CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Generating Character Animation for the Apex Game Engine using Neural Networks

Implementing immersive character animation in an industry-proven game engine by applying machine learning techniques

Master's thesis in Computer science and engineering
for a degree at MPIDE - Interaction Design and Technologies, MSc.

JOHN SEGERSTEDT

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2021

Generating Character Animation for the Apex Game Engine using Neural Networks

Implementing immersive character animation in an industry-proven game engine by applying machine learning techniques

JOHN SEGERSTEDT

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Generating Character Animation for the Apex Game Engine using Neural Networks
Implementing immersive character animation in an industry-proven game engine by applying machine learning techniques

JOHN SEGERSTEDT

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

MENTOR:
Andreas Tillema, Avalanche Studios Group

INITIAL SUPERVISOR:
Marco Fratarcangeli, Department of Computer Science and Engineering

SUBSTITUTE SUPERVISOR:
Palle Dahlstedt, Department of Computer Science and Engineering

EXAMINER:
Staffan Björk, Department of Computer Science and Engineering.

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Generating Character Animation for the Apex Game Engine using Neural Networks
Implementing immersive character animation in an industry-proven game engine by applying machine learning techniques

JOHN SEGERSTEDT

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The art of machine learning, here using neural networks to map pairs of inputs to outputs, has been greatly expanded upon recently. It has been shown to be able to produce generalizable solutions within multiple different fields of research and has been deployed in real-world commercial products. One of these research areas in which regular scientific achievements are made is game development, and specifically character animation. However, compared to other fields, even though there has been much work on applying machine learning techniques to character animation, few efforts have been made to applying them in real-world game engines. This thesis project aimed to research the applicability of one such piece of previous work, within the proprietary Apex game engine. The final results included an in-engine solution, producing character animation purely from a predicative phase-functioned neural network. Additionally, several different network configurations were evaluated to compare the impact of using, for example, a deeper network or a network that had trained for a longer period of time, in an attempt to investigate potential improvements to the original model. These alterations were shown to have negligible positive impacts on the final results. Also, an additional network configuration was used to investigate the applicability of this approach on an industry-used skeleton, producing promising but imperfect results.

Keywords: machine learning, phase-functioned neural network, locomotive character animation, Avalanche Studios Group, Apex, thesis

Generating Character Animation for the Apex Game Engine using Neural Networks
Implementing immersive character animation in an industry-proven game engine by applying machine learning techniques

JOHN SEGERSTEDT

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Acknowledgements

First and foremost, the author wants to show their appreciation for Andreas "Andy" Tillema for being an incredibly supportive mentor who has, especially during the later stages of this project, been a pillar of support. Without Andys guidance, this project would not have been possible.

Secondly, the two supervisors Marco Fratarcangeli and Palle Dahlstedt deserves thanks for their support and feedback; Marco for initial project scope and Palle for aiding in the process of turning a programming implementation into an actual thesis.

Thirdly, there have been multiple Avalanche employees that have offered their aid to this project. Some of these that deserve praise include Robert "Robban" Petterson, for helping with the .bvh retargeting, and Preeth Punnatjanath, for helping with .bvh to 3D model skinning. The later of which, able to provide aid on a short notice, even when tackling other deadlines.

John Segerstedt, Gothenburg, June 2021

Contents

1

Introduction

1.1 Background

The domain of machine learning techniques applied within video game development has been greatly expanded during the last few years with multiple AAA-level publishers funding machine learning dedicated departments. Amongst others, these include Ubisoft La Forge funded by Ubisoft Entertainment [1], and SEED funded by Eletronic Arts Inc [2]. Additionally, there has been great achievements within academia on this topic, such as the research done at the University of Edinburgh by Sebastian Starke, Daniel Holden, and others [3].

Within the specific sub-domain of generated character animation, considerable achievements in research has been made by some of the aforementioned entities, much of which published as recently as during the year 2020, see Section 2.2.

One of the reasons behind the new additions of machine learning within character animation is the need of automation for managing potentially hundreds of thousands of animation clips; as the demand for variation and fidelity, and also more adaptive and life-like animations, has increased, there has been an exponential demand in the number of animations within newer game titles. This great escalation of the problem space can be exemplified when there is an expectancy of animations that adapt to external factors, such as uneven terrain. Otherwise, the lack of more context-specific animations may lead to the players' sense of immersion being broken. By using scalable and context-free machine learning techniques to generate more environmentally feasible animations, players may be kept more emerged into the gameplay experience without having to manually link seemingly endless number of animation clips and states.

As a result of the video game industry's immense size, there is a myriad of stakeholders to potential revolutionary and commercially viable innovations using the newly emerging techniques within machine learning; amongst others within research, development, publishing, and consumption, of video games.

Within the context of this thesis, the video game development studio Avalanche Studios Group [4] is a direct stakeholder to the outcome of this project as a result of their direct collaboration with the project.

1.2 Research Problem

The aim of this master's thesis is to investigate the possibility of generating realistic character animations using a predicative neural network trained on previously captured animation data. This is to be achieved utilizing phase-functioned neural networks *(PFNN)*, based off of previous research by Holden et al. [5].

Additionally, this master thesis aims to contribute to the research field by developing an implementation solution within the Apex Game Engine [6], contrary to previous research. This proprietary game engine will be provided by the Avalanche Studios Group [4] for use within this thesis.

1.3 Research Question

Main research question:

- How can the applicability of a Phase-Functioned Neural Network approach for generating real-time locomotive character animation in modern game engines be further improved?

To answer the wicked problem that is the main research question, this thesis aims to investigate the following subsidiary questions:

- *Responsivity* - How much computational time is required for procedural single-character locomotive animations, in a industry-proven game engine, on consumer-grade hardware?
- *Accuracy* - How accurate are the generated locomotive character animations, in an industry-proven game engine, to the original animation data?
- *Architecture* - How can the phase-functioned neural network architecture presented in Holden et al. be improved?

Additionally, analysis and comparisons will be made between the quantitative results during the responsivity and accuracy research of the following networks, see Section 3.1.3:

- *Holden - Default*
- *Holden - Extra Trained*
- *Holden - Extra Layer*
- *Avalanche*

Also, a summary of some of the most important learnings produced by this thesis project will be written, see Section 6.3.

1.4 Scope

NETWORK TYPE
As of its simplicity, which is further discussed in Section 2.2, a phase-functioned neural network was selected as the chosen network architecture for this project.

NETWORK ARCHITECTURE
To answer the subsidiary research question regarding network architecture, and as a clear delimitation of the scope of this thesis project, comparisons will be made solely between the list of network archetypes listed in Section 1.3.

PREVIOUS RESEARCH
Since the phase-functioned neural network architecture was developed by Holden et al. at the University of Edinburgh [5], the publicized network pipeline and accompanying motion data will be used as a basis for this project. This previous research is based on a left-handed world-axis orientation, see Section 3.3.4. The motion capture data is of the .bvh filetype, see Section 2.3.1.

GAIT STYLES
To limit the space of character animations, only standard bipedal locomotive character animations are to be considered. Therefore, only animations such as walking, jogging, crouching, and strafing, are to be considered. Similarly, interactions with advanced terrain and environments, such as balancing on elevated narrow beams and dynamic crouching beneath low ceilings will not considered. As such, these movement styles will be obsolete from the responsivity evaluation, see Section 1.3. However, motion capture files associated with these movements will still be part of the data set for the evaluation process, see Section 1.3.

GAME ENGINE
This thesis will evaluate the feasibility of procedural animations only within the Apex game engine, provided by Avalanche Studios Group. This engine uses a right-handed world-axis orientation, see Section 3.3.4.

CONFIDENTIALITY
As a result of the Apex game engine being proprietary software solely used in-house, a certain level of discretion is required by Avalanche Studios Group. This includes, but is not limited to, potential omission of engine-specific details from the final report.

PHASE FUNCTION COMPUTATION
Out of the three methods of computing the phase variable, as presented in Holden et al. [5], only 'constant' is to be considered during this thesis project. This, in an effort to reduce the number of permutations of network settings.

TERRAIN AND INCLINATION
The original demonstration application produced by Holden et al. [5] uses a static heightmap for terrain height sampling. As this is not the case for the Apex game engine, the phase-functioned neural network implementation integration as part of this project will assume a fully flat terrain during runtime.

HARDWARE

The entire evaluation process, and the network training, will be limited to be performed on a single set of hardware specifications:

- *CPU* - Intel i7-8700k @ 3.70GHz
- *GPU* - NVIDIA GTX 1060 6GB
- *RAM* - 16.0GB

2

Theory

2.1 Artificial Neural Networks

This section provides an introduction to artificial neural networks, the machine learning model used for mapping input features to output targets by updating network weights and biases.

The concept of an artificial neural network is based off the human brain; given a sensory input, and as internal energy levels surpasses specific thresholds, synapses are fired between neurons connected in a graph network.

A neural network can be trained to model arbitrary input-output relationships using either:

- *Supervised learning* - Comparing network outputs to a ground truth; for an example, used for image and speech recognition.
- *Unsupervised learning* - Attempts to minimize a given error measure as no specific ground truths are given; for an example, used for clustering and classification.
- *Reinforcement learning* - Traverses the solution space by being provided intermediary encouragement and punishment given specific state spaces; for an example, used for self-driving vehicles.

For this thesis project, and for the rest of this theory segment, supervised learning is the considered context.

2.1.1 Network Layers

A neural network can be modelled as a feed-forward directed acyclical graph with multiple connected layers, see Figure 2.1.

When a neural network is given sensory input, this data is feed into the input layer. Then, through evaluating the outputs of each layer given its predecessor's, see Section 2.1.2, the resulting evaluation of the network is produced in the output layer. A network can have any number of hidden layers between the input and the output layers, and any different number of nodes in each layer.

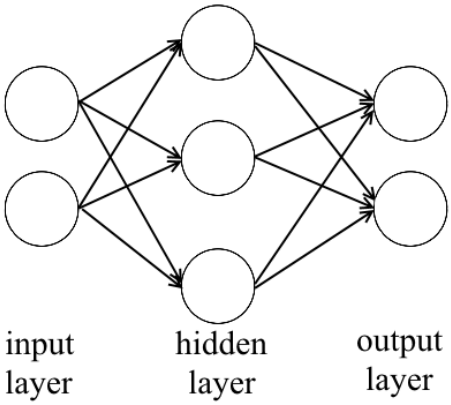

Figure 2.1: Simplified fully connected neural network model

Within the research field of machine learning, there are different network architectures consisting of other types of network layers than the fully connected layer type shown in Figure 2.1. Some other network types and examples of their usage are:

- *Recurrent Neural Networks* - By allowing for cyclical node connections, information is able to be passed and remembered between iterations. Used in text recognition and translation, amongst other fields.
- *Convolutional Neural Networks* - Through using feature convolving kernels that sequentially read subsets of the input, pattern recognition can be performed independent of the location of that pattern within the input data. Used in image recognition, amongst other fields.

2.1.2 The Mculloch-Pits Neuron

Named after its founders, the smallest component of a neural network is the single Mculloch-Putts neuron [7]. Such a neuron, see Figure 2.2, produces an output given an internal threshold and the weighted inputs of other neurons.

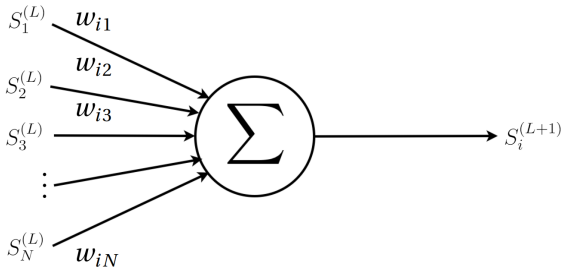

Figure 2.2: Simplified Mculloch-Pitts neuron model

Consider Equation 2.1; to evaluate the output signal of a Mculloch-Pitts neuron, one firstly considers the local field $b_i^{(L+1)}$ and inputs it into an activation function g. Popular activation functions include the ReLU ($= max(0, b_i)$) and the Sigmoid

$(= \frac{1}{1+e^{-b_i}})$ functions [8].

$$S_i^{(L+1)} = g(b_i^{(L+1)}) = g(\sum_j w_{ij}^{(L+1)} S_j^{(L)} - \theta_i) \tag{2.1}$$

where:

- g is the activation function of the neuron
- w_{ij} is the weight scalar from neuron j to neuron i
- $S_j^{(L)}$ is the output of neuron j in the previous layer L
- θ_i is the bias/threshold of neuron i

2.1.3 Supervised Learning of a Neural Network

By adjusting the weights and biases, a network can map any given set of input features X to any specific output Y. These are often referred to as pairs of input and output vectors.

To achieve this, a training process such as the following is performed:

Simplified training algorithm of a neural network

1. Split the data into two sets: training and validation
2. Initialize the network with random weights and biases
3. Use backpropagation to train the network using the training data according to the algorithm below *(= an 'epoch')*
4. Evaluate the accuracy of the network by performing complete prediction of all data points in the validation set, see Section 2.1.5
5. If the validation accuracy is increasing, according to some heuristic, go to step 3. Otherwise, terminate training *(= 'early stopping'), see Section 2.1.4*

Backpropagation is the technique of using the chain rule [9] to compute the update for weights backwards through the network, using the computed error in the output layer.

This is done by performing the following algorithm:

Backpropagation algorithm for a neural network

1. Forward propagate the input through the network
2. Calculate the output error using the difference to the ground truth
3. Propagate the errors back through the network
4. Update the weights using the backpropagated errors

5. Update the biases using the backpropagated error

equivalent to:

1. $S_i^{(L)} \leftarrow g(\sum_j w_{ij}^{(L)} S_j^{(L+1)} - \theta_i^{(L)})$, for all neurons i in layers L
2. $\delta_k^{(O)} \leftarrow g'(b_k^{(O)})(y_k - S_k^{(O)})$, for all neurons k in output layer O
3. $\delta_i^{(L)} \leftarrow \sum_j \delta_i^{(L+1)} w_{ij}^{(L+1)} g'(b_j^{L})$, for all neurons i in non-output layers L
4. $w_{ij}^{(L)} \leftarrow w_{ij}^{(L)} + \eta \delta_j^{(L)} S_i^{(L-1)}$, for all neurons i in layers L
5. $\theta_i^{(L)} \leftarrow \theta_i^{(L)} - \eta \delta_i^{(L)}$, for all neurons i in layers L

where $\eta \in (0, 1)$ is the learning rate of the network and which may decay during the training process. The learning rate, and other parameters such as number of epochs or the batchsize, are collectively referred to as hyperparameters.

2.1.4 Underfitting and Overfitting

When training a neural network, or when performing other types of regression model fitting, the choice of model complexity may give rise to issues as a result of a complexity level too low or too high.

Consider Figure 2.3, here one can observe the same data points and three different regression models. The optimal model for these types of problems is that which can most accurately represent the data distribution, and therefore can most precisely predict future data points belonging to the same data set. In the figure, the left model suffers from underfitting, wheras the right model suffers from overfitting.

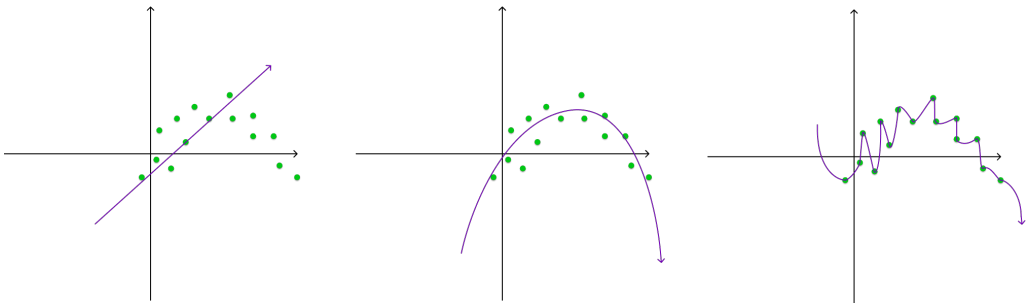

Figure 2.3: Simplified example of under/overfitting in 2D regression
Left: Underfitting, model fails to accurately represent data.
Middle: Optimal fit, model mimics the sampled distribution.
Right: Overfitting, model fails to generalize observations

In the previous example shown in Figure 2.3, the polynomial degree of a regression model was shown to be directly relating to any potential underfitting or overfitting. However, when it comes to neural network training, the complexity of the model is already decided upon previous to the training session. As such, the analogous parameter for neural network training is instead the number of training iterations.

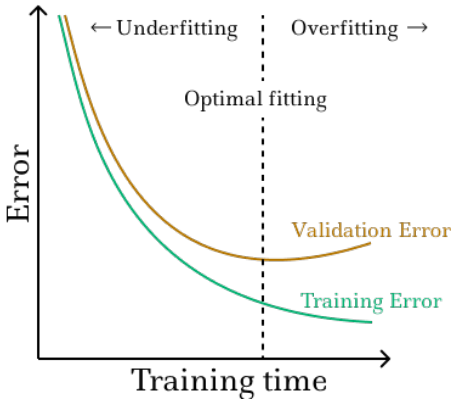

Figure 2.4: Simplified example of early stopping
As the prediction error on the training data reduces during the training process, the prediction error on the validation data initially decreases. After some time the prediction error starts to increase as generalizability is lost.

Consider Figure 2.4; here a neural network is trained repeatedly using a training data set of input and output vector pairs, see Section 2.1.3. A trivial expectancy of such a training process is that the prediction error on the training data set steadily declines as the training, often counted in the number of epochs, proceeds. However, to prevent the potential loss of generalization of the network, a separate validation data set is used. The network is at no point allowed to learn from, or update its network weights and biases in response to, being shown the validation data set. Instead, this data set is solely used to estimate the prediction qualities of the network given unseen data.

In other words, the calculated error of the network predictions on the validation data set is considered to be proportional to the generalizability of the network. As such, the ideal performance of the network is when the validation error reaches a minimum, at which the training process should terminate. This is visualized in Figure 2.4, where halting training before the validation error starts to increase leads to potential underfitting, and halting post this minima leads to overfitting.

2.1.5 Gradient Descent

The backpropagation algorithm presented in Section 2.1.3 attempts to minimize the prediction error by updating each weight and bias with regards to the function derivative of the error function with respect to each variable respectively.

Conceptually, one can visualize this process by using Figure 2.5, which shows how the prediction error of a neural network is directly dependent on the values of the weights and biases. Then, as a training process includes initializing random weights and biases, consider a marked spot at a random starting point on the line. From there one, through gradient descent during the network training process, that marked spot will move downhill as the network weights and biases are updated.

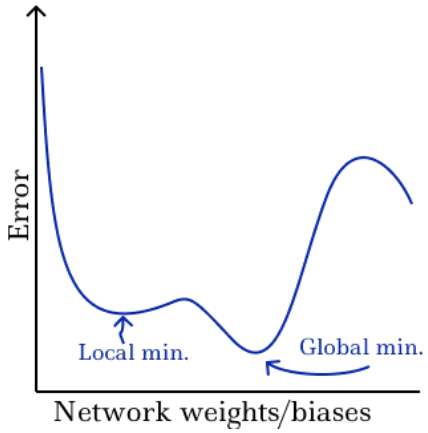

Figure 2.5: Simplified example of prediction error depending on weights/biases This model is heavily simplified. In actuality, a more realistic representation has one dimension for each weight value and for each bias value.

However, through using true gradient descent on an unknown error landscape, one might get stuck in local minimas. This can be visualized in Figure 2.5 if one were to initialize a network configuration close to the noted local minima. To avoid this serious issue, one introduces even more randomness to the network training procedure. As such, if we allow the network solution to occasionally move in an opposite direction of the error function gradient, a weights and biases configuration may escape a local minima.

This introduction of further randomness in the network training process can be done by updating the weights and biases after only having seen a randomly chosen subset, called a batch, of the training data set. This is called batch training and the technique of adding further randomness to the model training is referred to as stochastic gradient descent.

2.1.6 Adam Optimizer

The Adam optimization algorithm [10] [11] is an extension to stochastic gradient descent, see Section 2.1.5, first presented in 2015, aiming to increase efficiency during neural network training.

The most impactful difference between the Adam optimization algorithm and standard stochastic gradient descent is where the latter uses a single learning rate for all trainable parameters, the former uses individualized learning rates for each parameter that may decay individually. This decay is controlled through the hyperparameters beta1 and beta2.

2.2 Related Work

Recently, advancements in machine learning using deep neural networks has been made in a number of various fields:

Since ImageNet [12] 2015, the annual image recognition algorithm competition, competitors have been able to produce machine learning solutions that outperform humans in classifying photographs of objects [13].

In 2020, the AlphaFold agent developed by the Deepmind team, funded by Google, was able to make accurate predictions of protein shapes based off its sequence of amino acids, potentially beeing able to "accelerate research in every field of biology" [14].

The tech giant Google is using machine learning techniques for a multitude of their online services, such as: busyness metrics for public areas, individual passage indexing on webpages, song/music identification, breaking news detection, and language translation [15] [16].

Within gaming specifically, there has been multiple recent breakthroughs:

In 2016, the machine learning agent AlphaGo, produced by the Deepmind team, defeated Lee Sedol, winner of 18 world titles, in the board game of Go [17]. The achievement lead to AphaGo officially being the first ever computer agent being rewarded the highest ranking certification within the sport; 9 dan [17], a large step forward from the narrow matches between IBM's Deep Blue and Garry Kasparov in the much less complex game of Chess in 1997.

In a similar vein after AlphaGo's triumph in Go, computer games has become a new focus for deep learning agents. One of these is AlphaStar, also developed by Deepmind, which in 2019 became the first AI to achieve the highest ranking within a widely popular esport title without any game restrictions in the computer game of StarCraft II [18].

Another such agent is OpenAI Five, developed by the OpenAI team and funded amongst others by Elon Musk, which also in 2019 achieved both expert-level performance and human-AI cooperation in the computer game of Dota 2 [19].

However, world class competition is not the only use for neural network agents. Multiple games have officially launched with either fully, or partially, machine learned agents, such as the A.I. that players can play against in the game Planetary Annihilation [20]. Another example, where machine learning is only partially used, is the threat response, the fight-or-flight reaction, of the A.I. agent in the game Supreme Commander 2 [21]. Additionally, similar agents have been developed to the benefit of the game designers and developers, for reasons such as gameplay balancing and strategy win predictions [22].

Other uses of machine learning techniques to increase productivity, generalizability, or efficiency of the game development process include the field of procedurally generated game content. Applicable areas were machine learning techniques have been

applied such research include: level design, text and narration, music and sound effects, model textures, and character animations [23].

Some other specific examples of applied machine learning applied during game development is much of the research done at the machine learning research division Ubisoft La Forge [1] who have conducted research on, amongst others; 3D character navigation [24], motion in-betweening for character animations [25], data-driven physics simulations [26], automatic code bug detection [27], and motion capture data denoising [28].

As this report is on the topic of neural network generated locomotive character animations, the rest of this section is dedicated to presenting different advancements within this field and to, where relevant, relate their respective strengths and weaknesses in contrast to that of phase-functioned neural networks.

The following research is presented chronologically.

2.2.1 Phase-Functioned Neural Networks for Character Control (2017)

In this paper, Holden et al., at the University of Edinburgh, introduces a single network architecture for generating locomotive character animation over rough terrain using a phase function [5].

At each frame during runtime, this neural network takes as input the current pose of the character, any potential user input, and information of specific sample points along the ground ahead and behind the character, to produce the next character pose. This character pose consists of information regarding each joint within the character model, such as their position and orientation.

To accurately model the cyclical nature of the human walk, a phase variable producing function is introduced. This variable models the transitions between the contact of each of the two feet of the character with the ground. This ensures a perpetual forward animation of the character, stepping with each foot sequentially.

For a more in-depth presentation of the phase-functioned neural network used in this report, see Section 3.2.

2.2.2 Mode-Adaptive Neural Networks for Quadruped Motion Control (2018)

This paper, authored by researchers at the University of Edinburgh, introduces a dual neural network system for generating locomotive character animation for quadrupeds [29].

The first of these two networks is the motion prediction network, highly similar to the neural network used in a phase-functioned neural network, see Section 3.2. The second, a gating network that outputs blending coefficients that are used as inputs in the motion prediction network similar to the phase variable of a phase-functioned

neural network, see Section 3.2.

By controlling the motion weights of the motion network using another generative neural network, one trades manually labeling the phase of the motion training data for the requirement of training this separate gating network. This, however, is necessary for quadrupeds as the cyclical leg movement of these is heavily dependent on gait styles and cannot be modelled with a single phase variable [29].

Result-wise, the mode-adaptive neural networks approach achieve more realistic quadruped motion for flat terrain than that of a phase-functioned neural network [30]. However, direct comparisons in memory footprint or computational complexity, for either flat or rough terrain, was omitted from the original paper.

2.2.3 Neural State Machine for Character-Scene Interactions (2019)

Compared to other research presented here, this paper, also from the University of Edinburgh, specifically focuses on the interaction between a character and scene objects, such as opening doors, sitting on chairs, and lifting and carrying boxes [31].

For the training data, specific motion capture clips of the supported object interactions were recorded and a set of control points were manually labeled, such as the armrests of an interacted with chair. A data augmentation scheme was then used to generate a 16GB training set from the initial data.

At runtime, the state machine can transition and blend between animation modes such as walk, sit, open, carry, etc, triggered by user input. Additionally, this system is fed not solely the pose of the character and specific control points along the trajectory of the character, but also the geometry of the nearby surroundings through voxelization.

Contrary to the neural state machine which was built for this specific purpose, a phase-functioned neural network produces unsatisfactory and jittery results when attempting to produce character animation of scene object interactions [32]. However, for strictly locomotive tasks, a phase-functioned neural network produced comparable results in areas such as foot sliding and response time [31].

Additionally, the neural state machine presented in this research includes two different conjoined neural networks, similar to the system structure presented in Section 2.2.2, and one of which includes a phase variable similar to that of the compared to solution. Also, the comparatively massive training data adds considerably longer training time than that of a phase-functioned neural network [31].

2.2.4 Local Motion Phases for Learning Multi-Contact Character Movements (2020)

This paper, authored by researchers at Electronic Arts and the University of Edinburgh, presents the concept of local motion phases and shows successful applications

of this concept both within new fields of animations and within the context of previous research from the university [33].

The local motion phase is conceptually similar to the phase variable in a phase-functioned neural network, see Section [5]. However, rather than modelling the locomotive movement of an entire character with a single phase variable, local motion phases are automatically calculated and maintained on a per-bone basis. This allows for more realistic animations during highly detailed movements than that which can be generated by a phase-functioned neural network [34].

At its core, the system presented in this paper has a similar dual network structure to those mentioned in Section 2.2.2 and Section 2.2.3. However, in addition to the inclusion of the local motion phases, this paper introduces an autoencoder for the user input. This generative control model encodes and decodes user input at runtime, pretrained on the motion capture data.

This approach has been shown to produce high quality animations for tasks such as lifting boxes similar to those presented in Section 2.2.3, playing basketball, and for quadruped movement similar to those presented in Section 2.2.2. However, as this approach builds upon multiple advanced concepts, its final network structure is substantially more intricate than that of the phase-functioned neural network.

2.2.5 Learned Motion Matching (2020)

Traditional motion matching [35] consists of a system that regularly fetches the most appropriate pre-processed animation from an animation database, given a set of pose features, for a specific character. Such a database consists of highly structured and non-overlapping directional movement, as to minimize the number of recorded animation sequences. This, as the memory footprint of a motion matching system scales linearly with the size of the database as the latter must be kept fully in memory during runtime.

Learned motion matching [36], however, is a technique presented by Ubisoft La Forge [1] that introduces a neural network approach to motion matching that removes direct dependency on an animation database. Performance-wise, although a learned motion matching system is able to produce indistinguishable results to that of a traditional motion matching system with a substantially smaller memory footprint, it does so requiring considerably longer computational time [36].

Compared to using a phase-functioned neural network, a learned motion matching network uses slightly less memory and significantly less computational time at runtime while requiring an incredibly shorter training period [36] and while being able to produce similar or better results [37].

Although, a learned motion matching system does not require phase-labeling, it instead demands to be trained on a meticulously constructed database of encompassing motion capture data. Additionally, compared to a phase-functioned neural network, a motion matching system requires three distinct networks, each with their own features, targets, and error functions.

2.3 File Types & Software Libraries

This section aims to present relevant file types and software libraries used within this project, and to provide a brief introduction on how to use them.

2.3.1 The .bvh filetype

The .bvh, Biovision Hierarchy, filetype can be used to store motion capture data. These are human-readable text files, containing both a structural definition for the motion capture joint skeleton and the per frame joint data.

```
HIERARCHY
ROOT Hips
{
  OFFSET 0.000000 0.000000 0.000000
  CHANNELS 6 Xposition Yposition Zposition Zrotation Yrotation Xrotation
  JOINT LeftLeg
  {
    OFFSET 1.000000 -1.000000 1.000000
    CHANNELS 3 Zrotation Yrotation Xrotation
    End Site
    {
      OFFSET 0.000000 0.000000 0.000000
    }
  }
  JOINT RightLeg
  {
    OFFSET -1.000000 -1.000000 1.000000
    CHANNELS 3 Zrotation Yrotation Xrotation
    End Site
    {
      OFFSET 0.000000 0.000000 0.000000
    }
  }
}
MOTION
Frames: 3
Frame Time: 0.008333
2.801100 17.851100 -0.421913 -0.943466 0.030603 6.685755 -1.889587 17.864721 4.969343
-14.847416 -7.065584 -13.249440 1.025640
2.800815 17.848850 -0.421355 -0.932144 0.000126 6.685797 -1.904051 17.868369 4.931144
-14.870445 -7.095567 -13.284463 1.082333
2.800560 17.846750 -0.420267 -0.919127 -0.037791 6.682931 -1.919381 17.879059 4.896097
-14.894115 -7.119117 -13.317921 1.137596
```

Figure 2.6: Simplified .bvh file example

Consider the simple .bvh example in Figure 2.6.

Firstly, a 'HIERARCHY' of skeleton nodes are defined by name and parent offset. Additionally, each joint has a number of 'CHANNELS' associated with them. In this example, a skeleton of three joints is defined: a parent 'Hips' joint with the two children joints: 'LeftLeg' and 'RightLeg'.

Lastly, a .bvh file features a 'MOTION' section where the per frame motion captured data is listed. In order, each floating point value here corresponds to one of the 'CHANNELS' specified in the previous section. Each new row of data corresponds to a new frame. Joint orientations are stored in degrees, within the interval $(-180, 180]$.

Software, such as Blender, can import .bvh files and render the motion applied to the skeleton, as defined within the .bvh.

2.3.2 Theano

Theano [38] is a Python library built on top of Numpy [39] for efficient multi-dimensional array computations on the GPU. Theano was initially released in 2007 and further development on the project was shut down in 2017 [40].

Usage of Theano is done by creating 'theano.function':s that specify both input/output parameters and the actual operation to perform. For a simple example of Theano code, see Figure 2.7. However, Theano can be used for much more complex computations, such as neural network training, by passing the error function and learning rate update to the 'theano.function' function.

```
import theano
from theano import tensor

a = tensor.dscalar()
b = tensor.dscalar()

c = a + b
f = theano.function([a,b], c)

print(f(0.5, 1.5))
```

Figure 2.7: Simplified Theano code snippet for addition on the GPU
The code snippet is expected to print '2' to the console after performing an addition of the scalars 0.5 and 1.5 on the GPU.

2.3.3 Eigen

Eigen [41] is a C++98 library used to perform high-speed matrix and array operations. Eigen was first released in 2006 and has since been used in the creation of other software libraries, such as TensorFlow [41] [42].

In Eigen, one can define both statically-sized and dynamically-sized matrices and arrays. However, arithmetic inter-data type operations between matrices and arrays are not allowed, requiring users to cast objects between the types at runtime.

For a simple example on how to use Eigen arrays to represent a single-layed neural network, see Figure 2.8.

```
Eigen::ArrayXf W0;
Eigen::ArrayXf b0;
Eigen::ArrayXf Y;


void PerformNetworkPrediction(Eigen::ArrayXf X){
    Y = (W0.matrix() * X.matrix()).array() + b0;
}
```

Figure 2.8: Single-layered network implementation using Eigen
Note: this example requires variable initialization before use of the 'PerformNetworkPrediction()' function.

3

Methodology

3.1 To Answer the Research Questions

After the neural network solution has been fully integrated into the Apex game engine, its viability for generating locomotive character animations will be quantitatively evaluated in the following two ways.

3.1.1 Researching Responsivity

Responsivity will be measured in computational time required, per frame, during runtime for the network related code.

This will be tested for locomotive character animations around a static obstacleless track course, as to ensure deterministic user input. This obstacle course will be defined using waypoints, see Section 4.1.9, that both acts as positional checkpoints along the course and which dictates what movement style the character is expected to produce while traversing the environment toward the waypoint.

The track course will be defined using the following waypoints, see Table 3.1:

Index	Pos (m)	Gait	Speed	StrafeDir.
0	(-30, 40)	Walk	2.5	-
1	(-50, 0)	Walk	2.5	(-0.5, -0.5)
2	(-25, -25)	Jog	10.5	-
3	(25, -40)	Jog	10.5	-
4	(10, -10)	Crouch	2	-
5	(25, 50)	Walk	2.5	(0, -1)

Table 3.1: Track course details
Pos = the position of the waypoint in meters in world space
Gait = gait type, see Holden et al. [5]
Speed = goal root speed
StrafeDir. = normalized character facing direction vector when strafing

The final responsivity result will include the statistic for mean and standard deviations of frametime on a per lap basis of 19 laps. The in-engine representation of the

track course can be seen in Section 5.2.

The reason behind the choice of evaluating responsivity specifically, is how high responsiveness is a requirement for both immersive and interactive non-passive experiences, such as computer games, and that it that can be evaluated quantifiably. Additionally, low responsiveness may influence both player enjoyment and performance when playing computer games, interrupting a possible state of flow [43].

3.1.2 Researching Accuracy

The accuracy of the integrated network solution will be measured by comparison between the predicted output pose and the corresponding ground truth for the entirety of the training data.

To allow for this comparison, pairs of input and output vectors will be generated similarly as those during the database generation process, see Section 3.3.2, and will be evaluated as part of the final runtime package, see Section 4.1.8.

There are multiple different error definitions used within regression, such as mean-squared error (MSE), root mean-square error (RMSE), and mean-absolute error (MAE) [44].

The error definition to be used as part of the accuracy evaluation in this report will be mean-absolute error. This was chosen as firstly, mean-absolute error is more forgiving for outliers which may be expected in this particular data set, and secondly, mean-squared error is already used as part of the training process. A different error calculation for the evaluation process than what is used during training may be useful for testing generalization and to allow for comparisons between the two errors.

The error will be presented in both mean and standard deviation on a per-file basis, evaluated through a per-frame calculation according to the following mean-absolute error formula:

$$\frac{1}{|j|}\sum_{j}\frac{|t_j - p_j|}{t_j} \tag{3.1}$$

Where $|j|$ is the total number of frames in this file, t_j is the three-dimensional position of joint j as defined in the motion capture database, and where p_j is the three-dimensional network predicted output position of joint j. The values taken from the motion capture database, including t_j, is referred to as the ground truth.

As this definition includes the t_j denominating term, an error evaluated using the formula can be interpreted as the relative prediction error in percentage. In other words, an error of 0.01 equals an average joint position error of 1%.

Additionally, as a result of restrictive system memory, the maximum number of frames considered in each motion capture data file is that which equals at most 500'000 discrete joint positions. In other words, for a skeleton with 191 joints, only the first 2'617 unique frames will be considered.

3.1.3 Researching Architecture

To answer the subsidiary research question regarding architecture optimality, comparisons will be made between the results of the different network configurations, as presented in Section see Section 1.3.

This analysis is to be done by comparison of the evaluation results, as presented in Section 3.1.1 and Section 3.1.2, between the following neural network configurations:

- *Holden - Default* (HOLDEN) - The default network solution as presented in Holden et al. [5]: a phase-functioned neural network with a single hidden network layer of 512 nodes, trained for 2'000 epochs 3.3.3.
- *Holden - Extra Layer (HOLDEN-XL)* - The default network solution as presented in Holden et al. [5] but using two hidden network layers of 512 nodes each.
- *Holden - Extra Trained (HOLDEN-XT)* - The default network solution as presented in Holden et al. [5] but trained for 4'000 epochs.
- *Avalanche (AVA)* - The default network solution as presented in Holden et al. [5] but heavily altered to accompany an in-house skeleton.

The reasoning behind these specific choices in network configurations were that firstly, there must be a control case network that mimics the original implementation. Then, having a longer network, or a network that is trained for a longer period of time, could be used for conceptually straightforward comparisons. Additionally, evaluating a network with a longer trained process would also be interesting to investigate whether the original implementation by Holden et al. suffers from overfitting, see Section 2.1.4, given that that implementation uses no validation data or early stopping 2.1.4.

THE HOLDEN CONFIGURATIONS

The hyperparameters that will be used for the network configurations are directly based on the work by Holden et al. [5], see Section 3.2.

Additionally, the 31 joint .bvh, see Section 2.3.1, skeleton used for these configurations is that of the original .bvh files made public by Holden et al. [5], see Figure 3.1.

The HOLDEN and HOLDEN-XT networks will both have an input layer width of 342, a hidden layer width of 512, and an output layer width of 311. The extra hidden layer present in the HOLDEN-XL network configuration will also have 512 nodes.

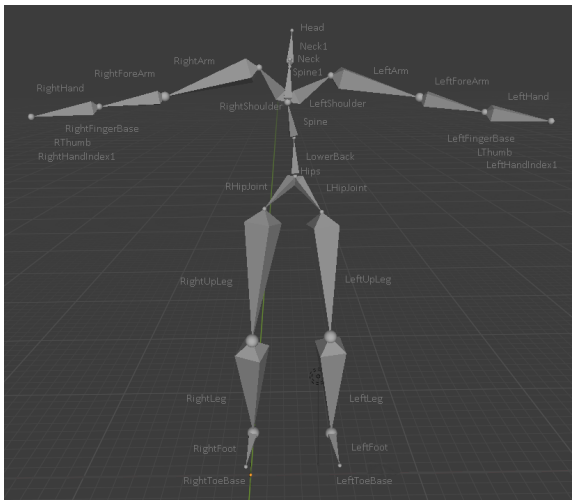

Figure 3.1: Visualization of the .bvh skeleton used by the Holden configurations This is the same skeleton as presented by Holden et al. [5].

THE AVALANCHE CONFIGURATION

The altered AVA network will general use the same network hyperparameters as that of the Holden configurations.

However, it will use a different skeleton, one that is used in a live Avalanche product. This skeleton is visualized in Figure 3.2.

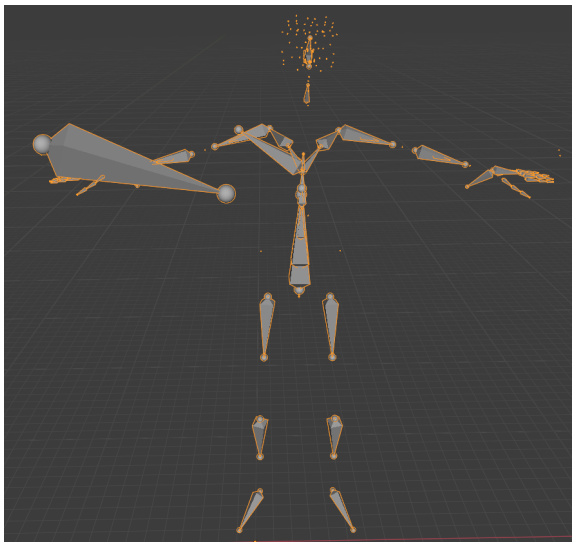

Figure 3.2: Visualization of the .bvh skeleton used by the AVA configuration The 'extra' joints that appear outside the character body are used for tasks such as deformation, object interactions, and player camera locations. Do notice the many extra, compared to in Figure 3.1, joints in the character head and hands. For the complete list of joint names in this skeleton, see Appendix A.3.

To accommodate for the AVA skeleton having 191 joints, rather than the 31 of the Holden configurations, the Avalanche network will have a input layer width of 1'302,

the same hidden layer width of 512, and an output layer width of 1'751.

Also, since this configuration aims to use an in-house Avalanche skeleton, the .bvh files made public by Holden et al. [5] will need to be re-generated. This retargeting step will be done by professionals employed at Avalanche.

Additionally, since the network is trained for 60Hz predictions, whereas the retargeted .bvh files was retargeted to the in-house standard of 30Hz, these .bvh files will need to be interpolated, see Figure A.1 and Figure A.2 in Appendix A.1.

As a consequence of the greater number of joints, the network training database used for the AVA configuration will only include every fourth motion capture frame. This, as otherwise the training database does not physically fit in the runtime memory of the system used as part of this thesis, see 1.4. To reemphasize: the AVA configuration will be trained on a fourth of the number of motion capture frames than that of the Holden configurations. However, each frame in the Ava training database will contain data of more than six times the joints than in the Holden training database. This issue, however, could have been resolved by rewriting the network training logic such that the network could onload, and offload, parts of the training database. This would lead to the network being able to indirectly train on the entire data set, including all movement frames, even though the database would be too large to fit in system memory at once. However, this procedure would require an extensive rewrite of the original implementation by Holden et al., and this would potentially drastically increase the time required during the training process as onloading and offloading such large chunks of memory is a slow process.

Also, a specific subset of the joints most equivalent to those of the Holden skeleton used in the Avalanche skeleton will be referred to as the Avalanche Masked skeleton: AVA-M. In other words, the AVA-M skeleton are the subset of Avalanche joints most similar to those in the Holden skeleton, see Section 3.3.3.

3.1.4 Simple Difference Significance Evaluation

To evaluate the statistical significance in the difference between two data sets, a and b, the following version of heuristic will be used:

$$\frac{2|mean(a) - mean(b)|}{sd(a) + sd(b)} \tag{3.2}$$

This is equivalent to evaluating the difference between the means of the two data sets in measurements of the average of their respective standard deviation.

The absolute difference is used here for the same reasons that mean-absolute error is used for the accuracy evaluation, see Section 3.1.2.

3.2 The Phase-Functioned Neural Network

A phase-functioned neural network, as presented by Holden et al. [5], is a neural network with weights generated by a cyclic phase variable produced by a phase function.

This section aims to describe the functional components of a phase-functioned neural network within the specific context of this project, as presented in Holden et al. [45].

3.2.1 Network Structure

The network architecture used in Holden et al. [5] is a neural network with the following structure, where each network node uses a trainable bias:

- H_0 - Input layer of 342 nodes, see Section 3.2.2.
- H_1 - Fully-connected hidden layer of 512 nodes.
- H_2 - Output layer of 311 nodes, ELU [8] activation function, see Section 3.2.3.

3.2.2 The Input Vector

The input vector $\mathbf{x}_i$, at frame i, is a concatenation of, amongst others; sample points on the terrain along the traversed and expected path of the animated character, see Figure 3.3, and the current joint positions and velocities of the character.

$$\mathbf{x}_i = \{\mathbf{t}_i^p, \mathbf{t}_i^d, \mathbf{t}_i^h, \mathbf{t}_i^g, \mathbf{j}_{i-1}^p, \mathbf{j}_{i-1}^v\} \in \mathbb{R}^n \tag{3.3}$$

where:

- $\mathbf{t}_i^p \in \mathbb{R}^{2t}$, the x, y positions of the sample points in character local space
- $\mathbf{t}_i^d \in \mathbb{R}^{2t}$, the x, y trajectories of the sample points in character local space
- $\mathbf{t}_i^h \in \mathbb{R}^{3t}$, the heights of each sample point and additional sub-sample points
- $\mathbf{t}_i^g \in \mathbb{R}^{5t}$, a vector containing the gait of the character along the sample points
- $\mathbf{j}_{i-1}^p \in \mathbb{R}^{3j}$, the position of all j character joints in the previous frame $j-1$
- $\mathbf{j}_{i-1}^v \in \mathbb{R}^{3j}$, the velocities of all j character joints in the previous frame $j-1$

where:

t is the number of sample points centered around, and including the at the feet of, the character. This value was set to 12 in Holden et al. [5], equaling five sample points ahead, and six sample points behind, the character.

j is the number of joints within the character model. This value is was set to 31 in Holden et al. [5].

Figure 3.3: Subset of PFNN input vector visualized
a: sample point positions - $\mathbf{t}_i^p \in \mathbb{R}^{2t}$
b: sample point trajectories - $\mathbf{t}_i^d \in \mathbb{R}^{2t}$
c: (sub-)sample point heights - $\mathbf{t}_i^h \in \mathbb{R}^{3t}$
source: Holden et al. [46].

3.2.3 The Output Vector

Similarly, the output vector $\mathbf{y}_i$, at frame i, is a concatenation of both predicted future states, the next pose of the character, and an update of certain metadata.

$$\mathbf{y}_i = \{\mathbf{t}_{i+1}^p, \mathbf{t}_{i+1}^d, \mathbf{j}_i^p, \mathbf{j}_i^v, \mathbf{j}_i^a, r_i^x, r_i^z, r_i^a, \dot{p}_i, \mathbf{c}_i, \} \in \mathbb{R}^m \tag{3.4}$$

where:

- $\mathbf{t}_{i+1}^p \in \mathbb{R}^{2t}$, the predicted x, y positions of the sample points in character local space of the next frame $i+1$
- $\mathbf{t}_{i+1}^d \in \mathbb{R}^{2t}$, the predicted x, y trajectories of the sample points in character local space of the next frame $i+1$
- $\mathbf{j}_i^p \in \mathbb{R}^{3j}$, the generated position of all j character joints
- $\mathbf{j}_i^v \in \mathbb{R}^{3j}$, the generated velocities of all j character joints
- $\mathbf{j}_i^a \in \mathbb{R}^{3j}$, the generated angles of all j character joints
- $r_i^x \in \mathbb{R}$, local character velocity in the relative x direction
- $r_i^z \in \mathbb{R}$, local character velocity in the relative z direction
- $r_i^a \in \mathbb{R}$, local character angular velocity around the world up vector
- $\dot{p}_i \in \mathbb{R}$, phase variable update delta
- $\mathbf{c}_i \in \mathbb{R}^4$, binary contact labels of heel and toe joints with the ground

3.2.4 The Phase Function

The Phase function blends between four sets of network weights, $\boldsymbol{\alpha}_{k_0}, \boldsymbol{\alpha}_{k_1}, \boldsymbol{\alpha}_{k_2}, \boldsymbol{\alpha}_{k_3}$, using cubic Catmull-Rom interpolation [47]. As such, the number of network weights needed to be stored in memory at runtime is multiple times that of a singular network configuration.

The phase function $\boldsymbol{\Theta}$ is evaluated:

$$\begin{aligned} \boldsymbol{\Theta}(p; \boldsymbol{\alpha}_{k_0}, \boldsymbol{\alpha}_{k_1}, \boldsymbol{\alpha}_{k_2}, \boldsymbol{\alpha}_{k_3}) = \\ \boldsymbol{\alpha}_{k_1} \\ +w(\tfrac{1}{2}\boldsymbol{\alpha}_{k_2} - \tfrac{1}{2}\boldsymbol{\alpha}_{k_0}) \\ +w^2(\boldsymbol{\alpha}_{k_0} - \tfrac{5}{2}\boldsymbol{\alpha}_{k_1} + 2\boldsymbol{\alpha}_{k_2} - \tfrac{1}{2}\boldsymbol{\alpha}_{k_3}) \\ +w^3(\tfrac{\mathbf{3}}{\mathbf{2}}\boldsymbol{\alpha}_{k_1} - \tfrac{3}{2}\boldsymbol{\alpha}_{k_2} + \tfrac{1}{2}\boldsymbol{\alpha}_{k_3} - \tfrac{1}{2}\boldsymbol{\alpha}_{k_0}) \end{aligned} \tag{3.5}$$

where:

$$w = \frac{4p}{2\pi} \pmod 1 \tag{3.6}$$

$$k_n = \left\lfloor \frac{4p}{2\pi} \right\rfloor + n - 1 \pmod 4 \tag{3.7}$$

Within this project, the phase function will be evaluated during runtime. An alternative approach would be to precompute the function and store its results in memory. This would reduce the computational load at runtime but increase the memory footprint [5].

3.3 The Full PFNN Pipeline

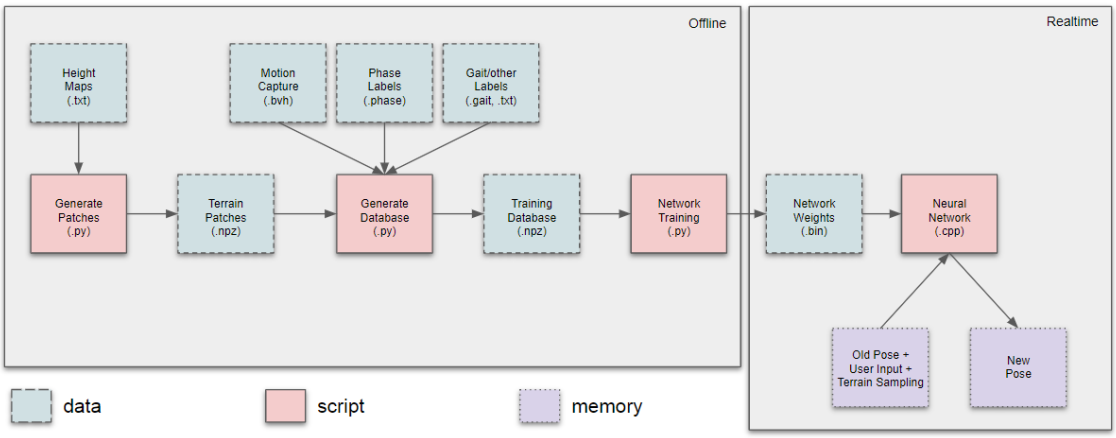

Figure 3.4: The full PFNN pipeline
'data' = offline storage
'script' = runnable files
'memory' = temporary, runtime

This section aims to provide an overview of the full phase-functioned neural network pipeline, as designed by Holden et al. [5] and as presented in Figure 3.4. The final integrated version of this model is presented in Section 4.1.

3.3.1 Generate Patches

To allow for the generation of locomotive character animations that adhere to the roughness of the topography, the training data used later must include different types of terrain. A solution to this is to fit heightmaps to the separately recorded motion capture data, firstly producing intermediate patches of terrain.

3.3.2 Generate Database

During this step, each input and output vector pair, see Section 3.2.2 and Section 3.2.3, is produced and stored. Each vector pair is created on a per-frame basis using motion captured data, see Section 2.3.1, and associated labels, such as the phase and gait variables. Additionally, for each motion capture clip, the ten most suitable heightmaps are fitted to the foot-to-ground contacts of the character.

3.3.3 Network Training

Training will be performed using the Theano [38], a Python library for multi-dimensional array computations on the GPU - see Section 2.3.2, implementation by Holden et al. [5], and an Adam optimizer, see Section 2.1.6. The result of this step will be the finalized trained network weights. The default hyperparamters for the training will be:

- batchsize = 32
- learning rate = 0.0001
- beta1 = 0.9
- beta2 = 0.999
- epochs = 2000
- error function = mean-squared error

For the order of the motion capture data files, see Table A.1 in Appendix A.2.

During the training process, the translation and orientation of the joints not on the following list, or equivalent to these in the case of the Avalanche configuration, within the input vector will be put to ≈ 0, as is done in Holden et al. [5]:

- Hips
- LeftUpLeg
- LeftLeg
- LeftFoot
- LeftToeBase
- RightUpLeg
- RightLeg
- RightFoot
- RightToeBase
- Spine
- Spine1
- Neck1
- Head
- LeftArm
- LeftForeArm
- LeftHand
- RightArm
- RightForeArm
- RightHand

Additionally, this training process will not make use of the early-stopping technique, see Figure 2.4.

3.3.4 Neural Network

This step includes the entire package necessary for runtime pose prediction. During initialization, all necessary trained network weights will be read and loaded in memory. Then, each frame, a prediction request is passed to the package, providing a character pose in the current frame and expecting an updated character pose as return value. In addition to the character pose, other metadata is feed to the network for prediction, such as sample points of the topography and user input, see Section 3.2.2.

In this step is where the bulk of the integration work will be. However, the overall package structure will be based of the demonstration codebase made public by Holden et al. [5], with the neural network model defined in Eigen, see Section 2.3.3, arrays and matrices.

Additionally, as mentioned in Section 1.4, the motion capture data, and therefore the trained neural network, uses left-handed world-axis, whereas the Apex engine uses a right-handed world-axis, see Figure 3.5.

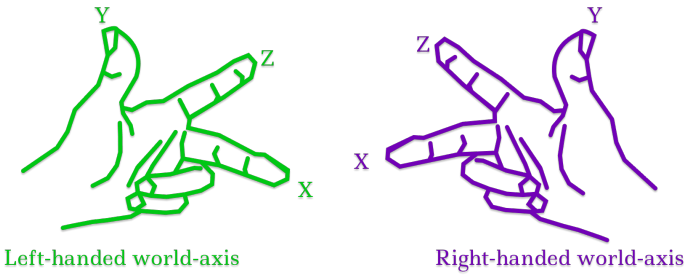

Figure 3.5: Visual representation of left/right-handed world-axis orientations
Left: Left-handed world-axis (green)
Right: Right-handed world-axis (purple)

For this reason, the runtime neural network package must be altered such that it can convert between the world-axis orientations. The character pose, living in a right-handed world-axis, is to be converted to the left-handed world-axis of the neural network. Then, the neural network outputted updated character pose must be converted back into right-handed world-axis before being applied to the character skeleton.

4

Process

4.1 The Runtime Package

This section aims to present the runtime package implemented for the phase-functioned neural network solution, originally based on the demonstration software made public by Holden et al. [5]. An overview of this package is presented in Figure 4.1.

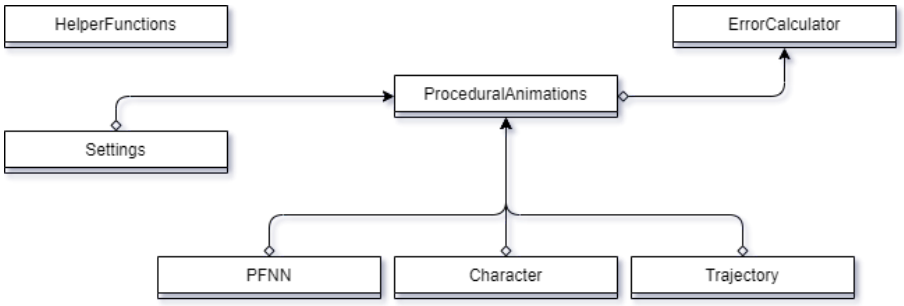

Figure 4.1: The Procedural Animations runtime package
The neural network solution is accessible either from the ProceduralAnimations class, or indirectly through the ErrorCalculator class.

4.1.1 Using the Runtime Package

The runtime package, see Section 4.1, is aimed to have a low level of coupling, such that other programmers need not to interact with, nor understand, the deeper machinations of the package.

As such, to use the runtime package, a programmer would only need to perform two things: initialize the Procedural Animations class and to call 'GetNextPose(...)' when wanting to use the network for predictions, see Section 4.1.2.

During initialization, the ProceduralAnimations constructor takes three optional parameters, see Figure 4.2:

- *new world transform* - A 4D matrix for character scaling/rotation/translation.
- *new setting* - A Setting enum, see Section 4.1.6, for network configurations.
- *new waypoint sptr* - A pointer to a vector of Waypoint:s, see Section 4.1.9.

```
CProceduralAnimations(
        AosMatrix4 new_world_transform = AosMatrix4(0.0f),
        CSettings::SETTING new_setting = CSettings::HOLDEN,
        std::vector<CProceduralAnimationsWaypoint*>* new_waypoints_ptr = nullptr);
```

Figure 4.2: Procedural Animations constructor

Then, during the constructor execution, the objects that the ProceduralAnimation class owns are initialized.

During runtime prediction, only the 'GenerateNextPose(...)' function is required, see Figure 4.3. This function takes two parameters: a pointer to the current character pose, and a pointer to the translational character-in-world offset. Then, the 'GenerateNextPose(...)' updates the two input parameters in place, given the outputs of the neural network.

```
void GenerateNextPose(CPose* pose, AosVector3* translation_offset);
```

Figure 4.3: Procedural Animations per frame prediction

4.1.2 ProceduralAnimations

This class owns the pointers to the PFNN, Character, Trajectory, and Settings representations. As the ErrorCalculator class is intended only for evaluation purposes, the ProceduralAnimations class is the default way to access the phase-functioned neural network solution.

Inside the 'GenerateNextPose(...)' function, see Figure 4.3, the flow of sub-function calls is organized as follows:

1. *Prepare* - Stores the input pose information in the Character object.
2. *Input* - Evaluates the Waypoint information and sets the Trajectory state.
3. *Insert* - Inserts the Character and Trajectory states into the input vector.
4. *Predict* - Runs the network prediction, setting the output vector.
5. *Output* - Stores the relevant output vector information in the return pose.
6. *Update* - Update Character and Trajectory states using output vector.

The time required to perform these six steps is recorded e ach frame for use in evaluating the systems responsivity, see Section 3.1.2.

Additionally, this class has debug rendering functionality for visually rendering network parameters, such as the joint skeleton, the sample points, character velocities, etc., in the engine.

4.1.3 PFNN

This struct holds the memory representation of the neural network and is responsible for the network prediction.

When initialized, the PFNN struct loads the network weights and biases into Eigen, see Section 2.3.3, matrices in memory from stored .bin files. The .bin directory and network configuration is fetched from the Settings object. Additionally, the PFNN struct is the only part of the runtime package dependent on the Eigen library.

During the prediction step, the PFNN struct performs the matrix multiplications necessary to propagate the input vector state, and then standardizes the result before storing it in the output vector data structure.

4.1.4 Character

The Character struct stores the positions and translations, in model space, of all character joints in the current frame. Additionally, the same information is stored for the few previous frames to allow for output blending when setting the return pose values.

4.1.5 Trajectory

Similar to the Character struct, the Trajectory struct holds all information regarding the sample points along the ground, see Figure 3.3, such as positions and velocities. These values are also stored between multiple frames to allow for output blending.

4.1.6 Settings

The Settings class is used to manage easy switching between the different network configurations, see Section 3.1.3, which is represented as an enum passed to the constructor.

To allow for a low level of coupling and extensibility, in the form of being able to add additional network configurations requiring minimal changes in the code base, the Setting class holds all data that may be affected by the choice of network configuration. In other words, if one wants to add another network configuration, one would only need to add support for it in the Setting class.

For an example, all paths to the network .bins are defined in the Setting class. This means that when a PFNN object initializes, it simply calls something similar to 'settings->GetWeightsPath()', without needing any logic, e.g. switch cases, that requires the knowledge of a network configuration enum or how that configuration would affect this class. This is shown in Figure 4.4

```
class Settings{
   enum CONFIG {HOLDEN, AVA};

   string path;

   Settings(CONFIG new_config){
       switch(new_config){
           case HOLDEN:
               path = "/holden_weights/"
               break;
           case AVA:
               path = "/avalanche_weights/"
               break;
       }
   }

   string GetPath(){
       return path;
   }
}
```

Figure 4.4: Simplified example of Settings implementation
(DISCLAIMER: PSEUDO CODE! NOT ACTUAL IMPLEMENTATION!)

4.1.7 HelperFunctions

This is a simple, fully static class that holds functions such as debug outprints and definitions for specific matrix operations.

4.1.8 ErrorCalculator

When evaluating the network, rather than creating an instance of the ProceduralAnimations class, one initializes an ErrorCalculator instead. This object acts as a wrapper around a ProceduralAnimations instance and, rather than depending on an input pose, uses stored input and output vector pairs, see Section 3.2.2 and Section 3.2.3.

This class is therefore responsible for calculating the evaluative results required in the answering of the research question regarding accuracy, see Section 1.3 and Section 3.1.2.

This evaluation process can either be run immediately on initialization, or on a per frame basis to allow for visualization of the network prediction, compared to the ground truth. This is controlled with a 'run-offline' flag.

Since the ErrorCalculator constructs an internal ProceduralAnimations instance, it also requires the same input parameters; both in the constructor, see Figure 4.5, and on the per frame prediction, see Figure 4.6.

```
CProceduralAnimations(
        AosMatrix4 new_world_transform = AosMatrix4(0.0f),
        CSettings::SETTING new_setting = CSettings::HOLDEN,
        std::vector<CProceduralAnimationsWaypoint*>* new_waypoints_ptr = nullptr,
        bool run_offline = false);
```

Figure 4.5: Error Calculator constructor

```
float CalculateError(CPose* pose);
```

Figure 4.6: Error Calculator per frame prediction

4.1.9 Waypoint

Each Waypoint instance is a simple datastructure, representing one checkpoint along the obstacle course that the characters will traverse as part of the responsivity research, see Table 3.1 in Section 3.1.1.

In addition to its inherent world translation, each Waypoint holds information representing the goal movement style that a character aims to perform when reaching it. This includes the gait styles; walking, jogging, crouching, etc, but also movement speed and facing direction. This, in an aim to deterministically simulate user input during the evaluation process.

The ProceduralAnimations instance keeps track of the current Waypoint index, and increments that number upon reaching the next checkpoint.

5

Result

5.1 Responsivity Results

All responsivity data, which is used to produce the figures and tables presented in this section, is available in Appendix A.4. For more information regarding the track course used, see Section 3.1.1.

The responsivity results presented in Figure 5.1 shows the average frame time computation in milliseconds per lap around the course. The same data is presented as a box plot in Figure 5.2, and summarized in Table 5.1.

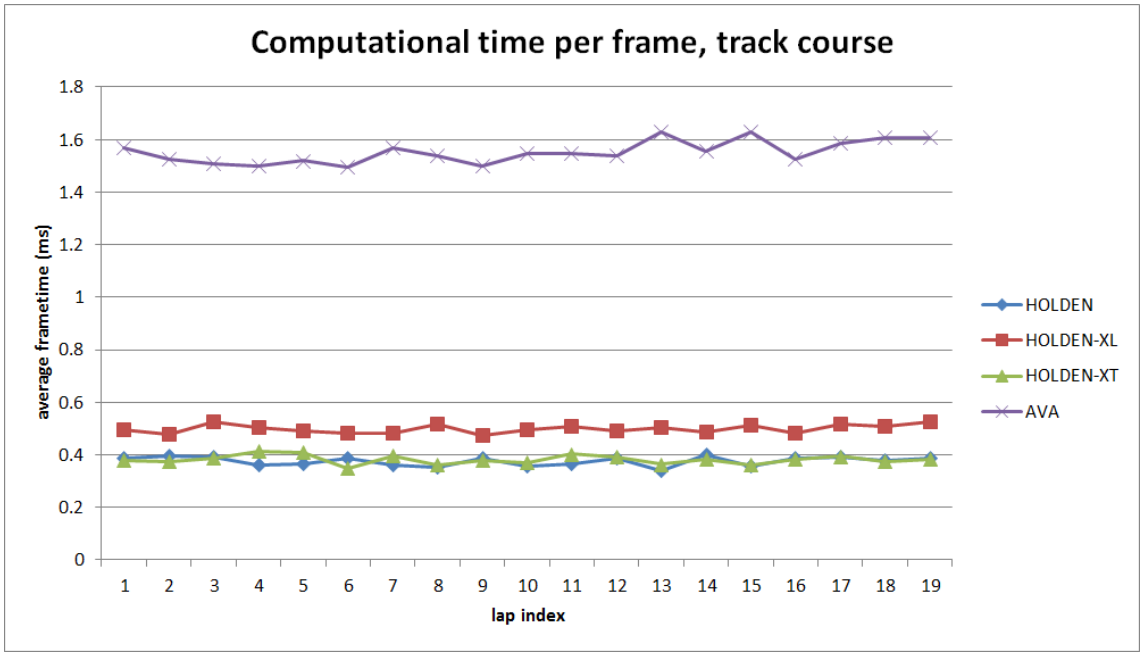

Figure 5.1: Line chart of responsivity results

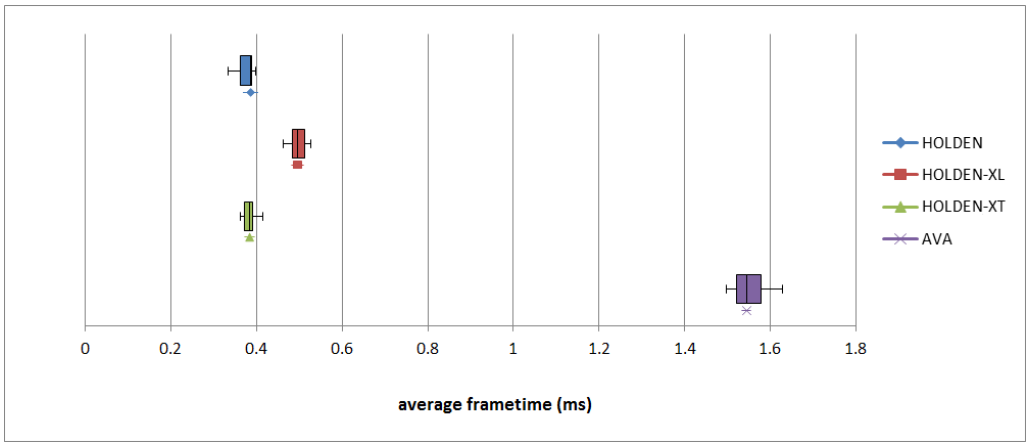

Figure 5.2: Boxplot of responsivity results

	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA
Mean	0.376	0.499	0.382	1.55
SD	0.0173	0.0161	0.0167	0.0430

Table 5.1: Mean and standard deviation results of responsivity evaluation Values are rounded to three significant digits.

By combining the visual results of the line chart in Figure 5.1 and the box plot in Figure 5.2, one can conclude that there is a considerably sized difference in computational time required for that of the AVA network configuration. A potential root cause of this is the great increase in number of joints for that network, see Section 6.1.1 for further discussion on this topic.

For the Holden configurations, the results of HOLDEN and HOLDEN-XT have almost perfect overlap in both Figure 5.1 and in Figure 5.2. As such, one can conclude that these two network configurations have practically equivalent responsivity. However, this is not too surprising as, in theory, a network having trained longer, with otherwise the same hyperparameters, should only result in a different set of network weights. Subsequently, two otherwise equivalent networks but with different weights should still be evaluated at runtime at the same speed.

Finally, for the HOLDEN-XL configuration, it is not as visually clear whether it at runtimes evaluates at a considerably different speed than that of the other Holden configurations. For this, the similarity metric defined in Section 3.1.4 can be used. This metric evaluates the absolute difference between each mean result, standardized by the average standard deviation of the two data series. In other words, the metric evaluates how many standard deviations two data points differ.

$$\frac{2|mean(a) - mean(b)|}{sd(a) + sd(b)} \tag{5.1}$$

- HOLDEN to HOLDEN-XT: $\frac{2|0.382-0.376|}{0.0173+0.0167} \approx 0.35$
- HOLDEN-XL to HOLDEN-XT: $\frac{2|0.499-0.382|}{0.0161+0.0167} \approx 7.1$
- AVA to HOLDEN-XT: $\frac{2|1.55-0.499|}{0.0430+0.0161} \approx 36$

These calculations, together with the visualizations in both Figure 5.1 and in Figure 5.2, can be combined to suggest the relative significance of the differences in standard deviations between the responsivity results. Even though the number of standard deviations between the results of the HOLDEN-XL configuration and that of the HOLDEN-XT results are much smaller than that to the results of the AVA configuration, one can still make the argument that there is a noticeable dissimilarity in computational time required for the HOLDEN-XL configuration. This difference could be explained through the fact that adding another layer in a neural network strictly increases the number of computations, and therefore the time, required for evaluation at runtime. For further discussion on this topic, see Section 6.1.1.

5.2 Responsivity Visualizations

Video recordings of these visualizations are available here [48].

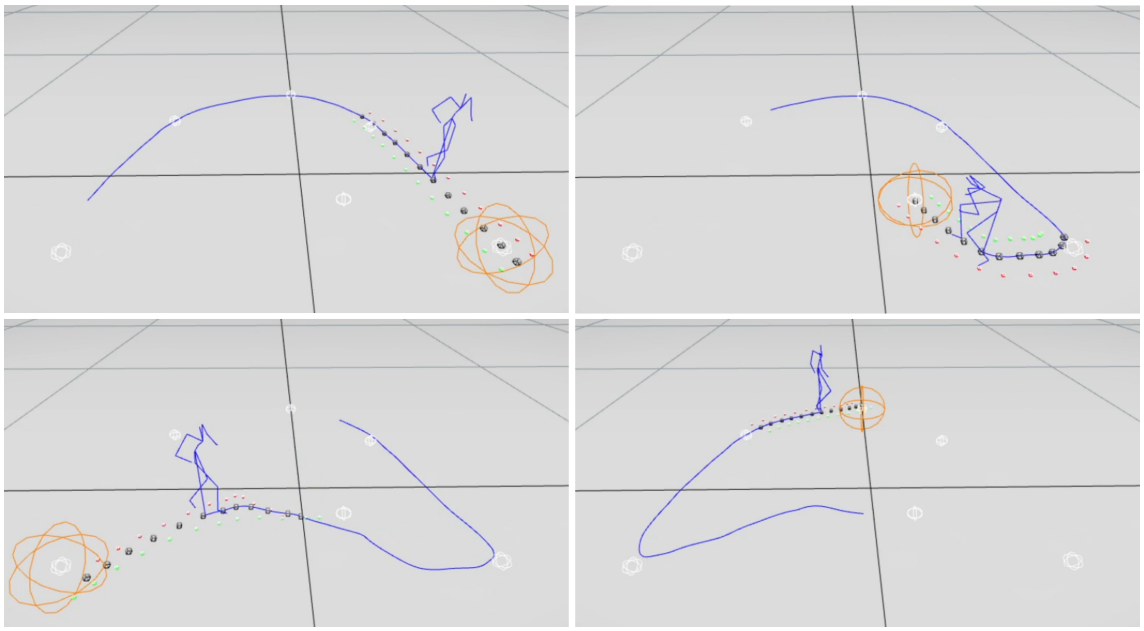

Figure 5.3: Some frames from the HOLDEN responsivity evaluation
Top left: jogging, Top right: crouching
Bottom left: strafing backwards, Bottom Right: walking
White globes are Waypoints, see Section 4.1.9.
The golden Waypoint is the next positional target of the network.

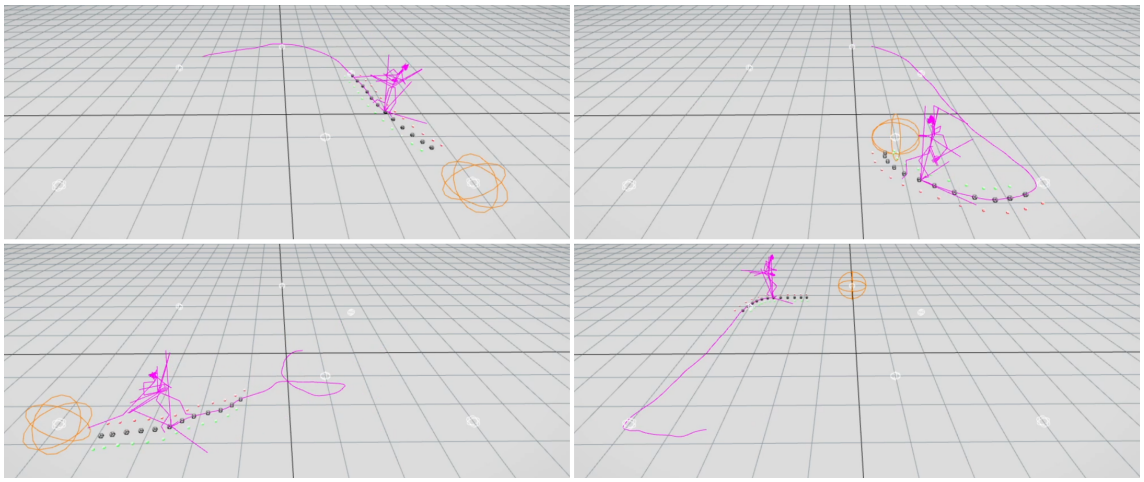

Figure 5.4: Some frames from the AVA responsivity evaluation
Top left: jogging, Top right: crouching
Bottom left: strafing backwards, Bottom Right: walking
White globes are Waypoints, see Section 4.1.9.
The golden Waypoint is the next positional target of the network.

In Figure 5.3 and Figure 5.4, one can see examples of the skeleton joint position outputs the HOLDEN and AVA network configurations produced during their respective responsivity evaluations.

For the AVA configuration, certain errors occurred, potentially as a result of the network not being trained on sufficient amount of data frames, see Section 6.1.3 and 6.4.1 for further discussion. For an example, notice how poorly the produced joint skeleton appears to be crouching in the bottom right photograph in Figure 5.4. Additionally, the AVA configuration failed to adapt to tight turns, making the outputted skeleton overshoot the target, see Figure 5.5.

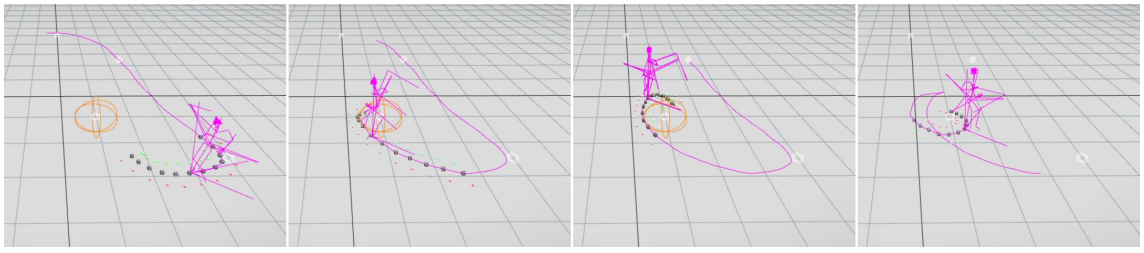

Figure 5.5: Directional overshoot during the AVA responsivity evaluation
The goal of the network is to move the skeletal character towards the golden Waypoint. However, the AVA configuration fails to sufficiently turn the character towards this goal before the character has passed it.

5.3 Accuracy Results

All accuracy data, which is used to produce the figures and tables presented in this section, is available in Appendix A.5 and Appendix A.6.

The accuracy results presented in Figure 5.6 shows the average error per motion capture data file for each of the four network configurations. The error calculation is defined as presented in Section 3.1.2:

$$\frac{1}{|j|}\sum_{j}\frac{|t_j - p_j|}{t_j} \tag{5.2}$$

Where $|j|$ is the total number of frames in this file, t_j is the three-dimensional position of joint j as defined in the motion capture database, and where p_j is the three-dimensional network predicted output position of joint j. The values taken from the motion capture database, including t_j, is referred to as the ground truth.

Additionally, the fifth data series 'AVA-M' shown in this figure represents the results of the AVA network limited to the subset of network outputs equivalent to those joints present in the original motion capture data made public by Holden et al. [45], see Figure 3.1 and Section 3.1.3.

Similarly, Figure 5.7 presents the standard deviations, a measurement of spread in the data distribution, of the per motion capture file network outputs for each of the network configurations. A smaller standard deviation equates to little difference between data points within a data series, whereas a higher standard deviation equates to more fluctuating data points.

The same accuracy data is presented as a box plot in Figure 5.8, and summarized in Table 5.2.

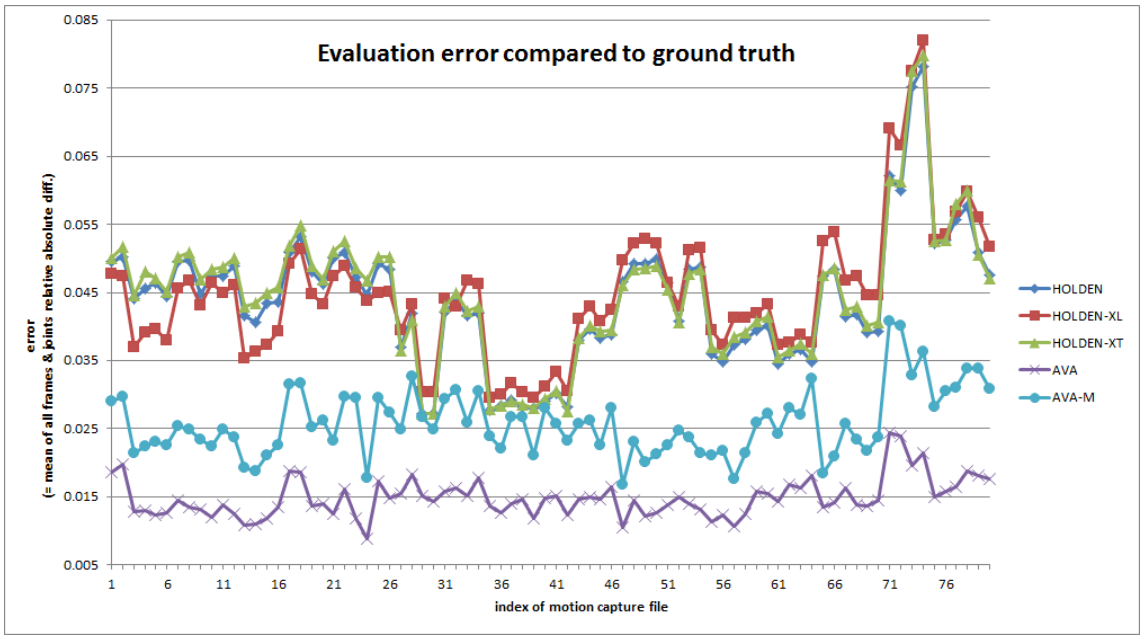

Figure 5.6: Line chart of mean results of accuracy evaluation
The error is defined as mean absolute error compared to the training data.
For full definition of error, see Section 3.1.2.
For indexing of motion capture files, see Appendix A.2.

What can be seen in Figure 5.6 is that the results of the three Holden configurations are greatly overlapping throughout the training data set. The blue diamond HOLDEN data series is almost perfectly obscured by the green triangle HOLDEN-XT series.

Somewhat similarly, the two Ava results appear to follow a slightly similar curvature, however vertically translated to a lower error level than that of the Holden configurations. Internally, however, the curvature of the two Ava data series is highly similar, though also vertically translated. In other words, if the AVA-M data series would be shifted downwards in the chart, there would be almost constant visual overlap between it and the AVA data series. However, visually there is almost no similarity in the curvatures of the Holden and Ava configurations.

Throughout the entirety of Figure 5.6, the Ava data series produce a lower error than that of the Holden configurations. This is visualized through how the AVA and AVA-M data series are consistently below the other three.

The motion capture files that all network configurations performed the worst at, data files indexed at 72-75, were that of the files containing movement interacting with more advanced terrain and environments, such as balancing on elevated narrow beams and dynamic crouching beneath low ceilings.

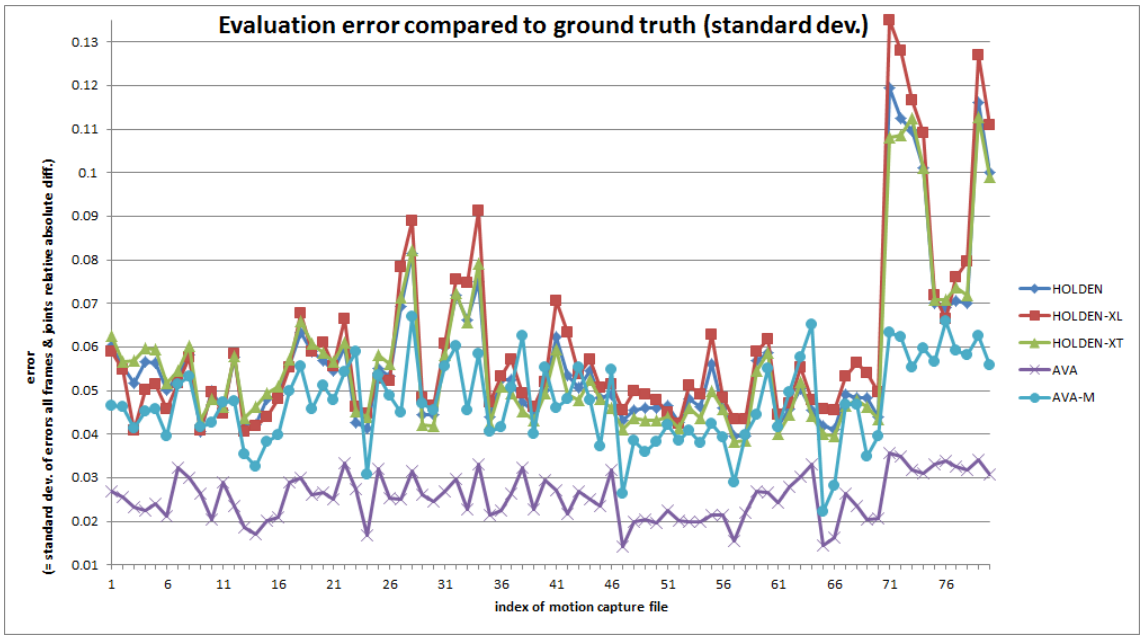

Figure 5.7: Line chart of standard deviation results of accuracy evaluation
The error is defined as mean absolute error compared to the training data.
For full definition of error, see Section 3.1.2. For indexing of motion capture files, see Appendix A.2.

As a similar trend to the means presented in Figure 5.6, the standard deviations shown in Figure 5.7 has almost perfect overlap for the three Holden configurations. Once again, the blue diamond HOLDEN data series is almost perfectly obscured by the green triangle HOLDEN-XT series.

However, the curvature of the AVA-M data series appears to be a a midpoint to that of the Holden configurations and that of the AVA data series. Visually, the AVA-M has local maxima and minima similar to both of aforementioned series. Additionally, the values of the AVA-M series are positionally closer to that of the Holden configurations than to that of the AVA data series. In other words, data points along the turqoise circle AVA-M line are further away from that of the purple crossed AVA line than to those of the other three data series.

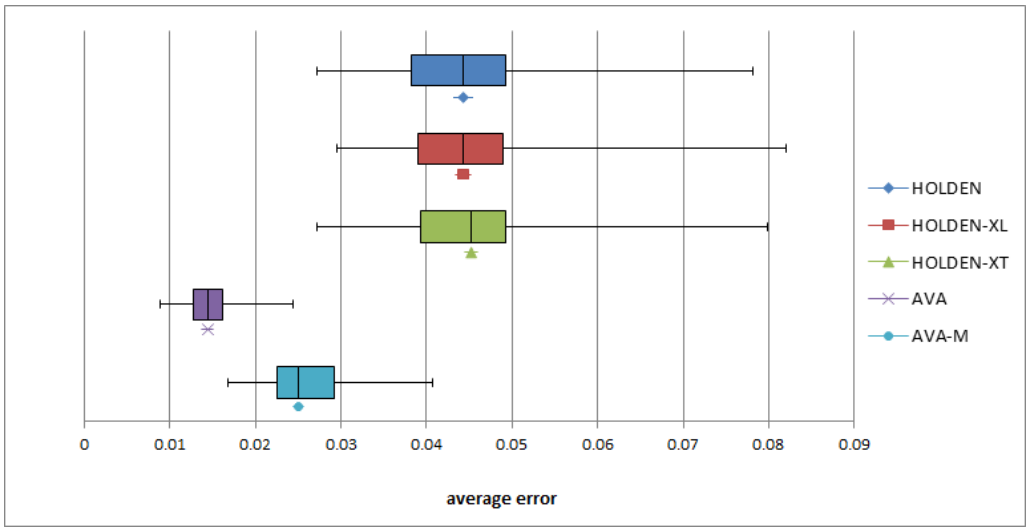

Figure 5.8: Boxplot of accuracy results
The error is defined as mean absolute error compared to the training data. For full definition of error, see Section 3.1.2.

As a reminder: an error of 0.01 equates to an average prediction error of 1%, see Section 3.1.2. As a concrete example; the predicted three-dimensional joint positions that the HOLDEN network configuration produced had, on average, a translational error of ≈ 4.4%.

	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA	AVA-M
Mean	0.0439	0.0448	0.0446	0.0148	0.0258
SD	0.00954	0.00980	0.00981	0.00283	0.00483

Table 5.2: Mean and standard deviation results of accuracy evaluation
Values are rounded to three significant digits.

For all three Holden configurations, the results have almost perfect overlap in both Figure 5.6 and in Figure 5.7. As such, one can conclude that these three network configurations have practically equivalent accuracy. This is rather interesting as both the HOLDEN-XL and HOLDEN-XT configurations each respectively have a specific advantage, in the form of extra network depth and extra training time, compared to the default HOLDEN configuration. These results suggest that there is no benefit to these specific network design alterations.

For the AVA configuration, by combining the visual results of the line chart in Figure 5.6 and the box plot in Figure 5.8, one can conclude that there is a significantly lower error in the AVA prediction than that of the Holden configurations.

Lastly, the AVA-M data series appear to share some similarity to both the Holden and AVA data series. To measure this similarity, one may utilize the difference metric used in Section 5.1 and originally presented in 3.1.1; calculating the number of standard deviations between the means. This produces the following results:

- HOLDEN to HOLDEN-XT: $\frac{2|0.0439-0.0446|}{0.00954+0.00981} \approx 0.072$
- HOLDEN to HOLDEN-XL: $\frac{2|0.0439-0.0448|}{0.00954+0.00980} \approx 0.0093$
- HOLDEN-XT to HOLDEN-XL: $\frac{2|0.0446-0.0448|}{0.0448+0.00980} \approx 0.0073$
- AVA-M to HOLDEN: $\frac{2|0.0258-0.0439|}{0.00483+0.00954} \approx 2.5$
- AVA to AVA-M: $\frac{2|0.0148-0.0258|}{0.00283+0.00483} \approx 2.9$

These differences can be summarized as the three Holden network configurations producing practically equivalent results, with an especially large overlap between HOLDEN and HOLDEN-XT, and the AVA-M results being slightly closer to that of the Holden configurations than to that of the AVA configuration.

5.4 Accuracy Visualizations

Video recordings of these visualizations are available here [48].

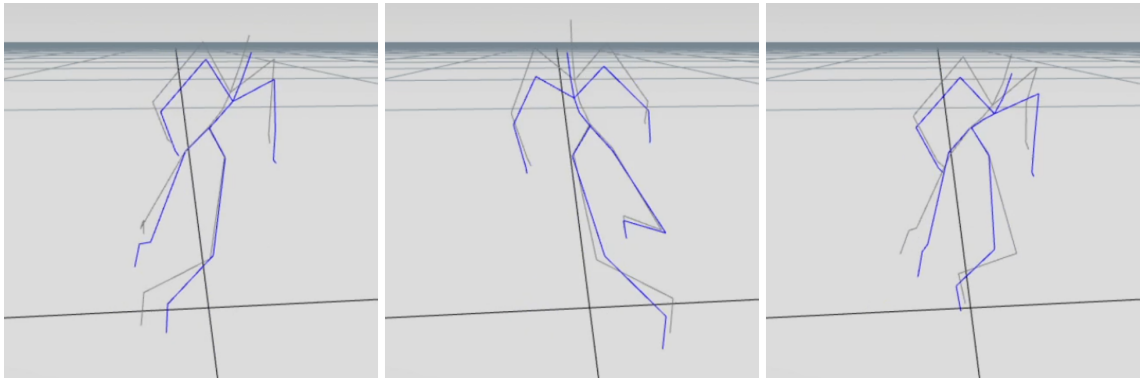

Figure 5.9: Some frames from the HOLDEN accuracy evaluation
Gray: Ground truth joint positions.
Blue: HOLDEN joint positions.

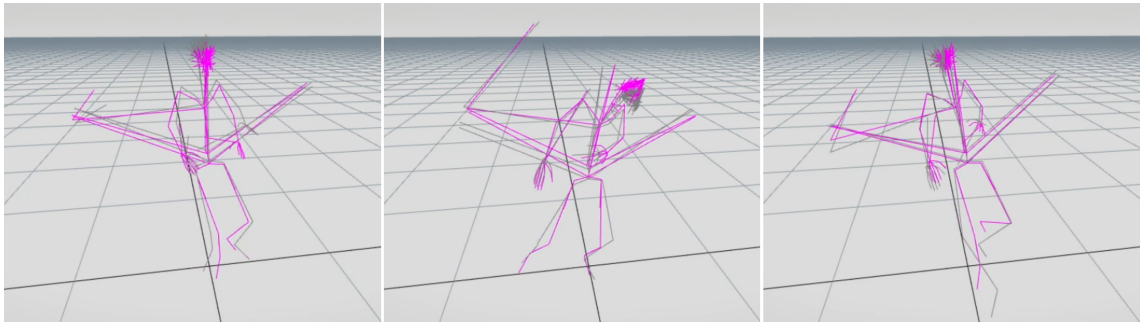

Figure 5.10: Some frames from the AVA accuracy evaluation
Gray: Ground truth joint positions.
Magenta: AVA joint positions.

In Figure 5.9 and Figure 5.10, one can see examples of the skeleton joint position

outputs the HOLDEN and AVA network configurations produced during their respective accuracy evaluations.

5.5 Training Process

During the training process, the prediction mean-squared error of the full output vector was recorded after each epoch. Do note the difference in error definition, and the fact that the entire output vector is used rather than just the predicted skeleton joint positions, compared to the one used in Section 5.3. This data is available in in full in Appendix A.7, and presented as a line chart in Figure 5.11.

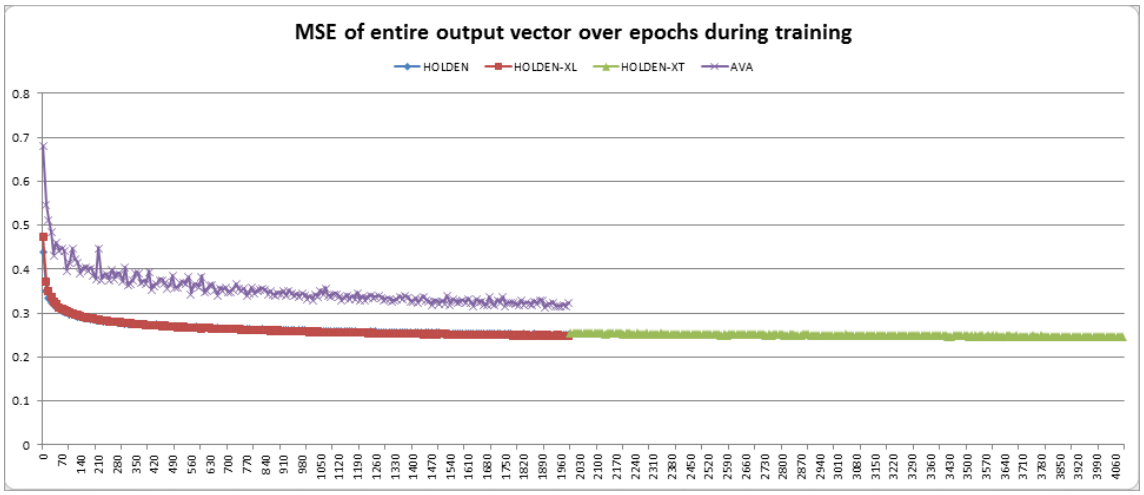

Figure 5.11: Mean-squared error of entire output vector during training
HOLDEN-XT is hidden during the first half of its training process as its design, and therefore results, is entirely equivalent to that of the HOLDEN configuration.

Figure 5.11 shows similarity through overlap between the HOLDEN and HOLDEN-XL configurations throughout their training period. Additionally, during the extra training period of the HOLDEN data series, here equivalent to that of the HOLDEN-XT configuration, the mean-squared error remains relatively unchanged. This further reemphasizes the similarity in accuracy argued in Section 5.3.

As a reminder; the green triangle HOLDEN-XT was trained for twice the number of epochs than the other network configurations, which results in a twice as long output error result.

For the AVA configuration, one can visibly determine a larger mean-squared error during the entirety of its training process compared to the Holden configurations. This is visualized through the fact that the purple crossed AVA data series lies relatively significantly above the others in Figure 5.11. Additionally, the mean-squared error of the entire output vector appears visibly more irregular between epochs during the training process than that of the Holden configurations.

5.6 Pipeline Overview

This section aims to present the computation time, see Table 5.3, and the data size, see Table 5.4, for each step of the full phase-functioned neural network pipeline, presented in Section 3.3.

Step	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA
Generate Patches	28min	-	-	-
Generate Database	90min	-	-	55min
Network Training	43h	47h	91h	31h
Neural Network	0.38ms	0.50ms	0.38ms	1.5ms

Table 5.3: Computational time required throughout the pipeline
Entries marked '-' share the HOLDEN results.

In summary, for the HOLDEN-XL configuration, Table 5.3 shows that there is little difference in training time when adding a new network layer to the Holden network.

The much longer training time of the HOLDEN-XT configuration was not unexpected, as a result of it being trained for twice the number of epochs, see Section 3.1.3.

Additionally, the table shows that the database generation, and to some extent the network training, is considerably quicker for the AVA configuration. This means that even though the AVA configuration had six times the number of skeleton joints, see Section 3.1.3, the fact that it only had a fourth of the frames compared to the Holden configurations, see Section 3.1.3, resulted in it being trained considerably faster.

Data	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA
Height Fields	134MB	-	-	-
Terrain Patches	606MB	-	-	-
Motion Capture Files	848MB*	-	-	7.62GB*
Phase Labels	7.04MB	-	-	-
Other Labels	56.9MB	-	-	-
Training Database	7.12GB	-	-	10.4GB
Network Weights	122MB	176MB	122MB	374MB

Table 5.4: Size of different data throughout the pipeline
Entries marked '-' share the HOLDEN results
*: Motion Capture Files are in 120Hz.

In summary, Table 5.3 shows the considerable increase in memory size between that of the Holden, to that of the AVA, network configurations. As mentioned in 3.1.3, do note that the AVA database only contains a number of frame data points equal to a quarter of that of the Holden configurations.

Additionally, as discussed previously, given that the HOLDEN-XT configuration differs from the default HOLDEN configuration solely through training time, it is expected that the network weights produced by the two have the same memory size.

In contrast, given that the HOLDEN-XL configuration has more network nodes than that of the HOLDEN configuration, it is expected that there are more weights, requiring more memory, for the former.

5.7 Skinning Visualization

Video recordings of these visualizations are available here [48].

These were the in-engine results of the network orientational outputs after switching the X- and Z-rotations, and inverting the Y- and switched X-rotations.

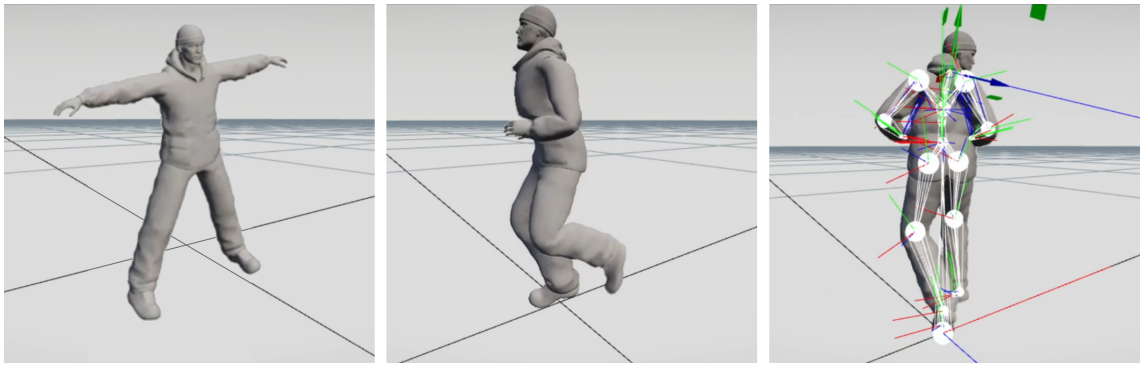

Figure 5.12: HOLDEN positional and orientational output skinned
Left: The skinned HOLDEN character model in default stance.
Middle: HOLDEN output skinned.
Right: HOLDEN output skinned with visible skeleton.

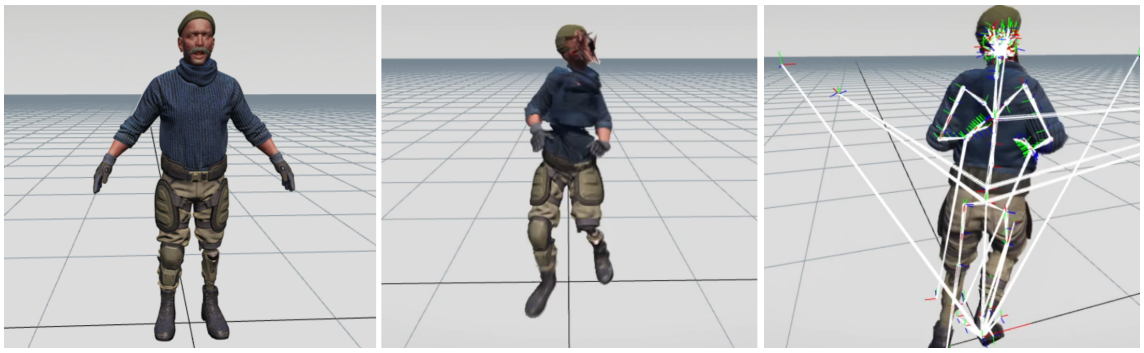

Figure 5.13: AVA positional and orientational output skinned
Left: The skinned AVA character model in default stance.
Middle: AVA output skinned.
Right: AVA output skinned with visible skeleton.

In Figure 5.12 and Figure 5.13, one can see examples of the skeleton joint translation

and orientation outputs the HOLDEN and AVA network configurations produced during their respective responsivity evaluations skinned to 3D character models.

The HOLDEN skinning, at a quick glance, appears correct in general. Occasionally, certain specific joints experience single-frame orientation errors. For an example, the head and torso sometimes rotate over 180 degrees around an axis in a single frame. As of writing, this error is still being investigated.

Similarly, the skinned AVA results also produce certain erroneous orientations, however much more frequently and for more than only two joints. In the middle panel of Figure 5.13, one can see the torso joint being rotated over 180 degrees. As of writing, this error is still being investigated. Additionally, the original skeleton file, on which the motion capture data was retargeted using, was lost and replaced with a new skeleton file for the runtime process. This new skeleton is perfectly equivalent to the old one, except for the facial structure. As such, the facial contortions shown in Figure 5.13 is to be expected.

However even though the character model occasionally appears incorrect, the underlying skeleton still moves correctly. This shows that the positional outputs of the neural networks are correct, however that is not always the case for the orientational. This is presumably a result of the fact that the network is trained in another set of world-axis orientations, see Section 3.3.4, compared to that of the Apex engine. The positional outputs of the neural networks are manually converted in runtime to match that of the engine, hence the apparently correct skeletal output. This inconsistency in world-axis orientations, however, does not affect the results of the responsivity or accuracy evaluations. This is further discussed in Section 6.4.2.

6

Discussion

6.1 Discussing the Research Questions

This section aims to, through discussion, answer the research questions as presented in Section 1.3.

6.1.1 Discussing Responsivity

The research question regarding responsivity asked how much computational time is required for procedural single-character locomotive animations, see Section 1.3. This is answered through testing an engine-integrated solution using the four different network configurations.

The responsivity results presented in Section 5.1 reveals a considerable computational difference between the Avalanche and the Holden network configurations.

This significant distinction in frametime could be explained by the difference in number of skeleton joints. As mentioned in Section 3.1.3, the two skeleton types have 31 and 191 joints, respectively. Since all skeleton joints are fed into the neural network during the runtime prediction process, the width of the networks, and in turn the number of computations each frame, depend on the number of joints.

Since the AVA network required a size of 1'302x512x1'751, whereas the two shallower configurations required only a size of 342x512x311, the number of computations required each frame to propagate the character pose through the network is therefore greatly increased in the former configuration.

A naive, since it assumes sequential computing, way of counting computations in a simple feed-forward network like the one used in this project is to sum the number of inter-layer network connections and the number of biases, like so:

$$\sum_{l} n_l(1 + n_{l+1}) \tag{6.1}$$

Where: 'l' is the index of all non-output network layers, '$l + 1$' is the index of the subsequent network layer following layer 'l', and 'n_l' is the number of nodes *(width)* in layer 'l'.

The number of computations per network configuration using the above formula can be seen in Table 6.1.

	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA
computations	335'501	598'157	335'501	1'566'701

Table 6.1: The number of computations required per network configuration

These data points can then be inserted into a line chart to display the linear relationship between the number of skeleton joints and the frametime, such as Figure 6.1.

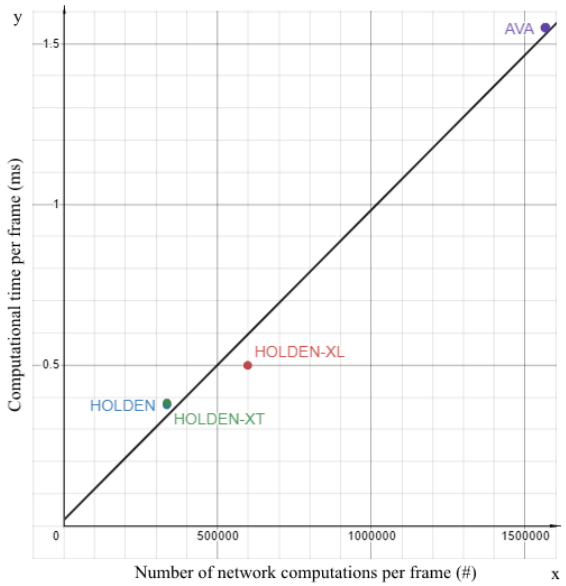

Figure 6.1: Linear regression of computational time over network calculations
Beware; this is a gross simplification simply to show a possible relation between network size and computational load at runtime.
Regression line: $y \approx 9.6x \cdot 10^{-7} + 0.018$

6.1.2 Discussing Accuracy

The second subsidiary research question asked how accurate the generated locomotive character animations are to the original animation data, see Section 1.3. This is answered here through relative comparisons between the network configurations.

DIFFERENCE BETWEEN THE HOLDENS

The three Holden network configurations maintain close resemblance with equivalent results throughout the error evaluation process. This can be seen through consistent overlap between the data shown in both Figure 5.6, Figure 5.7, and Figure 5.8, and with a near zero divergence metric, as presented in Section 3.1.1. Additionally, the mean-squared-error evaluation during the training process also shows no significant difference between the Holden configurations, see Figure 5.11.

Through these statistics, one can conclude that there was insignificant gain in either doubling the number of hidden layers in the neural network, as was the case for the HOLDEN-XL configuration, or doubling the duration of the training process, as was the case for the HOLDEN-XT configuration.

Additionally, the lack of improvement in results between the default HOLDEN and the long trained HOLDEN-XT configuration is evidence that the original implementation by Holden et al. does not suffer from underfitting, see Section 2.1.4 and Section 3.1.3.

AVA VERSUS AVA-M

Additionally, the AVA and AVA-M data series closely follow the same curvature in Figure 5.6, however translated vertically. This curvature is distinctively different from that of the Holden configurations.

Given that the AVA data series has consistently less of an error than the AVA-M series, one can deduce that the masked joints, those not present in AVA-M, give rise to a constantly lower error in comparison.

The mean absolute error definition used in this project includes a normalizing denominator, see Equation 3.1, as to allow the error to be relative to the ground truth positional values. For an example, an error of 0.01 equates to a predicted joint position that is 1% off the ground truth, see Section 3.1.2. However, this means that further away joints, joints at positions with high positional values, would need a larger absolute error to produce the same effective evaluated error than that of a joint closer to the axis origin.

As a concrete example, consider a joint a with a ground truth position at $a_t = 100$. For this joint to contribute with an error of 0.1, the predicted position would need to be at $a_p = 101$. However, if we consider a different joint b with a ground truth position at $b_t = 1$, the predicted position would need only to be $b_p = 1.01$ to contribute with the same error.

Out of the the masked AVA joints, a considerable number of these are highly concentration within the character head and face, see Section 3.1.3. As these joints are further away from the axis origin, the discrepancy in error between AVA and AVA-M may be a result of the inherent relativity design of the mean absolute error definition used in this project. As such, AVA-M might be more suitable for comparisons with the Holden configurations.

AVA-M VERSUS THE HOLDENS

Also, even though the mean-absolute prediction error at runtime of the AVA configuration was significantly lower than that of the Holden configurations, see Figure A.5, the mean-squared prediction error during the training process of the former was considerably higher than that of the latters, see Figure 5.11.

This difference could be a result of the fact that the runtime evaluation only considered the three-dimensional skeleton joint positions, wheras the training process evaluation considered the entire output vector. This means that it is possible that

the AVA network configuration is comparatively much better at joint position prediction than at predicting the other output features, such as: joint orientations and velocities, and sample point positions and trajectories, see Section 3.2.3.

Additionally, another potential reason behind this difference is the fact that the two evaluation processes used different error definitions; mean-absolute error and mean-squared error. The fact that the absolute error was lower than the squared error could be an indicator that the data, in this case the accuracy of the AVA network, had many extreme outliers. This, as the mean-squared error definition squares the per data point error, meaning that errors smaller than one get reduced and errors larger than one get amplified, compared to that of the mean-absolute error definition.

However, a potential source of the difference in error between the AVA and Holden configurations was the choice of evaluation frames per motion capture file, see Section 3.1.2. As a result of insufficient system memory for keeping each joint position in each frame in memory, a decision to only consider the first 500'000 three-dimensional joint positions in each datafile was made. This in combination with the fact that the AVA configuration only considered every fourth frame and that the AVA skeleton had more than six times the skeleton joints, see Section 3.1.3, means that it is probable that if the maximum joint position number is met, the two configurations would consider different blocks of frames within the motion capture data.

In other words, if the Holden configurations reached the memory limit of 500'000 joint positions, it will have considered the first $\frac{500'000}{31} \approx 16'000$ frames. On the other hand, if the AVA configuration reached the memory limit of 500'000 joint positions, it would have instead only considered the first $\frac{500'000 \cdot 4}{191} \approx 10'000$ frames. If the different network configurations was evaluated on a different set of motion frames, then that could make for an unfair comparison.

6.1.3 Discussing Architecture

The final subsidiary research question asked about how the phase-functioned neural network architecture presented in Holden et al. could be improved, see Section 1.3.

An initial hypothesis may be that allowing a neural network to train for a longer period of time, or to have a deeper network, may increase the predictive accuracy of the network. However, when comparing the accuracy results between that of the Holden configurations, see Section 6.1.2, one can conclude that is not the case, at least not for this particular model. This was shown through insignificant difference in accuracy between the three network configurations; default *(HOLDEN)*, double the hidden layers *(HOLDEN-XL)*, and double the training time *(HOLDEN-XT)*, see Section 6.1.2.

On the other hand, not just that there was no strictly positive sides of any of the two altered Holden configurations, there was still strictly negative ones. For an example, given that the HOLDEN-XL configuration required an additional layer of network weights and biases, it was shown to require both longer computational time, see

Figure 5.1 and Table 5.3, and more memory, see Table 5.4, at runtime. For the HOLDEN-XT configuration, the only clear downside of it, compared to the default HOLDEN network, was the increase in training time, see Table 5.4. As such, the altered Holden configurations have been shown to only perform equally, or worse, compared to that of the default HOLDEN configuration.

The fourth network configuration, AVA, that in one aspect may be used to evaluate the generalizability of the phase-functioned neural network approach, had its own fair share of issues. For the responsivity evaluation, the AVA network was shown to be significantly much slower to evaluate at runtime, compared to all other network configurations. This, however, may be quite unsuprising as it required many more computations for each prediction, see Figure 6.1, as a result of it being based on a character skeleton using more than six times the joints than that of the skeleton used for the other network configurations.

On the other hand, the accuracy evaluation implied an increase in prediction accuracy for the AVA configuration. However, it is questionable whether this comparison was truly fair. Certain special considerations were taken to avoid memory overflows as a result of, both directly and indirectly, the greatly increased number of skeleton joint. These include introducing a maximum number of considered frames during the accuracy evaluation, see Section 3.1.2, and having the network train on only a quarter of the training data frames, see Section 3.1.3.

Additionally, as shown in Section 5.2, the actual orientational outputs of the AVA configuration is erroneous to the extent that they are practically useless within the engine. This is assumed to be a result a side-effect of the inconsistent world-axis orientations, see Section 3.3.4.

To produce a more fair comparison between the AVA and the Holden network configurations, certain alterations in the AVA network configuration should be considered. These include only utilizing a subset of the skeleton joints, similar to the AVA-M subset - see Section 3.1.3, in the AVA skeleton that are equivalent to the Holden skeleton, see Section 6.4.1 , and implementing consistent world-axis orientations throughout the process, see section 6.4.2. This is further discussed in Section 6.4.

6.2 Ethical Considerations

Given that this project encompassed work with proprietary software, the Apex game engine, a certain level of consideration regarding confidentiality was taken, as previously mentioned in Section 1.4. The disclosure or misuse of sensitive information pertaining to the Avalanche Studios Group, including that of any of their systems or projects, may indirectly damage both the company and its intellectual properties. To ensure the personal liability of the thesis author towards any such consequences, a non-disclosure clause regarding sensitive information was signed as part of the collaborative contract. As such, no intricate details about the engine, in either text or photo, is included as part of this report.

As far as potential repercussions regarding the research content of this project, no

further ethical considerations were taken into direct consideration. Theoretically, revolutionary development within this field of research may lead to to industry-wide adoption of new techniques, nullifying certain current artist and engineering employment positions. Such a substantial alteration of the standard character animation pipeline might remove the need for the currently adapted animation state machines, and therefore eliminate certain common work tasks. However, a new animation pipeline, that uses neural networks, would still require plenty of similar talents, in addition to the need of new machine learning programmers.

Much consideration must be taken during the planning and recording of the motion capture data, or what other data is to be used during the training process. As such, there might be a shift in the work force towards the number of motion capture specialists, in favor of the number of animators. Animation programmers will still be needed to attach the neural network solution to the project. Additionally, even if a project were to shift to a neural network based animation engine, there might still be the need of standard animations. Certain specific animations, such as in object interaction or those used in cutscenes and cinematics, may utilize hand-made animations to achieve the highest level of detail in the bodily expression.

6.3 Takeaways

This section aims to summarize some of the most important learnings produced by this project on the topic of neural network integration into a modern game engine, or integration work in general. As such, the goal of these takeaways is to be broader lessons, based on the concrete scenario of this thesis project, valuable even within other contexts.

6.3.1 Integration Contextualization

Some of the obstacles encountered during thesis work, as a result of mismatching assumptions on the implementation, was handled in a way such as to avoid altering the to-be-integrated codebase and to rather adapt the solution at runtime. One such examplifying situation is where even though the original phase-functioned neural network pipeline by Holden et al., see Section 3.3, used a different set of world-axis orientations to that of the Apex engine, see Section 3.3.3 and Section 6.4.2, it was kept unaltered.

The original reasoning behind this was twofold. Firstly, as this thesis project aimed to only integrate this work within the specific context of the Apex game engine, it was assumed that manually taking care of an issue such as the inconsistent world-axis orientations would be a more rapid way of handling the problem, rather than attempting to make editions throughout the original phase-functioned neural network pipeline. Secondly, this decision was taken as to minimize the potential locations of integration bugs. If this thesis project included a fully altered pipeline, there would be many more places in the code base in where issues and programming mistakes could arise, given that the original implementation by Holden et al. was assumed to be fully functional.

However, even given these reasonings, there would be much to gain from fully contextualizing the to-be-integrated solution before initiating an integration process. As such, similar future work could be recommended to have further preemptive research on the integral differences between the two solutions of an integration procedure, identifying in which aspects alterations to the to-be-integrated solution may be necessary.

6.3.2 Integration Placement

Similarly to how certain steps were taken to as most rapidly be able to integrate the work by Holden et al. into the Apex engine by the inelegant solution of solving the world-axis orientation mismatch at runtime, certain shortcuts were taken on the other side of the integration work. The location in the Apex game engine where the phase-functioned neural network solution developed as part of this thesis work resides, was one where the integration could occur most effortlessly.

As such, the integrated solution was located within the game engine editor where it could be easily accessed and debugged by programmers. However, given the nature of its location, certain other parts of the engine became separated to the neural network solution. Therefore, the current implementation of the network integration has no access to neither the user input system nor the terrain sampling system, causing such features to be unavailable to the network solution.

A more rigorous integration process could potentially have unlocked the functionality of these systems, allowing the network to be able to produce terrain-adaptive animation as a result of being able to be fed the raycasted height data of nearby terrain, for an example, see Section 6.4.3.

6.3.3 Implementation Expertise

Something that may be overlooked in integration work is for there to be representatives within the integration team that together are both read-up on the work to be integrated and deeply experienced within the solution into which the integration will occur.

Within the context of this thesis project, both sides of the integration software coin, but especially the Apex engine, was brand new to the author. As such, much time was spent tackling with obstacles inherent to the engine that was novel to the author. Additionally, some less optimal integrational design choices could be attributed to this fact.

Even though the thesis mentor was able to provide guidance within this area, there is only so much such a person can provide while still maintaining their expected daily tasks and obligations.

6.3.4 Equipment Suitability

Certain areas within computer science are more computationally dependent than others. One of these is working with neural networks. Even though networks can be fluently used at runtime, the actual training process is extensive and may lock up a workstation for days. As such, it should be recommended to have designated training stations where the networks can complete this process without disturbing other operations.

Additionally, one limitation that became evident as part of this thesis work was the demand on system memory as a result of massive training databases. This issue can of course be resolved in other ways, such as implementing a runtime skeleton retargeting system as to avoid bloating the network pipeline when using a skeleton with a greatly increased number of joints, see Section 6.4.1. However, for this thesis project, simply having a training station with larger system memory capacity would resolve any such issues without having to alter the training process itself.

6.3.5 Neural Network Rigidity

Another inherent characteristic that became evident when working with the the neural network implementation of this project was their lack of flexibility.

A certain level of care must be taken, such as making sure that the framerate of the neural network during the training process matches with that in which it is expected to operate at runtime. Similarly, the joint skeleton, both in numbers and position, must be maintained when transitioning between the training and runtime applications, especially when considering skinning 3D models to the skeletons.

Therefore, other systems, such as inverse-kinematics or other pose-affecting features, cannot be allowed to operate on the character pose if that pose is part of a neural network feedback-loop.

The integration implementation produced during this project alleviated this issue by allowing the neural network to hold an internal copy of the character pose. This means that the outputting pose is able to be kept unaltered, for being inputted in the next frame, allowing other systems to operate on another copy, the one that the network broadcasts.

6.4 Future Work

This section presents three different avenues of potential improvements, or further advancements, on the topic of this project: runtime skeleton retargeting, consistent world axis orientations, and full pipeline integration.

6.4.1 Runtime skeleton retargeting

Many issues with the AVA network configuration are likely inherent to the much greater number of joints in the AVA skeleton, compared to what was necessary in

the Holden network configurations. These include the incredible computational time required during runtime, see Section 5.1, and the steps taken as a result of insufficient system memory, such as the max ceiling of evaluation frames, see Section 3.1.2 and the quartering of training frames, see Section 3.1.3.

Many of the joints used in the AVA skeleton are, by design, not meant to be used for locomotive animations. Many joints that exist in the skeleton are there for other purposes, such as for deformation or inverse-kinematics. Additionally, a considerably large portion of the joints in the skeleton are used to represent the structure and muscles of the face of the character.

A very clear avenue of improvement in the methodology of this report would be to only consider a subset of the joints in the AVA skeleton, such as the joints used in the AVA-M data series, see Section 3.1.3. This would not only mean that the network potentially could more easily discern the locomotive motion of the character, but would also mean that all frames of the motion capture data may be used in both the training and for the evaluation.

This means that a runtime retargeting system would need to be written that sets all updated joint values in the actual skeleton that is unaccounted for in the network. Otherwise, for an example, all the joints in the character face would stay perfectly still while the character moves, as these are not updated by the engine. Such a retargeting system would need to store a resting character pose, such that it can update the child joints accordingly after the network updates the chosen parent joints. In other words, when the neural network moves the singular head joint, the runtime retargeting system would need to correctly update the face joints to match that of the head.

Additionally, such a retargeting system would be highly useful in general if a neural network solution would be applied in a real-world project as the generated animations must be able to be reproduced across multiple similar skeletons. It is unrealistic to have separate neural networks, and separately recorded motion capture data, trained for each character, given that there is expected to be variations within the design of the physique of different characters.

6.4.2 Consistent world-axis orientations

As mentioned in Section 1.4 and explained in Section 3.3.4, the original work by Holden et al. assumes a left-handed world-axis orientation, different from the right-handed orientation used in the Apex game engine.

During this project, as the neural networks was trained using the motion capture data provided by Holden et al., the clash in world-axis orientation required much care. As such, the final runtime package includes multiple conversions at runtime between the right-handed character pose and the left-handed neural network.

This handling of the axis inconsistencies was done at runtime, where both the inputting, and outputting, data was flipped such that it was translated correctly between the world-axis orientations, depending on where the data was to be used; in

the neural network representation or in the engine. Additionally, these conversions were made such that the Holden configurations produced accurate positional and orientational outputs, without taking further care for the AVA configuration. That the AVA configuration may have been in need of special care to resolve the world-axis inconsistency issue could be reason why it produced such imperfect results, see Section 5.7.

If a motion capture data set was used that featured right-handed world-axis orientation, such that it is consistent for the game engine at runtime, the visual orientational errors in the skinnings could be potentially resolved, or at least more easily studied.

6.4.3 Full pipeline integration

The runtime neural network solution produced as part of this project was integrated into the Apex engine, as far as the solution executing as an internal part of the engine running.

However, this is still a far step away from a full integration into the runtime pipeline. The current network package is positioned where it would be most easy to be put; a part of the engine used by programmers during the development process. Ideally, the package would instead by integrated within the actual runtime engine pipeline.

Such a full integration would not just allow for more realistic evaluation of the in-game performance of the neural network solution, but would also open it up to be combined with multiple other engine sub-systems.

Currently, character input, in the form of where the character is moving towards, with what movement style, and at what speed, is deterministically determined by the obstacleless track course, as defined in Section 3.1.1. With a full integration, the neural network solution could be able to poll the user input system, such that the character could be controlled at runtime with a gamepad or mouse and keyboard.

Similarly, a full integration would allow the network package to utilize raycasts to sample the height of the terrain, such that the terrain adapting locomotive animations may be unlocked without using hard-coded simulative values.

Finally, by moving the network package to the runtime system, one would be able to combine animation techniques. In such a case, the neural network solution may be used to produce the underlying locomotive character animations, and then standard character animations may be blended on top to provide more intricate character animations, such as interacting with, or holding, equipped items and scene objects.

7

Conclusions

This section aims to provide closure to this thesis by firstly summarizing the answers, see Section 6.1, to the research questions, see Section 1.3. Additionally, this section features some short closing words where the outcomes of this thesis is attempted to be put into a broader context.

7.1 Summary

In an attempt to evaluate the phase-functioned neural network architecture presented in Holden et al. [5], and to investigate how it could be further improved upon, this project evaluated different versions of the network, see Section 1.3. These network configurations were to be integrated into the Apex game engine, provided by Avalanche Studios Group.

The first evaluated network was the default control configuration HOLDEN - the default network solution as presented in Holden et al. [5]: a network with a single hidden layer, see Section 3.3.3, trained for 2'000 epochs. For comparison, two otherwise direct replicas of this network was used: HOLDEN-XL, with two hidden layers, and HOLDEN-XT, which trained for 4'000 epochs.

The HOLDEN-XL network, as a result of it needing more network weights and computations for its additional hidden layer, required a larger computational and memory footprint. The HOLDEN-XT configuration, however, required a twice as long training process, but was at runtime not more cumbersome than the default HOLDEN network.

In summary, the two altered Holden networks did not yield any considerable improvement to that of the original network; especially not considered their respective downsides.

Additionally, a fourth network configuration, AVA, was evaluated that used a different character skeleton. The original motion capture data used in Holden et al. [5] was retargeted to this skeleton for use during the training and evaluation process.

The AVA network, based on a skeleton with more than six times the joints than the one used in the Holden configurations, required a significantly larger computational and memory footprint. To avoid system overflow, this great increase in memory was attempted to be compensated through sub-sampling of the motion capture data. As

such, the AVA network was feed with only a quarter of the motion capture frames used in the Holden configurations. Additionally, the great increase in memory usage lead to a cap in how many frames were considered in each motion capture data file during the evaluation process.

The final results of the AVA configuration was a network that was much less responsive, had questionable accuracy, and unstable joint orientations. However, these issues are not considered testaments against the applicability of the phase-functioned neural network as they may be side-effects of unsuitable conversions during integration, such as the overabundant number of skeleton joints or the mismatching of world-axis orientations, see Section 3.3.4.

7.2 Final Words

This thesis project aimed to evaluate a specific machine learning approach, phase-functioned neural networks, to locomotive character animations. By continuing on previous work made by Holden et al. [5], this thesis was able to evaluate the default phase-functioned neural networks, and similar configurations with minor alterations, rudimentarily integrated into a professional real-world game engine.

Additionally, this project also attempted to evaluate the generalizability of the phase-functioned neural network by attempting to utilize this prediction model with a new skeleton. Through this process, multiple affecting factors were identified that are in need of addressing to fully be able to generalize this algorithm into a real-world application, see Section 6.4.

Compared to the traditional approach to character animations, a machine learning perspective, like the phase-functioned neural network considered in this report, may require a massive readjustment of the entire animations pipeline. Through applying previous research within this topic, and in combination with a professional game engine, this thesis aimed to contribute to this research field - hopefully producing conclusions of value through both quantifiable results, summarized learnings, and suggestions for future work, specifically to the Avalanche Studios Group.

It is likely that as further achievements are made within the field of machine learning for character animations, or for machine learning in game development in general, various major actors may move to evaluate the possibility of integrating these strides into their products, in an effort to potentially reduce development costs and time.

Bibliography

[1] Ubisoft Entertainment, "Ubisoft La Forge," 2020. [Online]. Available: https://montreal.ubisoft.com/en/our-engagements/research-and-development/ Accessed on 12 December 2020.

[2] Electronic Arts Inc., "SEED // Search for Extraordinary Experiences Division," 2018. [Online]. Available: https://www.ea.com/seed Accessed on 12 December 2020.

[3] S. Starke, "AI4Animation: Deep Learning, Character Animation, Control," 2020. [Online]. Available: https://github.com/sebastianstarke/AI4Animation Accessed on 12 December 2020.

[4] Avalanche Studios Group, "Avalanche Studios Group," 2020. [Online]. Available: https://avalanchestudios.com/ Accessed on 12 December 2020.

[5] D. Holden, T. Komura, and J. Saito, "Phase-Functioned Neural Networks for Character Control," 2017. [Online]. Available: http://theorangeduck.com/media/uploads/other_stuff/phasefunction.pdf Accessed on 12 December 2020.

[6] Avalanche Studios Group, "OUR TECHNOLOGY," 2020. [Online]. Available: https://avalanchestudios.com/technology Accessed on 12 December 2020.

[7] A. Chandra, "McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron," 2018. [Online]. Available: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 Accessed on 29 January 2021.

[8] C. Hanses, "Activation Functions Explained - GELU, SELU, ELU, ReLU and more," 2019. [Online]. Available: https://mlfromscratch.com/activation-functions-explained/ Accessed on 28 January 2021.

[9] Q. Chen, "Deep learning and chain rule of calculus," 2018. [Online]. Available: https://medium.com/machine-learning-and-math/deep-learning-and-chain-rule-of-calculus-80896a1e91f9 Accessed on 24 March 2021.

[10] V. Bushaev, "Adam — latest trends in deep learning optimization." 2018. [Online]. Available: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c Accessed on

29 January 2021.

[11] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," 2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ Accessed on 14 April 2021.

[12] Stanford Vision Lab, Stanford University, Princeton University, "ImageNet," 2021. [Online]. Available: https://image-net.org/index.php Accessed on 4 May 2021.

[13] W. Zeng, "Toward human-centric deep video understanding," 2020. [Online]. Available: https://www.researchgate.net/publication/338566118_Toward_human-centric_deep_video_understanding Accessed on 4 May 2021.

[14] The Deepmind Team, "AlphaFold," 2020. [Online]. Available: https://deepmind.com/research/case-studies/alphafold Accessed on 4 May 2021.

[15] Kyle Wiggers, "Google details how it's using AI and machine learning to improve search," 2020. [Online]. Available: https://venturebeat.com/2020/10/15/google-details-how-its-using-ai-and-machine-learning-to-improve-search/ Accessed on 4 May 2021.

[16] Barak Turovsky, "Found in translation: More accurate, fluent sentences in Google Translate," 2016. [Online]. Available: https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/ Accessed on 4 May 2021.

[17] The Deepmind Team, "AlphaGo," 2017? [Online]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far Accessed on 4 May 2021.

[18] ——, "AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning," 2019, [Online]. Available: https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learningAccessedon4May2021.

[19] The OpenAI Team, "OpenAI Five," 2019? [Online]. Available: https://openai.com/projects/five/ Accessed on 4 May 2021.

[20] X. Matos, "Meet the computer that's learning to kill and the man who programmed the chaos," 2014. [Online]. Available: https://www.engadget.com/2014-06-06-meet-the-computer-thats-learning-to-kill-and-the-man-who-progra.html Accessed on 7 May 2021.

[21] M. Robbins, "Using Neural Networks to Control Agent Threat Response," 2016. [Online]. Available: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter30_Using_Neural_Networks_to_Control_Agent_Threat_Response.pdf Ac-

cessed on 7 May 2021.

[22] J. H. Kim and R. Wu, "Leveraging Machine Learning for Game Development," 2021. [Online]. Available: https://ai.googleblog.com/2021/03/leveraging-machine-learning-for-game.html Accessed on 7 May 2021.

[23] J. Liu *et al.*, "Deep Learning for Procedural Content Generation," 2020. [Online]. Available: https://arxiv.org/pdf/2010.04548.pdf Accessed on 7 May 2021.

[24] E. Alonso *et al.*, "Deep Reinforcement Learning for Navigation in AAA Video Games," 2020. [Online]. Available: https://montreal.ubisoft.com/en/deep-reinforcement-learning-for-navigation-in-aaa-video-games/ Accessed on 7 May 2021.

[25] F. G. Harvey *et al.*, "Robust Motion In-Betweening," 2020. [Online]. Available: https://montreal.ubisoft.com/en/automatic-in-betweening-for-faster-animation-authoring/ Accessed on 7 May 2021.

[26] D. Holden, "A data-driven physics simulation based on Machine Learning," 2019, [Online]. Available: https://montreal.ubisoft.com/en/ubisoft-la-forge-produces-a-data-driven-physics-simulation-based-on-machine-learning/Accessedon7May2021.

[27] J. Oliver, "US and China back AI bug-detecting projects ," 2018. [Online]. Available: https://www.ft.com/content/64fef986-89d0-11e8-affd-da9960227309 Accessed on 7 May 2021.

[28] D. Holden, "Robust Solving of Optical Motion Capture Data by Denoising," 2018. [Online]. Available: https://montreal.ubisoft.com/wp-content/uploads/2018/05/neuraltracker.pdf Accessed on 7 May 2021.

[29] H. Zhang *et al.*, "Mode-Adaptive Neural Networks for Quadruped Motion Control," 2018. [Online]. Available: https://github.com/sebastianstarke/AI4Animation/raw/master/Media/SIGGRAPH_2018/Paper.pdf Accessed on 12 December 2020.

[30] S. Starke, "[SIGGRAPH 2018] Mode-Adaptive Neural Networks for Quadruped Motion Control," 2018. [Online]. Available: https://youtu.be/uFJvRYtjQ4c?t=269 Accessed on 25 January 2021.

[31] H. Zhang *et al.*, "Neural State Machine for Character-Scene Interactions," 2019. [Online]. Available: https://github.com/sebastianstarke/AI4Animation/blob/master/Media/SIGGRAPH_Asia_2019/Paper.pdf Accessed on 12 December 2020.

[32] S. Starke, "[SIGGRAPH Asia 2019] Neural State Machine for Character-Scene Interactions," 2019. [Online]. Available: https://youtu.be/7c6oQP1u2eQ?t=393 Accessed on 25 January 2021.

[33] S. Starke *et al.*, "Local Motion Phases for Learning Multi-Contact Character Movements," 2020. [Online]. Available: https://github.com/sebastianstarke/AI4Animation/blob/master/Media/SIGGRAPH_2020/Paper.pdf Accessed on 12 December 2020.

[34] S. Starke, "[SIGGRAPH 2020] Local Motion Phases for Learning Multi-Contact Character Movements," 2020. [Online]. Available: https://www.youtube.com/watch?v=Rzj3k3yerDk Accessed on 25 January 2021.

[35] S. Clavet, "Motion Matching and The Road to Next-Gen Animation," 2016. [Online]. Available: https://www.gdcvault.com/play/1023280/Motion-Matching-and-The-Road Accessed on 12 December 2020.

[36] D. Holden *et al.*, "Learned Motion Matching," 2020. [Online]. Available: https://static-wordpress.akamaized.net/montreal.ubisoft.com/wp-content/uploads/2020/07/09154101/Learned_Motion_Matching.pdf Accessed on 12 December 2020.

[37] Ubisoft La Forge, "Learned Motion Matching," 2020. [Online]. Available: https://youtu.be/16CHDQK4W5k?t=92 Accessed on 25 January 2021.

[38] PyMC3, "Theano," 2020. [Online]. Available: https://github.com/Theano/Theano Accessed on 29 January 2021.

[39] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, accessed on 14 April 2021. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[40] OyMC Developers, "Theano, TensorFlow and the Future of PyMC," 2018. [Online]. Available: https://pymc-devs.medium.com/theano-tensorflow-and-the-future-of-pymc-6c9987bb19d5 Accessed on 14 April 2021.

[41] Eigen, "Eigen's Tuxfamily Main Page," 2021. [Online]. Available: https://eigen.tuxfamily.org/index.php Accessed on 30 April 2021.

[42] Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/ Accessed on 30 April 2021.

[43] K. Claypool and M. Claypool, "On frame rate and player performance in first person shooter games," 2007. [Online]. Available: https://web.cs.wpi.edu/~claypool/papers/fr/fulltext.pdf Accessed on 4 May 2021.

[44] S. Wu, "3 Best metrics to evaluate Regression Model," 2020. [Online]. Available: https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b Accessed on 4 May 2021.

[45] D. Holden, "Phase-Functioned Neural Networks for Character Control," 2017. [Online]. Available: http://theorangeduck.com/page/

phase-functioned-neural-networks-character-control Accessed on 26 January 2021.

[46] Yoshiboy2, "Phase-Functioned Neural Networks for Character Control," 2017. [Online]. Available: https://youtu.be/Ul0Gilv5wvY Accessed on 27 January 2021.

[47] J. Li and S. Chen, "The Cubic α-Catmull-Rom Spline," 2016. [Online]. Available: https://mlfromscratch.com/activation-functions-explained/ Accessed on 28 January 2021.

[48] J. Segerstedt, "Videos of thesis results," 2021. [Online]. Available: https://drive.google.com/drive/folders/1UDsGAKUhxKyQEd8yinwXLMOx1cRCvY-d?usp=sharing Accessed on 6 June 2021.

List of Figures

List of Tables

A

Apendix

A.1 .bvh Interpolator

To allow for easy conversion between framerate, given that the project needed to at least be able to interpolate 30Hz .bvh files to 60Hz/120Hz .bvh files, see 3.1.3, a Python scrip able to convert .bvh files to any given target framerate value was written. When run, this script takes all files in a given source folder, interpolates them, and stores the results in another given target folder.

```
# ----- bvh_interpolator.bvh -----
# This script takes all .bvh files in 'source_folder'...
# ...converts the framerate to 'target_fps' and...
# ...stores the resulting .bvh:s in 'target_folder'.

# Changable Parameters
source_folder = "source"
target_folder = "target"
target_fps = 60

########################################################
########################################################

# Imports
import os   # .listdir()
import re   # .split()
import math # .ceil()

# returns 'value', clamped between 'other_a' and 'other_b' ('other_a' < 'other_b')
def clamp(value, other_a, other_b):
    return max(other_a, min(other_b,value))

# returns 'value', clamped between 'other_a' and 'other_b'
def clamp_bi(value, other_a, other_b):
    return clamp(value, other_a, other_b) if other_a < other_b else clamp(value, other_b, other_a)

# returns linear interpolation between 'a' and 'b' with factor 'ratio'
def interpolate(a, b, ratio):
    return a * (1-ratio) + b * ratio

# returns linear interpolation between 'a' and 'b' with factor 'ratio', angles has to be [-180, 180)
def interpolate_angles(a, b, ratio):
    difference = abs(a-b)
    if difference > 170 and difference < 190:
        a = (a+180) % 360

    sensitivity = 90
    if a < -sensitivity and b > sensitivity:
        a += 360
    if b < -sensitivity and a > sensitivity:
        b += 360
    interpolated = (a * (1-ratio) + b * ratio)
    return_value = interpolated % 360 if (interpolated % 360) < 180 else (interpolated % 180) - 180
    return  return_value

# returns list of interpolatd, clamped, and rounded stored frame data correctly formated into row data
def formatList(interpolation_ratio, source_frame_data, prev_source_frame, next_source_frame):
    prev_data = source_frame_data[prev_source_frame]
    next_data = source_frame_data[next_source_frame]
    positions = [interpolate(prev_frame_data, next_frame_data, interpolation_ratio) \
            for prev_frame_data, next_frame_data in zip(prev_data[:3], next_data[:3])]
    orientations = [interpolate_angles(prev_frame_data, next_frame_data, interpolation_ratio) \
            for prev_frame_data, next_frame_data in zip(prev_data[3:], next_data[3:])]
    positions.extend(orientations)
    return positions
```

```
# Main Loop - For each file in source directory...
for filename in os.listdir(source_folder):

    # ------ VARIABLE INITIALIZIATION ------
    target_filename = "./"+target_folder+"/"+filename
    target_frametime = 1.0 / (float)(target_fps)
    target_num_frames = -1
    target_content = []

    source_filename = "./"+source_folder+"/"+filename
    source_frametime = -1.0
    source_num_frames = -1
    source_frames_index = -1
    source_fps = -1
    source_file = open(source_filename, 'r')
    source_content = source_file.readlines()
    source_frame_data = []

    conversion_ratio = -1


    # ------ FIND START OF FRAME DATA ------
    is_at_frames = False
    count = 0
    for line in source_content:

        # ---- STORE SOURCE FRAME DATA INDEX AND METADATA ----
        if not is_at_frames:
            if "Frames:" in line:
                source_num_frames = (int)((re.split(r'(\S[^:]+):\s*(.*\S)', line))[2])
            if "Frame Time:" in line:
                source_frametime = (float)((re.split(r'(\S[^:]+):\s*(.*\S)', line))[2])
                source_fps = int(1.0 / source_frametime)
                target_num_frames = (int)(source_frametime * source_num_frames / target_frametime)
                conversion_ratio = target_frametime / source_frametime
                is_at_frames = True
                source_frames_index = count+2
            target_content.append(line[:-1])

        # --- STORE FRAME DATA --
        if is_at_frames and count >= source_frames_index-1:
            source_frame_data.append([float(i) for i in line.split()])

        count += 1

    # ----- INTERPOLATE FRAME DATA -----
    for target_frame in range(0, target_num_frames):
        prev_source_frame = (int) (min(source_num_frames-2, max(0, target_frametime * target_frame \
        / source_frametime)))
        next_source_frame = (int) (min(source_num_frames-1, 1 + prev_source_frame))
        prev_source_time = prev_source_frame * source_frametime
        next_source_time = next_source_frame * source_frametime
        target_time = target_frame * target_frametime
        interpolation_ratio = clamp((target_time - prev_source_time) \
        / (next_source_time - prev_source_time), 0, 1) if target_time - prev_source_time != 0 else 0
        target_content.append(formatList(interpolation_ratio, source_frame_data, \
        prev_source_frame, next_source_frame))

    # ----- WRITE TARGET METADATA -----
    for line_index in range(0, len(target_content)):
        if "Frames:" in target_content[line_index]:
            row = re.split(r'(\S[^:]+):\s*(.*\S)', target_content[line_index])
            target_content[line_index] = "Frames: " + str(target_num_frames)
        if "Frame Time:" in target_content[line_index]:
            row = re.split(r'(\S[^:]+):\s*(.*\S)', target_content[line_index])
            target_content[line_index] = "Frame Time: " + str(target_frametime)


    source_file.close()

    # ----- WRITE TO TARGET FILE -----
    target_file = open(target_filename, 'w')
    target_file.writelines([re.sub('\,|\[|\]', '', "%s\n" % item)  for item in target_content])
    target_file.close()

    print("SUCCESS - Converted '"+filename+"' from "+str(source_num_frames)+"@"+str(source_fps)+"Hz to \
    "+str(target_num_frames)+"@"+str(target_fps)+"Hz")


print("TERMINATED SUCCESSFULLY")
```

Figure A.1: The .bvh Interpolator script

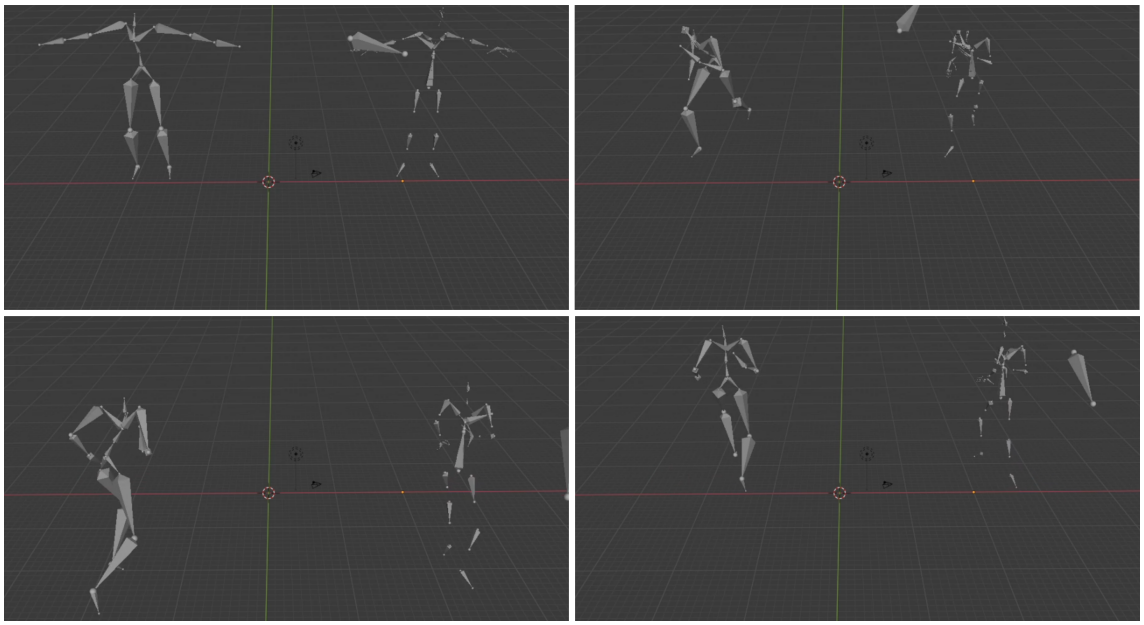

Figure A.2: Side-by-side comparisons of the original and retargeted .bvh skeletons
Left: Original .bvh skeleton as defined by Holden et al. [45].
Right: Retargeted and interpolated .bvh AVA skeleton.

A.2 Filenames

Index	Filename	Index	Filename
0	LocomotionFlat01-000	40	NewCaptures04-000
1	LocomotionFlat01-000-mirror	41	NewCaptures04-000-mirror
2	LocomotionFlat02-000	42	NewCaptures05-000
3	LocomotionFlat02-000-mirror	43	NewCaptures05-000-mirror
4	LocomotionFlat02-001	44	NewCaptures07-000
5	LocomotionFlat02-001-mirror	45	NewCaptures07-000-mirror
6	LocomotionFlat03-000	46	NewCaptures08-000
7	LocomotionFlat03-000-mirror	47	NewCaptures08-000-mirror
8	LocomotionFlat04-000	48	NewCaptures09-000
9	LocomotionFlat04-000-mirror	49	NewCaptures09-000-mirror
10	LocomotionFlat05-000	50	NewCaptures10-000
11	LocomotionFlat05-000-mirror	51	NewCaptures10-000-mirror
12	LocomotionFlat06-000	52	NewCaptures11-000
13	LocomotionFlat06-000-mirror	53	NewCaptures11-000-mirror
14	LocomotionFlat06-001	54	WalkingUpSteps01-000
15	LocomotionFlat06-001-mirror	55	WalkingUpSteps01-000-mirror
16	LocomotionFlat07-000	56	WalkingUpSteps02-000
17	LocomotionFlat07-000-mirror	57	WalkingUpSteps02-000-mirror
18	LocomotionFlat08-000	58	WalkingUpSteps03-000
19	LocomotionFlat08-000-mirror	59	WalkingUpSteps03-000-mirror
20	LocomotionFlat08-001	60	WalkingUpSteps04-000
21	LocomotionFlat08-001-mirror	61	WalkingUpSteps04-000-mirror
22	LocomotionFlat09-000	62	WalkingUpSteps04-001
23	LocomotionFlat09-000-mirror	63	WalkingUpSteps04-001-mirror
24	LocomotionFlat10-000	64	WalkingUpSteps05-000
25	LocomotionFlat10-000-mirror	65	WalkingUpSteps05-000-mirror
26	LocomotionFlat11-000	66	WalkingUpSteps06-000
27	LocomotionFlat11-000-mirror	67	WalkingUpSteps06-000-mirror
28	LocomotionFlat12-000	68	WalkingUpSteps07-000
29	LocomotionFlat12-000-mirror	69	WalkingUpSteps07-000-mirror
30	NewCaptures01-000	70	WalkingUpSteps08-000
31	NewCaptures01-000-mirror	71	WalkingUpSteps08-000-mirror
32	NewCaptures02-000	72	WalkingUpSteps09-000
33	NewCaptures02-000-mirror	73	WalkingUpSteps09-000-mirror
34	NewCaptures03-000	74	WalkingUpSteps10-000
35	NewCaptures03-000-mirror	75	WalkingUpSteps10-000-mirror
36	NewCaptures03-001	76	WalkingUpSteps11-000
37	NewCaptures03-001-mirror	77	WalkingUpSteps11-000-mirror
38	NewCaptures03-002	78	WalkingUpSteps12-000
39	NewCaptures03-002-mirror	79	WalkingUpSteps12-000-mirror

Table A.1: Ordering of motion capture (.bvh) file names
The files are available at the online repository uploaded by Holden et al. [45].

A.3 AVA skeleton joint names

Index	Joint Name	Index	Joint Name
0	Hips	40	fLeftBrowInnerADowner
1	UpperHips	41	fLeftLwrCheek
2	Spine	42	fLeftTemple
3	Spine1	43	fLeftMouthCorner
4	Spine2	44	fLeftEar
5	Spine3	45	fLeftMidCheek
6	Neck	46	fLeftNoseA
7	Head	47	fLeftUprCheek
8	offset-facialOrienter	48	fLeftNoseCheek
9	fJaw	49	fLeftBrowOuterA
10	fLeftChin	50	fLeftBrowMidA
11	fLeftLwrLip2	51	fLeftBrowInnerA
12	fLeftLwrLip	52	fRightUprLip
13	fLeftLwrLipSticky	53	fRightUprLipSticky
14	fRightChin	54	fRightEyeLidBulgeUp
15	fRightLwrLip2	55	fRightEyeLidBulgeDown
16	fMidLwrLip2	56	fRightBackCheek
17	fToungeRoot1	57	fRightCheekCrease
18	fToungeTip	58	fRightUprLid
19	fChinDowner	59	fRightLwrLid
20	fRightLwrLip	60	fRightEye
21	fRightLwrLipSticky	61	fMidUprLip2
22	fMidLwrLip	62	fRightUprLip2
23	fChin	63	fRightJawFlesh
24	fJaw-loose	64	fRightEyeCornerIn
25	fLeftUprLip	65	fRightEyeCornerOut
26	fLeftUprLipSticky	66	fRightBrowOuterADowner
27	fLeftEyeLidBulgeUp	67	fRightBrowMidADowner
28	fLeftEyeLidBulgeDown	68	fRightBrowInnerADowner
29	fLeftBackCheek	69	fMidUprLip
30	fLeftCheekCrease	70	fNoseback
31	fLeftUprLid	71	fRightLwrCheek
32	fLeftLwrLid	72	fRightTemple
33	fLeftEye	73	fRightMouthCorner
34	fLeftUprLip2	74	fRightEar
35	fLeftJawFlesh	75	fRightMidCheek
36	fLeftEyeCornerIn	76	fRightNoseA
37	fLeftEyeCornerOut	77	fRightUprCheek
38	fLeftBrowOuterADowner	78	fRightNoseCheek
39	fLeftBrowMidADowner	79	fRightBrowOuterA

Index	Joint Name	Index	Joint Name
80	fRightBrowMidA	120	FPAdjustorLeftShoulder
81	fRightBrowInnerA	121	LeftShoulder
82	fNoseFront	122	LeftArm
83	fThroat	123	LeftForeArm
84	fApple	124	LeftForeArmRoll
85	FPAdjustorRightShoulder	125	LeftForeArmRoll-DF-1
86	RightShoulder	126	LeftForeArmRoll-DF-2
87	RightArm	127	LeftHand
88	RightForeArm	128	LeftInHandPinky
89	RightForeArmRoll	129	LeftHandPinky1
90	RightForeArmRoll-DF-1	130	LeftHandPinky2
91	RightForeArmRoll-DF-2	131	LeftHandPinky3
92	RightHand	132	LeftHandPinky4
93	RightInHandPinky	133	LeftInHandRing
94	RightHandPinky1	134	LeftHandRing1
95	RightHandPinky2	135	LeftHandRing2
96	RightHandPinky3	136	LeftHandRing3
97	RightHandPinky4	137	LeftHandRing4
98	RightInHandRing	138	LeftHandIndex1
99	RightHandRing1	139	LeftHandIndex2
100	RightHandRing2	140	LeftHandIndex3
101	RightHandRing3	141	LeftHandIndex4
102	RightHandRing4	142	LeftHandMiddle1
103	RightHandIndex1	143	LeftHandMiddle2
104	RightHandIndex2	144	LeftHandMiddle3
105	RightHandIndex3	145	LeftHandMiddle4
106	RightHandIndex4	146	LeftHandThumb1
107	RightHandMiddle1	147	LeftHandThumb2
108	RightHandMiddle2	148	LeftHandThumb3
109	RightHandMiddle3	149	LeftHandThumb4
110	RightHandMiddle4	150	LeftInHandAttach
111	RightHandThumb1	151	LeftHandReverseIKOffset
112	RightHandThumb2	152	LeftArmRoll
113	RightHandThumb3	153	LeftArmRoll-DF-1
114	RightHandThumb4	154	LeftArmRoll-DF-2
115	RightHandReverseIKOffset	155	FPAdjustorWeaponAndCam
116	RightInHandAttach	156	CharacterCam2
117	RightArmRoll	157	CharacterCam2-PostAffector
118	RightArmRoll-DF-1	158	Sternum-weaponRoot
119	RightArmRoll-DF-2	159	Sternum-AimRef

Index	Joint Name
160	ChestToRightHandIK
161	ChestToLeftHandIK
162	Global-SternumRightHandAttach
163	SternumRightHandAttach
164	Global-SternumLeftHandAttach
165	SternumLeftHandAttach
166	LeftHandAttach
167	CharacterCam1
168	CharacterCam1-PostAffector
169	LeftHandIKTarget
170	RightHandIKTarget
171	RightHandAttach
172	HipsToLeftHandIK
173	HipsToRightHandIK
174	LowerHips
175	LeftUpLeg
176	LeftLeg
177	LeftLegRoll
178	LeftKnee
179	LeftFoot
180	LeftToeBase
181	LeftUpLegRoll
182	LeftAss-DF-1
183	RightUpLeg
184	RightLeg
185	RightLegRoll
186	RightKnee
187	RightFoot
188	RightToeBase
189	RightUpLegRoll
190	RightAss-DF-1

Table A.2: Ordering and names of joints in the AVA skeleton

A.4 Responsivity Data

Lap	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA
1	0.388623	0.496462	0.379074	1.473621
2	0.393882	0.478219	0.372569	1.487441
3	0.389684	0.527579	0.387475	1.496202
4	0.362003	0.505760	0.415037	1.456730
5	0.365993	0.489539	0.407157	1.450569
6	0.388100	0.483170	0.350097	1.554174
7	0.360687	0.483187	0.394324	1.487500
8	0.353109	0.516471	0.362927	1.483788
9	0.388259	0.474980	0.376687	1.487613
10	0.357007	0.494261	0.370764	1.521477
11	0.366729	0.509333	0.402152	1.485579
12	0.385496	0.490648	0.389647	1.462317
13	0.338432	0.504941	0.363286	1.471249
14	0.398865	0.486274	0.383576	1.471035
15	0.355426	0.513662	0.360077	1.456486
16	0.387522	0.482759	0.382811	1.449753
17	0.390639	0.517458	0.393618	1.514399
18	0.379178	0.506789	0.374275	1.490249
19	0.385632	0.524149	0.383185	1.397291

Table A.3: Full responsivity results - Average frametime per lap.
Values are in milliseconds.

A.5 Accuracy Data - Means

Index	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA	AVA-M
0	0.0496	0.0477	0.0501	0.0403	0.0505
1	0.0503	0.0475	0.0517	0.0414	0.0536
2	0.0442	0.0370	0.0446	0.0298	0.0388
3	0.0456	0.0391	0.0480	0.0311	0.0415
4	0.0464	0.0396	0.0470	0.0316	0.0420
5	0.0444	0.0379	0.0451	0.0313	0.0427
6	0.0496	0.0457	0.0503	0.0340	0.0413
7	0.0497	0.0467	0.0508	0.0330	0.0430
8	0.0448	0.0431	0.0469	0.0332	0.0415
9	0.0474	0.0465	0.0483	0.0305	0.0383
10	0.0473	0.0449	0.0488	0.0305	0.0429
11	0.0488	0.0460	0.0501	0.0299	0.0433
12	0.0416	0.0354	0.0429	0.0235	0.0303
13	0.0406	0.0364	0.0434	0.0243	0.0318
14	0.0434	0.0373	0.0449	0.0245	0.0324
15	0.0437	0.0393	0.0457	0.0260	0.0344
16	0.0509	0.0492	0.0520	0.0356	0.0449
17	0.0532	0.0514	0.0549	0.0368	0.0476
18	0.0481	0.0448	0.0488	0.0288	0.0406
19	0.0462	0.0433	0.0470	0.0291	0.0398
20	0.0500	0.0474	0.0511	0.0297	0.0406
21	0.0509	0.0489	0.0525	0.0301	0.0412
22	0.0473	0.0457	0.0486	0.0317	0.0440
23	0.0447	0.0437	0.0468	0.0313	0.0389
24	0.0493	0.0450	0.0503	0.0353	0.0458
25	0.0484	0.0451	0.0503	0.0354	0.0460
26	0.0369	0.0395	0.0364	0.0301	0.0383
27	0.0419	0.0432	0.0409	0.0303	0.0426
28	0.0272	0.0304	0.0272	0.0274	0.0388
29	0.0271	0.0304	0.0273	0.0272	0.0400
30	0.0421	0.0441	0.0429	0.0285	0.0379
31	0.0438	0.0429	0.0449	0.0305	0.0429
32	0.0416	0.0468	0.0422	0.0303	0.0392
33	0.0419	0.0462	0.0430	0.0305	0.0404
34	0.0277	0.0296	0.0278	0.0280	0.0395
35	0.0283	0.0301	0.0284	0.0278	0.0416
36	0.0292	0.0316	0.0290	0.0280	0.0403
37	0.0282	0.0304	0.0285	0.0284	0.0422
38	0.0280	0.0296	0.0280	0.0255	0.0363
39	0.0291	0.0312	0.0294	0.0283	0.0434

Index	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA	AVA-M
40	0.0303	0.0333	0.0305	0.0275	0.0383
41	0.0282	0.0305	0.0276	0.0276	0.0404
42	0.0381	0.0412	0.0383	0.0294	0.0383
43	0.0397	0.0430	0.0402	0.0322	0.0454
44	0.0382	0.0408	0.0394	0.0324	0.0419
45	0.0389	0.0424	0.0394	0.0331	0.0439
46	0.0465	0.0497	0.0461	0.0292	0.0386
47	0.0492	0.0522	0.0484	0.0278	0.0341
48	0.0492	0.0528	0.0486	0.0299	0.0372
49	0.0500	0.0522	0.0490	0.0304	0.0404
50	0.0465	0.0465	0.0454	0.0303	0.0394
51	0.0407	0.0430	0.0406	0.0325	0.0443
52	0.0483	0.0512	0.0478	0.0293	0.0381
53	0.0487	0.0516	0.0483	0.0300	0.0397
54	0.0359	0.0395	0.0370	0.0289	0.0383
55	0.0349	0.0374	0.0360	0.0287	0.0391
56	0.0373	0.0413	0.0385	0.0273	0.0341
57	0.0382	0.0413	0.0392	0.0288	0.0374
58	0.0395	0.0419	0.0408	0.0313	0.0403
59	0.0402	0.0432	0.0414	0.0327	0.0448
60	0.0345	0.0373	0.0355	0.0282	0.0381
61	0.0359	0.0376	0.0365	0.0289	0.0425
62	0.0366	0.0388	0.0374	0.0301	0.0400
63	0.0349	0.0376	0.0361	0.0300	0.0392
64	0.0474	0.0526	0.0476	0.0316	0.0358
65	0.0484	0.0539	0.0486	0.0334	0.0391
66	0.0414	0.0467	0.0424	0.0305	0.0377
67	0.0418	0.0475	0.0430	0.0313	0.0417
68	0.0392	0.0446	0.0401	0.0280	0.0339
69	0.0393	0.0447	0.0405	0.0280	0.0353
70	0.0621	0.0690	0.0614	0.0385	0.0503
71	0.0599	0.0666	0.0614	0.0402	0.0514
72	0.0752	0.0776	0.0776	0.0484	0.0595
73	0.0781	0.0821	0.0798	0.0511	0.0623
74	0.0522	0.0527	0.0525	0.0307	0.0433
75	0.0528	0.0535	0.0527	0.0317	0.0446
76	0.0557	0.0568	0.0581	0.0322	0.0431
77	0.0577	0.0599	0.0601	0.0339	0.0443
78	0.0509	0.0561	0.0506	0.0308	0.0424
79	0.0476	0.0517	0.0471	0.0327	0.0447

Table A.4: Full accuracy results - Mean errors
For definition of error, see Section 3.1.2.

A.6 Accuracy Data - Standard Deviations

Index	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA	AVA-M
0	0.0602	0.0590	0.0625	0.0403	0.0683
1	0.0556	0.0548	0.0566	0.0414	0.0793
2	0.0516	0.0410	0.0569	0.0298	0.0559
3	0.0565	0.0500	0.0596	0.0311	0.0622
4	0.0564	0.0514	0.0594	0.0316	0.0617
5	0.0501	0.0457	0.0517	0.0313	0.0644
6	0.0536	0.0517	0.0546	0.0340	0.0628
7	0.0581	0.0576	0.0603	0.0330	0.0691
8	0.0406	0.0407	0.0431	0.0332	0.0538
9	0.0498	0.0496	0.0480	0.0305	0.0584
10	0.0454	0.0448	0.0462	0.0305	0.0638
11	0.0577	0.0584	0.0580	0.0299	0.0815
12	0.0426	0.0406	0.0436	0.0235	0.0464
13	0.0422	0.0418	0.0464	0.0243	0.0526
14	0.0480	0.0439	0.0494	0.0245	0.0503
15	0.0489	0.0481	0.0509	0.0260	0.0583
16	0.0558	0.0553	0.0571	0.0356	0.0634
17	0.0633	0.0678	0.0658	0.0368	0.0717
18	0.0590	0.0590	0.0608	0.0288	0.0655
19	0.0568	0.0609	0.0587	0.0291	0.0645
20	0.0546	0.0555	0.0566	0.0297	0.0648
21	0.0601	0.0664	0.0611	0.0301	0.0664
22	0.0428	0.0462	0.0453	0.0317	0.0672
23	0.0413	0.0447	0.0439	0.0313	0.0536
24	0.0549	0.0527	0.0582	0.0353	0.0711
25	0.0531	0.0523	0.0560	0.0354	0.0728
26	0.0692	0.0783	0.0713	0.0301	0.0627
27	0.0813	0.0890	0.0823	0.0303	0.0834
28	0.0445	0.0483	0.0420	0.0274	0.0623
29	0.0446	0.0465	0.0418	0.0272	0.0679
30	0.0601	0.0608	0.0582	0.0285	0.0644
31	0.0718	0.0756	0.0724	0.0305	0.0734
32	0.0663	0.0748	0.0657	0.0303	0.0754
33	0.0750	0.0914	0.0791	0.0305	0.0784
34	0.0439	0.0474	0.0423	0.0280	0.0649
35	0.0524	0.0533	0.0510	0.0278	0.0765
36	0.0524	0.0572	0.0494	0.0280	0.0686
37	0.0474	0.0494	0.0454	0.0284	0.0764
38	0.0452	0.0462	0.0431	0.0255	0.0593
39	0.0502	0.0519	0.0493	0.0283	0.0859

Index	HOLDEN	HOLDEN-XL	HOLDEN-XT	AVA	AVA-M
40	0.0623	0.0706	0.0594	0.0275	0.0598
41	0.0536	0.0634	0.0491	0.0276	0.0674
42	0.0507	0.0538	0.0477	0.0294	0.0534
43	0.0546	0.0570	0.0526	0.0322	0.0716
44	0.0484	0.0507	0.0481	0.0324	0.0568
45	0.0492	0.0515	0.0459	0.0331	0.0664
46	0.0432	0.0454	0.0411	0.0292	0.0474
47	0.0455	0.0498	0.0436	0.0278	0.0403
48	0.0460	0.0490	0.0431	0.0299	0.0440
49	0.0460	0.0478	0.0433	0.0304	0.0527
50	0.0466	0.0448	0.0438	0.0303	0.0510
51	0.0427	0.0425	0.0414	0.0325	0.0663
52	0.0480	0.0513	0.0459	0.0293	0.0484
53	0.0462	0.0490	0.0437	0.0300	0.0509
54	0.0561	0.0627	0.0498	0.0289	0.0595
55	0.0459	0.0484	0.0457	0.0287	0.0651
56	0.0395	0.0435	0.0382	0.0273	0.0450
57	0.0394	0.0434	0.0385	0.0288	0.0509
58	0.0569	0.0590	0.0544	0.0313	0.0878
59	0.0586	0.0617	0.0587	0.0327	0.0712
60	0.0419	0.0445	0.0401	0.0282	0.0577
61	0.0456	0.0471	0.0445	0.0289	0.0627
62	0.0503	0.0554	0.0519	0.0301	0.0538
63	0.0456	0.0477	0.0442	0.0300	0.0545
64	0.0419	0.0457	0.0401	0.0316	0.0369
65	0.0410	0.0456	0.0396	0.0334	0.0441
66	0.0490	0.0532	0.0466	0.0305	0.0516
67	0.0484	0.0565	0.0478	0.0313	0.0676
68	0.0484	0.0540	0.0462	0.0280	0.0445
69	0.0438	0.0495	0.0435	0.0280	0.0508
70	0.1195	0.1350	0.1082	0.0385	0.0773
71	0.1124	0.1280	0.1086	0.0402	0.0799
72	0.1095	0.1166	0.1126	0.0484	0.1143
73	0.1012	0.1092	0.1011	0.0511	0.0913
74	0.0702	0.0720	0.0710	0.0307	0.0716
75	0.0690	0.0664	0.0709	0.0317	0.0720
76	0.0705	0.0759	0.0738	0.0322	0.0705
77	0.0699	0.0795	0.0719	0.0339	0.0730
78	0.1161	0.1270	0.1126	0.0308	0.0707
79	0.1002	0.1110	0.0990	0.0327	0.0745

Table A.5: Full accuracy results - Standard deviation errors
For definition of errorn, see Section 3.1.2.

A.7 Training Mean-Squared Error

0	0.442	400	0.275	800	0.265	1200	0.260	1600	0.256
10	0.354	410	0.275	810	0.265	1210	0.259	1610	0.257
20	0.336	420	0.274	820	0.265	1220	0.260	1620	0.256
30	0.326	430	0.276	830	0.266	1230	0.259	1630	0.255
40	0.319	440	0.273	840	0.265	1240	0.260	1640	0.255
50	0.315	450	0.274	850	0.264	1250	0.259	1650	0.257
60	0.311	460	0.273	860	0.265	1260	0.260	1660	0.256
70	0.307	470	0.273	870	0.264	1270	0.260	1670	0.256
80	0.304	480	0.272	880	0.264	1280	0.259	1680	0.255
90	0.303	490	0.273	890	0.263	1290	0.259	1690	0.256
100	0.300	500	0.272	900	0.265	1300	0.259	1700	0.256
110	0.298	510	0.271	910	0.264	1310	0.259	1710	0.256
120	0.295	520	0.271	920	0.263	1320	0.259	1720	0.255
130	0.295	530	0.271	930	0.264	1330	0.259	1730	0.256
140	0.292	540	0.271	940	0.263	1340	0.258	1740	0.254
150	0.292	550	0.271	950	0.263	1350	0.259	1750	0.255
160	0.291	560	0.270	960	0.262	1360	0.258	1760	0.256
170	0.290	570	0.270	970	0.263	1370	0.259	1770	0.255
180	0.288	580	0.271	980	0.263	1380	0.258	1780	0.255
190	0.287	590	0.270	990	0.263	1390	0.258	1790	0.254
200	0.286	600	0.269	1000	0.262	1400	0.257	1800	0.255
210	0.286	610	0.269	1010	0.263	1410	0.257	1810	0.255
220	0.285	620	0.269	1020	0.262	1420	0.258	1820	0.255
230	0.284	630	0.270	1030	0.262	1430	0.258	1830	0.255
240	0.284	640	0.269	1040	0.262	1440	0.257	1840	0.254
250	0.283	650	0.268	1050	0.262	1450	0.258	1850	0.253
260	0.282	660	0.269	1060	0.261	1460	0.257	1860	0.255
270	0.282	670	0.268	1070	0.262	1470	0.257	1870	0.255
280	0.281	680	0.267	1080	0.261	1480	0.257	1880	0.254
290	0.282	690	0.268	1090	0.261	1490	0.258	1890	0.254
300	0.280	700	0.267	1100	0.260	1500	0.258	1900	0.253
310	0.279	710	0.268	1110	0.261	1510	0.257	1910	0.254
320	0.279	720	0.267	1120	0.261	1520	0.257	1920	0.254
330	0.279	730	0.266	1130	0.260	1530	0.257	1930	0.254
340	0.278	740	0.267	1140	0.262	1540	0.257	1940	0.253
350	0.278	750	0.267	1150	0.261	1550	0.256	1950	0.254
360	0.277	760	0.266	1160	0.261	1560	0.257	1960	0.254
370	0.276	770	0.266	1170	0.261	1570	0.256	1970	0.253
380	0.278	780	0.266	1180	0.260	1580	0.256	1980	0.253
390	0.275	790	0.265	1190	0.260	1590	0.257	1990	0.254

Table A.6: The mean-squared errors of the entire output vector - HOLDEN

Ep.	MSE	Ep.	MSE	Ep.	MSE	Ep.	MSE	Ep.	MSE
0	0.474	400	0.273	800	0.261	1200	0.255	1600	0.251
10	0.371	410	0.272	810	0.262	1210	0.255	1610	0.250
20	0.350	420	0.272	820	0.261	1220	0.255	1620	0.251
30	0.336	430	0.271	830	0.260	1230	0.254	1630	0.251
40	0.326	440	0.271	840	0.261	1240	0.255	1640	0.251
50	0.319	450	0.271	850	0.261	1250	0.255	1650	0.250
60	0.314	460	0.271	860	0.260	1260	0.254	1660	0.250
70	0.309	470	0.270	870	0.259	1270	0.254	1670	0.250
80	0.306	480	0.270	880	0.260	1280	0.255	1680	0.251
90	0.304	490	0.269	890	0.260	1290	0.254	1690	0.251
100	0.301	500	0.269	900	0.259	1300	0.254	1700	0.251
110	0.299	510	0.268	910	0.259	1310	0.253	1710	0.250
120	0.297	520	0.268	920	0.259	1320	0.254	1720	0.250
130	0.295	530	0.268	930	0.259	1330	0.253	1730	0.250
140	0.293	540	0.268	940	0.258	1340	0.254	1740	0.250
150	0.292	550	0.268	950	0.258	1350	0.253	1750	0.251
160	0.290	560	0.267	960	0.259	1360	0.253	1760	0.249
170	0.289	570	0.267	970	0.259	1370	0.253	1770	0.250
180	0.288	580	0.266	980	0.258	1380	0.253	1780	0.249
190	0.287	590	0.266	990	0.259	1390	0.254	1790	0.250
200	0.285	600	0.265	1000	0.258	1400	0.253	1800	0.249
210	0.285	610	0.266	1010	0.257	1410	0.253	1810	0.249
220	0.283	620	0.266	1020	0.258	1420	0.254	1820	0.249
230	0.282	630	0.266	1030	0.257	1430	0.252	1830	0.250
240	0.283	640	0.265	1040	0.259	1440	0.252	1840	0.249
250	0.281	650	0.265	1050	0.257	1450	0.252	1850	0.249
260	0.280	660	0.264	1060	0.257	1460	0.253	1860	0.249
270	0.280	670	0.264	1070	0.257	1470	0.252	1870	0.250
280	0.280	680	0.264	1080	0.257	1480	0.252	1880	0.250
290	0.280	690	0.264	1090	0.256	1490	0.252	1890	0.249
300	0.278	700	0.264	1100	0.256	1500	0.253	1900	0.249
310	0.277	710	0.263	1110	0.256	1510	0.252	1910	0.249
320	0.277	720	0.263	1120	0.257	1520	0.253	1920	0.248
330	0.276	730	0.264	1130	0.256	1530	0.251	1930	0.248
340	0.276	740	0.263	1140	0.257	1540	0.251	1940	0.249
350	0.275	750	0.262	1150	0.256	1550	0.252	1950	0.249
360	0.275	760	0.262	1160	0.255	1560	0.251	1960	0.248
370	0.274	770	0.261	1170	0.256	1570	0.251	1970	0.248
380	0.274	780	0.261	1180	0.255	1580	0.252	1980	0.248
390	0.273	790	0.261	1190	0.255	1590	0.251	1990	0.248

Table A.7: The mean-squared errors of the entire output vector - HOLDEN-XL

Ep.	MSE	Ep.	MSE	Ep.	MSE	Ep.	MSE	Ep.	MSE
2000	0.253	2400	0.251	2800	0.250	3200	0.247	3600	0.247
2010	0.252	2410	0.251	2810	0.249	3210	0.248	3610	0.247
2020	0.253	2420	0.250	2820	0.249	3220	0.248	3620	0.246
2030	0.253	2430	0.252	2830	0.249	3230	0.248	3630	0.246
2040	0.252	2440	0.251	2840	0.250	3240	0.249	3640	0.246
2050	0.253	2450	0.250	2850	0.248	3250	0.247	3650	0.247
2060	0.253	2460	0.251	2860	0.249	3260	0.248	3660	0.247
2070	0.253	2470	0.251	2870	0.249	3270	0.247	3670	0.245
2080	0.253	2480	0.251	2880	0.250	3280	0.247	3680	0.246
2090	0.254	2490	0.251	2890	0.249	3290	0.248	3690	0.247
2100	0.252	2500	0.252	2900	0.249	3300	0.248	3700	0.246
2110	0.253	2510	0.251	2910	0.249	3310	0.248	3710	0.246
2120	0.253	2520	0.251	2920	0.249	3320	0.248	3720	0.246
2130	0.252	2530	0.251	2930	0.249	3330	0.248	3730	0.246
2140	0.252	2540	0.250	2940	0.248	3340	0.248	3740	0.246
2150	0.253	2550	0.250	2950	0.249	3350	0.247	3750	0.247
2160	0.253	2560	0.250	2960	0.249	3360	0.247	3760	0.246
2170	0.253	2570	0.249	2970	0.248	3370	0.247	3770	0.247
2180	0.252	2580	0.251	2980	0.248	3380	0.247	3780	0.247
2190	0.252	2590	0.249	2990	0.249	3390	0.247	3790	0.246
2200	0.252	2600	0.250	3000	0.248	3400	0.247	3800	0.246
2210	0.252	2610	0.250	3010	0.248	3410	0.248	3810	0.246
2220	0.253	2620	0.251	3020	0.248	3420	0.247	3820	0.245
2230	0.252	2630	0.250	3030	0.249	3430	0.246	3830	0.246
2240	0.252	2640	0.250	3040	0.250	3440	0.246	3840	0.245
2250	0.253	2650	0.250	3050	0.248	3450	0.247	3850	0.246
2260	0.251	2660	0.250	3060	0.248	3460	0.247	3860	0.246
2270	0.251	2670	0.249	3070	0.249	3470	0.247	3870	0.245
2280	0.252	2680	0.250	3080	0.248	3480	0.247	3880	0.246
2290	0.252	2690	0.250	3090	0.248	3490	0.247	3890	0.246
2300	0.252	2700	0.250	3100	0.248	3500	0.247	3900	0.246
2310	0.251	2710	0.249	3110	0.248	3510	0.246	3910	0.246
2320	0.251	2720	0.250	3120	0.248	3520	0.247	3920	0.247
2330	0.252	2730	0.250	3130	0.248	3530	0.247	3930	0.245
2340	0.252	2740	0.249	3140	0.248	3540	0.246	3940	0.246
2350	0.252	2750	0.250	3150	0.247	3550	0.247	3950	0.245
2360	0.251	2760	0.249	3160	0.247	3560	0.247	3960	0.245
2370	0.252	2770	0.250	3170	0.248	3570	0.246	3970	0.245
2380	0.251	2780	0.249	3180	0.248	3580	0.247	3980	0.245
2390	0.251	2790	0.249	3190	0.248	3590	0.246	3990	0.245

Table A.8: The mean-squared errors of the entire output vector - HOLDEN-XT

Ep.	MSE	Ep.	MSE	Ep.	MSE	Ep.	MSE	Ep.	MSE
0	0.782	400	0.542	800	0.464	1200	0.455	1600	0.463
10	0.711	410	0.572	810	0.562	1210	0.532	1610	0.460
20	0.633	420	0.506	820	0.499	1220	0.492	1620	0.511
30	0.670	430	0.520	830	0.480	1230	0.483	1630	0.451
40	0.616	440	0.501	840	0.484	1240	0.468	1640	0.463
50	0.575	450	0.495	850	0.517	1250	0.520	1650	0.520
60	0.603	460	0.502	860	0.530	1260	0.528	1660	0.464
70	0.578	470	0.529	870	0.493	1270	0.545	1670	0.462
80	0.599	480	0.583	880	0.478	1280	0.453	1680	0.460
90	0.555	490	0.479	890	0.454	1290	0.454	1690	0.580
100	0.539	500	0.484	900	0.528	1300	0.469	1700	0.464
110	0.619	510	0.593	910	0.498	1310	0.468	1710	0.491
120	0.548	520	0.496	920	0.464	1320	0.561	1720	0.519
130	0.618	530	0.494	930	0.512	1330	0.482	1730	0.444
140	0.561	540	0.479	940	0.476	1340	0.468	1740	0.444
150	0.558	550	0.479	950	0.491	1350	0.496	1750	0.477
160	0.556	560	0.507	960	0.474	1360	0.448	1760	0.461
170	0.552	570	0.545	970	0.494	1370	0.481	1770	0.446
180	0.586	580	0.493	980	0.477	1380	0.481	1780	0.475
190	0.548	590	0.509	990	0.510	1390	0.467	1790	0.473
200	0.527	600	0.540	1000	0.509	1400	0.517	1800	0.538
210	0.507	610	0.492	1010	0.494	1410	0.514	1810	0.456
220	0.524	620	0.491	1020	0.488	1420	0.520	1820	0.472
230	0.522	630	0.519	1030	0.456	1430	0.450	1830	0.457
240	0.520	640	0.555	1040	0.456	1440	0.494	1840	0.506
250	0.575	650	0.472	1050	0.488	1450	0.510	1850	0.447
260	0.625	660	0.505	1060	0.511	1460	0.465	1860	0.489
270	0.517	670	0.534	1070	0.458	1470	0.451	1870	0.496
280	0.518	680	0.470	1080	0.525	1480	0.465	1880	0.506
290	0.517	690	0.488	1090	0.490	1490	0.448	1890	0.489
300	0.547	700	0.472	1100	0.475	1500	0.465	1900	0.443
310	0.532	710	0.469	1110	0.488	1510	0.541	1910	0.473
320	0.559	720	0.534	1120	0.475	1520	0.491	1920	0.442
330	0.509	730	0.468	1130	0.489	1530	0.567	1930	0.472
340	0.510	740	0.486	1140	0.474	1540	0.451	1940	0.475
350	0.509	750	0.469	1150	0.534	1550	0.506	1950	0.470
360	0.539	760	0.534	1160	0.453	1560	0.509	1960	0.444
370	0.510	770	0.481	1170	0.469	1570	0.464	1970	0.472
380	0.526	780	0.519	1180	0.493	1580	0.447	1980	0.472
390	0.521	790	0.500	1190	0.468	1590	0.448	1990	0.461

Table A.9: The mean-squared errors of the entire output vector - AVA