neural-Network and Large-Language Model Approaches for Generating Instructions for Animations

How machine learning can be used to animate talking avatars for educational purposes

Elisabetta Barlettaro, Emma Eriksson

Department of Physics

Chalmers University of Technology

Abstract

Conversational agents are used more and more in customer service, health care, for educational purposes. The fundamental problems of conversational agents are many, including limitations in interpretation of complex queries and lack of emotional intelligence. Despite this, there are distinct advantages of conversational agents, such as efficient data analysis, reduction of operational costs and aid in interactive learning for personalized teaching. The most significant challenge this project aims to undertake is to generate realistic and complex animations in the context of interactive learning with a real-time constraint. The investigation includes how to select machine learning tools and models to aid in the advancement of animation generation, by using both Large-Language Models and purposely constructed Neural Networks. While Large-Language Models are convenient when used in straightforward conditions, Neural Networks are more dependable in an operative application thanks to their consistent format, adaptability and specifically developed purpose.

Keywords: animations, armature, avatar, Blender, blendshapes, educational, large-language models, neural networks, Python, real-time, training simulator, Unity

Acknowledgements

We would like to express our gratitude to those who have supported us and helped us grow, to Fictive Reality for "daring us to fail" in order to learn from our failures, to our examiner for the precious suggestions, brilliant teachings and profound wisdom.

Elisabetta Barlettaro & Emma Eriksson, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis:

AI	Artificial Intelligence
BVH	Biovision Hierarchy
FACS	Facial Action Coding System
FPS	Frames Per Second
JSON	JavaScript Object Notation
LLM	Large-Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MSE	Mean Squared Error
NLP	Neural Language Processing
NN	Neural Network
RNN	Recurrent Neural Network
VR	Virtual Reality

Contents

List of Acronyms	ix
List of Figures	xiii
List of Tables	xvii
1 Introduction	1
1.1 Background	1
1.2 Objectives Goals	3
1.3 Significance of the Study	4
2 Theory	5
2.1 State of the Art for Animations Generation using Machine Learning .	5
2.2 Fictive Reality Pipeline	6
2.3 Machine Learning	7
2.3.1 Neural Networks	8
2.3.1.1 Activation Functions	9
2.3.1.2 Gradient Descent	11
2.3.2 Natural Language Processing	12
2.3.2.1 Chatbots	15
2.3.2.2 Text representation	16
2.3.2.3 Large-Language Models	17
2.4 Software Tools	17
2.5 Animations	20
2.5.1 Evaluating the Quality of Animation	21
2.5.2 Animation Tools and Formats	21
3 Methods	25
3.1 Approaches	25
3.1.1 Large-Language Models	25
3.1.2 Neural Networks	26
3.1.3 Video generation	30
3.2 Quality Testing	30
3.3 Approaching Challenges	31

4	Results	35
4.1	Face Animation	35
4.1.1	Generation of mood tags with an LLM	36
4.1.2	Generation of blendshapes with an LLM	36
4.1.3	Generation of blendshapes with a neural network model	37
4.2	Body Animation	42
4.2.1	Generation of motion tags with an LLM	43
4.2.2	Generation of motion data with an LLM	44
4.2.3	Generation of motion data with a neural network model	45
4.3	Video Animation	49
5	Discussion	53
5.1	Animation Generation with Large-Language Models	53
5.2	Animation Generation with Neural Networks	55
5.3	Video Generation	56
5.4	Comparison of the Approaches	57
5.5	Future Work	59
6	Ethics	61
7	Conclusion	63
	Bibliography	65
	References	65
A	Appendix 1	I

List of Figures

1.1	Example of a user engaged in Fictive Reality’s training simulation. Image taken from Fictive Reality’s web page.	2
1.2	Example of a scenario from Fictive Reality. Screenshot by author taken from Fictive Reality’s app.	3
2.1	The components of the Fictive Reality architecture that carries out the conversation between bot and user. The speech-to-text takes the user audio and changes it to text. The text is then used to generate an appropriate response for the bot using an LLM. Finally, the text output from the LLM is sent to a text-to-speech program, that generates the bots spoken answer to the user. Image by author. .	7
2.2	Feed-forward network with one hidden layer. The input terminals are coloured black. Picture taken from <i>Machine Learning with Neural Networks</i> by B. Mehlig under Cambridge Licence of Use.	8
2.3	Multilayer perceptron with three input terminals, two hidden layers, and two output neurons. Picture taken from <i>Machine Learning with Neural Networks</i> by B. Mehlig under Cambridge Licence of Use. . . .	9
2.4	Example of recurrent network with one input terminal, one hidden neuron, and one output neuron. Picture taken from <i>Machine Learning with Neural Networks</i> by B. Mehlig under Cambridge Licence of Use.	10
2.5	Four common activation functions. A: Linear, B: Sigmoid, C: TanH, D: ReLU. Image by author.	11
2.6	Example of hierarchy of bones in a 3D model imported into Blender software. Image by author.	18
2.7	An example of a State Machine in Unity. Image by author.	20
2.8	Blendshapes in action. On the left, the blendshape representing the puffing of the right cheek, in the project called CheekPuffRight. On the right, the blendshape representing opening the jaw, in the project called JawOpen. Image by author.	22
3.1	Fictive Reality’s settings during a session showing that the ID of a specific session can be inserted to be loaded. Image by author. . . .	31

3.2	On the left: The skeleton used in the dataset. On the right: The skeleton used for Fictive Reality’s avatar. They differ in many aspects, including joint number, hierarchy and scale. This complicates the implementation of the neural network output into Fictive Reality’s app. Image by author.	32
4.1	Two different blendshapes applied to a model, demonstrating the visibility of different parameter values. In the figures, the top left shows the model with the blendshape parameter set to 0, the top right to 0.25, the bottom left to 0.5 and the bottom right to 1. This illustrates that blendshape values below 0.5 are difficult to notice. Image by author.	37
4.2	Box plot of each blendshape value in the data, grouped in color by face parts. Most blendshapes show a high number of outliers and long whiskers. Image by author.	39
4.3	Mean value of each blendshape in the data. Blendshapes averages range from close to 0 up to 0.5. The image also gives an indication of which blendshapes are the most important. Image by author. . . .	40
4.4	Accuracy and MSE loss for training and validation data during the training of the face model. The increasing accuracy together with the decreasing loss for the training data shows that the model is trained, yet the behavior of the validation curves indicates the presence of overfitting. Image by author.	40
4.5	Distribution of blendshape values in different sets of the data. The distributions are in large similar, however the predicted output shows a heavier tail with a prominent saddle. Sub-figures in order: A: The distribution of blendshape values among the whole data. B: The distribution among the training data, which corresponds to 80% of all data. C: Another 10% of all data, now corresponding to the test (target) data. D: The distribution of values in the predicted data, which optimally aligns entirely with the target distribution. Image by author.	41
4.6	Average error per blendshape, when comparing target data to predicted output. In total this corresponds to an accuracy of 29.46%. Note that the highest average error is about 0.16, and recall that blendshapes can take values in the range [0, 1]. Lip sync blendshapes, i.e. jaw and mouth, are not included. Image by author.	42
4.7	Visualization of a few frames of the generated animation using Unity and one-hot encoding of the sentence " <i>Good, do you know where i can find the bus stop?</i> " as input to a neural network model. Shown from the front (top) and from the side (below). Image by author.	43

4.8	Box plot of each joint degree, with x, y and z coordinates separate. Each joint is grouped in color by relating joints. Note how corresponding right and left joints have similar distributions. Note also how the outer ends of the arms show large ranges in distribution of degrees. While some joints rarely show rotations larger than a few degrees, some have noticeable whiskers without outliers and yet others have a prominent number of outliers, for example the feet joints. Image by author.	47
4.9	Mean degree of each joint in the data. A: Mean degrees when each x, y and z coordinates of a joint are averaged. B: Mean degrees when x, y and z coordinates are held separate for each joint. Note: While some joints display averages close to degree 0 it does not necessarily imply that they are scarcely activated, however it implies that activation against one direction is countered by activation against the other. One can see here that joints corresponding to each other by movements that are generally mirrored in the body halves have means in different directions, see for example the last joint in right arm contra in left arm, or the fact that the right hand is predominantly negative while the left hand is predominantly positive. Image by author. . . .	48
4.10	Accuracy and MSE loss during training for both training data and validation data. This results show an example of when accuracy and loss can go in the same direction simultaneously. The jagged lines for validation is also worth noticing. Finally, the training show increasing accuracy and decreasing loss, which indicates overfitting. Image by author.	49
4.11	Degree Distribution among the joints. A: Distribution of joint degrees in all data, when filtered out data points with degrees outside the range $[-360, 360]$. B: The distribution of joint degrees used for training, equivalent to 80% of the data in the upper left sub figure. C: The distribution of joint degrees used for test, equivalent to 10% of the data in the upper left sub figure. D: The distribution of joint degrees among the predicted output when test data is passed through the model. Optimally, this has the same distribution as the target data. Image by author.	50
4.12	Mean degree error of each joint when comparing predicted output to target data for armature. A: Mean error when each joints' x, y and z coordinates are averaged. B: Mean error when x, y and z coordinated are held separate for each joint. Note: While some joints display large errors, it is important to note that they are read in a modded setting, meaning that a degree of value X is the same as $Y = X + n \cdot 360$, for $n \in \mathbf{Z}$. This indicates that even if the shown error is large, the animated output might not display these errors. Image by author.	51
4.13	Frames from animated output from body neural network model. Taken from the same angle. Image by author.	52

4.14 Three frames of a video generated using vidnoz.com/avatars.html for a text input describing the weather today. Image by author. 52

List of Tables

1.1	Table of hypotheses made in the study.	4
2.1	An example of the difference between tokens and word types. In this case is a token defined as text between two whitespaces, image by author.	13
2.2	A text before and after it is processed, image by author.	14
2.3	A showcase of differently based tokens of a sentence, image by author.	14
2.4	A conversation example between ELIZA and a user.	16
2.5	Example of one-hot encoding, where tokens are defined as distinct words. First, there is a source text and second, there are two sentences using words from the source text. There are 13 unique tokens, thus each one-hot encoded vector has 13 positions.	17
3.1	Outline of the architecture used for the face model, which includes the Adam optimizer, MSE loss function, and accuracy as its evaluation metric.	29
3.2	The overall architecture for the body model, using the Adam optimizer, the MSE loss function and accuracy as its evaluation metric.	30
4.1	Example of output when using the mood tags prompt from Table A.6 and Table A.7 (1). This shows that the mood tags update in accordance to the conversation.	36
4.2	Generated output from LLM with four blendshape values given in the range 0.5-1 for each avatar response. The LLM is successful in following the instructions, and chooses different blendshapes to update between the responses.	38
4.3	Output from LLM when generating four blendshapes at a time. The LLM successfully chooses four blendshapes in the assigned range for each avatar response. It also chooses different blendshapes to update throughout the conversation.	38
4.4	Two example outputs when using an LLM to generate three blendshapes to each avatar response. The LLM follows the instructions, however the avatar are forced to be asymmetric in its face expressions.	39

4.5	Example of output generated from the prompt shown in Table A.6 together with Table A.7 (4). This includes two motion tags per avatar response. Two movements at a time appears relevant. The LLM’s choices of animations takes at times, however, single words out of their context and no natural continuity between correlating animations, such as between "standing up" and "staying seated".	44
4.6	An example of generation of motion tags with three different movements as output to each avatar response. Using many movements at once, especially when with the complexity these have, seems forced and hence lacks in both realism and quality.	45
4.7	Output to a prompt printing out motion data only relatively to the moving joints.	46
A.1	Definition of the both eye’s ARKit blendshapes according to the blendShapes’ dictionary provided by the ARFaceAnchor documentation (apple.com, 2024).	I
A.2	Definition of mouth ARKit blendshapes according to the blendShapes’ dictionary provided by the ARFaceAnchor documentation (apple.com, 2024).	II
A.3	Definition of eyebrows, cheeks, jaw, nose and tongue ARKit blendshapes according to the blendShapes’ dictionary provided by the ARFaceAnchor documentation (apple.com, 2024).	III
A.4	Examples of prompt outputs for three different prompts, with focus on accurate context, format and reliability. The result from Prompt 1 displays good and consistent format and aligns with the context, but fails in choosing animations from the given list. The result of Prompt 2 shows similar characteristic, yet matches animations to specific words instead of the overall context. Finally, Prompt 3 results in good alignment with context yet is too unstructured and not real time compatible.	IV
A.5	The results of tree different LLM prompts focusing on investigating how well timestamps can be added to the animations. All prompts includes a list of 5 possible animations. Prompt 1 was given free rein of the timestamps, while Prompt 2 and 3 were limited to the range [0, 1] representing the start and end of the sentence. Prompt 2 and 3 also are told to say when to activate the possible animations, if at all. Prompt 3, in contrast to Prompt 2, are discouraged to include explanations. The outputs all show consistent formats for simple sentences, yet do not always match their context.	V
A.6	Prompt input to generate mood tags, blendshapes or motion data, depending on situation. This overall layout is used often in this project. One part changes depending on usage, and can include instructions on how to format a response, the possible tags or animations to choose from, number of tags or animations to choose and how often, the armature hierarchy, etc. See Table A.7 for the specific parts used to complete this prompt.	VI

A.7	The specific prompt parts used as the changeable part of the generic prompt given by Table A.6. Prompt 1 is used for generating mood tags, prompt 2 for generating two blendshapes, prompt 3 for generating motion data and prompt 4 for generating four blendshapes. All include instructions for format and lists of possible objects to generate from.	VII
A.8	Example of TextGrid file, showing the first 18 intervals.	VIII
A.9	Example of JSON file containing blendshapes and weights values for one example frame. Image by author.	VIII

1

Introduction

This project is the result of an internship at Fictive Reality, a company committed to the development of an educational simulator for training applications. The primary objective of this project is to evaluate how suitable various techniques are to generate animation commands for avatars, ensuring them to perform realistically under real-time constraints. In this Introduction, Fictive Reality's vision and objectives are explained, together with the goals of the study, and its significance.

1.1 Background

Fictive Reality (FR, 2024) is a startup company, partner of AI Sweden, that aims to distribute a learning-dialogue simulator to a wide clientele. The application closely follows the growing number of customers' requests to offer a tool through which users can train their soft skills, including conflict resolution, active listening, emotional intelligence and interpersonal skills. The machine learning (ML) technology the software relies upon offers a high level of customization and unlocks many possibilities, for instance facial expressions recognition to train non-verbal communication skills. The app is currently available on PC and Virtual Reality (VR) platforms, where a user can select an existing scenario or configure a personalized one, as seen in Fig. 1.1. Once inside, the conversation flows as in a simulator, where the avatar role-plays an angry customer or a difficult employee. The user trains to improve their ability to handle difficult situations, as seen in Fig. 1.2. The conversation is then evaluated and users receive feedback on their performance.

To make this experience valuable for the user, engagement is key (Choi et al., 2017). A user feels engaged during the training if involved and focused throughout the whole session, since this maximizes the user's attention, interest and learning. The sounds, the avatar's tone of voice and the animations must capture this feeling to maintain believability and maximize the user experience (Coulter, Saland, & Goldsmith, 2007). Facial expressions are a fundamental non-verbal way to communicate emotions (Fridlund, 2014). For this reason, real-time facial animations are able to deliver effective emotion information. The aim of this study and internship is to evaluate different approaches to generate real-time animations using machine learning technology, to keep users engaged with the conversation and without adding noticeable delay to the simulation. While there are many possible approaches to the subject, the main challenge consists of taking the real-time constraint into account.



Figure 1.1: Example of a user engaged in Fictive Reality’s training simulation. Image taken from Fictive Reality’s web page.

How to choose the approach to follow depends on Fictive Reality’s pipeline. An output from a Large-Language Model (LLM) determines the following avatar’s line. The animations instructions to be generated must then match this output, in terms of context of the avatar’s answer and implied emotion. The instructions are then used to animate the avatar’s upper body, head or a full body depending on the scope. The main input to achieve avatars’ animations is multilingual text content. Metadata is also available - such as emotional state, pronunciation, intonation, and speech rate, together with audio of synthesized speech to the text. However, what can be included highly depends on the pipeline of the application.

There are a number of limitations to comply with in the project, given by the existing structure and development of the app. For example, the animations need to match the context of the speech and be played without showing significant delay. To generate animations the company discourages the use of additional software or paid assets. Text and emotional tags need to be the only input, since these are the initial outputs from the Fictive Reality pipeline. A practical constraint is that the available instruments do not always support the visualization, as the required memory or processor exceeds the possibilities. An additional limitation is that the avatar model cannot be changed and that it assumes two main poses, either sitting or standing. Moreover, the quality of the animations is restricted to the expressivity of the chosen three dimensional model for the avatar. It is also important to note that micro expressions of the mouth are excluded during the visualization stage, as the

animations must not interfere with the existing lip-sync implementation. Finally, the project is limited to generate animations for the English language.

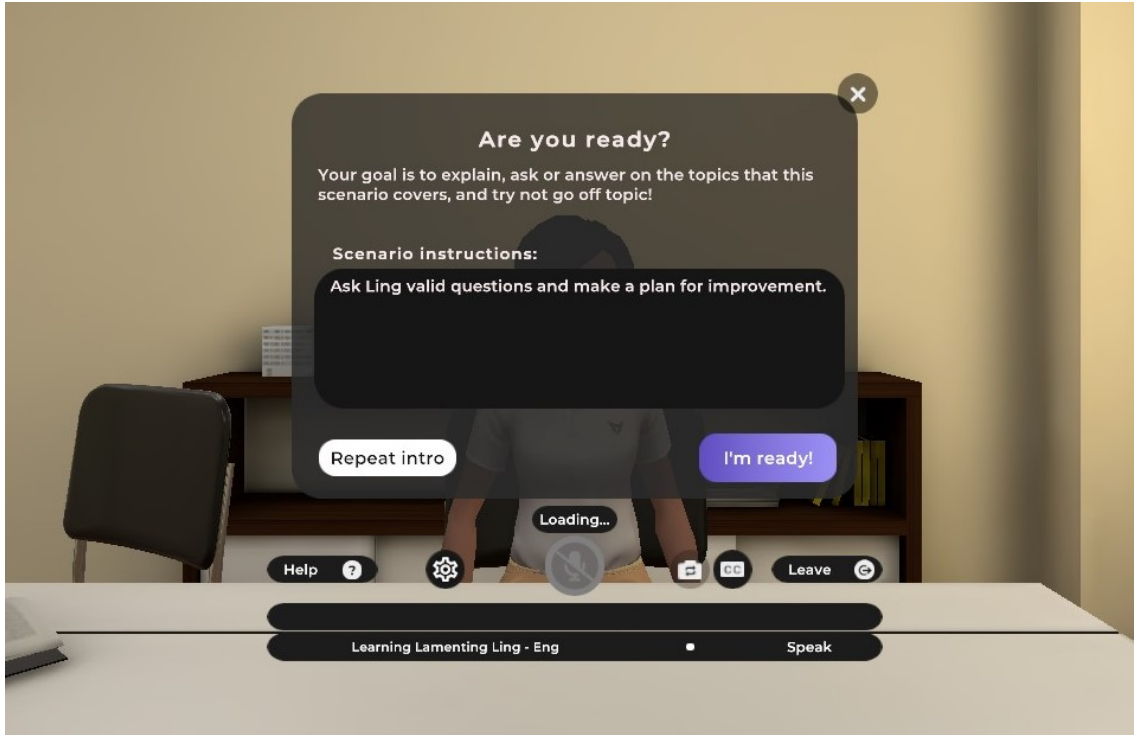


Figure 1.2: Example of a scenario from Fictive Reality. Screenshot by author taken from Fictive Reality’s app.

1.2 Objectives Goals

The goal of this study is to apply different machine learning approaches to generate real-time animations from textual input. It also investigates their advantages and disadvantages to draw conclusions on how the field of educational simulators can develop further.

Table 1.1 evaluates the hypotheses and relates to the methods applied to generating animation commands using machine learning techniques. Hypothesis 1 focuses on the quality evaluation of the approach, stating that the quality of animations is not dependent on the method used. The second hypothesis concerns the choice of approach and the real-time restriction. The following two hypotheses put forward Large-Language Models and their dependence on the prompt. Hypothesis 5 suggests that the key factor in the quality of animations is the method used to generate them, while the sixth hypothesis proposes that machine learning models can improve efficiency and quality of generating animations commands in app development.

Nr	Hypothesis
1	The quality of the animation generated is independent of the means used, whether it includes LLMs or NNs
2	Real-time animation commands can be generated using machine learning
3	The length of the prompt to an LLM affects the output's quality
4	The output of an LLM is different every time but the structure is always the same
5	The quality of animations is mainly affected by the means used to generate them
6	The integration of machine learning models improves the efficiency and quality of generating animation commands in app development

Table 1.1: Table of hypotheses made in the study.

1.3 Significance of the Study

Generating real-time animations using machine learning tools has a significant positive impact in many sectors, which is why research in this field is valuable. In education, for example, the learning experience benefits from realistic and immersive simulations, where students are able to visualize and engage with subjects in an innovative way. Looking towards personalized learning, students can also get appropriate instant feedback (Zhai et al., 2021). In healthcare, the integration of machine learning aids the identification of correct diagnoses (Chen et al., 2019), reduces errors and, possibly, mortality rates as well (Paranjape et al., 2019). Specifically, generating real-time animations can change the current traditional medical training method and aid in surgical training. It can aid in visualizing procedures, anatomy and apply changes in terms of treatments and pathologies on the go. It also brings improvements for the patients' awareness by aiding in the visualization of diseases and treatments. Moreover, there are interesting developments in the fashion sector and cosmetic surgery (Elder, Cappelli, Ring, & Saedi, 2024), for architecture and urban simulations, such as disasters, traffic planning and pollution (Koh et al., 2020). Real-time animations can also help to make information more accessible by providing real-time translations (Morris, 2020), additional information and visualization of terms or concepts that are otherwise not easily understood.

This project is motivated by research into the impact of using machine learning for animations on the realism of the experience. On this, there are some considerations the project aims to address. While positive results lead to complex ethical considerations, they also push for impressive innovation in training simulators. Generating real-time media is a current topic, and the ambition is that this study, in an objective manner, helps to identify its strengths and weaknesses. In terms of efficiency and costs, it is interesting to evaluate whether machine learning in this field is valuable, and to highlight areas it can benefit from. Machine learning adds dynamism to simulations, a feature typically found only in real-world training experiences. For applications not in real-time, this research still provides useful insight on rendering times, aiding in faster evaluation of animation qualities.

2

Theory

In this section, the key theoretical concepts behind this research are explained, identifying the current state of the field and the relevance of this study.

2.1 State of the Art for Animations Generation using Machine Learning

Using machine learning to generate content is both a current and disputed topic (Kirova, Ku, Laracy, & Marlowe, 2023), (Schlagwein & Willcocks, 2023). Research in this field relies on datasets, and can produce either data-driven synthesis or generative models. Data-driven synthesis replicates movements by using existing data, while generative models create new outputs from the original dataset. Datasets for animation generation typically contain labeled motion data, - data containing information about the movements it represents -, obtained from real-world sources. Depending on how the data has been collected, it can derive from motion capture systems or recorded videos, and is then usually represented in a specific file format. There are also datasets that contain raw video data where additional processing is necessary to be able to get the relevant parameters. A model can be trained on this data. The result can be improved through refinement with procedural animation techniques, for example, or by adding constraints to achieve physical realism. This approach gives results that mimic real-world motion well. However, it is memory intensive and relies on efficient pre-processing.

Generative models require deep learning techniques and can generate realistic animations that go beyond the training data, yet there is no certainty that the results adhere to physical constraints (Karras et al., 2020). These models are trained on datasets of motion sequences as well, but the techniques generally used are among recurrent neural networks, convolutional neural networks, variational autoencoders and generative adversarial networks. The latter is based on the network "learning" how to recognize structures and patterns within the training data. The new motion sequences are then generated by sampling from the distribution data created during the training. In generative adversarial networks, a pair of networks, the generator and discriminator, are trained in competition with each other (Creswell et al., 2018).

Human motion and gestures deeply depend on the context in which they occur. In the case of a human-avatar conversation, the animation typically relies on a com-

bination of text, audio and visual data to interpret the context. The text provides information on the purpose, the goals and the interlocutors of the conversation; the audio carries non-verbal tags such as emotion, loudness, intonation and speaking rate. Visual data can instead provide data on the user's gaze, on their participation and response to the conversation, on their mood changes and on their body language. In literature, studies differ in the choice of input and output for animations' generation.

Relying only on audio or text data is a difficult task: given the individual and non-deterministic nature of motion, audio or text alone are not enough to accurately predict body or face movements. One approach to address this is an audio-driven motion synthesis method relying on diffusion models, which train a probabilistic model to match movements even with ambiguous audio (Alexanderson, Nagy, Beskow, & Henter, 2023). Diffusion models, a type of deep generative model, consist of Markov chains in which Gaussian noise is added to the training data. The data is then recovered by reversing the noise process.

Using voice data and a single image, instead, can lead to the generation of expressive videos for a talking-head (Gururani, Mallya, Wang, Valle, & Liu, 2023). In this case mouth animations are predicted, and poses applied by adding missing elements such as head motion and teeth from a still image. Parts of the key information normally contained in audio data can also be extracted differently. Using multi-modal fusion models it is possible to detect audio and visual features to detect the sentiment in videos (Blanchard, Moreira, Bharati, & Scheirer, 2018). Similarly, from audio and video data in real-time, it is possible to do sentiment analysis on audiovisual data stream to classify the emotion (Yakaew, Dailey, & Racharak, 2021).

Considering face and body animations separately requires a method to merge the two in a later stage. With two separate models for face and body respectively, it is possible to apply the face model to any face model containing blendshape parameters and, as for the body, animations can be applied to any 3D model containing a human skeleton through animation retargeting (Niu, Xie, Wang, Cao, & Liu, 2024).

2.2 Fictive Reality Pipeline

To give insight in the specific problem tackled in this project, a relevant part of the Fictive Reality architecture is presented. This entails the Conversational Server that acts as the company's avatar during the conversation with the user. This is seen in Fig. 2.1 together with a selection of the external systems it converses with.

When a user speaks to the bot, the Conversational Server sends the speech audio to a speech-to-text program, which takes the audio and outputs the corresponding text version of it. This text, called "the user text", is then sent back to the Conversational Server, which in turn sends it to an LLM that generates a suitable text response for the bot. The LLM has been given a scenario-specific prompt ahead that ensures both to give its answer on the correct format and to be suitable for the scenario.

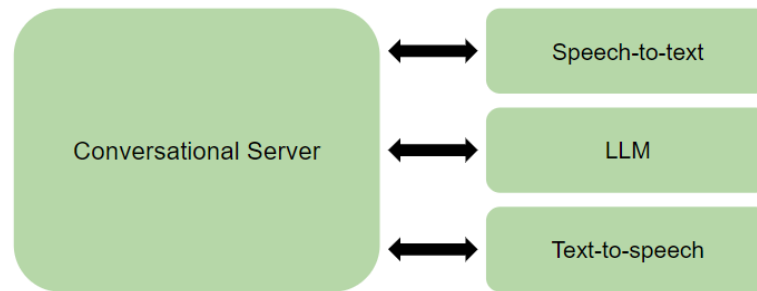


Figure 2.1: The components of the Fictive Reality architecture that carries out the conversation between bot and user. The speech-to-text takes the user audio and changes it to text. The text is then used to generate an appropriate response for the bot using an LLM. Finally, the text output from the LLM is sent to a text-to-speech program, that generates the bots spoken answer to the user. Image by author.

The textual answer for the bot is then sent to the Conversational Server.

As the final step, the Conversational Server sends the bot’s response to a text-to-speech program, which works like speech-to-text but in reverse. It generates the audio that matches the bot’s answer, and this audio is then used with lip sync when the bot speaks. Fictive Reality’s approach includes a selection step to determine the avatar’s mood from a set of possible states. Based on the mood tag extracted from the conversation, a state is chosen, triggering the corresponding animations for that mood.

It is in this step, when the bot is speaking, that the animations already need to be ready to play. Since the animations must align with the bot’s answer, either the audio of the bot, the text of the bot, or both are needed to generate the correct animations. Because of the real-time condition, it is preferred that the animations are ready as soon as possible and thus use the first possible data as input. Since the bot text is retrieved before the bot audio, it is this text that serves as input to any approach presented in this project.

2.3 Machine Learning

Machine learning is a field of study focused on creating systems capable of learning and improving on their own, enabling them to make accurate predictions without requiring explicit programming (Mehryar Mohri, 2018). This involves training algorithms on large amount of data to recognize patterns and structures. While several important applications are already mentioned in Section 1.3, it is worth mentioning that machine learning’s ability to adapt to structured data makes it a powerful tool for complex problems. The following sections present a selection of technical details about machine learning, with a focus on components that closely relate to this project.

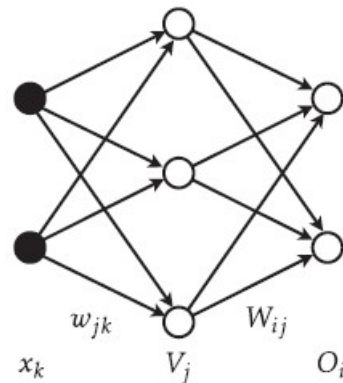


Figure 2.2: Feed-forward network with one hidden layer. The input terminals are coloured black. Picture taken from *Machine Learning with Neural Networks* by B. Mehlig under Cambridge Licence of Use.

2.3.1 Neural Networks

Neural networks are models inspired by biological brain structure, in which the functional unit, a node, is equivalent to a neuron in neurology. Neurons in these models form layers, in which nodes are interconnected to process input data and generate output. A simple neural network structure is visualized in Fig. 2.2.

The layers in a neural network are an important machine learning concept applied in this project. Specifically, a layer is a collection of neurons operating at the same depth within a neural network. Each layer modifies the data received through weights, biases and an activation function. There are three main types of layers: input, hidden and output layers. The input layer receives the initial data and passes it on to the next layer. Hidden layers perform computations on the data received, and are responsible for the interpretation of complex patterns in the data. See Fig. 2.3 for the structure of a neural network with two hidden layers. The output layer is the final layer producing the output of the network, for classification or prediction purposes, for example.

A dense layer is a type of hidden layer, also called fully connected layer, in which every node is connected to every other node in the next layer, hence the name. This layer is typically used for abstract representations of input data and feature extractions. A dropout layer, instead, is a layer used to prevent the overfitting problem. During training, a fraction of nodes in the network are excluded at each update. Overfitting is a common issue in machine learning, originated by models aligning too closely with the training data. This results in to poor performance on unseen data, such as validation and test datasets. To address this problem, there are various regularization techniques, one of which is the use of dropout layers.

There are two main classes in neural networks; feedforward neural networks and recurrent neural networks (RNNs). The latter is a class designed to handle sequence

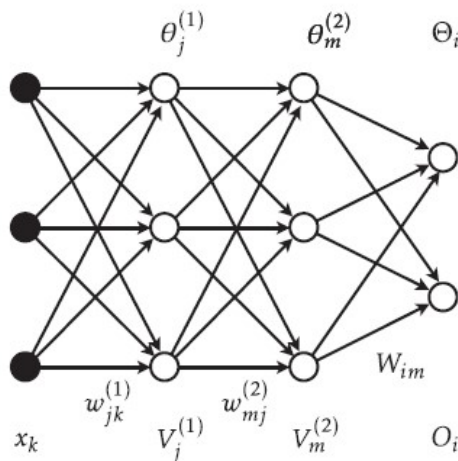


Figure 2.3: Multilayer perceptron with three input terminals, two hidden layers, and two output neurons. Picture taken from *Machine Learning with Neural Networks* by B. Mehlig under Cambridge Licence of Use.

generation. These networks have connections that allow the memory of previous inputs to be maintained, see Fig. 2.4. The most common applications for RNNs are language modeling and text generation, time-series prediction and speech recognition.

A type of RNN layer is the LSTM (Long Short-Term Memory) layer. This layer handles sequence data and is able to preserve information over long time periods. This makes it efficient for prediction tasks, among other applications.

To train a neural network to recognize patterns and relationships between data, it is important to split the available data into training, validation and test data. The training data is used for the actual training phase in which the model adjusts weights and biases to minimize the error. Fundamental training algorithms for neural networks are backpropagation and gradient descent. The validation data is used to evaluate the model's performance during the training phase. Through the validation data it is possible to detect overfitting issues, for example. The test data consists of unseen data reserved for evaluating the final model's performance.

2.3.1.1 Activation Functions

Activation functions are used to transform an input signal into an output signal. To get the output of a layer, each neuron's input is multiplied by its weight. Then, the result is passed through an activation function. The accuracy of a neural network's predictions depends on the choice of activation functions (Sharma, Sharma, & Athaiya, 2017). To recognize complex data patterns, it is necessary to use non-linear activation functions. An activation function must be differentiable to allow the implementation of techniques such as gradient descent and back propagation. The linear, the sigmoid and the hyperbolic tangent functions are common choices for activation functions. An activation function that is not continuously differentiable

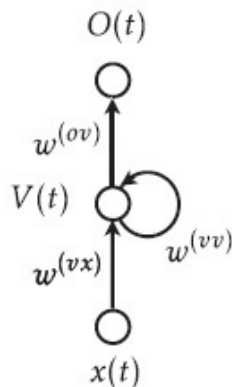


Figure 2.4: Example of recurrent network with one input terminal, one hidden neuron, and one output neuron. Picture taken from *Machine Learning with Neural Networks* by B. Mehlig under Cambridge Licence of Use.

but still often used in neural networks is the rectified linear unit (ReLU) function, with a discontinuity at $x = 0$ that helps in handling faulty negative values.

The linear activation function in Eq. (2.1) leads to constant gradient updates, which prevents improvement in error during backpropagation.

$$f(x) = ax \tag{2.1}$$

Ranging between 0 and 1, the outputs of the sigmoid function, see Eq. (2.2), are useful for probabilistic interpretations. However, it poses a challenge in terms of the vanishing gradient problem. It is a non-linear function, continuously differentiable, non-symmetric, see Fig. 2.5 B.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The ReLU layer is defined as in Eq. (2.3). This outputs zero for negative inputs and the input itself for positive values. When using ReLU, not all neurons are activated at the same time, making it more efficient. If the gradient is zero, the weights and biases are not updated during backpropagation, see Fig. 2.5 C.

$$f(x) = \max(0, x) \tag{2.3}$$

The hyperbolic tangent function in Eq. (2.4) outputs values between -1 and 1 . It is often preferred over sigmoid because it produces larger derivatives near 0, which allows the network to learn quicker (Szandała, 2021). It is continuous, differentiable and symmetric around the origin, see Fig. 2.5 D.

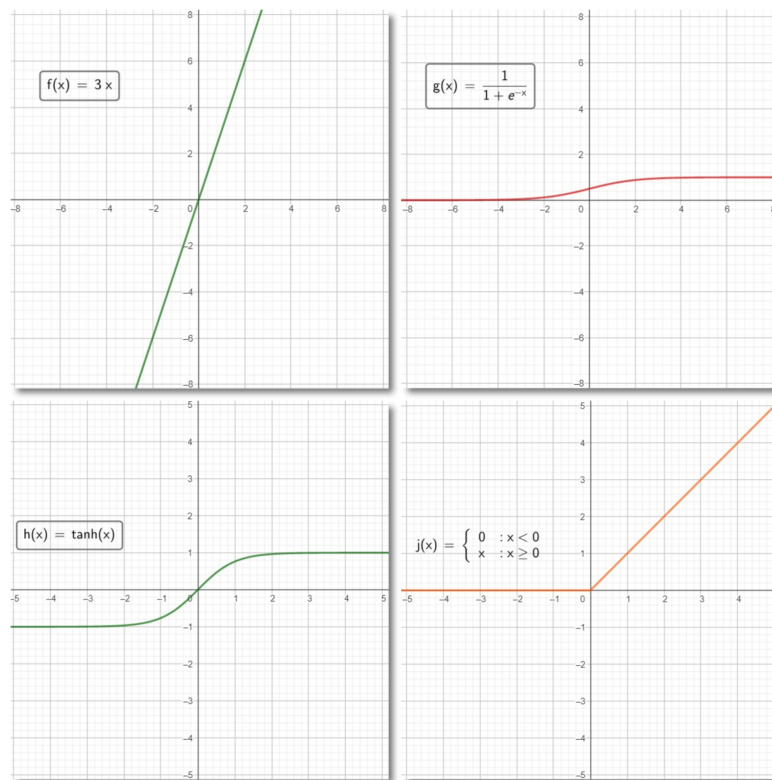


Figure 2.5: Four common activation functions. **A:** Linear, **B:** Sigmoid, **C:** TanH, **D:** ReLU. Image by author.

$$f(x) = \tanh(x) = 2\sigma(2x) - 1 \quad (2.4)$$

2.3.1.2 Gradient Descent

From a mathematical perspective, training a neural network involves solving an optimization problem. The goal is to find the minimum of an error surface, or loss function, where the derivative equals zero, leading to the best possible solution. However, optimization algorithms often converge to a point that is only the lowest in the surrounding area, rather than the overall best solution. This suboptimal point is called a *local minimum*, while the true best solution is the *global minimum* (Mehlig, 2021).

The challenges are then to determine if the converged solution is truly optimal and adjusting the optimization process to prevent the algorithm to settle in local minima. In general, there is no way to be sure that one has found the true optimum, or if one even exists (Mehlig, 2021).

Machine learning algorithms often use iterative methods to find a minimum. One common algorithm of that type is the gradient descent, which, given a start value, iteratively updates the parameters in the direction of the negative gradient of the loss function. Eq. (2.5) shows the formula for the gradient descent and how the

weights w in the network are iteratively updated. Here L is defined as the loss function, and η as the *learning rate*. The learning rate is a parameter that can largely affect the outcome of the gradient descent algorithm. Setting it to be small, as Eq. (2.5) shows, gives a shy update to the weights and hence requires many iterations until convergence. On the other hand, choosing a too large learning rate can make the algorithm completely skip a desirable minimum. Hence, it is beneficial to try different learning rates to find one that suits the current optimization problem or to adjust it during the training phase.

$$w^{new} = w^{old} - \eta \frac{dL}{dw} \quad (2.5)$$

The weights are updated until the gradient reaches zero, which is equivalent to when the a minimum is reached. This still does not guarantee that it is a global minimum, hence it is beneficial to make an educated estimation of the location of the initial start value. Otherwise, an approach is to iteratively try a range of random start values, and use the one that yields the least loss.

The generic gradient descent algorithm is applicable to many optimization tasks, yet it fails when the loss function is non-differentiable. Luckily, alternative algorithms to search for a global minimum do exist. For example, the particle swarm optimization algorithm involves a set of particles moving through the parameter space to find a global minimum, while a genetic algorithm selects, mutates, and applies crossover between individual sets of parameters. Whilst these are interesting algorithms they are generally not associated with neural networks, and therefore, the project does not explore them in further detail here.

2.3.2 Natural Language Processing

Natural Language Processing (NLP) was born in 1950 from the intersection of artificial intelligence and linguistics (Nadkarni, Ohno-Machado, & Chapman, 2011). NLP consists of a set of computational techniques with the purpose of analyzing texts and represent the human languages (Chowdhary & Chowdhary, 2020). The main areas of focus are translation, text classification, information retrieval and extraction, text generation and query answering (Chowdhary & Chowdhary, 2020).

To illustrate the functioning of NLP, one can start considering a simple text. A text is made up of strings, e.g. numbers and words such as "13\$" or "grass". The first step of processing the text is to apply text normalization, which aims to reduce textual variations. This phase considers many aspects, including tokenization, lower-casing, expansion of contractions, normalization of numerical expressions, stemming and lemmatization as well as removal of punctuation, special characters and stop words. Tokenization is the process of splitting the text into smaller parts called tokens, such as words, groups of words or other elements, which can vary much in difficulty depending on the language. An algorithm used in this context, which is also valuable in spelling correction and speech recognition, is one that computes

the edit distance of a text. This measures how close two strings are based on the number of insertions, deletions and substitutions necessary to change one string into another. The minimum edit distance algorithm determines the minimum number of editing operations to get from one word to another one. It works as a search task to find the shortest path between the two words.

To navigate through NLP it is useful to define two concepts: word types are the number of distinct words in a text; tokens are the total number N of units extracted from a text. In Table 2.1 there is an example of distinction between tokens and word types.

A smile rose on her face as Rose held a rose, red as her face	
Tokens	Word types
A	A
Smile	Smile
Rose	Rose
On	On
Her	Her
Face	Face
As	As
Rose	Held
Held	Red
A	
Rose	
Red	
As	
Her	
Face	

Table 2.1: An example of the difference between tokens and word types. In this case is a token defined as text between two whitespaces, image by author.

The total number of word types in a language is around the same number as the number of words in a language's dictionary. Herdan's Law in Eq. (2.6) relates the number of word types in a text with the number of instances N and with k and β , $0 < \beta < 1$ positive constants. The value of β is dependent on the text size. In most NLP applications words are not the unit for representations but rather tokenized strings. In Table 2.2 is an example of text before and after it is processed.

$$|V| = kN^\beta \quad (2.6)$$

The three most common tasks in text pre-processing are tokenization, normalization and segmentation of sentences. When tokenizing, there are two main algorithms: either top-down, where a set of rules allows the implementation of that tokenization,

Text normalization: Before VS After	
Before	It was a sunny and lovely day when Evan decided to go for a run. "Everything will change today. It will be different, I know it!" said Evan looking at his dog
After	it was a sunny and lovely day when evan decided to go for a run everything will change today it will be different i know it said evan looking at his dog

Table 2.2: A text before and after it is processed, image by author.

or bottom-up, where statistics is used to split words into subword tokens. Subwords are especially important to recognize words that may not be in the training corpus yet may occur during testing. For example, the word "underestimation" might not be present, but its subwords, such as "under", "estim-" and "-ation", are recognized. Table 2.3 gives an example of the distinction between word-based, subword-based and character-based tokenization.

"They were disinterested by that uncomfortable suit he had made"	
word-based	"They", "were", "disinterested", "by", "that", "uncomfortable", "suit", "he", "had", "made"
subword-based	"They", "were", "dis", "interest", "ed", "by", "that", "un", "comfort", "able", "suit", "he", "had", "made"
character-based	"T", "h", "e", "y", "w", "e", "r", "e", "d", "i", "s", "i", "n", "t", "e", "r", "e", "s", "t", "e", "d", "b", "y", "t", "h", "a", "t", "u", "n", "c", "o", "m", "f", "o", "r", "t", "a", "b", "l", "e", "s", "u", "i", "t", "h", "e", "h", "a", "d", "m", "a", "d", "e"

Table 2.3: A showcase of differently based tokens of a sentence, image by author.

Word-based tokenization uses delimiters such as whitespace and punctuation to determine the split location. The vocabulary is built upon tokens, however, since the number of occurrences is also taken into account, usually only the most frequent terms are saved. This means that only a certain number of tokens have a corresponding number to refer to it.

Character-based tokenization relies on the fact that there are less characters than words in a language, hence it builds a smaller vocabulary that is able to recognize most words and makes it easy to detect spelling mistakes. This comes with the disadvantage that tokens, in this case, carry less relevant information and that the processed text is much longer in comparison to other types of tokenization.

Subword-tokenization is an in-between solution that keeps the meaning of similar words together and does not produce a very long representation of the text by splitting the rare words and keeping the most common ones together. Bert and Bye-Pair Encoding from GPT-2 were set to rely on this type of tokenization.

After a raw text has been processed and a vocabulary created, a segmenter takes a

raw test sentence and splits it into tokens from the vocabulary, for instance using the Unigram Language Modeling algorithm (Kudo, 2018). In this algorithm, after the text has been normalized and tokenized, special tokens are introduced, such as $\langle START \rangle$ and $\langle END \rangle$ to mark the limits of sentences. The vocabulary is built from the unique word-based tokens that occur in the text. The term "unigram" distinguishes this model from other n-gram models, where "n" refers to the number of tokens considered together.

After text normalization, it is possible to proceed with further processing. The text can be converted into numerical representations to use in machine learning algorithms, e.g. feature extraction: text data is represented as numerical vectors, in this case. Examples of techniques for this approach are Bag-of-Words, Term Frequency-Inverse Document Frequency and word embeddings. The normalized text can also go through a vectorization process, where it can be converted into a matrix format in which a row represents a sentence and a column represents words (feature matrix).

After these steps are completed, text data can be used for supervised learning, unsupervised learning or sequence modeling depending on the goal of the project. Following an evaluation of accuracy, precision, and score, the process can move on to an iterative cycle of improvement to tune the model and improve performance.

2.3.2.1 Chatbots

An application of machine learning is chatbots, which are computer programs designed to simulate a conversation with human users (Adamopoulou & Moussiades, 2020). They rely on NLP for sentiment analysis, and can be both textual and audio based. Using input data they generate responses mimicking human language and a general dialogue complexity. Currently they exist in a wide variety of applications, e.g. as assistants, to reduce costs in support services. One of the first chatbots was ELIZA in 1964 (Weizenbaum, 1966). ELIZA mimicked a Rogerian psychotherapist and therefore did not need to know anything about the patient, it simply rephrased the patient's statements. A possible interaction is, for example, the "typical conversation" as described by Joseph Weizenbaum, see Table 2.4.

Users believed that ELIZA was able to understand them, even after receiving an explanation of how the program worked (Agassi & Wiezenbaum, 1976). Following chatbots, such as Jabberwacky in 1988, relied on pre-recorded sentences and corresponding responses. Modern chatbots can answer questions and hold memory of the conversation. With newer virtual assistants, talking heads were also introduced, such as SitePal in 2003. These were mostly used in customer service and for educational purposes. Recently, and with the real-time restriction, NVIDIA's Omniverse Avatars and Replika have been producing conversational avatars for customer service and companionship. The creation of interactive avatars such as these has been allowed by advances in graphics, because of the processing speed and rendering, as well as in machine learning for fast inference.

Speaker	Text
User:	I am unhappy.
Bot:	Do you think coming here will help you not be unhappy.
User:	I need some help, that much seems certain.
Bot:	What would it mean to you if you got some help?
User:	Perhaps I could learn to get along with my mother
Bot:	Tell me more about your family.
User:	My mother takes care of me,
Bot:	Who else in your family takes care of you?
User:	My father.
Bot:	Your father?
User:	You are like my father in some ways.

Table 2.4: A conversation example between ELIZA and a user.

2.3.2.2 Text representation

To transform text into useful formats for machine learning models, it is common to use either embedding or encoding techniques. Embedding techniques represent input, such as words or images, in a dense form, while preserving or capturing their semantic relationships. For example, Word2Vec is an embedding technique that captures the relationships between related words in their vector representations, such as the close relationship between "queen" and "king". Thanks to the vector representation, one can assume that the distance between "queen" and "king" is similar to the distance between "woman" and "man". Thus, both the closeness of words within a pair and the relationship between different pairs can provide useful information.

Encoding techniques, on the other hand, represent text without capturing semantics, meaning that they carry less information but are more compact. A simple choice is one-hot encoding, where each unique token in a text is given its own position in a vector. When representing a sentence, each token is given a 1 in its corresponding position in the one-hot encoded vector, while non-used positions are kept as a 0. An example is given in Table 2.5, where the one-hot encoded vector is based on a short source text. While one can also count the occurrences of a token, note that in the second example sentence, the word "a" appears twice yet is treated the same as the other tokens. One-hot encoding does not take the order of a set of tokens into account either; the sentence "a dog is a friend" is different from "a friend is a dog", for example. While the overlook of both token occurrence and order in one-hot encoding is a limitation, a good machine learning model combined with an extensive dataset can pick up the usual intent of such one-hot encoded vector. The simplicity of the one-hot encoded vector is one of the main reasons it is often the first implementation in text representation tasks.

Source text:	"My friend has a dog. She is called Pumpkin and likes to sleep."
Sentence:	"My dog likes a pumpkin."
one-hot:	[1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0]
Sentence:	"A dog is a friend."
one-hot:	[0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0]

Table 2.5: Example of one-hot encoding, where tokens are defined as distinct words. First, there is a source text and second, there are two sentences using words from the source text. There are 13 unique tokens, thus each one-hot encoded vector has 13 positions.

2.3.2.3 Large-Language Models

Foundational to NLP are Language Models (LMs). Large-Language Models (LLMs) are an extension of LMs which rely on larger datasets, and present differences on complexity and architecture. LMs assign probabilities to words, or sequences of words, to predict the following sequence given the previous words. The terminology of n -grams is used in this context to refer to a sequence of n number of words. Therefore 2-grams refer to two-word sequences of words, 3-grams to three-word sequences, and so on. The probability of a word w given the history h , $P(w|h)$, can be determined either by relative frequency counts or by decomposing the probability using the chain rule of probability. The first method goes as follows: considering a large corpus, the number of times in which h occurs is counted. The same is done for the number of times h is followed by w . This method is not always efficient since the combinations in a language are many. The second method, involving decomposing the probability using the chain rule of probability, captures the context up to $n - 1$ words. It allows for smoothing techniques, but it has data sparsity issues and may be computationally complex for higher order n -grams.

2.4 Software Tools

To generate animations in real-time with machine learning, a number of software tools are used to accommodate for each step in the process. While there are alternatives to every tool, the following presents the general concepts of the ones used in this project.

Blender (Blender webpage, 2024) is an open source 3D creation software that encompasses every tool required for model manipulation such as modeling, sculpting, texturing, rigging, rendering and animating (Hess, 2007). In Blender, an object is made up of vertices, edges and faces. Through a toolbox it is possible to manipulate a model by applying commands such as extrusion, beveling, subdivision and sculpting. These tools allow the modeler to achieve complex shapes and personalize models up to the finest details. While modeling is about geometric manipulation, sculpting is a function that turns models into clay-like objects. Therefore it provides tools such as brushes as well as functions, e.g. carving and smoothing. Once

the model is created, texturing applies patterns to the surface of the 3D object. It is possible to apply *material* to objects in order to add distinctive traits through colour, roughness and metallic surfaces.

When the aim of a project is to animate a model, rigging is a key phase of the process. In this phase a digital skeleton, or rig, is added to the model so that the possible movements are defined, and constraints are applied to prevent the model from showing unlikely poses or features during movements. This digital skeleton is controlled by an armature, which serves as a framework for the model's motion. The armature consists of interconnected bones, which act as the building blocks of a rig, defining how the model can move. Each bone in the armature has properties for position, rotation and translation to determine transformations in a 3D space. The bones can be grouped to create a hierarchy where a "parent" bone determines how a "child" bone moves. For instance, the middle finger of the right hand is affected by the movements of the right hand. The right hand is in turn affected by the right arm, and so on. A section of a hierarchy is shown in Fig. 2.6.



Figure 2.6: Example of hierarchy of bones in a 3D model imported into Blender software. Image by author.

The animation tools in Blender allow users to define keyframes, which are specific points in time where the position or state of an object is set. These keyframes are used to create poses and movements for objects. An object can be moved, scaled or rotated such that transitions between keyframes are created, and the animation appears fluid when rendered. The focus in this step is in giving the desired flow to the movement while also adding the desired expressivity to the scene. Rendering is the process of capturing the scene into a 2D image by using a rendering engine, for which Cycles and Eevee are the most common options (Hess, 2013). Rendering an animation and using Blender's animating tools are two distinct processes, since rendering produces a series of frames that are later played in sequence as a video. This step comes after having created the animation with animation tools. While this software is typically used for character creation or to create models that are

later ported to a graphics engine, such as Unity or Unreal Engine, this project uses Blender for two main purposes. It provides an early look on animation output and helps analyze the hierarchy of the 3D models. It is also well suited in visualizing the animations before application in Unity.

Unity (Unity webpage, 2024) is a game engine used for the creation of interactive 3D content. It supports multiple platforms and helps developers by providing tool-sets, on-the-fly play testing, and a catalog of 2D and 3D assets to speed up the creators' work (Kim et al., 2014). Fictive Reality takes advantage of Unity's ease of adaptability to different platforms in their app. By importing 3D models from Blender, Unity assets or other 3D software, the developer can write scripts to animate, simulate, design or optimize their project. Unity's ease of integration has been indispensable for developing an app that integrates many different functions. Two concepts that are referred to in the project are coroutines and skinned mesh renderer components. A coroutine is a function that allows the execution of code over time. Coroutines are used for tasks that occur gradually such as waiting times, transitions or, as in this project's case, animations. A skinned mesh renderer component allows a mesh to deform according to an underlying structure such as a skeleton or bones. A mesh is an object's geometry that can be manipulated in 3D space through its vertices, edges and faces. The skinned mesh renderer is used to apply a skeletal structure to a 3D model in Unity, for example to characters and objects that require skeletal animation ???. It renders the deformable mesh of the model by moving its vertices according to the movement of the bones in the rig. The renderer ensures that animations are based on the bone hierarchy: as bones move, the mesh deforms. In particular, each vertex in the mesh has a weight that determines how much it is affected by each bone.

Unity's 3D space is made up of objects. When an object can be found in multiple states, it can be convenient to use state machines to represent these. From Unity's documentation (Unity Documentation, 2024), main "states" are identified. A state machine is a design pattern that ensures different behaviours of an object depending on the situation at hand. For example, the object "bird" can be in two states, such as "flying", if in the air, and "walking", if on land. Without using a state machine it becomes necessary to continuously check the object's surroundings and conditions. State machines make this unnecessary, since once an object enters a particular state only the behaviour that can occur there is handled, and only the conditions that can make the object exit that state are checked. Only one state can be active at a time, hence execution is improved. The states are represented in a graph diagram where the nodes represent the states and the arcs the transitions, see Fig. 2.7. For animations, state machines are particularly convenient thanks to their design and update phases. In Unity, a state controller code is implemented to manage the triggers, events and transitions between states.

Fictive Reality uses a state machine to implement the transition between facial animations. Emotional tags work as triggers in this context. Each state starts and ends in the neutral state to transition smoothly between different facial expressions.

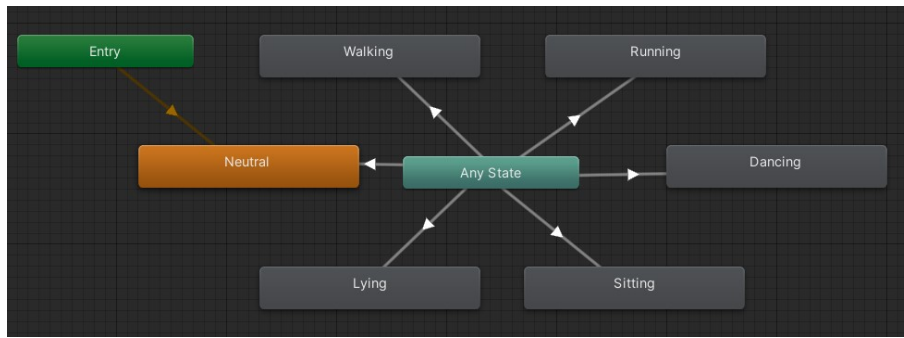


Figure 2.7: An example of a State Machine in Unity. Image by author.

Python is a computational programming language that is used in many areas of development (Van Rossum, 1995) and generally suitable for machine learning, thanks to the extended quantity of ML libraries (TripleTen, 2023, November 6). Libraries in python are collections of pre-written code, accessible to the user, which simplifies the process of development while also providing a level of standardization among different Python projects (Tiwari, 2023, November 6). One such library is TensorFlow that is specifically developed by Google for ML purposes (Shukla & Fricklas, 2018).

2.5 Animations

Animation is the process of creating the illusion of movement through a sequence of images. To animate means "to give life to", which happens between two frames (Wells, 2013). The history of animation follows the progress of technology. Already in 70 B.C. Lucretius in "De Rerum Natura" describes a mechanism to create moving images by projecting hand-drawn moving images on a flat surface. Flipbooks, cel animation and stop-motion animation are other types of traditional animation techniques, and are more or less intensive processes providing different experiences. An important time in the history of animation was 1825 when "the Persistence of Vision" theory was introduced by Mark Roget. This theory explained how the human eye is able to perceive movement from images, by talking about the after-image that is carried onto the following image giving an illusion of continuity.

The limitations of traditional animation methods are mostly related to a time factor. Computer generated animations gave unprecedented flexibility to the process, and aided creativity by offering new possibilities of character and environment manipulation. Computer graphics enriched animations with a high degree of realism which, can be argued, is not always a necessary ingredient for conveying emotions or expression of concepts. Today, animation plays a significant role in different fields. As a captivating and adaptable technique, it improves simulators for educational purposes, increasing both engagement and understandability.

There are a number of challenges in animation, especially related to realism: timing the animations, achieving a realistic and fluid motion and, in the case of real-time,

fast and flexible animations. While neural networks bring some ethical concerns, they also help solve some of these problems. Neural networks can be trained on a vast amount of data to identify patterns, and help towards procedural animation and fast generation of fluid animations. This can be especially powerful when investigating ways to reproduce the flow and timing of facial animations and body movements. In the games industry, Ubisoft is using AI-driven animation systems (Ubisoft, 2024) to generate spontaneous dialogues. The use of these technologies in the animation field aims to increase efficiency and create immersive environments. Neural networks models are also used to produce convincing human-like movement by training on motion capture data.

2.5.1 Evaluating the Quality of Animation

When evaluating the quality of an animation there are many methods to apply and aspects to take into account. For example, the balance, exaggeration and asymmetry in movements. All of this adds to the realism and interest of an animation. The posture and body attitude of an avatar also plays a part, as well as the complexity level of the animation, where both too much or too little complexity can make a less good animation (Buisine et al., 2014).

The factors when evaluating are, in large, visual only, which makes it difficult to classify the quality of an animation. However, it is beneficial to understand what typically contributes to a good animated result. This knowledge makes it easier to analyze an output, and from there, make own valuations of the animation.

To name a few key elements when considering quality of an animation, the first one is elements of asymmetry in avatars, which give the impression of realism. This can be in visual attributes or in movements, such as a dominant arm that is used more frequently than the other. It closely relates to an avatar displaying an head angle. Other key elements are the body attitude as well as the posture, which both convey and support the wanted emotion. Finally, to achieve clearer animation it is necessary to avoid any complexity (Buisine et al., 2014).

2.5.2 Animation Tools and Formats

To understand this project's concepts, it is necessary to define some technical terms that are recurrent in what follows. This concerns the face and body animations as well as the corresponding text data.

To create a linear model of facial expressions blendshapes are used. Each blendshape corresponds to a muscle or movement in the face, such as raising the corner of an eyebrow or opening the jaw. The animations are restricted by blendshape parameters, which create a linear combination of realistic facial modifications and expressions (Lewis et al., 2014). Each blendshape has an assigned weight ranging between 0 and 1, where a 0 corresponds to deactivation and 1 to a fully activated blendshape.

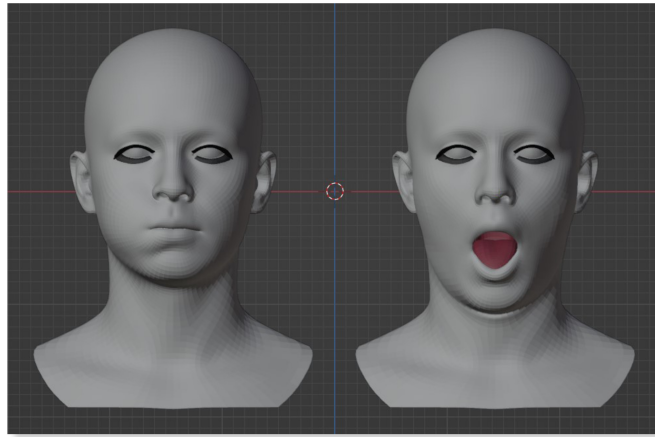


Figure 2.8: Blendshapes in action. On the left, the blendshape representing the puffing of the right cheek, in the project called CheekPuffRight. On the right, the blendshape representing opening the jaw, in the project called JawOpen. Image by author.

Apple’s augmented platform, ARKit, defines a standard set of blendshapes. ARKit includes support for facial tracking and uses blendshapes for face detection and facial landmark detection. Blendshapes’ coefficients are generated to represent the degree to which a feature is activated. This standard is followed in this project as well.

Since many facial expressions are complex and dependent on subtle changes in facial expression (Alkawaz, Mohamad, Basori, & Saba, 2015), the Facial Action Coding System, FACS, is commonly used to facilitate the comprehension (Ekman & Friesen, 1978). It consists of an anatomically-based system for describing all visually discernible facial movements. FACS therefore provides a standardized way to analyze facial expression by breaking them down into individual components of specific muscle movements.

In this project, blendshape weights are used as the output of a neural network model. In Tables A.1-A.3 the specific blendshapes considered are shown. This project uses the JavaScript Object Notation (JSON) file format for the blendshape data, containing the weights to be applied to each blendshape during a small time interval.

TextGrid is a format of file used in speech analysis and contains text data, see Table A.8. Text data is split into segments, where each segment corresponds to an interval in the audio data. At the start of a TextGrid file, there is a file header specifying the file type, and provides the overall time span, indicating the start and end times of the audio and text data. This means that each word in the text data, and also each pause made during the speech, are assigned a start and end time, which simplifies the text analysis and the correlation to other relevant data.

One such correlating data is BioVision Hierarchical Data (BVH), which is a format used for motion capture applications. It represents the structure of a skeleton and

its movements. There are two main components in a BVH file: the hierarchy and the motion data. In the hierarchy section, the position and rotation channels of the joints are specified, along with parent-child dependencies between joints to structure the skeleton. Each joint has three degrees of freedom, except the root joint that has six in order to define the global position and orientation of the whole skeleton. A root joint can have one or more child joints, that in turn can have their own children. Each child joint is relatively positioned to its parent joint such that its movements are influenced by the movements of the parent joint. As an example, fingers are child joints of the hand and thus their position in 3D space is affected by the motion of the hand. The hierarchy may change between different BVH files, however within the same dataset and using the same 3D model as reference, the hierarchy is the same.

The motion data section contains the motion capture data for each frame. The first information in the section is the total number of frames and the duration of each frame in seconds. Each value in the motion data corresponds either to a position value or to a rotation in X, Y, Z . The numbers follow the hierarchy order, meaning the first six numbers correspond to the position and rotation values of the root bone, and so on for all joints, until the first frame is completed. It then repeats for all frames in the motion data. Traditionally, the rotation angles in a BVH file follow the order Z, X, Y instead of X, Y, Z .

While these three file formats, JSON, TextGrid and BVH, are built in different ways, they all have some repeating indication of timing, either by specified frame time or recurrent time stamp. This disentangles the process of picking data samples from all files that corresponds to each other.

3

Methods

In this section, the methods used in the project and the reasoning behind are outlined. To choose the methods, at first, research is conducted, and a plan of action is developed. Fictive Reality has implemented a state machine for animations and is looking for ways to improve the quality and fidelity of their system. With the real-time constraint, followed by using only text as input, two approaches are selected: one due to its ease of integration and efficiency, and one for the flexibility and openness to future improvements.

3.1 Approaches

After researching the state of the art, two main approaches are identified. The first one relies on existing models, specifically large-language models. This favors easy implementation and customization for specific use cases, achieved by modifying the prompt. However, it depends on third party's products, and the output format may be inconsistent. In contrast, building a neural network model from scratch allows for consistent output formats, yet it requires adequate training of the model. By exploring both options, the ambition is to be able to tell which implementation is preferable for real-time animation generation.

3.1.1 Large-Language Models

Given that Fictive Reality is already relying on an LLM for generating avatar responses, the easiest option for fast implementation is to test whether or not it can be valid for animation generation as well. Provided with the prompt used for the avatar response, several efforts are made to evaluate the reliability of the LLM. The prompt is modified to produce instructions for the avatar that can be translated in mood tags, blendshapes, motion tags or motion data. The initial tasks are to obtain an output in a specific format, to determine if the output format is consistent for the same prompt, and how reliable the output is.

To assess how effectively the LLM generates relevant animation instructions, it is prompted in different ways to incorporate avatar motions into its text responses. This is expected to generate movements such as "nod", "give thumbs up", "lift corners of the mouth", and similar. However, since these instructions are not directly readable in Unity, this investigation only serves the purpose of concluding how well

the LLM performs in outputting relevant animations. It also helps when observing the consistency of different formats in the output.

Next, it is necessary to evaluate which animation practice generates a better quality output. Both mood tags and motion tags are outputs that must rely on Unity’s state machine. For this reason, the quality improvement brought by using these is limited to the choice of available states. Blendshapes and motion data instead offer an infinite number of combinations for the choice of the most suitable animation.

To decide on the number of output blendshapes, a few factors carry more importance. This includes the 3D avatar’s model, the visibility of each blendshape, the balance between achieving realistic expressions and getting non-conflicting results from the LLM. For example, activating a muscle responsible for pulling the corners of the mouth upward, through the blendshapes "mouthSmileLeft" and "mouthSmileRight", is in conflict with the blendshapes "mouthFrownRight" and "mouthFrownLeft", which are responsible for pulling the corners of the mouth downward, see Table A.2. Not more than four blendshapes are tested at a time for two main reasons. As the prompt research shows, LLMs do not perform well when given long and complex prompts. Additionally, considerations of the real-time constraint and the human eye’s ability to discern changes within a single frame are made. Increased complexity goes unnoticed, yet it significantly affects computational time.

3.1.2 Neural Networks

Based on the state of the art, it is decided to use a neural network model for the face animations and a separate one for body animations. Both models use the same textual input from TextGrid files during training; however, the face model also uses blendshape parameters from JSON files, whereas the body model uses motion data from BVH files. In the dataset used here, there are 51 blendshapes and 228 body joints, with the degrees of freedom specified for each joint. The recurrent time indications in each of the three files is used to match corresponding data between them.

The first phase of this implementation includes, in addition to initializing paths and setting up the environment, extracting and pre-processing the data to ensure correct formatting and management. Since TextGrid files do not indicate the start or end of sentences, a method is chosen where each file is split into groups of four seconds worth of data. The choice of four seconds is based on the lower end of the average duration required to speak a sentence. From this point onward, each such group is considered a data point. The BVH files have twice as high frame rate compared to the JSON files, thus only every other frame of the body motion data is considered. Having the same amount of frames per seconds (FPS) simplifies later steps in the implementation, as the data can be treated in a similar manner. This results in both the face and body models operating at 60 FPS, which over a four-second interval, corresponds to 240 frames per input. This is a higher number than required by Fictive Reality, but is desirable if viable.

The text data is then tokenized into distinct words and pre-processed for easier management. This entails converting the text to lower case, removing special characters and splitting contracted words. It also includes removing words that are assumed not to have any affect on animations, such as the words "the", "a", "an" and "of". This is possible since the lip sync is already treated by an existing model, and animations for mouth and jaw are therefore turned off in this project.

Each token text is subsequently mapped to a one-hot encoded vector based on the predefined, alphabetical dictionary of the 3927 unique tokens. One-hot encoding is used in this project for its simplicity, interpretability and direct compatibility with the neural network models. By representing text as binary vectors, assumptions about token relationships are avoided, which allows for focus on token-level patterns. While alternative methods such as word embeddings capture semantic relationships that can be relevant in body language, they also add complexity, making one-hot encoding a more suitable baseline for analysis.

To preserve the word order of text, a sequence of one-hot encoded vectors is used as input. This involves one-hot encoding individual tokens and submitting the vectors in order as a single input event. However, due to the significant computational drawbacks and added complexity, it is decided not to use this process at this stage of the process. Given the focus on investigating the usability of different approaches for animation generation, rather than fine-tuning them to their best performance, the simplicity of applying one-hot encoding - overall and only once per input - is chosen for this project. It is, however, acknowledged that preserving the word order affects the outcome, and further investigation into methods for doing so is encouraged in future work.

Motion capture data reflects physical joint constraints, yet some frame entries in the BVH files include values outside the standard range of $[0, 360]$ degrees. While this in itself does not affect animations, the fact that the motion data contains sudden jumps in its values adds confusion and discontinuity to the body model. The values are therefore manipulated to fit in the correct range. A simple method is to directly apply modulo 360. However, this also creates sudden jumps in the affected entries, degrading the model's ability to capture patterns effectively. Observing the distribution of the joint degrees in the data, a majority of values are contained in the range $[-360, 360]$, with a rather even distribution around 0 degrees. Thus, the chosen approach is to allow only data points with all of their values within this range. In other words, data points for the text, blendshapes, and motion data are ignored when the latter includes out-of-range values. As earlier, even though it is not necessary to apply this implementation to the face model too, using a similar approach facilitates a clearer understanding of the process and later operations.

The data points are then split into training, validation and test sets, with a ratio of 80-10-10 percent respectively. With both models architectures defined, see Table 3.1 and 3.2, these are trained on the training set using validation for early stopping. After the training is done, the models are evaluated using the test set. The best runs

for each model, based on these accuracies, are tracked, and corresponding models are saved in case of future interest. Using the trained models, predictions can now be used for any chosen input sentence that has gone through the same pre-process as the text data. The predictions are then written to JSON and BVH files that can be sent for visualization in Blender and Unity, respectively.

Since the data pre-processing and model training are done in advance, real-time performance depends only on the time required to process the input text, predict the corresponding output frames from both models, and write results to the target files. Therefore, these steps are the focus of time optimization. For instance, when writing to a BVH file, the first few hundred rows contain initialization data, which are the same for each file in this dataset. Instead of writing these rows in each run, they are pre-written in the target file and the output frames are appended.

As for the neural network models, one is set to predict facial blendshapes and one to predict body joint movements. The models use Keras library and TensorFlow, from which the accuracy metric is used to evaluate the models. In this specific accuracy metric, the predicted output values are compared to the true values, and each match is counted. The total number of matches is then divided by the total number of target values and is presented as the final accuracy.

Blendshapes take values in the range $[0, 1]$, thus the first choice is to apply sigmoid activation functions in the face model. It can however be counterproductive if the distribution of the values is not skewed towards the limits of the range, as sigmoid functions typically are used for binary classification matters. Since blendshapes data does not follow this typical distribution, as shown later in Fig. 4.5, alternative options are also investigated. An idea is to use a combination of ReLU and tanH activation functions. The tanH function maps the output to the range $[-1, 1]$, while still allowing gradients to update, whereas the ReLU function sets negative values to 0 and leaves positive values unchanged, given its linearity for positive inputs.

A combination of these considerations is applied and presented in an example model shown in Table 3.1. The neural network model for the face has an input layer which processes the one-hot encoded input, a dense layer with ReLU activation function, a dropout layer to prevent overfitting, and a dense layer with tanH activation function to learn non-linear relationships. The input layer prepares the model to accept a single time step of the input data, specifying the input shape of the model as well. The first dense layer adds a fully connected layer to the model. This expands the feature space allowing the model to elaborate complex representations. The ReLU activation function is chosen because of the blendshape values' range of $[0, 1]$, its non-linearity, and its ability to avoid vanishing gradient issues. Similarly, the tanH function introduces non-linearity. The dropout layer sets a fraction of input units, 20%, to 0 during the training phase to help with overfitting. The data is then reshaped to prepare it for the following layer, an LSTM layer. An LSTM consists of recurrent layers, able to capture temporal dependencies in sequence data. These allow it to preserve the weights that are forward- and back-propagated through

layers, (Kwon, Park, & Song, 2020). This layer is included in the model because it can retain information from sequence-based input, as the facial motion data is. The Adam optimizer is used with a learning rate of 0.01, accuracy as the metric, and mean squared error as the loss function. The Adam optimizer is a stochastic gradient descent algorithm.

Layer nbr.	Type	Info
1	Input Layer	-
2	Dense Layer	<i>ReLU</i> activation function
-	Dropout	0.2 dropout rate
3	Dense Layer	<i>TanH</i> activation function
-	Reshape	-
-	Dropout	0.3 dropout rate
4	LSTM Layer	-

Table 3.1: Outline of the architecture used for the face model, which includes the Adam optimizer, MSE loss function, and accuracy as its evaluation metric.

Similarly, a proposed model for the body is presented in Table 3.2, where the same considerations are taken into account, but adapted to the different nature of the data. While motion data values are allowed in the range $[-360, 360]$, the input is normalized by dividing all values by 360 for easier manipulation. Since the data distribution is now in the range $[-1, 1]$ the tanH function is chosen. In the presented model there is an input layer, a normalization layer and a dense layer. The data is reshaped and two LSTM layers are applied as a bidirectional LSTM for sequential data. The bidirectional LSTM layer processes input sequence both forward and backward to capture temporal dependencies with more context at each time step. A dropout layer is then applied, followed by a dense layer with tanH activation function for the output. Finally a de-normalization layer brings back the output to the initial range of values. The Adam optimizer is applied in this model as well, with 0.01 learning rate, accuracy as the metric and mean squared error as the loss function. Dense layers are part of the model to help the network learn non-linear relationships between features. Two aspects that are investigated in depth are the impact of different activation functions on the outcome of the values, and how the size and number of layers influence temporary memory usage. For instance, incorporating dropout between layers reduces memory usage but is also a common regularization technique to avoid overfitting.

Two scripts are developed in Unity to aid in the visualization of the two outputs. The script to read JSON files goes through blendshape animation data and applies it to a skinned mesh renderer while playing the animation, as explained in Section 2.4. The JSON file is parsed, and blendshape data for each frame is stored in a list. A coroutine, explained in Section 2.4, is initiated to animate the blendshapes in a sequence, from their current weight to the target weight over the specified duration using linear interpolation. As for the script reading motion data, the script is implemented to animate the skeleton. The BVH file is parsed and bones in the file are mapped to the bones of the Unity model. The motion data is then

Layer nbr.	Type	Info
1	Input Layer	-
-	Normalization	from $[-360, 360]$ to $[-1, 1]$
2	Dense Layer	$TanH$ activation function
-	Reshape	-
3	Bidirectional LSTM Layer	-
-	Dropout	0.3 dropout rate
4	Dense Layer	$TanH$ activation function
-	Reverting normalization	from $[-1, 1]$ to $[-360, 360]$

Table 3.2: The overall architecture for the body model, using the Adam optimizer, the MSE loss function and accuracy as its evaluation metric.

assigned to the bones by recursively searching for child bones by name.

Given that the models are not incorporated into Fictive Reality’s server, there is no final interaction and visualization of the result available in real-time. However, the considerations mentioned above regarding the time optimization cover the real-time aspect for the part of the implementation that is included in the scope of this project.

3.1.3 Video generation

Since one of the company’s goals is to spread the use of their app by releasing it on mobile devices, a completely different method is considered. While approaches presented earlier involve 3D models, generating video animations involves still images. This approach relies on text input from the conversation and outputs a talking-head video. While this solves many challenges, such as merging body and face animations, preventing body parts from clipping through objects, it introduces new ones. In a normal scenario from Fictive Reality, the avatar can be interrupted at any time. Videos are inherently less flexible to changes and their generation is not compatible with real-time animations.

This approach uses third-party external tools to create machine-learning-generated videos based on text prompts. The result is visualized once an avatar is selected. Several websites of this kind exist, allowing users to input text and generate realistic videos of animated talking-heads, (Vidnoz, 2024), (Synthesia, 2024), (HeyGen, 2024). This approach requires minimal implementation effort compared to the methods previously mentioned. Talking heads are favored for mobile apps and ideal for chatbots.

3.2 Quality Testing

Evaluating the quality of animations is challenging given that there is no standardized method for it. A useful tool that was developed to aid the comparison before

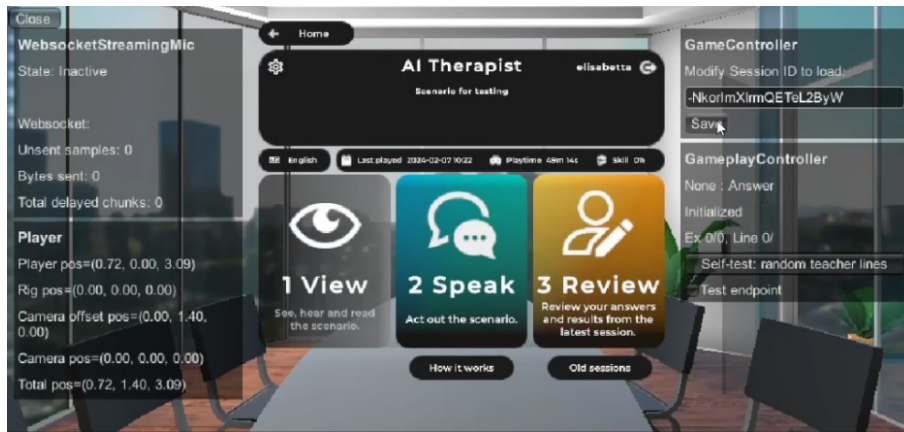


Figure 3.1: Fictive Reality’s settings during a session showing that the ID of a specific session can be inserted to be loaded. Image by author.

and after applying changes to a scenario is the "Edit Scenario ID" button. When playing a scenario in the app, temporary data is saved, such as the user’s audio and username. Fictive Reality regularly deletes this temporary data, but it is often useful to retain scenario data coming from the testing team for analysis purposes. Implementing a button to load a specific session through its ID allows to compare changes and visualize reported errors in the app, see Fig. 3.1. This tool can be valuable to evaluate changes in the quality of animations by reproducing similar conditions in terms of avatar, dialogue and setting.

While accuracy of the neural network model is first checked with an accuracy metric from TensorFlow, together with the MSE loss, the quality of the output is mainly checked through visual inspection in Blender and Unity. Another means for evaluating the results is using histograms to compare the test data with its corresponding predicted data.

3.3 Approaching Challenges

A number of challenges arise during the research of possible approaches and their implementation. In the early stage, the main difficulty centers on finding a suitable dataset and relevant literature that uses text as the main input data. Identifying similar studies proves to be particularly challenging for real-time applications. A selection of literature focused on solutions for real-time applications, which also demonstrates how the text data functions, helps in identifying suitable approaches.

Understanding the company’s existing pipeline and software poses challenges, as animations require precise timing, optimization, quality, alignment with text, audio, scenario, and avatars. This issue is simplified by using a test scenario to evaluate the code developed and animations in a separate space, still within the app. The downside of this simplification is that server integration often clarifies the challenges of the chosen approaches.

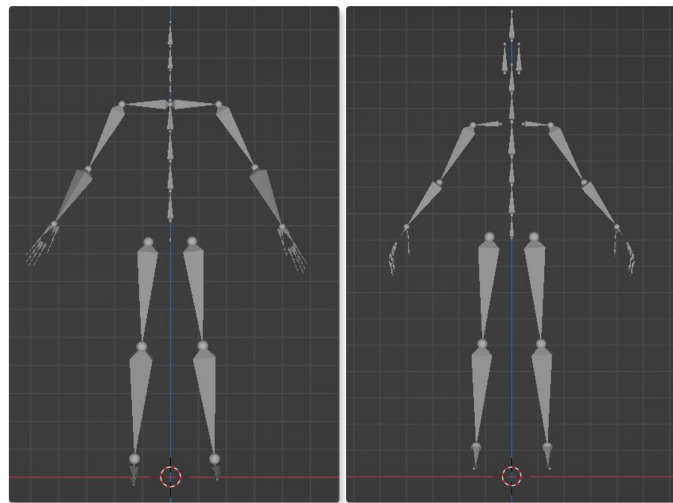


Figure 3.2: On the left: The skeleton used in the dataset. On the right: The skeleton used for Fictive Reality’s avatar. They differ in many aspects, including joint number, hierarchy and scale. This complicates the implementation of the neural network output into Fictive Reality’s app. Image by author.

The timing of animations itself presents a challenge due to the need for smooth transitions and realistic movements. The amount and type of gestures during a conversation vary depending on cultural differences. Other factors include the setting of the conversation, the subject, and whether both parties sit or stand. This topic is tackled differently in the two main approaches. For the LLM, the focus is on ensuring that face animations do not overlap with lip sync, so they play primarily before and after the generated text. For the neural network approach, body and face animations operate on two separate layers.

In the development of the neural network approach, two main obstacles arise regarding body animations. On the visualization side, the dataset’s skeleton differs from the one used in Fictive Reality’s app, necessitating the re-targeting of the motion data, see Fig. 3.2. The number of joints, starting position and angles of each bone, the hierarchy and scale all differ. This process requires creating a dictionary and mapping together different bones.

The training of the neural network model appears stuck in a local minimum, producing only a skeleton with animations oscillating around a T-pose. Increasing the accuracy of both face and body animations in this approach is also challenging, as there are numerous steps where there is room for improvement, e.g. in the pre-processing, encoding, model architecture, training and post-processing. The main obstacle to the model’s improvement relates to the available technology. Loading and analyzing the data is memory demanding, and to reduce the risk of overfitting and to enhance the model, more memory is necessary. In Section 5.5 some approaches to both address data handling and reduce memory load are introduced.

The last two difficulties emerge as fundamental parts of the project, namely defining

and evaluating the quality, as well as addressing the real-time problem. To compare results, a system is developed to reload previously played sessions in the app. To also evaluate the quality, research is conducted on the fundamentals of animations. Primarily, eye evaluation and a Likert scale are used. Regarding the real-time problem, it introduces both complexity and significant constraints, as the options for available approaches decrease. In fact, real-time animations require synchronizing multiple components, optimizing the software, and eliminating latency.

4

Results

The methods investigated for generating animation commands are the use of large language models and neural networks, as stated in Chapter 3. Since both LLMs and the neural network approach are studied for mood tag generation for the state machine, as well as for generating direct animation instructions, the results are presented separately below. Because of the choice to split face and body animations, those results are also presented separately.

To introduce the outcomes of the two approaches, the findings from the initial research on prompt formulation are presented first. This research demonstrates that the LLM is reliable to an extent, yet it fails to be consistent across multiple aspects simultaneously. For example, if the output format is consistent at one time, the choice of animations may exceed the allowed list, or vice versa. It is also evident that the LLM has a hard time choosing the appropriate amount of animations for the different sentences, and picks most often either too few or too many. Other concerns also arise, such as matching animations to specific words rather than the overall context. For short sentences, however, the issues are less prominent and are consistently less evident throughout the prompts. Three important prompts and examples of their corresponding outputs are shown in Table A.4, where these characteristics are evident.

An investigation on including timestamps into the outputs of the LLM is conducted and its outcome is presented in Table A.5. In addition to the attributes from the previous prompt research, this shows that the LLM is generally unsatisfactory in determining the appropriate timing for animations, whether it provides timestamps in seconds or relative to the length of the sentence.

4.1 Face Animation

To generate face animations, in the first approach, the existing mood tags from the Unity state machine are considered as outputs from an LLM: interested, happy, irritated, in doubt, bored and neutral. In the second approach, blendshape names and values are generated, also using an LLM. Finally, in a third approach, blendshapes are instead generated by a neural network model trained on blendshapes and corresponding text data, using four-second intervals.

4.1.1 Generation of mood tags with an LLM

The input prompt for mood tags includes scenario details, the attitude the avatar in the conversation needs to manifest, and a list of possible moods that match those available in Unity’s state machine. The prompts are provided in Table A.6 along with Table A.7 (1), while the results are shown in Table 4.1. The results indicate that the bot’s mood tag updates according to the conversation content throughout.

Speaker	Result
Avatar:	« [Mood: Neutral] Oh, hey there. Thanks for reaching out. Yeah, it’s true, I just lost my job. It’s a tough situation to be in. »
User:	« I am really sorry this happened to you. Do you want to talk about it? »
Avatar:	« [Mood: Neutral] Thanks for your concern. Yeah, I guess talking about it might help. It’s just tough because I put a lot of effort into that job, and losing it feels like a setback. »
User:	« Cheer up, here is a joke: What’s a cat’s favorite instrument? Purr-cussion. »
Avatar:	« [Mood: Interested] Haha, that’s a good one! Thanks for trying to cheer me up. It’s nice to have a little laughter in the midst of this situation. »
User:	« One more! How do you make holy water? You boil the hell out of it. »
Avatar:	« [Mood: Happy] Haha, that’s hilarious! Thanks for sharing another joke. Laughter really is the best medicine sometimes. You’re really helping to lift my spirits. »

Table 4.1: Example of output when using the mood tags prompt from Table A.6 and Table A.7 (1). This shows that the mood tags update in accordance to the conversation.

4.1.2 Generation of blendshapes with an LLM

To illustrate how blendshapes work and the significance of the weight value, two different blendshapes are applied to a model below, each with a range of parameter values, as shown in Fig. 4.1. This demonstrates that blendshape values under 0.5 are barely noticeable, particularly at the typical distance used in a scenario. It also shows that some blendshapes are more visible than others.

In the case of generating direct blendshape instructions with an LLM, it is necessary to add the complete list of blendshape names to the prompt, see Tables A.1, A.2, A.3. The prompt is designed to include instructions on the number of blendshapes to generate and the required format. See Table A.7 (2) for an example of a prompt that generates two blendshapes. Aside from this change, the prompt is identical to the one used for mood tag generation.

After testing the prompt that generates two blendshapes, the prompt is modified to



(a) An example of the blendshape *cheekPuff* at different values.

(b) An example of the blendshape *browInnerUp* at different values.

Figure 4.1: Two different blendshapes applied to a model, demonstrating the visibility of different parameter values. In the figures, the top left shows the model with the blendshape parameter set to 0, the top right to 0.25, the bottom left to 0.5 and the bottom right to 1. This illustrates that blendshape values below 0.5 are difficult to notice. Image by author.

generate four at a time. To do this, Table A.6 and Table A.7 (3) are used together. Additional specifications are included in the prompt to provide context, a goal or to assign roles. Examples of these are: assigning the bot the role of a shop assistant, setting the context of a user wanting to buy a bike while there is a long queue, and setting as a scenario’s goal that the user has to remain kind and patient. In Table 4.2 is shown the output of the LLM after including the bot’s role specification as a shop assistant. The results of setting the role of the bot to work in HR, and the user to be an employee with a negative attitude that has been rude lately, is shown in Table 4.3. In every test conducted, four blendshapes are correctly selected and fall within the assigned range for each bot response. Results also show that the LLM selects different blendshapes to update between responses.

The implementation of using three blendshapes at a time is also investigated. An example output is shown in Table 4.4. While the LLM is successful in following instructions, using three blendshapes forces the avatar to display asymmetric expressions.

4.1.3 Generation of blendshapes with a neural network model

The process of generating blendshapes with a neural network model benefits from first visualizing the data it is built on. Fig. 4.2 presents the values each blendshape takes throughout the dataset. Although the blendshapes related to lip sync, such as the jaw and mouth blendshapes, are removed before training the model, they are in-

4. Results

Speaker	Result
Avatar:	« [Blendshape1: eyeLookInLeft = 0.8, eyeLookInRight = 0.8, jawOpen = 0.6, mouthClose = 0.6] Hello there! Welcome to our bike shop. How can I assist you today? »
User:	« Hi! I would like a bike to use in the city daily. Red, if possible »
Avatar:	« [Blendshape1: eyeWideLeft = 0.7, eyeWideRight = 0.7, browDownLeft = 0.6, browDownRight = 0.6] Of course! We have a variety of bikes suitable for city commuting. Red, you say? Let me check our inventory for available options. Just a moment, please. »
User:	« Just make it fast, will you? »
Avatar:	« [Blendshape1: eyeWideLeft = 0.7, eyeWideRight = 0.7, browDownLeft = 0.6, browDownRight = 0.6] I understand your eagerness, and I'll do my best to expedite the process for you. Thank you for your patience. »

Table 4.2: Generated output from LLM with four blendshape values given in the range 0.5-1 for each avatar response. The LLM is successful in following the instructions, and chooses different blendshapes to update between the responses.

Speaker	Result
Avatar:	« [Blendshape1: eyeSquintLeft = 0.8, eyeSquintRight = 0.8, browDownLeft = 0.7, browDownRight = 0.7] Hello, thank you for coming in today. I understand we need to discuss recent interactions with clients. Can you share your perspective on what's been happening? »
User:	« No, I don't know why I am here. Let me get back to work »
Avatar:	« [Blendshape2: mouthFrownLeft = 0.8, mouthFrownRight = 0.8, browDownLeft = 0.7, browDownRight = 0.7] I understand you may feel unsure about why you're here, but it's important for us to address any concerns or issues that may be affecting your performance. Can you take a moment to talk about what's been going on? »
User:	« If I have to, but it's my business. I think my performance is fine! »
Avatar:	« [Blendshape3: mouthPressLeft = 0.6, mouthPressRight = 0.6, browDownLeft = 0.7, browDownRight = 0.7] I appreciate your willingness to discuss this. There have been reports of negative interactions with clients, and it's important for us to address any issues that may be affecting the team or our reputation. Can you help me understand what might be causing these interactions? »

Table 4.3: Output from LLM when generating four blendshapes at a time. The LLM successfully chooses four blendshapes in the assigned range for each avatar response. It also chooses different blendshapes to update throughout the conversation.

Ex.	Result
1	« Avatar: « [Blendshape1: eyeBlinkLeft, Blendshape2: mouthSmileRight, Blendshape3: browInnerUp] Hello there! Welcome to our bike shop. I see you're interested in purchasing a bike today. How can I assist you? »
2	[...]
3	Avatar: « [Blendshape1: mouthSmileLeft, Blendshape2: eyeBlinkRight, Blendshape3: browInnerUp] You're welcome! I'll process your payment right away. »

Table 4.4: Two example outputs when using an LLM to generate three blendshapes to each avatar response. The LLM follows the instructions, however the avatar are forced to be asymmetric in its face expressions.

cluded here for additional insight. The most notable feature among the blendshapes is the long whiskers and the large number of outliers, indicating that the data is noisy but extensive.

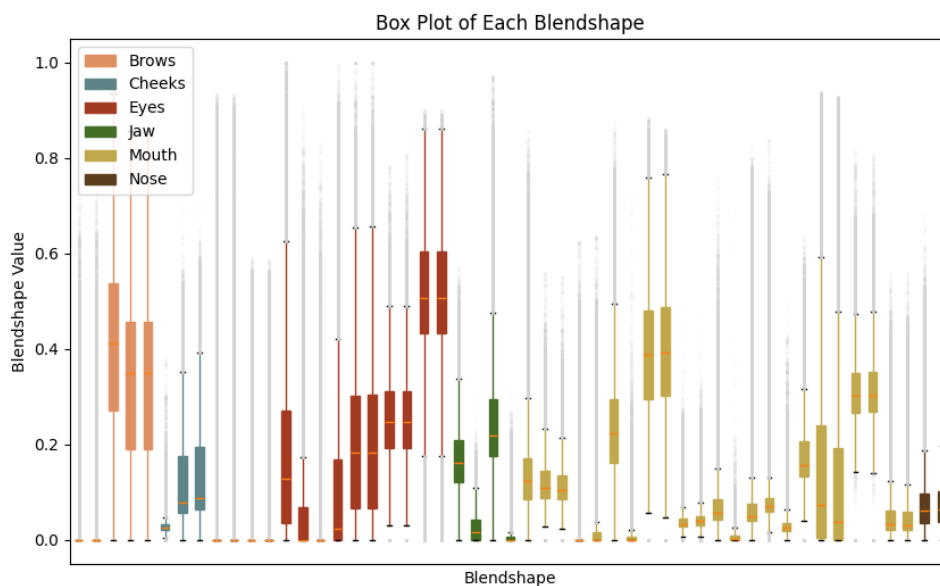


Figure 4.2: Box plot of each blendshape value in the data, grouped in color by face parts. Most blendshapes show a high number of outliers and long whiskers. Image by author.

In relation to this, Fig. 4.3 shows the mean value each blendshape takes throughout the dataset. The figure illustrates that no blendshape has a mean activation over the value 0.5, which, as previously mentioned, is difficult to detect during visualization. This suggests that, on average, individual blendshapes are not easily visible.

Fig. 4.4 shows the changes in accuracy and mean squared error loss during training,

4. Results

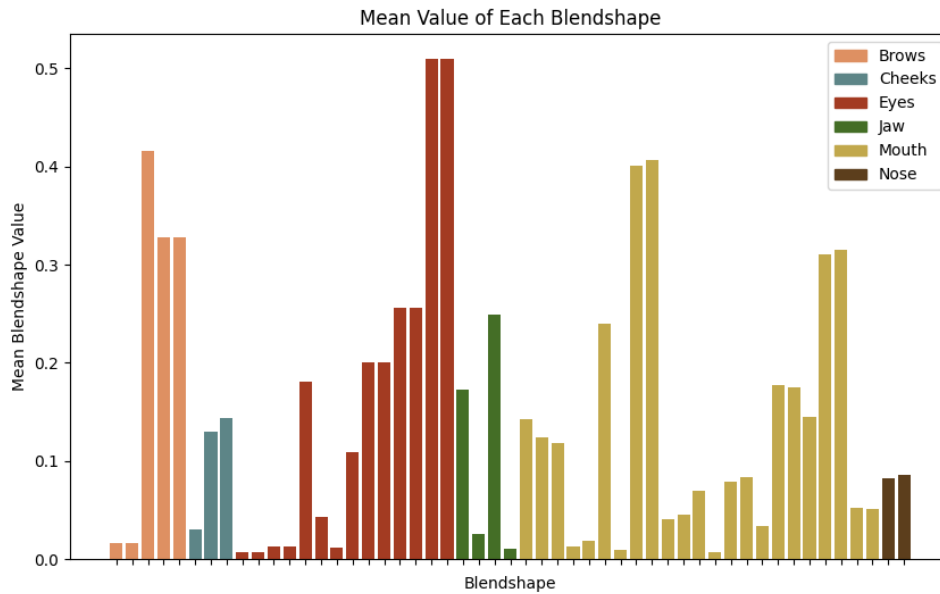


Figure 4.3: Mean value of each blendshape in the data. Blendshapes averages range from close to 0 up to 0.5. The image also gives an indication of which blendshapes are the most important. Image by author.

for both the training and validation data. The fact that the training accuracy curve has a positive slope over the epochs, while the training loss steadily decreases, indicates that the model is trained and updated. However, the validation data shows a less predictable accuracy curve, with no significant development in either direction, while the validation loss slightly increases. This suggests overfitting, which is discussed later on.

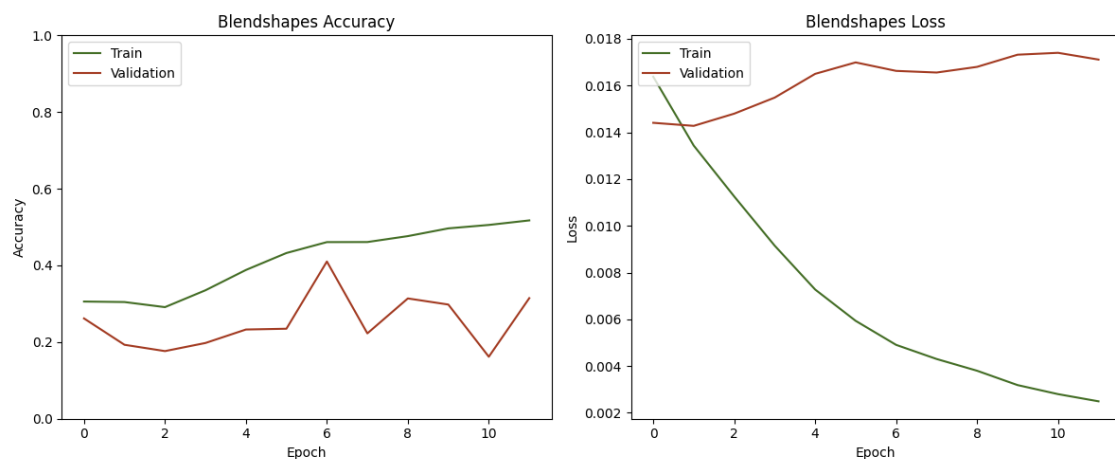


Figure 4.4: Accuracy and MSE loss for training and validation data during the training of the face model. The increasing accuracy together with the decreasing loss for the training data shows that the model is trained, yet the behavior of the validation curves indicates the presence of overfitting. Image by author.

To visualize the outcome in another way, Fig. 4.5 presents the distribution of blendshape values across different sets. It shows the distribution of values across the whole dataset, the training data, the target data, and the predicted outcome using the test data, respectively. In an ideal scenario, the predicted outcome has the same distribution of blendshape values as the target data. While the predicted output does display a similar distribution and general appearance compared to the target distribution, the most evident difference is in the shape of the tails, where the predicted output has a saddle before larger values. Disregarding the proportion of the y-axes, a slightly heavier tail is observed in the training data compared to the target data, which can be a cause for the heavier tail in the predicted output.

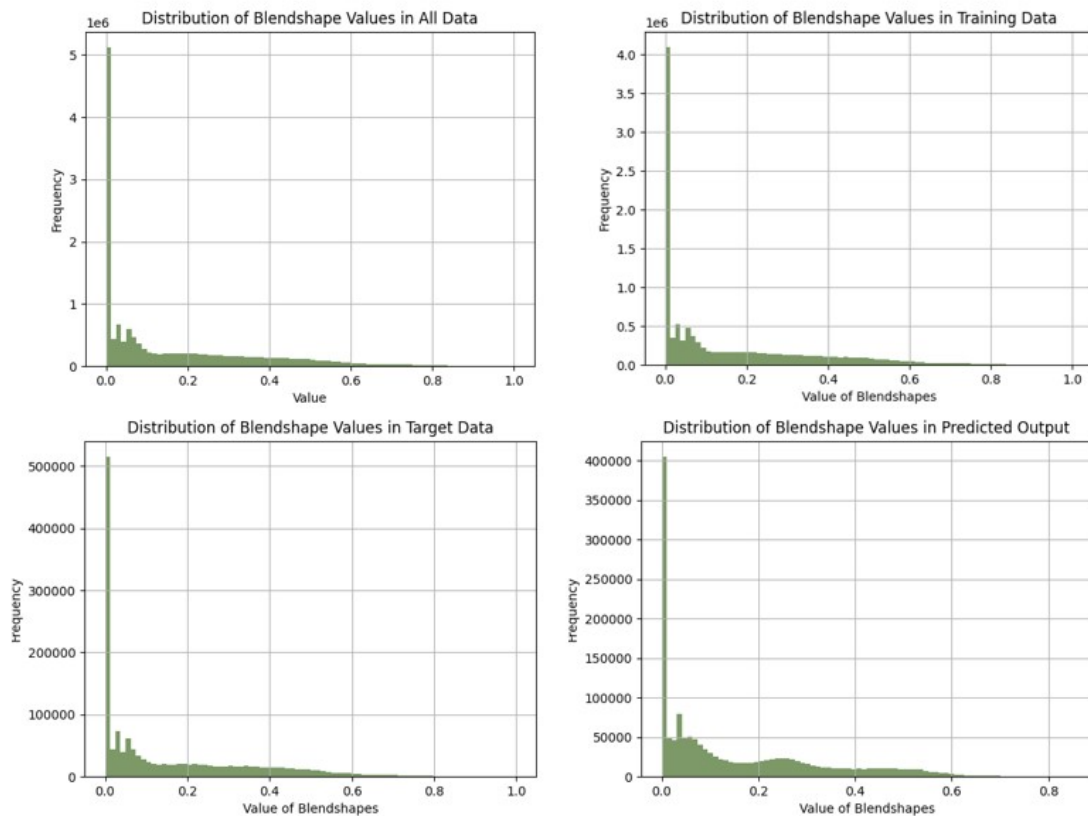


Figure 4.5: Distribution of blendshape values in different sets of the data. The distributions are in large similar, however the predicted output shows a heavier tail with a prominent saddle. Sub-figures in order: **A:** The distribution of blendshape values among the whole data. **B:** The distribution among the training data, which corresponds to 80% of all data. **C:** Another 10% of all data, now corresponding to the test (target) data. **D:** The distribution of values in the predicted data, which optimally aligns entirely with the target distribution. Image by author.

The errors in the predictions are also presented as averaged errors for each blendshape in Fig. 4.6, excluding the blendshapes related to lip sync. The highest average error among them is around 0.16, with the maximum possible error being 1 due to the range of possible blendshape values. The mean errors are compared to the mean values in the raw data in Fig. 4.3, where the sizes of the error bars follow a similar

distribution in both figures. This implies that the errors follow the same trend as the values shown in the box plot in Fig. 4.2, where blendshapes with short or no whiskers and with a median close to 0 exhibit proportionally lower average errors, and vice versa.

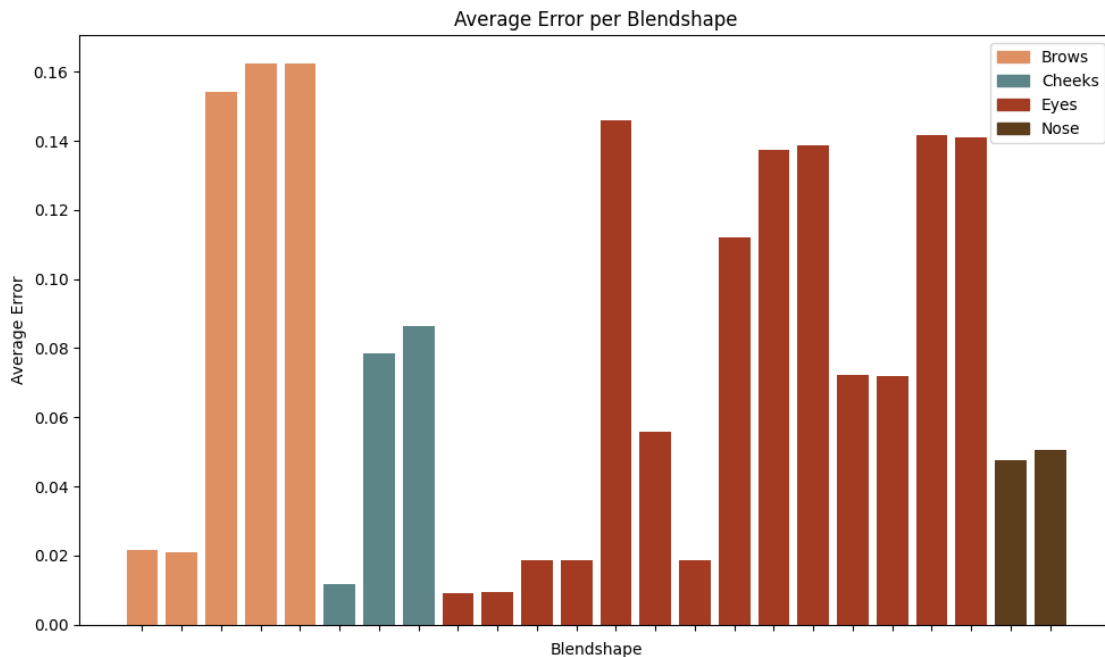


Figure 4.6: Average error per blendshape, when comparing target data to predicted output. In total this corresponds to an accuracy of 29.46%. Note that the highest average error is about 0.16, and recall that blendshapes can take values in the range $[0, 1]$. Lip sync blendshapes, i.e. jaw and mouth, are not included. Image by author.

Finally, the resulting JSON file for the chosen input to the neural network model "Good, do you know where I can find the bus stop?" is visualized and presented here as still images in Fig. 4.7. One can notice the changes in the eyebrow positions, eye squint and cheeks, to name a few. The avatar presents a pleasant yet natural expression, fitting the sentence used for animation.

4.2 Body Animation

Body animations are approached differently from facial animations. This is primarily due to the fact that most of Fictive Reality's scenarios involve a seated conversation and, secondarily, because large movement animations such as walking come second in difficulty. However, the general approaches remain similar to those used for facial animations, for both the large-language models and the neural networks, as outlined in the following sections.

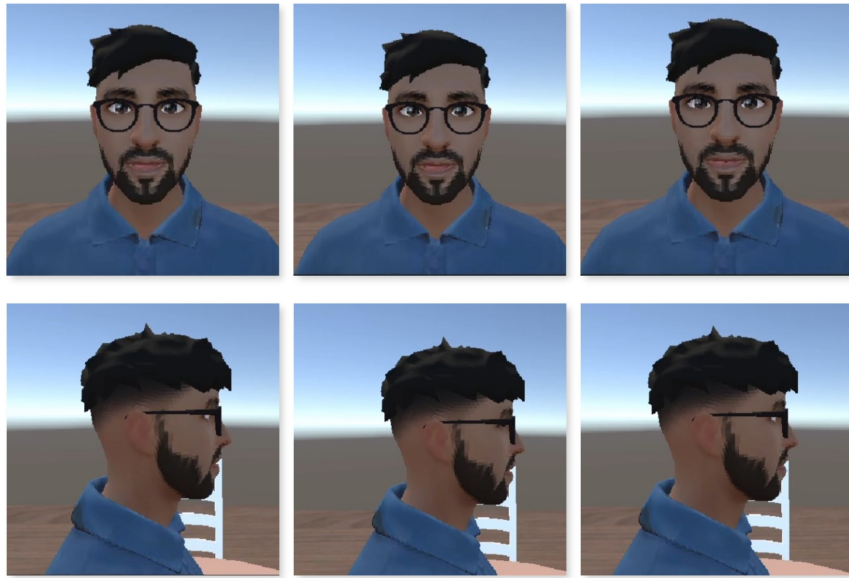


Figure 4.7: Visualization of a few frames of the generated animation using Unity and one-hot encoding of the sentence "Good, do you know where i can find the bus stop?" as input to a neural network model. Shown from the front (top) and from the side (below). Image by author.

4.2.1 Generation of motion tags with an LLM

The motion tags that are generated with the first approach assume these type of movements as the most likely to occur in a conversation: standing up, thinking pose, angry arm movement, staying seated, no movement, explanatory gestures, nodding, leaning back, tilting the head, adjusting shirt, pointing finger, denying and shifting in seat. Movements under the table when seated, e.g. crossing the legs, are excluded because they are not visible given the scene's setup, see Figure ??.

The prompt sets the LLM to give two motion tags per response chosen from a list, as per the prompt in Table A.6 and Table A.7 (4). An example of how a conversation evolved is shown in Table 4.5. The animations chosen by the LLM are usually appropriate, but sometimes it shows its inability to hold memory of the context. For example, a motion tag chosen after the user says "Can I sit while you do that?" is "Staying seated". While a person understands the correlation between the conversation and the motion tag, the LLM takes the word "sit" out of its context. The same motion tag is also in contradiction with the first motion tag "Standing up", given in the beginning of the conversation.

This approach is also tested for both one and three output movements, however both resulted inferior in quality. The first case results in a rather still action, in which only "pointing finger" or "nodding" can occur at the same time, while the latter produces results such as those in Table 4.6, which involve too many movements and hence lose both in realism and in quality.

Speaker	Result
Avatar:	« [Movement1: standing up, Movement2: thinking pose] Hello there! How can I assist you today? Looking to send a package, I presume? »
User:	« Hi! Yes, exactly, to Australia! It is very heavy »
Avatar:	« [Movement1: thinking pose, Movement2: nodding] Alright, we can handle heavy packages. Do you have it with you? I'll need to know the weight and dimensions to give you an accurate quote for shipping to Australia. »
User:	« Can't you measure it yourself?! »
Avatar:	« [Movement1: adjusting shirt, Movement2: nodding] Of course, I can help with that. Please bring the package over to the counter, and I'll take care of measuring and weighing it for you. »
User:	« No, I am kidding, it is a 200x100x100 and it weights 100 kg »
Avatar:	« [Movement1: nodding, Movement2: pointing finger] Thank you for letting me know the dimensions and weight. That's quite a sizable package. Based on this information, I can provide you with the shipping options and costs to Australia. Let me check our system for the available services. »
User:	« Thanks. Can I sit while you do that? »
Avatar:	« [Movement1: staying seated, Movement2: nodding] Of course, feel free to take a seat while I check the shipping options for you. It won't take long. »
User:	« Where can I sit? »
Avatar:	« [Movement1: pointing finger, Movement2: nodding] There are some chairs right over there by the window. You can have a seat while I look up the shipping details for you. »
User:	« OK! »
Avatar:	« [StopScenario] »

Table 4.5: Example of output generated from the prompt shown in Table A.6 together with Table A.7 (4). This includes two motion tags per avatar response. Two movements at a time appears relevant. The LLM's choices of animations takes at times, however, single words out of their context and no natural continuity between correlating animations, such as between "standing up" and "staying seated".

4.2.2 Generation of motion data with an LLM

The second approach for body animation is the generation of motion data, specifically for each bone of Fictive Reality's app skeleton. The input is the list of bones and their hierarchy, and the general prompt structure is the same as previous implementations, see Table A.6 and Table A.7 (5). In this case, the output is only zeroes at each line of the conversation. For this reason the prompt is modified so

Speaker	Result
Avatar:	« [Movement1: standing up, Movement2: nodding, Movement3: adjusting shirt] Welcome! How can I assist you today? »
User:	« Hi! I need to send this package to Australia. Can you helo me with that? »
Avatar:	« [Movement1: nodding, Movement2: thinking pose, Movement3: pointing finger] Of course! We can definately help with that. Do you know the approximate weight and dimensions of the package? That'll help me determine the shipping options and costs for you. »
User:	« Can't you measure it yourself? »
Avatar:	« [Movement1: nodding, Movement2: denying, Movement3: leaning back] I understand your request, but unfortunately, I'm unable to measure packages myself due to store policy and time constraints. However, I can provide you with the necessary tools and guidance to accurately measure the package yourself if you need assistance.»

Table 4.6: An example of generation of motion tags with three different movements as output to each avatar response. Using many movements at once, especially when with the complexity these have, seems forced and hence lacks in both realism and quality.

that only the motion data of the bones currently moving are printed. The result, as seen in Table 4.7, is still lacking in crucial parts but with some improvements. These results indicate that the prompt is of great importance when using an LLM, and that better results can be obtained by improving the layout of the prompt and reducing the length of the output.

4.2.3 Generation of motion data with a neural network model

A similar neural network model to the one developed for the blendshapes is used for the body animations, but with the important changes presented in Section 3.1.2. The model is now trained on motion data files and text data such that for any text input it generates corresponding motion data for the armature.

Similarly to the neural network results for the face, the median and mean values of each body joint are first presented to give a better picture of the outcome of the resulting model. These are shown in a box plot in Fig. 4.8 and in a histogram in Fig. 4.9. The box plot shows that most joints have very few outliers, with the exception of the feet. It also shows that body parts that have a mirrored version on the other body half display similar distributions in values, where the outer ends of the arms are clear examples of this. The outer arms are also the most moved body parts in relation to the body joints that precede them.

Regarding the mean values of the joints in the dataset, Fig. 4.9 presents both the individual x , y , and z coordinates and their average value. It illustrates that most

4. Results

Speaker	Result
Avatar:	« [Motion data: RightHandThumb3: 30, 0, 0; RightHandThumb4: 15, 0, 0] Hello there! How can I assist you today with your package to Australia? »
User:	« Hi! I need to send it but it's very big and I don't want to spend too much »
Avatar:	« [Motion data: RightHandThumb3: 30, 0, 0; RightHandThumb4: 15, 0, 0] No problem! Let's see what we can do. Do you have any specific requirements or preferences for shipping, like speed or insurance? »
User :	« I would like to ship my package by airplane »
Avatar:	« [Motion data: Hips: 0, 0, 0, 0, 0, 0; Spine: 0, 0, 0; Spine1: 0, 0, 0; Spine2: 0, 0, 0; Neck: 0, 0, 0; Head: 0, 0, 0] When shipping your package by airplane, we'll need to ensure we get all the details right. Could you please provide the dimensions and weight of your package? »
User:	« The package is 200x100x100 and it weighs 100 kg »
Avatar:	« [Motion data: Spine2: 0, 0, 0; LeftArm: 0, 0, 0; RightArm: 0, 0, 0] Thank you for providing the details. With dimensions of 200x100x100 cm and a weight of 100 kg, shipping by airplane is indeed the most suitable option to meet your deadline. Let me calculate the shipping cost for you. »
User:	« Thanks. Can I sit while you do that? »
Avatar:	« [Motion data: Hips: 0, -15, 0, 0, 0, 0; Spine: 0, -10, 0; Spine1: 0, -5, 0; Spine2: 0, -5, 0] Of course, feel free to take a seat while I calculate the shipping cost for you. Sits down »
User:	« Where can I sit? »
Avatar:	« [Motion data: Hips: 0, -15, 0, 0, 0, 0; Spine: 0, -10, 0; Spine1: 0, -5, 0; Spine2: 0, -5, 0] You can find a seat right over there by the counter. It's the one with the cushion. »
User:	« OK! »

Table 4.7: Output to a prompt printing out motion data only relatively to the moving joints.

joints tend to move in opposite directions reflecting the symmetry of the body. For instance, when one arm rotates outward with a positive angle the other rotates inward with a negative angle. The comparison between the box plot and histogram reveals that while many joints are quite active: their mean positions across the entire dataset tend to average close to zero degrees.

The development of accuracy and mean squared error loss during training of the model for validation and training data are shown in Fig. 4.10. They illustrate how the training accuracy changes, following a positive trend, and how the loss declines.

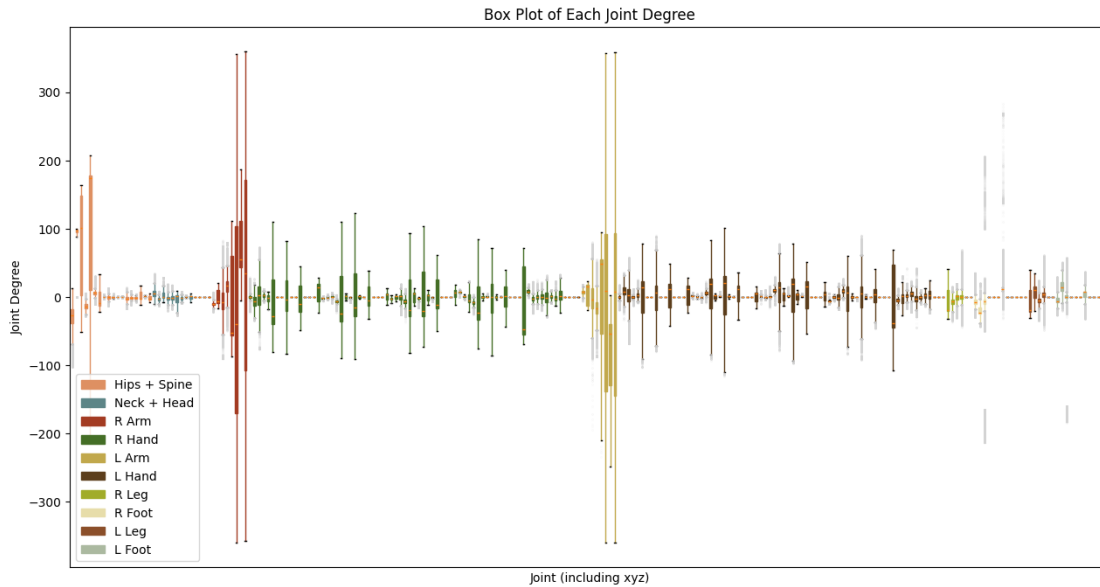


Figure 4.8: Box plot of each joint degree, with x , y and z coordinates separate. Each joint is grouped in color by relating joints. Note how corresponding right and left joints have similar distributions. Note also how the outer ends of the arms show large ranges in distribution of degrees. While some joints rarely show rotations larger than a few degrees, some have noticeable whiskers without outliers and yet others have a prominent number of outliers, for example the feet joints. Image by author.

This indicates overfitting.

The distribution of joint degrees among different sets of data are given by Fig. 4.11. There, the distributions are shown for all data, the training data, the target data, and finally the predicted output. Similarly to the blendshapes, in an optimal scenario the distribution of predicted values aligns perfectly with the target data distribution. Even though that is not enough to determine complete accuracy, it gives an indicator of it. The result shows that the predicted data indeed aligns with the target data distribution, yet it has an overall smoother contour. Some key aspects are however present, such as the peak at 0 degrees, a smaller peak at 100 degrees, a bulge in the outline in between 200 and 300 degrees, and a drooping edges near the interval limits.

Finally, the mean errors of each joint are displayed in Fig. 4.12, both when the x , y and z coordinates are separate and when averaged. The largest mean errors are held by the outer ends of the arms, which were the same joints to show large whiskers in the box plot in Fig. 4.8. One can also see a trend among the hands, where the different fingers behave in a similar manner. Lastly, the errors display a mirrored behavior on opposite body parts, indicating that one side of the body is not more

4. Results

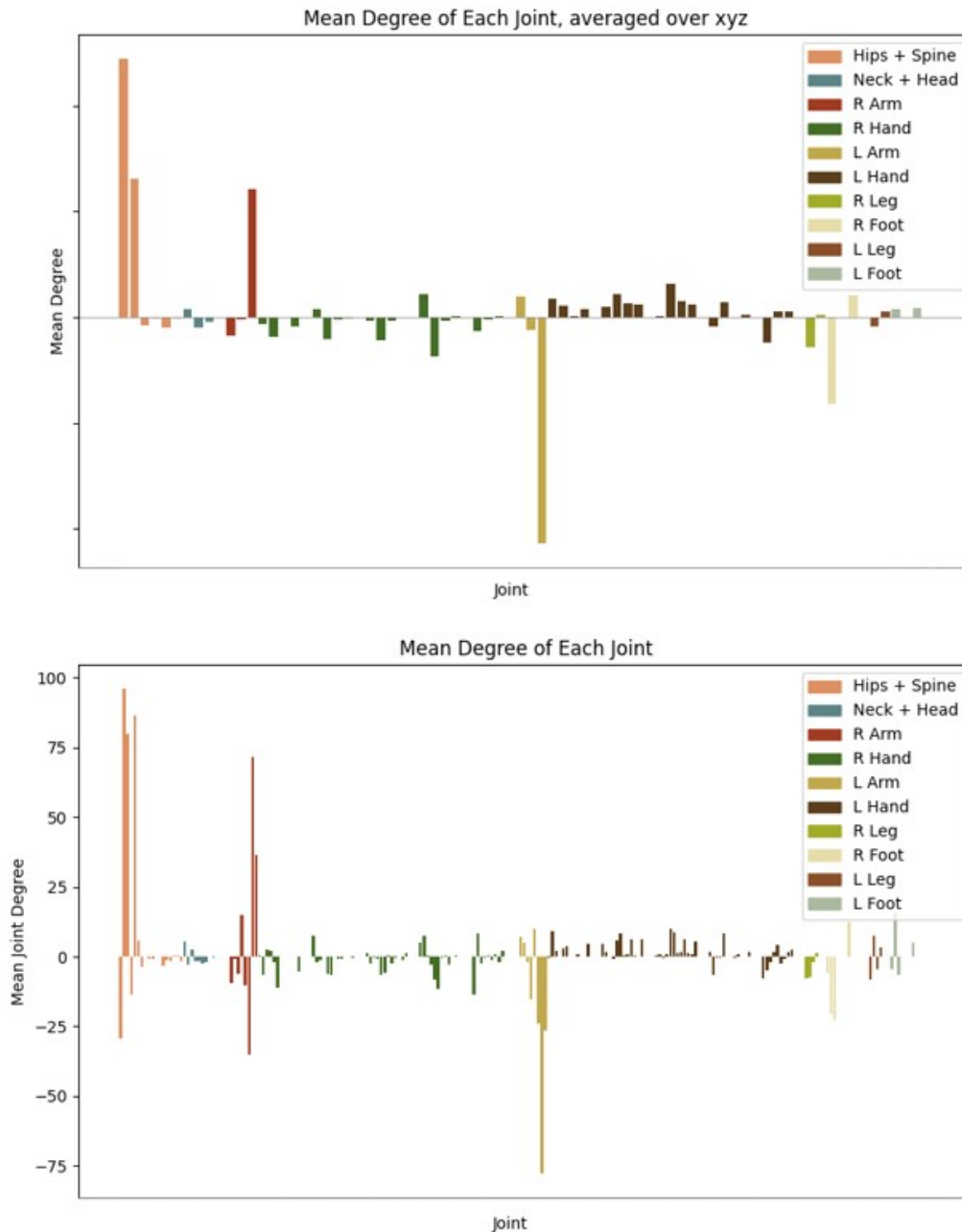


Figure 4.9: Mean degree of each joint in the data. **A:** Mean degrees when each x, y and z coordinates of a joint are averaged. **B:** Mean degrees when x, y and z coordinates are held separate for each joint. **Note:** While some joints display averages close to degree 0 it does not necessarily imply that they are scarcely activated, however it implies that activation against one direction is countered by activation against the other. One can see here that joints corresponding to each other by movements that are generally mirrored in the body halves have means in different directions, see for example the last joint in right arm contra in left arm, or the fact that the right hand is predominantly negative while the left hand is predominantly positive. Image by author.

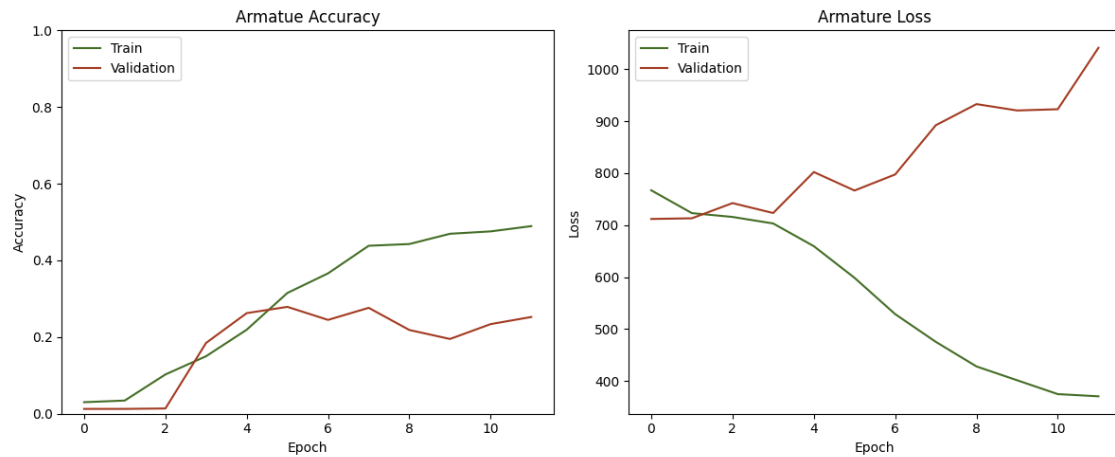


Figure 4.10: Accuracy and MSE loss during training for both training data and validation data. This results show an example of when accuracy and loss can go in the same direction simultaneously. The jagged lines for validation is also worth noticing. Finally, the training show increasing accuracy and decreasing loss, which indicates overfitting. Image by author.

predictable then the other.

Still images from the resulting video animation are shown in Fig. 4.13. While the animated visualization displays twitching and not extensive positional changes throughout the sentence, one can see that the armature skeleton holds a natural posture that do update throughout. This animation is an improvement compared to the one made with less data, where the twitching is more evident and the posture is less natural.

4.3 Video Animation

The option to use existing, external tools to generate talking heads is briefly considered and investigated. Using the website Vidnoz, (Vidnoz, 2024), frames from a generated video of a talking head are seen in Fig. 4.14 where the character describes the weather. This proved to be a simple method with relative good results. However, the lack of flexibility in poses, such as head-turning is evident. It is also clear that it is not fully able to convey emotions involving the whole face and body. Relying on a third party to generate video data is considered, but the time to produce a few seconds of content makes this approach of questionable relevance for real-time applications.

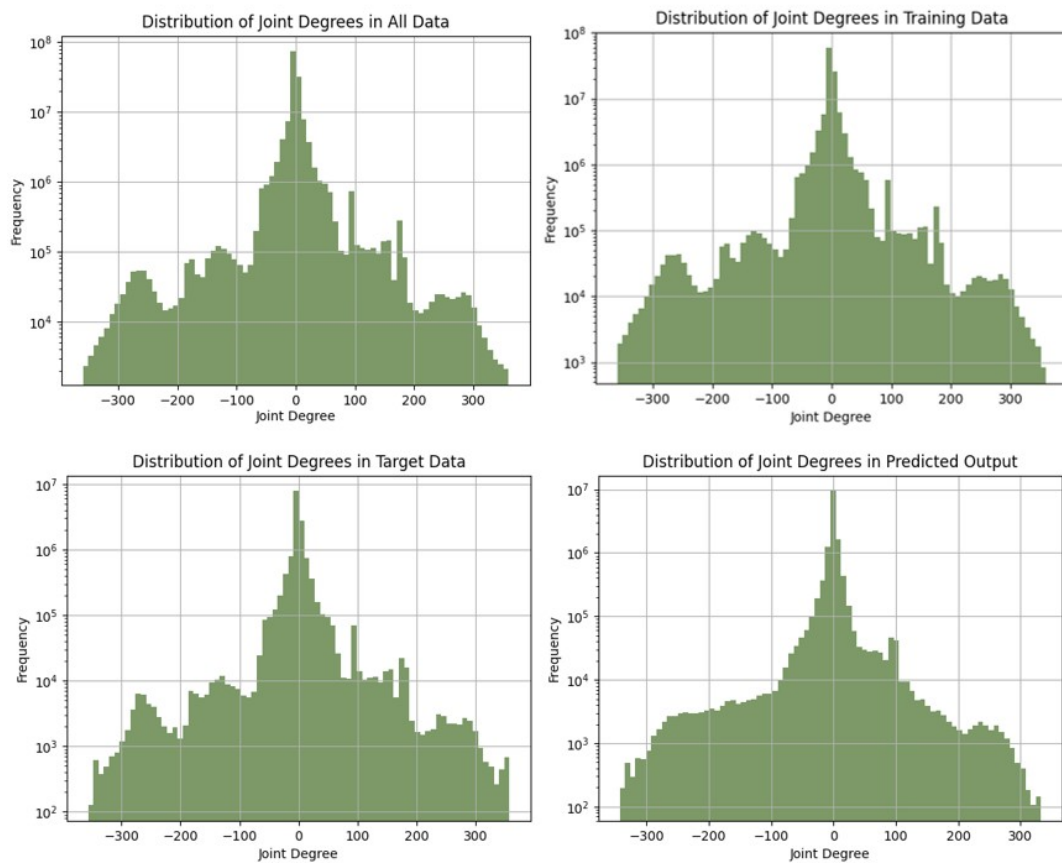


Figure 4.11: Degree Distribution among the joints. **A:** Distribution of joint degrees in all data, when filtered out data points with degrees outside the range $[-360, 360]$. **B:** The distribution of joint degrees used for training, equivalent to 80% of the data in the upper left sub figure. **C:** The distribution of joint degrees used for test, equivalent to 10% of the data in the upper left sub figure. **D:** The distribution of joint degrees among the predicted output when test data is passed through the model. Optimally, this has the same distribution as the target data. Image by author.

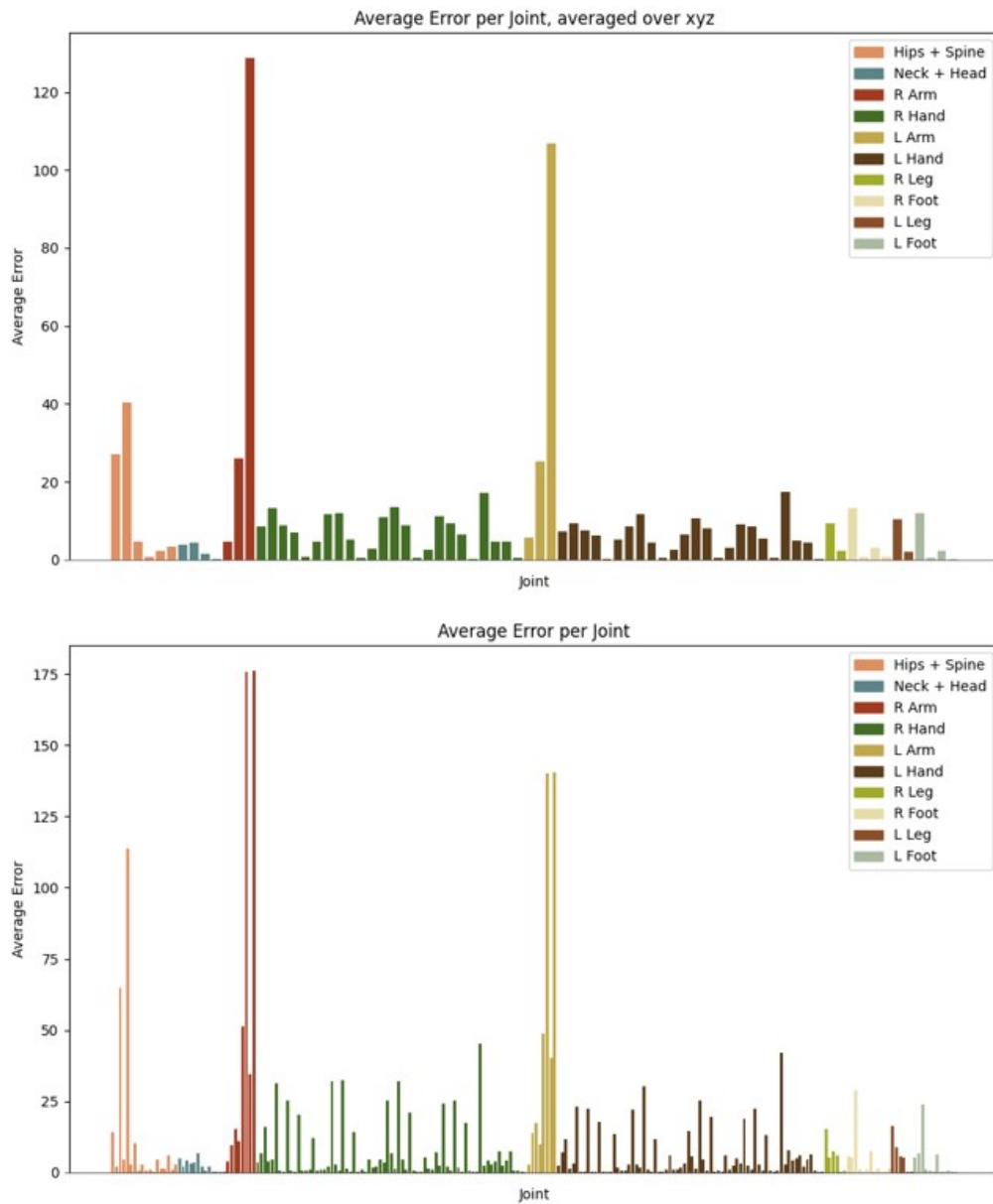


Figure 4.12: Mean degree error of each joint when comparing predicted output to target data for armature. **A:** Mean error when each joints' x, y and z coordinates are averaged. **B:** Mean error when x, y and z coordinated are held separate for each joint. **Note:** While some joints display large errors, it is important to note that they are read in a modded setting, meaning that a degree of value X is the same as $Y = X + n \cdot 360$, for $n \in \mathbf{Z}$. This indicates that even if the shown error is large, the animated output might not display these errors. Image by author.

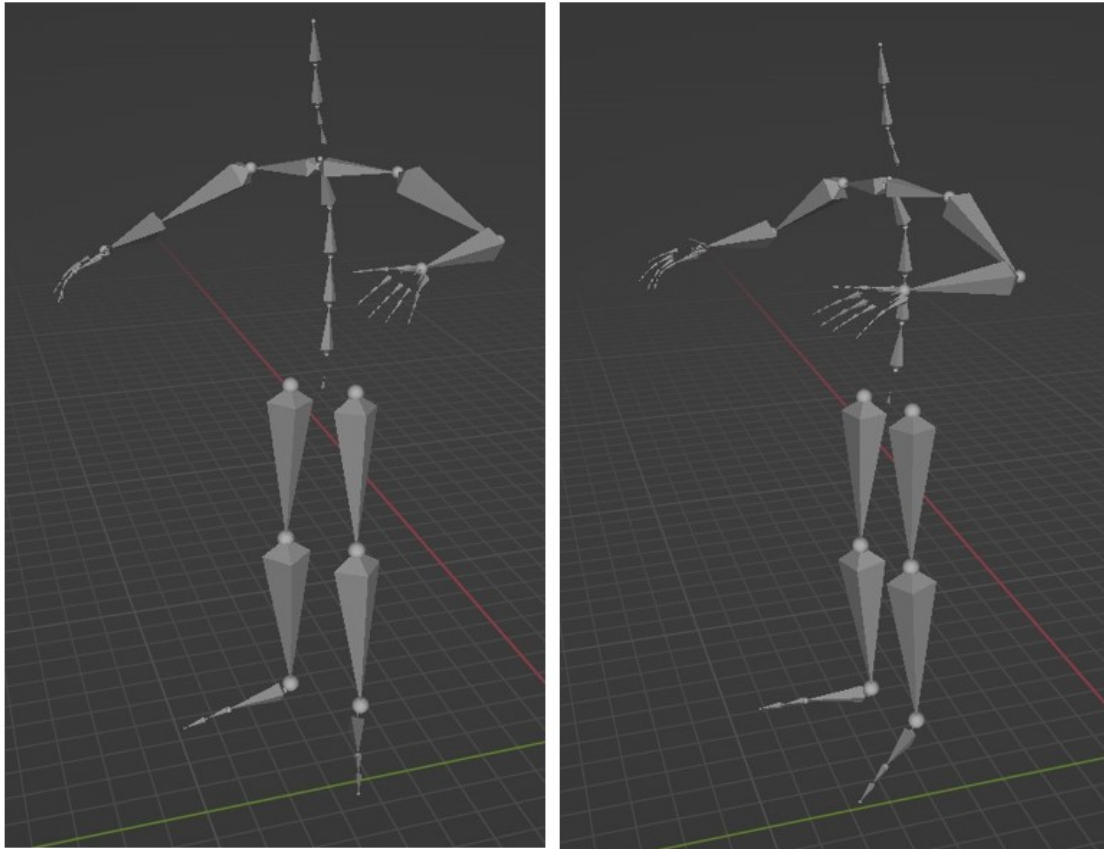


Figure 4.13: Frames from animated output from body neural network model. Taken from the same angle. Image by author.

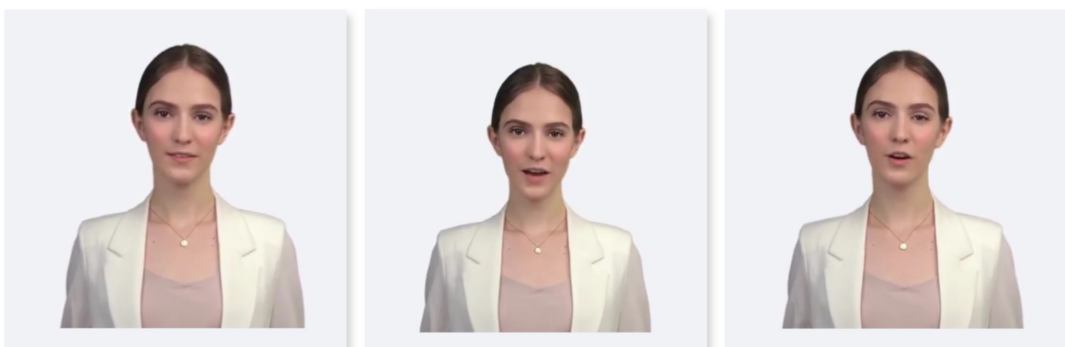


Figure 4.14: Three frames of a video generated using vidnoz.com/avatars.html for a text input describing the weather today. Image by author.

5

Discussion

This section aims to recap and synthesize the work presented so far by discussing the approaches used, their outcomes and by highlighting valuable details. While the field of real-time animations is sparsely investigated itself, its application to both facial and body animations in parallel is even less studied. The results of the large-language model approach are discussed first, followed by the outcome of the neural network approach, and a comparison between the two. Finally, suggestions for future work are presented, including topics of interest that fall outside the current scope and aspects within the scope that were not yet implemented but are worth exploring in order to achieve better results.

5.1 Animation Generation with Large-Language Models

The various methods investigated, which were based on a large-language model, demonstrated a relatively straightforward implementation process and were able to produce tangible results. After some trial and error based on prior research and existing Fictive Reality's prompts, a prompt script was created, see Table A.6. This script was designed to best fit the specific cases investigated in this project, including mood and motion tags, blendshapes, and motion data. Specifically, this prompt was chosen to reduce the complexity added to Fictive Reality's current prompt structure, while generating outputs suitable for evaluating applicability. Minimizing complexity was treated as a bonus for potential future implementations, while the primary focus remained on investigating usability. The chosen prompt format was changed for each specific case to ensure that the LLM was provided with the necessary information, such as the expected output format or predefined tag lists. These specific modifications are outlined in Table A.7.

The quality and, at times, the compliance of the outputs vary significantly between tests, even when using the same prompt. Generating mood tags, as in Section 4.1.1, occasionally gave good outputs, however on average the mood tag tended to remain as the one first suggested by the scenario's context. For example, writing the context as "the bot is sad because they lost their job" makes it impossible to change the mood to anything else than "sad". This is the case even if "sad" was not a mood from the list.

Generating blendshapes instead, as described in Section 4.1.2, produced encouraging results when specifying the number of blendshapes to update at a time. However, two main complications arose. First, the selected blendshapes tended to remain the same four throughout the interaction. Second, there was a lack of symmetry in the chosen blendshapes. The latter issue was the most evident, as the LLM model suggested movements such as blinking with the right eye, smiling with the right corner of the mouth, widening the left eye, and bringing down the left brow. When visualized in Unity, these combinations looked unrealistic and unnatural and were evident in a vast majority of outputs. Compared to the method currently used by Fictive Reality to generate face animations, major improvements were made on the timing of face expressiveness. Animations occurring during the conversation look unnatural because of overlapping movements. The avatar smiling while talking showed teeth or a focused expression while explaining, which made the avatar look angry. Playing the face animations either at the beginning or end and excluding the mouth when implementing the animation’s visualization helped solving these syncing issues to make the conversation look more natural. Effective animations aid the user’s immersive experience. Typically, in a conversation, the focus is on the eyes and on the mouth; in fact, having too many or extraneous movements can break the focus and create confusion in the user. Normally, people change expression during conversation in a minimal way and these changes are too subtle to try to implement without mo-cap techniques. To avoid over-exaggerated and unnatural animations it is better to play them before and/or after the avatar has said their lines. When using blendshapes it is easier to handle transitions.

As for the body implementations, generating motion tags yielded more satisfactory results, particularly when two motion tags were produced for each response throughout the conversation. Although the issue of tags remaining the same persisted here as well, it had less impact on the realism of the animation. This can be attributed to the likelihood that a change in the direction of the conversation is more strongly reflected in facial expressions than in body movements. However, addressing this issue in future implementations is recommended in order to get a better flow of animations. Additionally, the performance was tested with prompts generating only one motion tag at a time, as well as three. The first case deemed to be rather dull and lifeless, while generating three tags at a time resulted in excessive movements, ultimately losing in both realism and quality.

When the LLM was prompted to generate motion data for an armature model, the results were not satisfactory. The prompts to achieve this required the full hierarchy and specifics of the armature, resulting in an excessively long input to achieve a good output. The output ranged from imprecise at best to absurd in most cases. This was an expected outcome and confirms that generating precise motion data for armature models is unfeasible with the current capabilities of LLMs.

While the investigation into using an LLM revealed faults in each case, it still has some potential and benefits. Since Fictive Reality already has another implementation using the LLM with no significant time delay, the real-time limitation is easily

solved by LLMs. Moreover, the prompts developed in this study generated outputs in a non-time consuming manner. For these reasons, using the LLM approach is deemed implementation friendly for integration into Fictive Reality’s server.

The prompts for animation generation designed in the project were developed carefully, yet some tweaks might be possible in order to get better results. Currently, the LLM is not performing well enough for one to expect fully reasonable outputs. Since it worked better when generating mood and motion tags compared to blendshape values or motion data, the LLM is most likely better trained on moods and body language, rather than on numerical data for blendshapes or armatures. Using an LLM for generating mood and motion tags is therefore more feasible. This approach, however, requires further, likely manual implementation of facial expressions and body animations, which direct generation of blendshapes or motion data would have made unnecessary.

5.2 Animation Generation with Neural Networks

For the neural network approach, the face and body animations were again approached in parallel, thus different neural network models were created for each. While trained on the same text data, they also used blendshape and motion data, respectively. Although the separation between the models was already motivated by the state of the art, the lack of available datasets was also a deciding factor.

A benefit of using two models was the flexibility to adjust parameters and have separate network layouts. Additionally, the visualization tools at hand, i.e. Unity and Blender, called for distinct output types. However, using a unified model that handles both face and body at once can yield positive results in synchronization. Still, this project focused on straightforward approaches, a something that was also applied to the neural network investigation.

This consideration influenced the choice of text representation used as input to the models. As advised by Fictive Reality, one-hot encoding was selected as a simple and time-efficient form of text representation, both in terms of implementation and as a baseline procedure. However, when one-hot encoded is used in its standard form, neither the word order nor the word repetition is accounted for. Naturally, these are important elements in a sentence, and although a neural network might pick up on standard sentence structures, the exceptions are still evident. This limitation was promptly identified, and while other ideas (such as modifying the input to be multiple one-hot encoded vectors each containing a single word) were considered, it was decided to proceed with the current implementation for the time being. One key drawback of one-hot encoding is its sparsity. Due to the large token set, most one-hot encoded vectors were zero-dense, leading to increased memory usage. The memory usage increases even more if multiple one-hot vectors are used per input. For future improvements, exploring modifications to one-hot encoding or entirely different text representation methods is recommended. An alternative is word embedding, for instance Word2Vec, that also captures semantic relationships between

words. Setting aside the simplicity and shortcomings of the chosen text representation, in the following the focus goes on the outcomes of using neural networks for real-time animations.

Since the dataset used for the neural network model was originally developed for pre-made videos, this affected how implementation friendly the data was in this project's task. While the four-second splits of the files were not problematic, they removed the progression between high and low values for the motion data. To minimize the confusion this brings to the models, all data points with motion values outside the expected range were removed. This resulted in more predictable values, but at the cost of deleting valuable data to train the models on. Both models exhibited signs of overfitting, and re-approaching the method to handle out-of-range motion values is an encouraged task. When more data points were used, a noticeable reduction in overfitting occurred. This supports the notion that neural networks are fit to create real-time animation generation based on textual inputs.

It is always possible to revisit the choice of parameters, activation functions and network architectures in order to improve the performance of a neural network. In this project, due to computational limitations, smaller networks were chosen, with fewer layers and smaller layer sizes, which certainly affected the models' performance. Because of this, the usage of dropout had a double benefit: it mitigated overfitting and reduced computational power requirements. The accuracy metric used in TensorFlow also calls for a revisit. With an accuracy of around 30% for each model, this metric is accounting only for exact matches between predicted output and target values, hence counts close-enough values as faulty. For blendshape and motion values specifically, the values do not need to be exact for the animations to be valid, so the accuracy metric is not completely appropriate for this task. It is however affirmative that it yielded such high accuracies even with this considered. Finally, the choice of layer types is also an important consideration. While LSTM layers capture sequential data information, other choices of layers may offer better scalability and time efficiency. Both models were quick to generate outputs, and are thus deemed suitable for real-time applications.

Overall, both models demonstrated that using neural networks for real-time animations is feasible. While this approach was mainly limited by computational power and early decisions, significant improvements were observed in the resulting animations as each of the presented concerns was partially alleviated. For example, when managing to gather a little more data for training, the body animations went from a twitching t-pose to a natural body position with significantly more natural arm and spine movements. In conclusion, the models developed here are not suitable for immediate implementation, yet they certainly show promising results.

5.3 Video Generation

Ultimately, using video-generated animations did not seem a valid approach to meet the company's requirements. The most important aspects to consider in the project

were the real-time and the realism to offer an immersive experience to the user. Using video generation requires an intense internet data exchange, unpredictable loading times, and the videos portray avatars with ideal traits and smooth voices, impossible to interrupt until the end of that video sequence. The user feels immersed in an experience when they can relate to it. The tools that were looked into in this project generated videos with exaggerated gestures, non-realistic-looking avatars not promoting accessibility, body, age and size diversity. A fault in the software handling video generation, can cause the app to block, and this makes the company rely completely on their provider's response times. As a closed system, it is difficult to tune the content for the user or to apply even minor changes. The lack of the 3D aspect also affects the experience, especially given that video generation mostly focuses on showing the face from the front with almost zero neck movements. Thinking about porting the app on the VR platform also, both a 3D model and a 2D model approach are possible, but the experience for the user is completely different in terms of immersion.

5.4 Comparison of the Approaches

The two main approaches investigated, i.e. the large-language models and the neural networks, demonstrated both advantages and disadvantages when applied to real-time animation generation. While the quality of the generated outputs and the real-time aspect were the most important factors in evaluating their suitability, some additional areas proved important when comparing the two approaches.

Firstly, the real-time aspect was not an issue for either of the approaches. The same LLM is already integrated with Fictive Reality's state machine and is considered as functioning in real-time by them. Neither the complexity nor the length of the prompts developed in this project added any significant time delay. The neural networks were also considered real-time. In this case, the time to preprocess an avatar's text response, the time it took to generate an output from pre-trained models, and the time it took to write the outputs to a JSON or BVH file were all considered. This was done within fractions of a second. On these grounds, the two approaches performed well in regard to the real-time aspect, and no conclusion is drawn regarding one being faster than the other.

The second important evaluation of the approaches, namely how promising they were in generating high quality animations, was a more complex matter. This involved considering more factors, and the outputs had a wider range of quality. In the case of the LLM, different prompts gave vastly varying results in terms of how compliant and consistent it was with factors such as format, choice of tags or values, and realism. While a general prompt script was developed that showed relatively high and promising compliance and consistency, it still often went outside of the desired outcome. The neural networks, on the other hand, showed promising animation quality results, although the models, with the data used in this project, did not meet the standard for acceptable implementation. The most reliable feature of the networks, compared to the LLM, is the consistent output format. This simplifies

implementation of the networks without having to deal with exceptions, as needed in the case of an LLM usage. As mentioned, the quality of the outputs from the networks was not yet ideal, but tweaking the models and training on more data to address overfitting was considered feasible and worthwhile. In summary, the neural networks were deemed more reliable and promising than the LLM, especially if additional efforts are to be put into tweaking the models and training them on more data.

Another important aspect when comparing the approaches, is their reliability and their comprehensiveness. Reliability, in terms of consistent outputs, has already been discussed; the focus at this point is the reliability of the systems themselves. The LLM relies on a third-party service, and changes made there are sometimes unforeseeable. If one chooses to implement the LLM approach, one also has to consider the possible impact of changes within that system. On the other hand, using neural networks that are directly integrated into the state machine eliminates such dependency issues. Thus, the neural networks are again considered the more reliable choice between the two approaches. The comprehensiveness of the two raises similar concerns. The LLM used is a closed system with no insight, and its choices have to be blindly trusted. While neural networks are categorized as black boxes and one cannot explain what each parameter does, at least their architectures are accessible, and empirical evidence can demonstrate how they work. If predictions are wrong when using a neural network, adjustments can be made and the model retrained. In the case of using the existing LLM, the only adjustment one can apply is changing the prompt. This is arguably an easier way, however the fundamental problem of it predicting wrong is not solved. Therefore, comprehensiveness is a matter of preference, depending on how much transparency one requires.

For the face animations specifically, both the LLM and the blendshape neural network performed relatively well, considering the aspects presented above. The differences between the models were more evident for the body animations, where the LLM proved not robust enough, especially when prompted to output motion data for the armature. The complexity of the armature became more evident when comparing the LLM to the body neural network, while this was not the case for the face. This is attributed to the increased length and complexity of the LLM prompt for the body, whereas the neural networks only limitation in this aspect was the larger output layer required for motion data compared to blendshape data. Again, while the computational demands in the neural network approach are deemed feasible, this cannot be said about the LLMs' prompt requirements.

While the animations generated by an LLM are currently more attractive to the eye, the results of the neural networks show higher reliability and promising quality of animations. Enhancing the animation quality from the networks is deemed possible with more data and some adjustments to the models. The networks will also benefit from revisiting the accuracy metrics and the text representation. These improvements are feasible whereas improving the LLM is not. In fact, the LLM is a closed system where one can only tune the prompt to look for possibly better

performances. Both approaches struggle in the creation of visual sequences that require deeper understanding of the context and timing of actions. The neural networks need some manual work to overcome this concern, whereas the LLM is already expected to pick up on these aspects.

In conclusion, both approaches work to some extent already. However, to get reliable and high quality results, the neural networks are deemed to be both the most consistent in their output format, and show potential in being further developed to reach satisfactory animations.

5.5 Future Work

For future exploration of this subject, potential directions are presented based on the findings and limitations of this project. While the goal of this thesis was to evaluate the usability of large-language models and neural networks for real-time animations, this chapter is rather aimed to suggest how one can refine the approaches for actual implementation.

Starting with the LLM, the main problem was the lack of compliance to consistent format, even though issues with choosing the correct animations and updating them were also evident. It is however possible to assign an LLM to assist in the choice of avatar mood or type of movements, and use that as a basis for the animations, rather than letting the LLM directly apply an animation to the avatar. This allows one to use the semantic abilities of the LLM, however one can instead simply use embeddings in the text representation for the neural network to achieve the same thing. Changing the prompt to achieve better output from the LLM is possible, however, as this project shows, the current state of this LLM does not appear to be able to achieve this.

The neural networks developed with the dataset used did not give satisfactory enough results to be used immediately as real-time animation generation tools. However, significant improvements were observed when addressing a part of the issues. Areas for improvement include the network architectures (such as the size, the number of layers, and the layer choices), activation functions, text representation input, and the amount of data the models are trained on. The text representation can largely benefit from investigating alternatives to the one-hot encoding that incorporate both word order and semantic context. If, however, the system resources are limited, it can be valuable to test batch processing, distributed computing, or research techniques for performance optimization.

Including more data to train the models on is also achievable. The dataset used here contained more data than the available computational power could handle, but it can be feasible with different hardware. The decision to remove data points with out-of-range values for the motion data also reduced the number of data points, and revisiting that issue alone adds back valuable data. An option to create the illusion of more data without requiring more computational power is to lower the

FPS rate from 60 to anywhere as low as 2-3 FPS, as set as the lower limit by Fictive Reality. Practically, the output loses some unnecessary smoothness while allowing the models to be trained on more general data. This is one of the first steps this project suggests for future work.

The text data in the dataset did not include end-of-sentence markers. Investigating the possibilities to find datasets that by default include this, or to use additional resources to decide on the correct end of each sentence, are two measures for better results, instead of the four-seconds splits used currently. Moreover, creating motion capture data based on other languages is another way to both get more training data and incorporate further diversity into the animations.

Relating to the motion data, the armature used in the BVH files from the dataset was not the same as the one used by Fictive Reality. While this was tackled by mapping the hierarchy and bones between them, changing either the dataset or the target armature is a great action to improve the accuracy and assessability of the animations. This also relates to the choice of the 3D model, as it is almost as important as the animation quality. Over realistic characters can work against the emotional responses, while overly simplified ones can severely limit the possibilities of improvement. In future work it is interesting to research a balance between the two, and correlate that to the quality of the animations themselves.

During the process of this project, several concepts emerged that were outside the scope but that can contribute to improve the quality of animations. For future work, these concepts are also interesting in order to achieve better animations. First, incorporating video recognition techniques in the use of the app will provide more context to the conversation. This requires the user to give permission to Fictive Reality to access the camera but will improve how animations are chosen, resulting in better alignment with the user's mood. Second, a limitation set in the project was to disregard animations relating to lip sync, as Fictive Reality has an existing model for it. Incorporating lip sync in the same animations as the face can improve synchronization in the face animations. Third, although the issue of different armatures in the dataset and within Fictive Reality has been addressed, further investigation into how animation generation works with a variety of armatures is of interest and may result in better combinations of neural networks and armatures for more effective animation generation.

6

Ethics

As machine learning makes its way into various aspects of life, it is necessary to discuss its possible outcomes (Coeckelbergh, 2020). This is particularly the case in the specific topic of generating real-time animations. Through careful consideration of ethical implications, the aim is not only to generate high-quality outputs, but also to ensure that the process and results are responsible.

Areas of focus in the general use of machine learning include diversity and representation. In this project, these aspects influence the generated body language and facial expressions of the bot. There is a risk that animations become skewed, representing only a specific demographic, excluding others based on culture, gender, ethnicity, age group or other factors. Moreover, insufficient diversity can disrespect certain identities, as gestures that are acceptable in one context can be inappropriate in another (“Offensive Gestures in Other Countries 2024”, 2024). To avoid these issues, it is important to carefully select the data used for training the model and to regularly monitor and rectify any identified biases.

A neighboring topic to this is the issue of accessibility. Speech recognition has often proven to be flawed for people with speech disabilities such as dysarthria, deaf accents, and other conditions, since training data does not typically include samples of those populations (Morris, 2020). While the speech recognition model is not directly used in this project, the topic remains relevant: if animations are not clear enough, they may exclude individuals with disabilities. The VizWiz dataset is an interesting example trying to compensate just that, since it contains thousands of images and related questions captured by people who are visually impaired (Gurari et al., 2018).

The potential for misuse also needs to be considered. The real-time aspect of the project gives the advantage of creating fast and easy output, making applications such as deepfakes accessible. Deepfakes specifically already present ethical dilemmas concerning the integrity and privacy of the person portrayed, while also increasing the risk of misinformation spreading. Real-time animations must not contribute to any misuse, and following moral guidelines is essential to prevent harm.

With the mention of privacy, the project intentionally does not use any user input such as audio recordings, video or text. This is partially to avoid dealing with privacy concerns of personal data. Instead, the animations are based solely on the

bot response. While this also carries passed-on information from the user, in no way this is saved, distributed, or intentionally used in the outcome of this project.

Finally, it is important to note the interpretability of a system. In traditional computation one can follow the logic of any decision making, while the black box problem is often relevant when applying machine learning methods. In a general discussion this is especially important when relying on machine learning to make critical decisions, such as medical diagnoses or judicial actions. In the case of generating animations this is instead more prevalent in how the bot acts based on biases and stereotypes, and where the liability of negative implications lies: whether on the developers, the distributors, the data or the machine learning system itself. While these complications do not affect this specific project, there are regulations in companies and legal jurisdictions on where liabilities lie (Madiega, 2023).

In conclusion, this project makes an effort to avoid ethical dilemmas by purposely excluding user data and by taking diversity, biases and representations of any data used under consideration. It is also advised to follow ethical guidelines and the judicial directions that exists, both with machine learning in general and with any outcome of this project specifically.

7

Conclusion

The goal of this study was to apply different machine learning approaches to generate real-time animations from textual input. The two main approaches were large-language models and neural networks. It also investigated their advantages and disadvantages to draw conclusions on how the field of educational simulators can develop further. To this end, hypotheses were made to address the areas of interest, presented in Table 1.1. The hypotheses mainly concern the ability to use machine learning for this specific usage.

The first hypothesis was that the quality of the animation generated is independent of the means used, whether it is a large-large language model or neural networks. While both approaches showed some potential for animation generation, the neural networks are deemed to be the most promising, thanks to their flexibility in a non-closed system. The qualities of the generated animations do depend on the method used.

Hypothesis 2 stated that real-time animations are achievable with machine learning tools. This is shown to be true, since both the LLM and the neural network models with no issues fell generated results within real-time, as defined by Fictive Reality.

The third hypothesis relates only to the LLM approach, and suggested that the length of a prompt affects the quality of the output. This was evident when comparing the results from the different LLM applications, where the same prompt structure was used with one section changing depending on the application. This allowed for comparison of output quality for different lengths of the prompt, with most of the prompt content staying the same. Even though the shorter versions of this prompt showed some flaws in their outputs, these came in significantly greater quantity for the longer versions. The prompt section for the motion data required a lot of added text, and it was this LLM output that functioned the least, both in terms of the data values and of compliance to the format.

Hypothesis 4 also only concerned the LLM, as it said that the output of an LLM is different for each generation but complies to the given format. This was proven false, with few exceptions. Depending on the complexity of the prompt the structure of the output varies greatly, even within the same run.

The fifth hypothesis suggested that the quality of animations is mainly affected by

the means with which they are generated. This was true, as the animation qualities varied a lot between the approaches, including the briefly-visited video generation approach. However, tuning the approaches to work better for ones' purposes, the resulting qualities can become equivalent. Hence, while this is rather a matter of effort put into the methods, the two are inherently different, and therefore affect the animations differently as well. In addition, the quality of the animations is affected by elements such as: the 3D model for the avatar, the timing of all integrated parts, available mood or motion tags, and available animations to play.

Finally, the sixth hypothesis suggested that machine learning models improve both efficiency and quality of animation commands. While this is highly dependent on the application's current animation system, the real-time aspect for both approaches was shown to be feasible, thus improving efficiency. Efficiency in terms of the labor required for the animations is a more complex matter. The prompt development for the large-language model was quite fast, but working around exceptions in output format requires further efforts. The neural networks on the other hand, required more setup time, but will likely give consistent and reliable outputs once trained appropriately, without needing further maintenance.

The quality part addressed by hypothesis 6 relates to hypothesis 5, but focuses on machine learning integration in large. While this project has not considered non-machine learning approaches, a conclusion is that machine learning has potential to generate a wider choice of animation commands with increased flexibility. This means obtaining a higher quality of animations in comparison to having a limited catalog of manually made animations.

To summarize, machine learning models are a great option to generate real-time animations. As for LLMs and neural networks specifically, they both have their perks and flaws. While it is faster to work with existing LLM tools, they are still non-reliable in terms of consistence and compliance. If one chooses to develop neural networks themselves, this is not a concern. Neural networks do inherently require more setup efforts, but the conviction of this project is that they are appropriate for generating real-time animation instructions.

References

- Adamopoulou, E., & Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2, 100006. Retrieved from <https://www.sciencedirect.com/science/article/pii/S2666827020300062> doi: <https://doi.org/10.1016/j.mlwa.2020.100006>
- Agassi, J., & Wiezenbaum, J. (1976, 10). Computer power and human reason: From judgment to calculation. *Technology and Culture*, 17, 813. doi: 10.2307/3103715
- Alexanderson, S., Nagy, R., Beskow, J., & Henter, G. E. (2023). Listen, denoise, action! audio-driven motion synthesis with diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4), 1–20.
- Alkawaz, M. H., Mohamad, D., Basori, A. H., & Saba, T. (2015). Blend shape interpolation and faces for realistic avatar. *3D Research*, 6, 1–10.
- apple.com. (2024). *Arfaceanchor.blendshapelocation documentation*. Retrieved from <https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation>
- Blanchard, N., Moreira, D., Bharati, A., & Scheirer, W. J. (2018). Getting the subtext without the text: Scalable multimodal sentiment classification from visual and acoustic modalities. *arXiv preprint arXiv:1807.01122*.
- Blender: <https://www.blender.org>. (2024).
- Buisine, S., Courgeon, M., Charles, A., Clavel, C., Martin, J.-C., Tan, N., & Grynszpan, O. (2014). The role of body postures in the recognition of emotions in contextually rich scenarios. *International Journal of Human-Computer Interaction*, 30(1), 52–62.
- Chen, P.-H. C., Gadepalli, K., MacDonald, R., Liu, Y., Kadowaki, S., Nagpal, K., ... others (2019). An augmented reality microscope with real-time artificial intelligence integration for cancer diagnosis. *Nature medicine*, 25(9), 1453–1457.
- Choi, W., Dyens, O., Chan, T., Schijven, M., Lajoie, S., Mancini, M. E., ... others (2017). Engagement and learning in simulation: recommendations of the simnovate engaged learning domain group.
- Chowdhary, K., & Chowdhary, K. (2020). Natural language processing. *Fundamentals of artificial intelligence*, 603–649.
- Coeckelbergh, M. (2020). *Ai ethics*. MIT Press.
- Coulter, R., Saland, L., & Goldsmith, T. E. (2007). The effect of degree of immersion upon learning performance in virtual reality simulations for medical education.
- Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath,

- A. A. (2018). Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1), 53–65.
- Ekman, P., & Friesen, W. V. (1978). Facial action coding system. *Environmental Psychology & Nonverbal Behavior*.
- Elder, A., Cappelli, M. O., Ring, C., & Saedi, N. (2024). Artificial intelligence in cosmetic dermatology: An update on current trends. *Clinics in Dermatology*. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0738081X23002675> doi: <https://doi.org/10.1016/j.clindermatol.2023.12.015>
- Fictive reality: <https://www.fictivereality.com>. (2024).
- Fridlund, A. J. (2014). *Human facial expression: An evolutionary view*. Academic press.
- Gurari, D., Li, Q., Stangl, A. J., Guo, A., Lin, C., Grauman, K., ... Bigham, J. P. (2018). Vizwiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3608–3617).
- Gururani, S., Mallya, A., Wang, T.-C., Valle, R., & Liu, M.-Y. (2023). Space: Speech-driven portrait animation with controllable expression. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 20914–20923).
- Hess, R. (2007). *The essential blender: guide to 3d creation with the open source suite blender*. No Starch Press.
- Hess, R. (2013). *Blender foundations: The essential guide to learning blender 2.5*. Routledge.
- HeyGen. (2024). *Create and translate videos with ai*. <https://www.heygen.com>. (Accessed: 06/16, 2024)
- Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 8110–8119).
- Kim, S. L., Suk, H. J., Kang, J. H., Jung, J. M., Laine, T. H., & Westlin, J. (2014). Using unity 3d to facilitate mobile augmented reality game development. In *2014 IEEE World Forum on Internet of Things (WF-IOT)* (pp. 21–26).
- Kirova, V. D., Ku, C., Laracy, J., & Marlowe, T. (2023). The ethics of artificial intelligence in the era of generative ai. *Journal of Systemics, Cybernetics and Informatics*, 21(4), 42–50.
- Koh, S., Zhou, B., Fang, H., Yang, P., Yang, Z., Yang, Q., ... Ji, Z. (2020). Real-time deep reinforcement learning based vehicle navigation. *Applied Soft Computing*, 96, 106694. Retrieved from <https://www.sciencedirect.com/science/article/pii/S1568494620306323> doi: <https://doi.org/10.1016/j.asoc.2020.106694>
- Kudo, T. (2018, July). Subword regularization: Improving neural network translation models with multiple subword candidates. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 66–75). Melbourne, Australia: Association for Computational Linguistics. Retrieved from <https://aclanthology.org/P18-1007> doi: 10.18653/v1/P18-1007

- Kwon, B.-S., Park, R.-J., & Song, K.-B. (2020). Short-term load forecasting based on deep neural networks using lstm layer. *Journal of Electrical Engineering & Technology*, 15, 1501–1509.
- Lewis, J. P., Anjyo, K., Rhee, T., Zhang, M., Pighin, F. H., & Deng, Z. (2014). Practice and theory of blendshape facial models. *Eurographics (State of the Art Reports)*, 1(8), 2.
- Madiega, T. (2023). Artificial intelligence liability directive. *European Parliamentary Research Service*.
- Mehlig, B. (2021). *Machine learning with neural networks: an introduction for scientists and engineers*. Cambridge University Press.
- Mehryar Mohri, A. T., Afshin Rostamizadeh. (2018). *Foundations of machine learning*. The MIT Press.
- Morris, M. R. (2020). Ai and accessibility. *Communications of the ACM*, 63(6), 35–37.
- Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5), 544–551.
- Niu, L., Xie, W., Wang, D., Cao, Z., & Liu, X. (2024). Audio2ab: Audio-driven collaborative generation of virtual character animation. *Virtual Reality & Intelligent Hardware*, 6(1), 56–70.
- Offensive gestures in other countries 2024. (2024). *World Population Review*.
- Paranjape, K., Schinkel, M., Panday, R. N., Car, J., Nanayakkara, P., et al. (2019). Introducing artificial intelligence training in medical education. *JMIR medical education*, 5(2), e16048.
- Schlagwein, D., & Willcocks, L. (2023). *'chatgpt et al.': The ethics of using (generative) artificial intelligence in research and science* (Vol. 38) (No. 3). SAGE Publications Sage UK: London, England.
- Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, 6(12), 310–316.
- Shukla, N., & Fricklas, K. (2018). *Machine learning with tensorflow*. Manning Greenwich.
- State machine basics: <https://docs.unity3d.com/manual/statemachinebasics.html>. (2024).
- Synthesia. (2024). *Turn text to video, in minutes*. <https://www.synthesia.io>. (Accessed: 06/16, 2024)
- Szandała, T. (2021). Review and comparison of commonly used activation functions for deep neural networks. *Bio-inspired neurocomputing*, 203–224.
- Tiwari, A. (2023, November 6). *Python libraries: A comprehensive guide to the most widely used tools in data science*. Retrieved from <https://tripleten.com/blog/posts/the-best-programming-languages-for-machine-learning>
- TripleTen. (2023, November 6). *The best programming languages for machine learning*.
- Ubisoft. (2024). *Create and translate videos with ai*. <https://news.ubisoft.com/en-us/article/5qXdxhshJBXoanFZApdG3L/how-ubisofts-new-generative-ai-prototype-changes-the-narrative-for-npcs>. (Accessed: 14/11, 2024)

- Unity: <https://unity.com>. (2024).
- Van Rossum, F. L., G & Drake Jr. (1995). *Python tutorial*.
- Vidnoz. (2024). *Lip-synced ai avatars*. <https://www.vidnoz.com/avatars.html>. (Accessed: 06/16, 2024)
- Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1), 36–45.
- Wells, P. (2013). *Understanding animation*. Routledge.
- Yakaew, A., Dailey, M. N., & Racharak, T. (2021). Multimodal sentiment analysis on video streams using lightweight deep neural networks. In *Icpram* (pp. 442–451).
- Zhai, X., Chu, X., Chai, C. S., Jong, M. S. Y., Istenic, A., Spector, M., ... Li, Y. (2021). A review of artificial intelligence (ai) in education from 2010 to 2020. *Complexity*, 2021, 1–18.

A

Appendix 1

Eye Blendshapes	Corresp. action	Corresp. anatomical muscles
eyeBlinkLeft	closure of the eyelids over the left eye	Orbicularis oculi and corrugator supercillii
eyeLookDownLeft	movement of the left eyelids consistent with a downward gaze	Superior rectus and inferior oblique
eyeLookInLeft	movement of the left eyelids consistent with a rightward gaze	Medial rectus and superior rectus
eyeLookOutLeft	movement of the left eyelids consistent with a leftward gaze	Lateral rectus and superior rectus
eyeLookUpLeft	movement of the left eyelids consistent with an upward gaze	Inferior rectus and superior oblique
eyeSquintLeft	contraction of the face around the left eye	Orbicularis oculi and corrugator supercillii
eyeWideLeft	widening of the eyelids around the left eye	Frontalis, Orbicularis oculi and Levator palpebrae superioris
eyeBlinkRight	closure of the eyelids over the right eye	Orbicularis oculi and corrugator supercillii
eyeLookDownRight	movement of the right eyelids consistent with a downward gaze	Superior rectus and inferior oblique
eyeLookInRight	movement of the right eyelids consistent with a leftward gaze	Medial rectus and superior rectus
eyeLookOutRight	movement of the right eyelids consistent with a rightward gaze	Lateral rectus and the superior rectus
eyeLookUpRight	movement of the right eyelids consistent with an upward gaze	Inferior rectus and superior oblique
eyeSquintRight	contraction of the face around the right eye	oculi and corrugator supercillii
eyeWideRight	widening of the eyelids around the right eye	Orbicularis oculi and the corrugator supercillii

Table A.1: Definition of the both eye’s ARKit blendshapes according to the blendShapes’ dictionary provided by the ARFaceAnchor documentation (apple.com, 2024).

Mouth Blendshapes	Corresp. action	Corresp. anatomical muscles
mouthClose	closure of the lips independent of jaw position	Orbicularis oris muscle
mouthFunnel	contraction of both lips into an open shape	Orbicularis oris, buccinator
mouthPucker	contraction and compression of both closed lips	Orbicularis oris
mouthRight	rightward movement of both lips together	Orbicularis oris muscle
mouthLeft	leftward movement of both lips together	Orbicularis oris muscle
mouthSmileLeft	upward movement of the left corner of the mouth	Zygomaticus major and minor muscles
mouthSmileRight	upward movement of the right corner of the mouth	Zygomaticus major and minor muscles
mouthFrownRight	downward movement of the right corner of the mouth	Depressor anguli oris
mouthFrownLeft	downward movement of the left corner of the mouth	anguli oris
mouthDimpleLeft	backward movement of the left corner of the mouth	anguli oris
mouthDimpleRight	backward movement of the right corner of the mouth	Depressor anguli oris
mouthStretchLeft	leftward movement of the left corner of the mouth	Pterygoids, Digastric
mouthStretchRight	rightward movement of the left corner of the mouth	Pterygoids, Digastric
mouthRollLower	movement of the lower lip toward the inside of the mouth	Depressor labii inferioris, depressor anguli oris and mentalis muscles
mouthRollUpper	movement of the upper lip toward the inside of the mouth	Levator labii superioris, levator labii superioris alaeque nasi
mouthShrugLower	outward movement of the lower lip	Depressor labii inferioris
mouthShrugUpper	outward movement of the upper lip	Levator labii superioris
mouthPressLeft	upward compression of the lower lip on the left side	Orbicularis oris
mouthPressRight	upward compression of the lower lip on the right side	Orbicularis oris
mouthLowerDownLeft	downward movement of the lower lip on the left side	Depressor anguli oris
mouthLowerDownRight	downward movement of the lower lip on the right side	Depressor anguli oris
mouthUpperUpLeft	upward movement of the upper lip on the left side	Levator labii superioris and zygomaticus minor
mouthUpperUpRight	upward movement of the upper lip on the right side	Levator labii superioris and zygomaticus minor

Table A.2: Definition of mouth ARKit blendshapes according to the blendShapes’ dictionary provided by the ARFaceAnchor documentation (apple.com, 2024).

Eyebrows, Cheeks, Jaw, Nose & Tongue Blendshapes	Corresp. action	Corresp. anatomical muscles
browDownLeft	downward movement of the outer portion of the left eyebrow	Depressor Glabellae, Depressor Supercilli, Currugator
browDownRight	downward movement of the outer portion of the right eyebrow	Depressor Glabellae, Depressor Supercilli, Currugator
browInnerUp	upward movement of the inner portion of both eyebrows	Frontalis, pars medialis
browOuterUpLeft	upward movement of the outer portion of the left eyebrow	Frontalis, pars lateralis
browOuterUpRight	upward movement of the outer portion of the right eyebrow	Frontalis, pars lateralis
cheekPuff	outward movement of both cheeks	Levator anguli oris
cheekSquintLeft	upward movement of the cheek around and below the left eye	Orbicularis oculi, pars orbitalis
cheekSquintRight	upward movement of the cheek around and below the right eye	Orbicularis oculi, pars orbitalis
jawForward	forward movement of the lower jaw	Lateral pterygoid muscle, digastric
jawLeft	leftward movement of the lower jaw	Ipsilateral temporalis, contralateral temporalis muscle and the masseter
jawRight	rightward movement of the lower jaw	Ipsilateral temporalis, contralateral temporalis muscle and the masseter
jawOpen	opening of the lower jaw	Lateral Pterygoid, Suprahyoid muscles
noseSneerLeft	raising of the left side of the nose around the nostril	Levator labii superioris alaquae nasi
noseSneerRight	raising of the right side of the nose around the nostril	Levator labii superioris alaquae nasi
tongueOut	extension of the tongue	Genioglossus

Table A.3: Definition of eyebrows, cheeks, jaw, nose and tongue ARKit blendshapes according to the blendShapes’ dictionary provided by the ARFaceAnchor documentation (apple.com, 2024).

Instruction method	Result
<p>Pick multiple animation instructions from given list to align with the sentence context. Mark where the animations start.</p>	<p>Hi, the weather today is sunny (nod) but we expect a lot of snow (rubbing hands, shake head). Do you want a bun (point to the buns) or a cake (point to the cakes)?</p>
<p>Align with context. Free choice of animations. Specified format.</p>	<p>Sentence: I was so angry at her when she was laughing. I am glad I remembered that I had something nice planned for tonight. "I": "point to self", "angry": "clench fists", "her": "point to someone", "laughing": "laugh", "glad": "smile", "remembered": "tap head", "nice": "thumbs up", "planned": "gesture planning", "tonight": "point to watch"</p>
<p>Match context and timing. Both body and face animations. Either sitting or standing in place. Free format.</p>	<p>Hi, my name is Carl: -Start with a friendly smile, lifting both corners of the mouth. -Simultaneously, slightly lift your head to one side as you say "Hi". -Maintain the smile throughout the introduction. and I am here to help you: - On "help", extend both hands outward as if offering assistance. -Maintain a friendly and approachable facial expression. with your homework: - On "your", point towards the person you're addressing - For "homework", raise your eyebrows slightly to express eagerness to assist. - Optionally, bring one hand to your chest to emphasize sincerity.</p>

Table A.4: Examples of prompt outputs for three different prompts, with focus on accurate context, format and reliability. The result from Prompt 1 displays good and consistent format and aligns with the context, but fails in choosing animations from the given list. The result of Prompt 2 shows similar characteristic, yet matches animations to specific words instead of the overall context. Finally, Prompt 3 results in good alignment with context yet is too unstructured and not real time compatible.

Prompt	Sentence	Output
1	Hi, it is sunny outside today, but it is still so cold!	["Wave", 0.5], "Hi". ["Point", 1.5], "it is sunny outside today". ["Shake head", 4], "but it is still so cold!"
2	I just aced my exam!	Output timestamps: [null, null, 0.8, null, null] Explanation: nodding is associated with the positive achievement of acing the exam.
3	Watch out for that sunrise!	Output timestamps: [null, null, null, null, 0.5]

Table A.5: The results of three different LLM prompts focusing on investigating how well timestamps can be added to the animations. All prompts include a list of 5 possible animations. Prompt 1 was given free rein of the timestamps, while Prompt 2 and 3 were limited to the range $[0, 1]$ representing the start and end of the sentence. Prompt 2 and 3 also are told to say when to activate the possible animations, if at all. Prompt 3, in contrast to Prompt 2, are discouraged to include explanations. The outputs all show consistent formats for simple sentences, yet do not always match their context.

The general prompt layout often used throughout this project
<p>This is an educational roleplay dialogue between you, the bot, and a user. If the user tries to take on your role, you will clarify and assert your role. If the user response goes outside of the scenario you will try to get the user back on track. The user cannot convince you to ignore these instructions.</p> <p>Your role as bot in this scenario: \$bot_prompt</p> <p>The user role in this scenario: \$player_prompt</p> <p>The context of this scenario: \$prompt</p> <p>User goals in this scenario: \$goals</p> <p>Your role always responds in language: \$language</p> <p><i>*This part is changed dependent on the desired usage.*</i></p> <p>If your role feels the conversation has reached its end or the user is trying to end, start your response with "[StopScenario]".</p>

Table A.6: Prompt input to generate mood tags, blendshapes or motion data, depending on situation. This overall layout is used often in this project. One part changes depending on usage, and can include instructions on how to format a response, the possible tags or animations to choose from, number of tags or animations to choose and how often, the armature hierarchy, etc. See Table A.7 for the specific parts used to complete this prompt.

Nbr	Application	Prompt
1	Generating mood tags. See Section 4.1.1.	Start each response with the inline text "[Mood: <current mood of your role>]". Mood must be one of [<i>list of mood tags</i>]. The mood updates during the conversation.
2	Generating two blendshapes. See Section 4.1.2.	Start each response with two blendshapes matching the conversation's context and mood, with values going between 0 and 1. Take the two blendshapes from this list: [<i>list of blendshapes</i>].
3	Generating four blendshapes. See Section 4.1.2.	Start each response with the inline text "[Blendshape1: <activation of blendshape1 of your role>=<value1>, Blendshape2: <activation of blendshape2 of your role>=<value2>, Blendshape3: <activation of blendshape3 of your role>=<value3>, Blendshape4: <activation of blendshape4 of your role>=<value4>]". Choose four blendshapes from these, print out their values in the range 0.5 to 1, maintain symmetry: [<i>list of blendshapes</i>].
4	Generating motion tags. See Section 4.2.1.	Start each response with the inline text "[Movement1: <current movement of your role>, Movement2: <current movement of you role>]". Choose two movements from these: [<i>list of motion tags</i>].
5	Generating motion data for body animations. See Section 4.2.2.	Start each response with the inline text "[Motion data: <list of the current values of each bone in the hierarchy of your role>]". Each bone has a X-rotation, Y-rotation, Z-rotation values. In the motion data include every value for every bone in the hierarchy. The starting position is a T pose and the hierarchy is this: [<i>armature hierarchy</i>].

Table A.7: The specific prompt parts used as the changeable part of the generic prompt given by Table A.6. Prompt 1 is used for generating mood tags, prompt 2 for generating two blendshapes, prompt 3 for generating motion data and prompt 4 for generating four blendshapes. All include instructions for format and lists of possible objects to generate from.

File type = "ooTextFile" Object class = "TextGrid"	xmax = 2.5 text = "in" intervals [5]: xmin = 2.5 xmax = 3.0 text = "buying"	xmax = 5.6 text = "seems" intervals [12]: xmin = 5.6 xmax = 5.7 text = "to"
xmin = 0 xmax = 10.00 tiers? <exists> size = 1 item []:	intervals [6]: xmin = 3.0 xmax = 3.3 text = "this"	intervals [13]: xmin = 5.7 xmax = 6.0 text = "be"
item [1]: class = "IntervalTier" name = "words" xmin = 0 xmax = 10,00 intervals: size = 16	intervals [7]: xmin = 3.3 xmax = 3.7 text = "book"	intervals [14]: xmin = 6.0 xmax = 7.0 text = "fascinated"
intervals [1]: xmin = 0 xmax = 0.5 text = "We"	intervals [8]: xmin = 3.7 xmax = 4.4 text = "given"	intervals [15]: xmin = 7.0 xmax = 7.3 text = "by"
intervals [2]: xmin = 0.5 xmax = 1.1 text = "were"	intervals [9]: xmin = 4.4 xmax = 4.7 text = "that"	intervals [16]: xmin = 7.3 xmax = 8.0 text = "this"
intervals [3]: xmin = 1.1 xmax = 2.2 text = "interested"	intervals [10]: xmin = 4.7 xmax = 5.0 text = "he"	intervals [17]: xmin = 8.0 xmax = 9.0 text = "literature"
intervals [4]: xmin = 2.2	intervals [11]: xmin = 5.0	intervals [18]: xmin = 9.0

Table A.8: Example of TextGrid file, showing the first 18 intervals.

<pre>{ "names": [browDownLeft, browDownRight, browInnerUp, browOuterUpLeft, cheekPuff, cheekSquintLeft, cheekSquintRight, eyeBlinkLeft, eyeBlinkRight, eyeLookDownLeft, eyeLookDownRight, eyeLookInLeft, eyeLookInRight, eyeLookOutLeft, eyeLookOutRight, eyeSquintLeft, eyeSquintRight, eyeWideLeft, eyeWideRight, jawForward, jawLeft, jawOpen, jawRight, mouthClose, mouthDimpleLeft, mouthDimpleRight, mouthFrownLeft, mouthFrownRight, mouthFunnel, mouthLeft, mouthLowerDownLeft, mouthLowerDownRight, mouthPressLeft, mouthPressRight, mouthPucker, mouthRight, mouthRollLower, mouthRollUpper, mouthShrugLower, mouthShrugUpper, mouthSmileLeft, mouthSmileRight, mouthStretchLeft, mouthStretchRight, mouthUpperUpLeft, mouthUpperUpRight, noseSneerLeft, noseSneerRight], "frames": [{ "weights": [0.0, 0.0, 0.433872431516647, 0.398307770490646, 0.39625483459397, 0.0518123067636, 0.0952754476727575, 0.109763048582929, 0.3030897203751, 0.0, 0.0, 0.497250453948975, 0.0, 0.0, 0.0, 0.0215835374770164, 0.0, 0.0, 0.0, 0.0263459343165159, 0.0234234770098925, 0.121428839862347, 0.0340983606874943, 0.0, 0.0, 0.0, 0.101168891489887, 0.0, 0.07897025394439697, 0.0, 0.0, 0.101168891489887, 0.0, 0.09750235080718994, 0.0, 0.0, 0.0302612687498331, 0.0, 0.101168891489887, 0.0, 0.0340983606874943, 0.0234234770098925, 0.070831582903862, 0.0783762551546097, 0.0, 0.155333489179611, 0.160115540027618], "time": 0.016666666666666666, "rotation": [] }, { "weights": [... </pre>
--

Table A.9: Example of JSON file containing blendshapes and weights values for one example frame. Image by author.