SAARLAND UNIVERSITY

Faculty of Mathematics and Computer Science
Department of Computer Science
MASTER THESIS

Simulating Fast-Movements with Mixture-of-Expert Models: An Interactive Boxing Controller

submitted by
Chirag Bhuvaneshwara
Saarbrücken
November 2021

Supervisor

Janis Sprenger
Deutsches Forschungszentrum für Künstliche Intelligenz
Saarland Informatics Campus
66123 Saarbrücken, Germany

Reviewers

Prof.Dr.-Ing. Philipp Slusallek
Computer Graphics Lab
Saarland Informatics Campus E1 1
66123 Saarbrücken, Germany

Dr. Fabrizio Nunnari
Deutsches Forschungszentrum für Künstliche Intelligenz
Saarland Informatics Campus
66123 Saarbrücken, Germany

Advisor

Prof.Dr.-Ing. Philipp Slusallek
Computer Graphics Lab
Saarland Informatics Campus E1 1
66123 Saarbrücken, Germany

Saarland University
Faculty MI – Mathematics and Computer Science
Department of Computer Science
Campus - Building E1.1
66123 Saarbrücken, Germany

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Beachten Sie jedoch, dass die Seiten mit den Nummern 53 und 54 in der elektronischen Version zur besseren Lesbarkeit gedreht sind, aber der Inhalt dieser Seiten entspricht genau dem der gedruckten Version.

Statement in Lieu of an Oath

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

However, note that the pages bearing numbers 55 and 56 are rotated in the electronic version for better readability, but the contents of these pages are exactly the same as the printed version.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Acknowledgements

Throughout this thesis, I have received a great deal of support and assistance.

I would first like to thank Janis, whose deep technical knowledge helped formulate the research questions and methodology and in structuring the thesis text. Your advice and work ethic helped me to push myself and brought my work to a higher level.

I want to thank everyone at DFKI who have supported me. In particular, the motion synthesis team in the Agents and Simulated Reality research branch at DFKI helped me with the motion capture and many aspects of my implementation, either directly or indirectly. And Prof. Slusallek, who is the head of this branch, helped me with his valuable inputs and advice. I sincerely appreciate all the assistance from each of you.

I am grateful to Dr. Fabrizio Nunnari for reviewing the finished work.

I also sincerely thank my direct DFKI colleagues at the Affective Computing Group. Everyone in my team extended the conducive work atmosphere and allowed me to also focus on my thesis. Your silent support and encouragement, especially towards the end of my thesis journey, is much appreciated!

Working through the coronavirus pandemic as an international student has been hard. Luckily, many friends in Saarbrücken helped me get through this arduous journey. I sincerely thank every one of you for taking care of me.

Also, special thanks to my friends Daksitha, Yogesh, Ayan and Suraksha for their valuable inputs that helped in phrasing and formatting the text in this thesis document.

I want to thank my family, especially my parents and uncle, for their continued belief in me and their wise counsel throughout.

Finally, I thank my dear grandmother for everything she did for me throughout her life.

Abstract

Recently, neural networks have been successfully applied to various types of data like images and text, and they are also being adapted for the complex real-time task of interactive motion synthesis. The latter has led to novel neural models and associated controller systems that produce responsive and natural-looking motions. However, there are very few neural-based data-driven approaches for interactively synthesising hand-related motions of humanoid characters. Almost none of these approaches focus on boxing, even though the actions contained within boxing have broad applications in video games.

In this thesis, we hypothesise that a neural model based on the mixture of experts concept can be developed to synthesise fast boxing actions interactively. As the primary goal here is to synthesise natural-looking motions, a novel boxing dataset consisting of the fast actions of punching and stepping is prepared. This is followed by the design of effective representations of motion states used to train our neural models that are incorporated in a boxing controller system to facilitate user interaction with the neural synthesised motions.

Our system allows a user to specify the exact target of the punches and the direction of the steps to produce motions in the boxing style. This system can synthesise high-quality boxing actions despite being trained on limited data. To achieve the best synthesis results using such data, several experiments are conducted to determine effective motion states for producing more accurate punching and stepping actions. Our experimental results show that formulating the motion states by considering the fastness of the boxing actions leads to more accurate and natural-looking results for both targeted punching and user-directed stepping actions.

Contents

Chapter 1
Introduction

Animated virtual characters have recently been used for various exciting applications. One such application is the use of virtual characters by Baur et al. [4] to create a simulation environment for a job interview. Antakli. et al. [3] use computer-generated characters to prepare simulated environments for verifying the safety of human-robot collaboration operations. Another exciting application is a study, by Bönsch et al. [8], exploring peer pressure in human-human interaction, using virtual characters as a stand-in for humans. However, these new applications of virtual characters are still developing. The most common applications remain to be animated movies and computer games.

Whatever the application area might be, the challenge is to synthesise realistic motions in virtual environments. Nevertheless, synthesising such natural character movements for animated movies is fundamentally different from the one used for computer games. In the former case, motion synthesis systems can utilise massive computing power to generate the results in an offline mode. As a result, the users, who typically form the audience at the cinemas, view only the final and polished result. In addition, motion synthesis systems at industry-leading companies like Pixar allow animators to hand clean the generated motions as explained in [58]. However, coming to the latter scenario of computer games, the synthesis process is very different.

In computer games, the motions need to be synthesised frame by frame on systems with varying levels of computing power. Additionally, computer games include player-controlled and computer-controlled characters. So interactive motion synthesis must factor in either the user input or the changes in the environment and respond with natural motions. Besides these challenges from the application areas, motion synthesis systems, in general, need to tackle the inherent problem of the high degrees of freedom associated with a virtual skeleton as explained by Rotenberg [48]. Along with the varied hardware issues, the requirement of responsiveness, and the problem of dimensionality, the system must also produce realistic motions. Considering these challenges, motion synthesis, in general, is quite complicated and the additional requirement of interactivity makes it even more so.

Lee et al. [35] developed an approach to tackle the above challenges while enforcing realism by using a method called motion fields (MoFi). This method produces motion similar to a set of reference motion data clips captured from actual human actions.

However, the MoFi method has scaling problems, which modern learning algorithms can potentially solve, as they can be utilised to learn a good interpretation of the data as presented by Holden et al. [18].

Simultaneously, the recent breakthroughs in machine learning (ML) such as AlexNet [31] have started the deep learning (DL) revolution and reignited the interest in neural network-based ML models. The successful results from AlexNet [31] have led to the application of DL models to various domains, including the complicated domain of motion synthesis. Consequently, Holden et al. [21] adapted the findings from the DL literature to the complex real-time problem of interactive motion synthesis. Their approach is designed specifically for the actions related to human locomotion, and it achieves realistic and responsive results. While locomotion-related animations are prevalent in computer games, many games involving humanoid characters, such as [25, 12], also require punching actions.

Boxing is one sport where the action set involves both stepping and punching. This data has its own style, which influences all the character motions. For example, the stepping action from one point to another is different from simple walking actions in the locomotion dataset used by Holden et al. [21]. Additionally, stepping occurs in short bursts with the character in the boxing stance. Similarly, the non-cyclic punching actions are also typically executed over a short duration. In this thesis, we aim to use modern ML solutions inspired by the neural-based approach of Zhang et al. [63] and adapt it to synthesise such fast boxing movements. Our model is then used as the motion synthesis unit of a system that enables a user to control the boxing actions of a virtual character.

However, before presenting the related works and our approach, a clear description of our motivation is presented in the next section.

1.1 Motivation

Video games utilise virtual characters in simulated worlds and allow human players to interact with virtual objects. Modern computer games include large virtual worlds filled with characters performing realistic actions that a player can control. In addition, the kind of interaction offered is realistic as all the characters move and behave in a more natural way than in games of the past. One such video game which involves a thriving virtual world with realistic character movements is Grand Theft Auto 5 [25]. It is a popular modern video game that has been consistently generating millions of dollars in revenue. Thus animating virtual characters for video games is an important application.

The animation of virtual characters in the video game industry is a complicated process as the characters not only need to move realistically but also need to react to changes in the virtual video game environment. In addition, the animations must also consider user input to make the characters responsive. For example, such user input can move a virtual character to a target location in a video game environment without considering the relationship between the limbs of the character. However, such rudimentary methods would distract the players, and they will not be able to immerse themselves in the virtual world. To overcome this, methods have been developed to use motion capture (mocap) data that is obtained from the motions of an actual human participant.

Computer games can now synthesise good quality motions using such data or modern synthesis methods, but they still mainly restrict the character to executing a fixed set of actions. For example, in Grand Theft Auto 5 [25], punching and stepping around an

opponent character is necessary when the player character is in peril. In this particular scenario, the game allows the player to punch the opponent. However, nearly all the punches are targeted to the head of the character, and the player can only launch the punch, but not direct it towards a target chosen by the player. This leads to failure cases as seen in Figure 1.1. Even though the system employed in this game always tries to punch the face, it fails when the target has moved, and the user has no control to direct the punch, and thus, it leads to a missed punch.

Figure 1.1: This figure shows the limitation of the punching mechanism in a AAA video game called Grand Theft Auto 5 [25]. The opponent character is in motion in this figure, and the system-generated punch misses the target. Additionally, the punch variety is low, and most are always directed at the face of the opponent.

If additional targeted control is provided to users, the failure cases as in Figure 1.1 might not happen. In addition, such user control will increase the immersion of the player in the virtual environment. However, most video games do not provide such control, and there are also not many approaches in the research literature that can provide such targeted control. Moreover, since the character movement style is distinct when fighting an opponent, the stepping action to move from one point to another is different from simple walking actions. Such punching and stepping actions are present in boxing motions with a distinct boxing motion style, and this can be utilised for synthesising realistic targeted punching and user-directed stepping motions.

In the character animation research domain, there have recently been significant improvements in data-driven human locomotion synthesis [21, 55]. Such data-driven approaches utilise mocap data and have already been applied in the video game industry. For example, motion matching [10] has been implemented successfully in a popular video game called For Honor [12]. Moreover, approaches from Lee et al. [33] and Starke et al. [56] have allowed interactive control of characters in playing basketball with realistic motions and interactions. These approaches [33, 56] utilise neural networks to obtain good results.

However, when this thesis was conceptualised, there was a complete lack of neural-based approaches that focused on synthesising martial arts movements, such as the motions in boxing. Boxing motions involve stepping, punching and blocking, and all of these actions are non-periodic and fast. An action such as walking, which has a natural rhythm, is referred to as a periodic motion, but boxing actions have no such rhythm. They are typically executed randomly over a short duration of time. Thus, they are non-periodic and fast. Synthesising such motions using traditional animation techniques is possible, but there will be limitations in relation to naturalness and responsivity. However, neural-based data-driven approaches such as [21, 63, 55] can be adapted to provide realistic and responsive results.

Thus, in this thesis, we develop a novel neural-based data-driven method to synthesise the boxing actions of punching and stepping. The punching action is designed to allow targeted control, and the stepping action allows user-specified directions. These actions are collected using a mocap system and it is transformed into an effective motion encoding that improves the synthesis quality of our data-driven model.

However, note that around the end stages of this thesis, Starke et al. [57] presented an approach aimed at interactive control of mixed martial arts movements. This approach and its motivation in relation to ours will be discussed more in Chapter 2. In the next section, we describe the contribution of this thesis with a brief introduction to the approaches that are directly relevant to this thesis.

1.2 Contribution

As already mentioned, Holden et al. [21] showed that it is possible to develop a data-driven interactive controller that can synthesise locomotion actions, which is achieved with the phase function neural network (PFNN). While the PFNN obtains responsive and natural-looking results, a part of the model is still not based on learning approaches. This gap was bridged by Zhang et al. [63] with their presentation of a completely neural-based alternative to the PFNN model. Their model is called the mode adaptive neural network (MANN) model. However, it is not a *deep* neural network like the computer vision models from [49]. We cover the details about both the PFNN model and the MANN model in Chapter 2.

Nevertheless, we hypothesise that the MANN model is capable of synthesising good quality boxing actions. This hypothesis is based on the fact that neural-based approaches perform better when they are allowed to extract their own features directly from the data as shown in [49]. Furthermore, while the motion state representations are not the direct equivalent of raw image data, these representations still carry a substantial amount of information that facilitate good synthesis results ([35], [21], [63]). Therefore, MANN-type neural models should be capable of addressing the challenges that arise while synthesising boxing motions.

Hence, in this master thesis, we develop neural models in the spirit of the MANN model and provide user interactivity through an associated controller system. For this purpose, we constructed a novel mocap dataset for the boxing motions of stepping and punching. This data is processed and labelled using an automatic labelling scheme. These labels are utilised to develop effective motion state representations suitable for targeted punching and user-directed stepping. Several versions of these motion states are prepared by considering different ways to sample the boxing dataset and experimenting with the variables to include in the motion states. Based on these motion representations,

several versions of our MANN-based neural model are trained. Furthermore, a specific evaluation setup is presented for determining the synthesis performance of both stepping and punching actions.

So our main contributions in this thesis are twofold: 1) the implementation and evaluation of several neural models trained on different versions of the motion states that we designed specifically for boxing, and 2) the development of a controller system for interactive control of the boxing actions of stepping and punching. In most cases, the resulting system can generate actions from the boxing dataset responsively and accurately. We also show that our controller is a critical system capable of applying post-prediction fixes and improving the results. Moreover, this thesis shows that it is possible to interactively synthesise good-quality targeted boxing actions even when the amount of training data is restricted. Finally, the experimental results demonstrate that effective motion state encodings based on dataset statistics of the captured actions are necessary to improve the synthesis results in relation to the interactive synthesis of targeted punching and user-directed stepping actions.

Having decided on a model capable of countering the challenges in boxing motion and briefly considering our contributions, we now look into the structure of this thesis.

In Chapter 2, we will first cover different existing methods for motion synthesis aimed at animators and computer gamers. Here, the data-driven motion synthesis models will be covered in greater detail, with an emphasis on the MANN model, which will be extended in this thesis. Subsequently, Chapter 3 covers information about the mocap session and the tools that we used to record the boxing motions and post-process the results. This section also covers how the data is labelled and presents a statistical analysis of our data.

In Chapter 4, information about the motion state representations that we use to train our models are explained. It also contains our model definition and the details about the training of our neural models. Next, Chapter 5 covers the design of the interactive boxing controller system and information about the functions performed by the different units in our system. This chapter is followed up with Chapter 6 which presents information about the evaluation scheme employed. The same chapter also contains details about the experimental setup, the models considered for the experiment, and the evaluation results. Then, the many fascinating and peculiar findings from the experiments are discussed in more depth in Chapter 7 with an elaboration of the shortcomings of our system and possible directions for future work. Finally, the same chapter summarises the work done in this thesis with conclusions about the most important results.

Chapter 2
Related Work

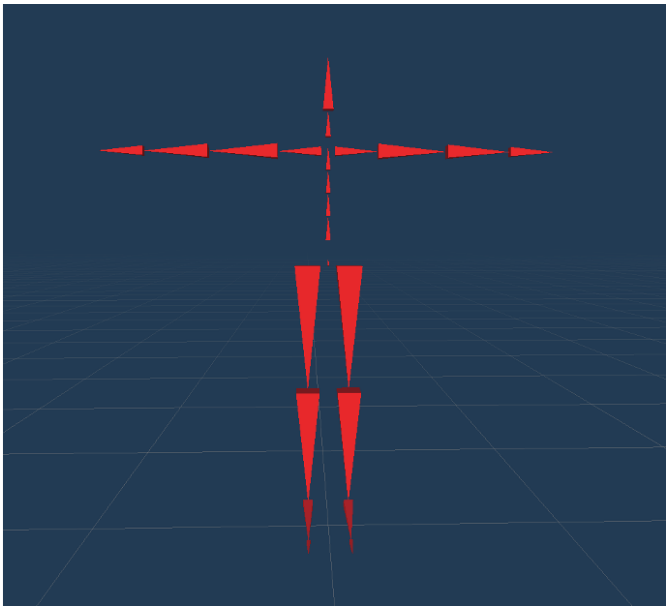
Motion synthesis is the task of generating motions of a character in a virtual world. The models enabling such synthesis can focus purely on generating the actions, or in addition to generating actions, they can also consider the laws of physics during synthesis. The former is referred to as a kinematic model and the latter as a dynamic model. However, both of these models usually utilise a kinematic skeletal structure, and a dynamic system is typically added on top of a kinematic foundation as explained in [48].

In Figure 2.1a, we show a typical kinematic skeleton that consists of a connected set of rigid segments. These segments correspond to the limbs of the human body, and they are connected by the joints listed in Figure 2.1b. There is a specific hierarchical structure that defines the relationship between these joints. Each joint in the hierarchical structure is the child of the joint above it and the parent of the joint below it. As seen in Figure 2.1b, the root joint, i.e. the hip, is the parent of all joints, and it is responsible for tracking the motion of the skeleton in the global space of the virtual world. The other joints are only responsible for producing the motion in the local space of their immediate parent joints.

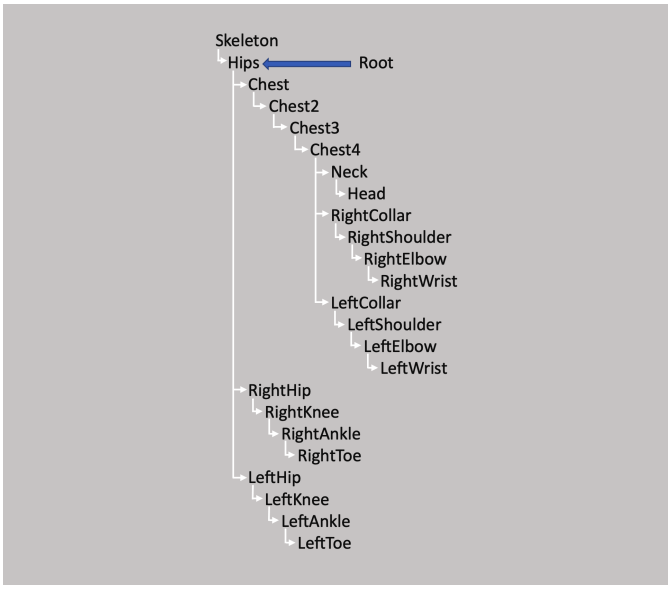
Kinematic models use a skeleton such as the one explained in Figure 2.1a and traverse the joint hierarchy in Figure 2.1b while applying computations that move a child joint in relation to its parent. The idea of applying the computations in a top-down manner, i.e. from the root joint to the end joints such as the wrist and the toe, leads to forward kinematics and the inverse order of computations, i.e. starting from the wrist or toe and traversing up to the root leads to inverse kinematics. These computations are only focused on moving the joints to the desired position, even if it might look unrealistic. Such computations rely on functions of positions, velocities or acceleration to achieve motion [29].

However, a dynamic system considers physical characteristics such as forces, torques and energies while moving the joints of a skeleton. To apply such characteristics, the dynamic models require to operate in a physics engine where the laws of physics are applied on each joint in the skeleton [29].

In either case, the skeletons will usually have a high degree of freedom regarding their movements. To make better decisions in such a high dimensional space, the research in both models has started shifting towards employing neural networks as



(a)



(b)

Figure 2.1: A typical virtual skeleton is shown in Figure 2.1a and it consists of several joints arranged in a joint hierarchy as shown in Figure 2.1b. As the heirarchy indicates, the hip joint is at the highest level of the heirarchy and it tracks the motion of the character in the virtual world. All the other joints produce motions in the local space of their parent joints.

they have produced good results with high dimensional data. However, many of the best performing neural models from computer vision require a large amount of training data [49]. This might also be applicable for neural-based motion synthesis, and for this purpose, we can gather large amounts of data through mocap systems. There are many commercially available mocap systems such as the Xsens MTw Awinda [6]. These systems have associated software that helps to transfer the motions made by an actor onto a virtual skeleton.

The data produced by Mocap systems can be used by both kinematic and dynamic models that use neural networks for motion synthesis. However, these motion synthesis models are not limited to using mocap data as explained in [61], which is a review of all motion synthesis methods in the literature. So, while there exist other methods, the literature survey in this thesis will focus on methods that rely on mocap data.

Considering the relevant approaches for this thesis, we classify the prevalent motion synthesis methods into data-driven and simulation-based approaches. If the motion synthesis is focused on generating models similar to a reference dataset, we classify them as data-driven methods, and if the synthesis system is constrained to respect the laws of physics in a virtual world, we categorise them as simulation-based methods. As we want to apply neural-based approaches to learn a motion manifold of our boxing data as described in Section 1.1, the model developed in this thesis will be using the data-driven approach, and we present the works using this approach in greater detail in Section 2.2.

However, since the use case of our model is in the domain of computer games, it might be the case that the synthesis must respect the physics rules inside the game. Moreover, as presented further, there are also proposals in the literature to use data-driven models similar to ours in conjunction with simulation-based approaches to achieve dynamic motion synthesis. Thus, while data-driven approaches will ensure the realism of synthesised boxing motions, they might require assistance from simulation-based methods to make the boxing motions more dynamic.

Well-preforming simulation-based approaches are currently using DL in combination with reinforcement learning (RL), which is a method focused on helping a computer agent to determine the best policy to solve a problem by considering the constraints in a system. RL achieves this by defining rewards in a system that an ML model will be trained to maximise. Thus, RL methods rely on learning by taking actions in a simulation, and these methods have been applied in the literature to synthesise dynamic human motions. Such simulation-based approaches using DL and RL together come with their own problems, but they might be helpful for future work on our model. Thus, in addition to data-driven approaches, we also briefly present simulation-based approaches in this chapter and begin our literature survey presentation with simulation-based approaches in the next section.

2.1 Simulation-based Methods

Motion synthesis methods based on a simulation focus on synthesising motions that respect the laws of physics in the virtual world. When such methods use mocap data, they will usually enforce the system to not only produce motions that are accurate in the virtual world but also try to be similar to the motions in their reference mocap dataset. Other approaches that do not use mocap data usually try to define a better search space to generate more realistic motions. The literature survey in this thesis will focus more on the former type of simulation-based methods that utilise mocap data as these methods

can be used as an additional layer on our model to constrain it to produce realistic boxing motions in a simulated environment.

The motion synthesis in simulation-based approaches is dependent on the quality of the virtual world. Fortunately, there are various game engines available such as Unity [59], and Unreal Engine [24] that can support high-quality virtual worlds. These engines provide an ideal simulation environment for setting up motion synthesis methods based on the dynamic models and are used in developing the simulation-based synthesis methods that use DL and RL in conjunction.

The advances in DL research have led to breakthroughs in the RL domain. For example, Silver et al. [52] showed that with the help of deep neural networks (DNNs), it is possible to develop a system trained with RL that has a very high winning rate in the game of Go. This approach is referred to as deep reinforcement learning (DRL) in the literature, and its most important consequence for motion synthesis is that DNNs can be utilised to process enormous search spaces and produce appropriate policies based on current scenarios.

With the availability of modern game engines and the potential of DRL, motion synthesis methods have made significant breakthroughs, such as in [16]. The approach by Heess et al. [16] shows that human locomotion behaviours such as run, jump, crouch and turn can be synthesised using DRL. This work is improved upon by Yu et al. [62] as they set up a method that can produce relatively more realistic looking motions without using any reference clips. While the previous methods dealt with manipulating the joint torques directly, Jiang et al. [28] show that it is possible to translate the problem from the muscle actuation space to the joint actuation space and doing so automatically provides a relatively more realistic action space for the agent to explore. Lee et al. [34] develop a similar approach, but their approach uses a musculoskeletal model with DRL to produce realistic human movements by developing policies that are based on muscle contraction dynamics.

The alternative approach that employs DRL and synthesises realistic motion depends upon existing mocap data similar to the approach by Heess et al. [16] mentioned above. The goal in such methods is to be similar to the mocap reference data used for training the DNN model. Such methods can be referred to as DRL methods performing imitation tasks. Using a DRL model, Peng et al. [44] show that a biped character can be instructed to do relatively complex tasks, like dribbling a ball and following a path through obstacles, while the synthesised motions are constrained to be similar to the reference clips. Peng et al. [45] extend this work by developing a method that can imitate a large variety of motions, adapt to the environment and react to user-specified goals. Park et al. [42] employ the recurrent neural network (RNN) in an imitation based DRL method to develop a control policy that enables interactive control of characters that can react to inanimate objects and other virtual characters in the environment. Similarly, Bergamin et al. [5] propose a responsive simulated character controller developed using unstructured mocap data.

While the research mentioned above has made progress in generating simulation-based models that can also produce sufficiently realistic motions, some problems persist. For example, DRL methods that are not using mocap reference clips to enforce realism need to utilise very complicated representations to achieve realistic motions. Essentially, when reference clips are not used, an expert RL developer will need to define a reward function that can synthesise realistic motions. Expert intervention is also required when using mocap data in DRL-based methods, but the results tend to look more realistic, and the systems are slightly less complicated. Using such expert-defined representations is in

contrast to the findings from the DL literature [49] as it has clearly been shown that the DNNs perform better when they are allowed to extract features of their own. This finding should apply to DRL-based motion synthesis models as well.

Due to the need of the expert-defined reward functions explained above, the successful DRL synthesis methods are overly complicated to set up, test and tune to obtain good results. For example, as seen in [45], the loss function needs to be adjusted specifically for each new mocap dataset and for each goal action being considered. Additionally, the training of the DRL models needs to be adjusted by using complicated workarounds to avoid very long training cycles. As explained by Peng et al. [45], the workarounds are necessary because of the high dimensionality of the state space being considered in DRL methods.

Considering the complicated reward functions and training problems, it is hard to arrive at a boxing synthesis model that can produce realistic motions. Moreover, the success of some simulation-based methods depends on data-driven methods. For example, the simulation-based approach in [5] depends on the data-driven motion matching [10] method internally. Combining a method such as motion matching, which has scaling issues, and DRL methods that have to deal with high dimensional state spaces will be very complicated. Thus, data-driven approaches need to be improved, and this will simultaneously assist simulation-based approaches.

In summary, physics-based DRL systems can react to external forces in a simulation, and this would be beneficial for boxing. However, these methods are hard to set up and are not always stable, resulting in awkward movements, which will be more pronounced if the method does not rely on mocap data. Additionally, data-driven methods can be used as a unit within such simulation-based systems and thus, improve the overall results. Therefore, we believe that the data-driven approach for boxing must be thoroughly explored first. Hence, we will dive into a detailed overview of the prevalent data-driven motion synthesis approaches in the next section.

2.2 Data-driven Methods

Setting up a mocap session requires time, money, and effort. So, it is more beneficial to reuse the mocap data that has already been captured. To this end, animators utilise motion editing systems to produce new motions from existing mocap data. Such systems enabling animators are synthesising motions similar to the reference mocap dataset while considering the input from the animator. Thus, these systems help animators work on considerable amounts of mocap data at a time. However, the motion synthesis systems employed for computer games have the constraint of responding in real-time while still producing a reasonably good motion quality.

Accordingly, the interactive synthesis systems used by animators can work offline as the emphasis is on the quality of motions only. However, the systems in computer games are restricted to online synthesis as these games are real-time systems where the user expects responsive synthesis. Based on this, we categorize the research literature into *Motion capture data editing* and *Interactive motion synthesis* methods. Both of these methods are presented in this section. However, since modern neural-based approaches are more relevant in this thesis, our presentation of the editing and interactive methods will focus more on works using learning-based approaches.

2.2.1 Editing Motion Capture Data

There are several methods designed to facilitate animators to edit mocap data. One such method, called layering, allows the addition of offsets to the body parameters of a virtual skeleton to slightly change the mocap data enacted by the skeleton. Seol et al. [51] apply this method of layering to change the style of human motion to that of various creatures. Another editing method called splicing was applied by Ikemoto et al. [23] to synthesise novel motions by supplanting the joint trajectories from one motion clip to another. An optimisation-based method to allow animators to edit mocap data is presented by Liu and Popović [37]. They apply constraints to produce realistic-looking motion that follows the instructions given by the animator.

Recently, neural networks have also been applied to allow animators to edit mocap data. For example, Holden et al. [18] developed a neural autoencoder that can help animators to correct the mocap data. Their approach can correct corrupted motion, compute the similarity between motion clips and avoid blending artefacts. This work is extended by Holden et al. [19] which allows an animator to define the path to be followed by the virtual character. Subsequently, Harvey et al. [15], who were motivated by the traditional animation process of inbetweening, designed a system that synthesises realistic motion between sparse keyframes using adversarial RNNs.

The methods we have presented so far were focused on enabling animators to synthesise motions by considering their active inputs. However, there exists a method called *Motion style transfer* that can reduce the load on animators, especially for projects that require them to change the style of mocap data to another. This method can transform the input motion style to the desired style, and one way of doing this is through feedforward neural networks as in [20]. Sprenger et al. [53] present a way to infer the style information from mocap data to produce locomotion in the male or female style and also allow for the results to be varied in terms of the expressivity of the chosen style. Improvements in the style transfer domain led to Aberman et al. [2] developing a method that can work on unpaired motion styles. This work can also extract motion styles directly from videos, which is advantageous as this method can work on data that was never seen during model training. However, while style transfer helps synthesise new styles, it remains a method for reducing the workload of animators as it does not allow more fine-tuned control.

While motion editing methods have improved significantly, the approaches presented in this section cannot be used in a computer game requiring a real-time boxing motion synthesis system. On the contrary, the concepts from motion style transfer can be utilised for non-player characters in video games by combining with simulation-based methods. The other data-driven editing approaches presented in this section are primarily suitable for offline applications such as animated movies. However, since our model is expected to perform interactive motion synthesis, we shall present the relevant, interactive data-driven approaches in the following section.

2.2.2 Interactive Motion Synthesis

In this section, the focus is on research works that allow a user to control the motion of a virtual character interactively. These approaches can be adapted to allow interactive motion synthesis using our boxing dataset. Therefore, we explore various methods that are relevant to our work in this section. We start by broadly classifying the prevalent

methods into search-based and learning-based methods for interactive motion synthesis, and they will be explained in more detail in the following section.

Search-based methods

Search-based methods rely on pre-captured motion data stored in a database. The primary approach here is to define conditions regarding how the motion clips in the database can be combined to produce the required motion. However, these methods need to store the entire mocap database in the computer storage of the user. The early search-based methods could not overcome this storage requirement as the motion synthesis domain was focused on developing a system capable of efficiently navigating the database. To this end, Kovar et al. [30] presented a directed graph-based model that captures the relationships between the motions in the database and allows for interactive character control. This led Lee et al. [35] to introduce motion fields (MoFi) where the mocap data is transformed and organised into a motion database, which can be used to compare the similarity with the current pose. Based on the user input, a Markov decision process is then employed to select a similar action from the generated motion database, and it is visualised for the user.

Inspired by the MoFi approach, Clavet [10] presented a simplified concept known as motion matching. In motion matching, the mocap data is processed to extract the vital motion information to form a matching features database. Then, the current pose is combined with the user input to form a query vector. This vector is matched with the database, and the entire motion clip associated with the matched feature is played back as a response to the user input. However, motion matching has a scalability issue due to the search for a match in the feature space of the motion database. Thus, Holden et al. [22] presented learned motion matching, which aims to address the problems of motion matching by using several neural networks. This method learns manifolds that can generate responsive motions instead of relying on the motion database at runtime. Therefore, we will explain this work along with other learning-based methods presented in the following text.

Learning-based methods

In learning-based interactive synthesis methods, the mocap database is used to train an ML model that can remove the requirement of the mocap database in the computer storage. Essentially, the ML models are expected to learn the motion manifold by taking in representative features of the current motion state as input and producing the representative features of the following motion state as output. This output can then be used to animate the following pose. Therefore, there is no need to access a mocap database after training the ML model. In addition, ML models try to generalise from the training data to unseen data points, while a search-based method can only interpolate between existing examples in the database.

Continuing from the database-driven method of motion matching, we now look at the succeeding method from Holden et al. [22]. Their primary motivation was to remove the requirement of the mocap database and thereby extend the motion matching method to a more significant amount of mocap data. Accordingly, their method removes the reliance on all databases by using three feedforward neural networks and one neural autoencoder. Essentially, after the neural models are trained, all the search problems are

transformed to forward passes of neural networks, and the produced output can be used directly to animate the next pose.

While learned motion matching [22] uses neural networks to remove the dependence on the database, it is still motivated by the search-based MoFi method. On the other hand, there are methods wherein the motivation has been to develop good motion state representations and a capable ML model that can directly map the current motion state to the following one. Hence, these methods are not motivated to perform any matching operations like in search-based methods. We hypothesise that a direct mapping method is better for a neural model than performing matching operations. Furthermore, for the case of our boxing data, it is better to synthesise the motions directly, as the actions we are considering occur in a short duration of time, and matching operations for synthesising such fast motions might reduce the responsivity of the system. Thus, the research approaches that help in designing models capable of direct mapping are presented next.

Mocap data represents how the different joints of a virtual skeleton move w.r.t. time in a virtual world. This inherent time-based nature motivated Fragkiadaki et al. [13] to apply the encoder recurrent decoder (ERD) neural model for the task of interactive motion synthesis. This model utilises RNNs internally, but it converges to an average pose. Zhou et al. [64] try to solve this problem by applying teacher forcing, which encourages the network to continuously generate longer motion sequences during training. Subsequently, Lee et al. [33] show that such time series based neural models can be applied to map control inputs that represent user intentions to responsive character animations in tasks such as locomotion and playing basketball. While these time series based models make logical sense, they have convergence issues as they cannot distinguish between the different stages in the motions.

Another interactive motion synthesis approach was developed that considers the extent to which the motion is currently completed. This approach uses a neural network that maps the current motion state to the next motion state. For this purpose, it is possible to use a simple feedforward neural network and include variables depicting the extent of motion completion in the motion states. However, only this setup would lead to convergence issues similar to the case of the ERD models. To overcome this, the neural network weights should be varied dynamically w.r.t. a subset of the motion state. Such networks enable high quality and responsive, interactive motion synthesis. The foundational work in this direction was presented by Holden et al. [21] through the phase function neural network (PFNN).

The PFNN allows a user to interactively control the locomotion of a virtual character through a rough terrain having varying heights and being filled with obstacles. Holden et al. [21] designed this model specifically for cyclic motions such as walking. Their main idea was to dynamically change the weights of the neural network as a function of the phase of the walking motion. To achieve this, the PFNN uses a phase function, given by the cubic Catmull-Rom spline cyclic function. This function considers the current walking phase to generate the new weights of the network as shown in Figure 2.2. These weights will be changed at each frame to map the the current motion state $\mathbf{x} \in \mathbb{R}^{n \times 1}$ at frame i to the next one $\mathbf{y} \in \mathbb{R}^{m \times 1}$ at frame $i + 1$. For this, the phase function utilises the phase $p \subset \mathbf{x}$ to map to a mixture of K network weight configurations α_k with $1 \leq k \leq K$.

Holden et al. [21] also verify that the PFNN performs better than the ERD architecture on locomotion tasks. However, the PFNN is not explicitly designed for interactive control of targeted combat motions, but it was extended to such motions in a master thesis by Freda [14] that focused on achieving interactive control of attack movements with a sword.

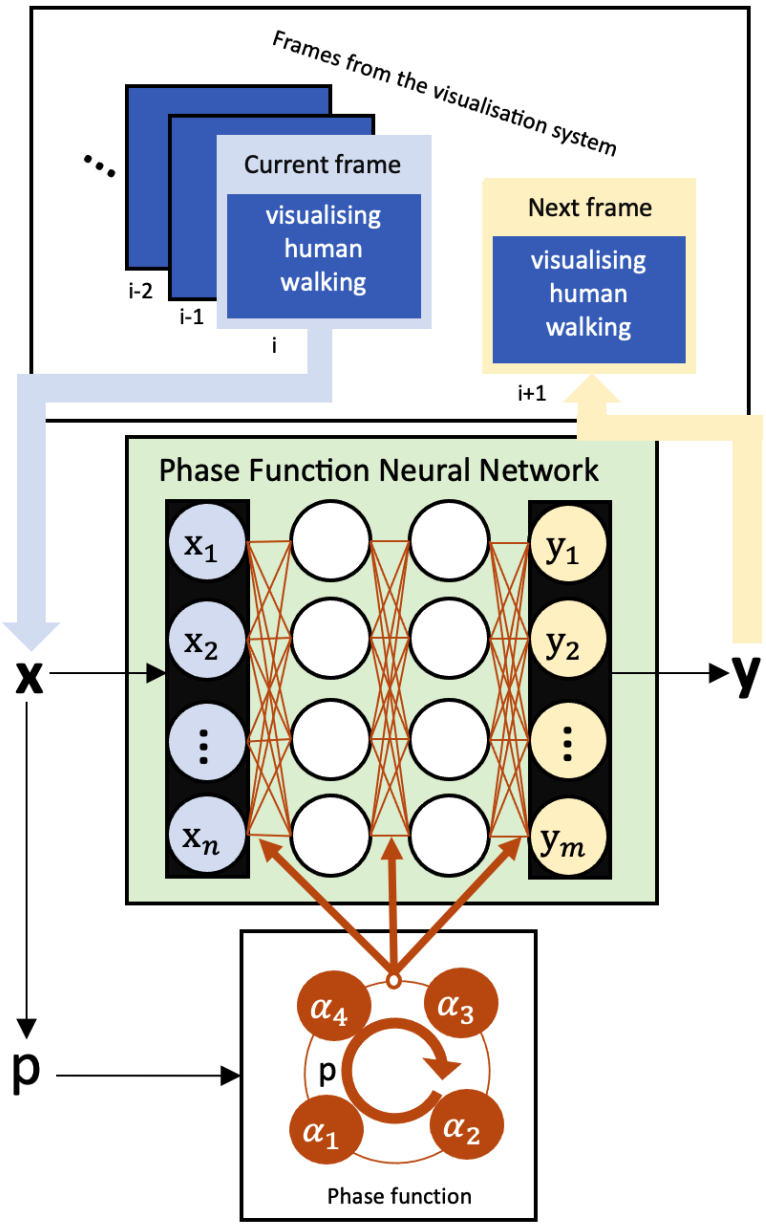


Figure 2.2: A visual representation of the PFNN model. This figure shows that while mapping the current motion state $\mathbf{x} \in \mathbb{R}^{n \times 1}$ at frame i to the next motion state $\mathbf{y} \in \mathbb{R}^{m \times 1}$ at frame $i+1$, the weights of the network are changed by the phase function, which is *not* a neural network. The walking phase $p \subset \mathbf{x}$ drives the phase function to select a mixture of the four network weight configurations $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$.

The main contribution was the introduction of the attack phase and the target position in the motion states. In contrast to the original findings from Holden et al. [21], Freda [14] concluded that the ERD model performs better than the PFNN. However, Freda [14] did not consider the trajectory of the hand joints and that of the sword. Therefore, the performance of PFNN-type models remains to be tested thoroughly.

However, PFNN has certain shortcomings for synthesising quadruped motions and that motivated Zhang et al. [63] to develop the MANN model. This model replaces the phase function with another feedforward neural network and achieves interactive motion synthesis for quadruped locomotion. The work presented in this thesis is based on the MANN model, and thus, the MANN model and the shortcomings of the PFNN are presented in more detail in Section 2.3.

The success of MANN has led to the use of a similar architecture for several interactive synthesis tasks. Starke et al. [55] first extended MANN for controlling a character to perform more complicated actions in the virtual environment, such as sitting on a chair and opening a door. The first main addition is that the motion state considers information in a bidirectional format (i.e. in the point of view of the character and in that of the object). Starke et al. [55] also introduced a volumetric representation capturing information about the presence of the character near an object and achieved good synthesis results for interaction with inanimate objects. The natural next step is to move from interaction with inanimate objects to more complex interactions involving other virtual characters. To this end, Starke et al. [56] extended the MANN model to interact with other moving characters in the environment in a one-on-one basketball setting. This work introduced local motion phases that avoid labelling limb movements w.r.t the global movement, and this representation enables the network to learn the motion of the body parts locally without aligning the entire body motion using a global phase. Similar to considering the point of view of the inanimate objects in [55], the model in [56] also considered opponent state information to produce more realistic motion around an active opponent.

However, the success of the approaches mentioned above depends on the foundation of the MANN model. Thus, in the next section, we present the MANN model in more detail and explain why it is relevant for synthesising boxing actions.

2.3 Mode Adaptive Neural Network

In this section, we present the mode adaptive neural network (MANN) model developed by Zhang et al. [63]. This model has been extended by Starke et al. [55, 56, 57] for interacting with inanimate objects, interacting with an opponent character and generating martial movements, respectively. We will first explain the nuances of this model and present our reasoning for choosing MANN-type models to develop our interactive boxing motion synthesis system.

In the previous section, we came across the PFNN model [21] as well as the MANN model [63]. The main difference between these two models is in the network structure. The PFNN, which forms the foundation for neural network-based interactive motion synthesis, consists of a simple four-layered neural network and a phase function as shown in Figure 2.2. However, the network has multiple sets of weights instead of just one. The number of sets is a parameter to be decided upon by a developer, and a higher number usually results in sharper motions. The phase function uses the walking phase to blend the different sets of weights into one set that is used for the current forward pass of the PFNN. The problem with the phase function is that while it may be suitable for

locomotion, it is not trainable. Hence, the phase function might not be suitable for other tasks. So, it is better to replace this function with a neural network that can be trained for each type of data that is considered. Such a neural architecture that replaces the phase function with another neural network has been implemented by Zhang et al. [63] in the MANN model.

In the pursuit of interactive control of quadruped locomotion and gait styles, Zhang et al. [63] came across the problem of labelling. Quadruped locomotion can have multiple different gaits like walk, pace, canter and gallop. For such motion, the movements and the phases of the torso and limbs are difficult to identify. So, even skilled annotators might mislabel the gaits. However, a direct application of the PFNN model requires explicit gait labels. Therefore, the primary motivation for Zhang et al. [63] to replace the phase function with a neural network was to avoid the problem of gait mislabelling for quadruped mocap data. However, this approach also benefited from allowing the neural network to extract its own features rather than using an explicitly defined function like the phase function. Such a design allows the MANN model to learn from unlabeled periodic gaits but also learn from non-periodic actions. The latter part is relevant for this thesis as the punching motion from our boxing mocap data is non-periodic.

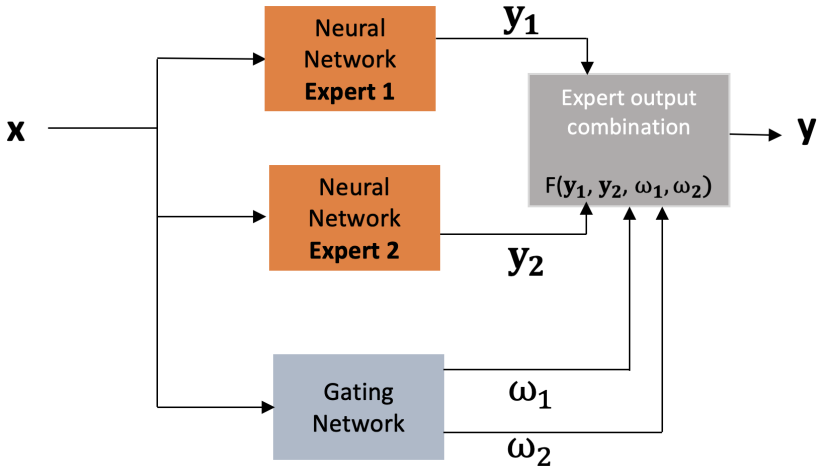


Figure 2.3: This figure shows a possible neural network MoE architecture for motion synthesis in the style of the original MoE approach [27]. As seen here, the original approach would involve at least two neural network experts that view the current motion state $\mathbf{x}$ and provide individual predictions ($\mathbf{y_1}$ and $\mathbf{y_2}$). For obtaining the next state $\mathbf{y}$, these individual predictions must be combined with a function $\mathbf{F}$ that uses the gating network outputs (ω_1 and ω_2) to prepare the final result $\mathbf{y}$ that can be animated.

The solution of the neural only architecture in MANN was inspired by the impressive results obtained by Chang et al. [9] in the DL literature. Their approach obtains state-of-the-art results in the computer vision task of person re-identification. The approach in [9] is based on the approach by Jacobs et al. [27], who originally presented the idea of combining the results from several expert models to improve the overall result using a gating network. These models are known as the mixture of experts (MoE) models in the ML literature. While the MANN model is inspired from such MoE approaches, the process of mixing in MANN is fundamentally different from the original approach in [27]. A possible direct implementation of the original approach for motion synthesis would be

as shown in Figure 2.3. As the figure indicates, the original approach of mixing would combine the results ($\mathbf{y_1}$ and $\mathbf{y_2}$) of the two expert networks with the gating network predictions (ω_1 and ω_2) to improve the overall result $\mathbf{y}$ (for more details, refer to the caption of Figure 2.3). However, the approach for mixing in MANN is fundamentally different.

Going into a bit more detail about the MANN architecture, we see that it consists of a motion prediction network and a gating network as shown in Figure 2.4. The motion prediction network is similar to the feedforward network from PFNN, and the gating network replaces the phase function. Both the networks in the MANN model are simple three-layer neural networks. The motion prediction network maps the current motion state $\mathbf{x} \in \mathbb{R}^{n \times 1}$ at frame i to the next motion state $\mathbf{y} \in \mathbb{R}^{m \times 1}$ at frame $i + 1$. Before this mapping occurs, the gating network first takes a subset $\hat{\mathbf{x}} \in \mathbb{R}^{n' \times 1}$ of the current motion state $\mathbf{x}$ and maps it to the blending coefficients ω. These coefficients are used to blend the K sets of weights $\boldsymbol{\alpha_k}$ (with $1 \leq k \leq K$) into a single set $\boldsymbol{\alpha}$ that is relevant to the motion prediction network to process the current motion state. For example, in Figure 2.4, $K = 4$ weight sets are combined to get the final weights $\boldsymbol{\alpha}$.

Therefore, we can clearly see that the neural network experts in MANN are utilised in a fundamentally different manner compared to the original approach from Jacobs et al. [27]. This difference is also apparent from looking into the combination and blending processes indicated in Figures 2.3 and 2.4. Essentially, in the original approach [27], the mixture is of the outputs of the expert networks. In the MANN model, the mixture is of the expert weights that result in the best weight configuration for the motion prediction network to handle the current input. Basically, as explained in [63], after training the MANN model, the expert weights (i.e. $\boldsymbol{\alpha_k}$ in Figure 2.4) will become locally specialized for some subdomain of the motion data. The mapping of a particular expert weight to a particular subdomain of the motion data is handled by the trained gating network. This ability of the gating network to learn effective subdomain mappings produces good motion synthesis.

Thus, with the ability to learn effective subdomain mappings, the MANN model has been extended in [55, 56] to cover various types of motions (explained in Section 2.2.2). The MoE-type models in [55, 56], produce realistic and responsive results for interactive motion synthesis. Additionally, the MANN model allows neural networks to extract their own features. This approach is in line with the DL literature and is shown to produce better results [49].

The quadruped data used in training the MANN consists of non-periodic motions such as jumping. A motion such as jumping occurs in short bursts and at random times in the mocap data. This motion is similar to the non-periodic motion of punching that we consider in this thesis. The other motion we consider is that of stepping, which moves the character from one point to another in the style of boxing. This stepping action is similar to walking, and MANN-type models have already been employed to achieve humanoid walking by Starke et al. [55, 56].

Therefore, we hypothesise that utilising the MANN model as the basis for the MoE models developed in this thesis would result in high-quality boxing motion synthesis. However, there is an added complexity of considering the exact target for the punches controlled by the user. Additionally, the stepping motion is sudden and occurs in bursts of a few frames and is not frequently occurring as a periodic motion in real boxing scenarios. In this thesis, we ensure that our MoE model can handle such nuances of boxing data.

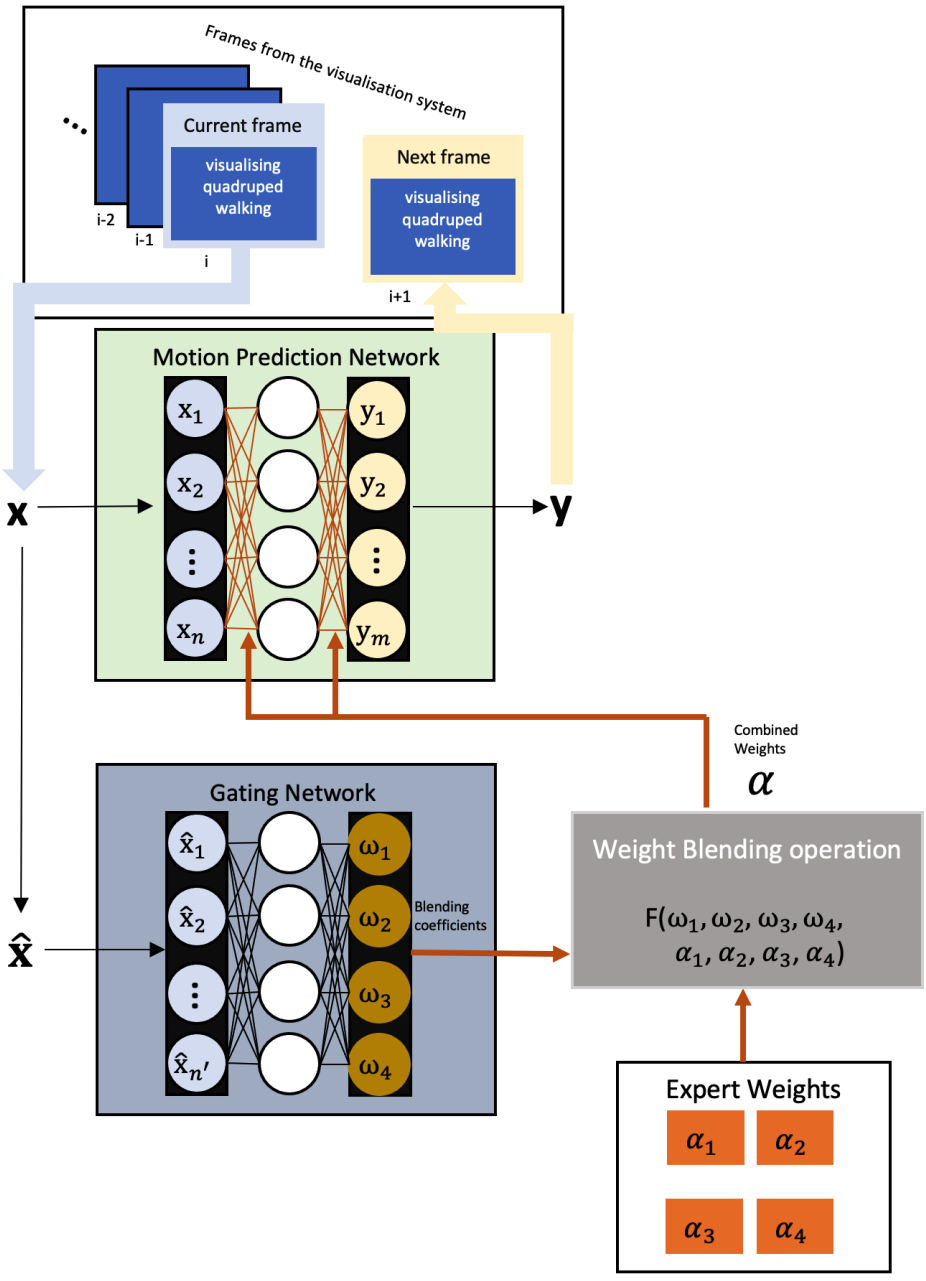


Figure 2.4: In this figure, we see the MoE style neural architecture from Zhang et al. [63] called the mode adaptive neural network. Here, the gating network considers a subset $\hat{\mathbf{x}} \in \mathbb{R}^{n' \times 1}$ from the current motion state $\mathbf{x} \in \mathbb{R}^{n \times 1}$ to predict a set of blending coefficients ω. These coefficients are utilised in a blending operation $\mathbf{F}$ to change the weights of the motion prediction network to $\boldsymbol{\alpha}$ which will be best for mapping the current motion state $\mathbf{x}$ to the next motion state $\mathbf{y} \in \mathbb{R}^{m \times 1}$. The best set of weights $\boldsymbol{\alpha}$ is obtained by combining the four expert weights $\boldsymbol{\alpha_k}$ with $1 \leq k \leq 4$.

Note that Starke et al. [57] presented another approach, after the implementation of our MANN-based boxing model, that can not only be used for editing data by animators but can also be used for interactive motion synthesis. This work can generate martial arts actions such as punching and kicking and handle interactions with other characters. The motivation for [57] is to allow animators to edit the motion intuitively without requiring them to change their workflow as they are already used to techniques such as layering. Along with their neural model, Starke et al. [57] also present a controller that allows for both animator editing and interactive character control. Thus, the motivation in [57] is different from ours as we focus only on interactive synthesis as presented in Section 1.1. We also discuss how our results compare against the ones in [57] in Chapter 7.

However, before the discussion, we provide more information about how our MANN-based approach considers the nuances of boxing data and develops MoE models capable of interactive boxing motion synthesis in Chapter 4. Nevertheless, before explaining our MoE model, we will first present the mocap setup for collecting our boxing data and the statistics about our data in the next chapter.

Chapter 3
Data Acquisition

As we saw in Chapter 1, motion synthesis systems have many applications, and therefore, there exists an active community that has produced several open-source mocap datasets. Searching through these databases ([39] or [32]) shows that punch action data is already available. However, it cannot be used easily for our boxing application because several factors such as the capturing technology and skeletal representations used varies. Moreover, most of the available motion clips contain a single punch, which is insufficient for our neural-based method.

In addition, boxing motions have a specific style because both the stepping and punch actions occur in short bursts. There is also a significant difference between punches that start from a still posture and those that start while the character is in motion. Hence, to ensure that boxing style is represented in the data and to obtain a wide variety of data, we decided to capture our own data. The wide variety of actions is ensured by giving specific instructions to the mocap actor.

Motion capture is the method of mapping the motions of an actor from the real world to a virtual environment. Several commercially available mocap systems enable us to achieve such a mapping. These systems can be set up using optics-based sensors or non-optics based sensors. There are also video-based mocap techniques that do not need any markers on the mocap actor. We will limit our discussion to the non-optics based sensors, as our mocap system is based on such sensors. However, more information can be found about the other methods in a review on mocap systems by Menolotto et al. [40].

Non-optical based methods usually require an actor to don a mocap suit. This suit consists of sensors that are placed close to the joints in a typical human body. The system will also contain associated software to map the motions to the virtual environment. As explained by Menolotto et al. [40], while there are other non-optical mocap systems like exoskeletons, the best ones utilise inertial measurement units (IMUs) as the primary sensors in the suit. These systems can record outdoors, and the algorithms in the associated software can automatically deal with occlusions. The motion recorded and processed by the software is stored in one of the ways explained by Meredith et al. [41].

Most IMU-based mocap systems require that the mocap suit be close enough to a linked computer running the motion recording software. However, actions such as stepping for

our boxing data requires an ample amount of space. To overcome this problem Troje [60] suggests using a Treadmill, but this is not feasible for our data. Moreover, IMU-based systems struggle to synchronise and record motions when there is magnetic interference nearby. Therefore, we cannot use the treadmill approach from Troje [60]. Considering these points, the mocap setup for this thesis employs a relatively large area which is described further in Section 3.1.

Another critical point brought up by Troje [60] is that if the actor is hyper-focused on the motion, it might lead to unrealistic movements. So, Troje [60] suggests using a 5-minute interval between capturing sessions to allow the actor to get habituated to the suit and the recording process. This interval is designed to avoid biases that might enter the data. Therefore, as we aim to capture realistic boxing motions, the mocap setup for this thesis utilises a small interval between each mocap trial, which is explained further in Section 3.1.

After the capturing process, we apply processing algorithms available in the associated software (see Section 3.1). The punches in the resulting motion data are first automatically labelled using the scheme presented in Section 3.2, and the labels are utilised to analyse the punches in our boxing data. Next, the stepping action is analysed by visual inspection and all the analysis results are presented in Section 3.3.

The following section presents more detailed information about the mocap session. In particular, we will cover the instructions given to the actor, information about the actor and the tools used for the capturing process.

3.1 Motion Capture

In this section, we see the precise details about the components involved in our mocap process and the procedure we followed to collect the boxing data.

Motion capture system

The mocap system used in this thesis is the Xsens MTw Awinda [6]. This system is a wireless inertial-magnetic motion tracker that consists of 17 IMU sensors that were mounted using velcro body straps on the following body parts: head, sternum, pelvis, upper legs, lower legs, feet, collar bones, shoulders, elbows and wrists as shown in Figure 3.1. The information from these 17 sensors is used to drive a virtual skeleton (same as Figure 2.1) in the Xsens MVN Animate software. In our case, the software sampled the sensors of the suit at 60 Hz, which means that one frame from the generated data corresponds to approximately 0.017s of motion. The suit and the associated software have been thoroughly analysed by Paulich et al. [43] and Schepers et al. [50], respectively, showing the capability of capturing high-quality data.

However, one drawback of IMU-based sensors is that they can result in the feet of the character skating along the virtual ground. This problem is solved by the foot skating correction offered by the software as explained in [7]. Another problem is that the presence of nearby magnetic fields will result in poor recording quality. To overcome this, we hold our motion-capture recording session outdoors in a sufficiently large area.

Figure 3.1: Images showing the Xsens MTw Awinda mocap suit donned by our mocap actor. The orange coloured blocks located near the body joints are the IMU sensors.

Motion capture actor

Due to the ongoing coronavirus pandemic, it was difficult to recruit participants for the mocap session. Additionally, as the goal is to capture boxing specific data, the participants need to be trained in boxing. Therefore, we restricted the motion capturing to one actor with beginner level of boxing experience. This actor was also familiar with motion capturing technology, and thus, he was not hyper-aware of motion capturing. So the resulting actions we recorded were as realistic as possible.

This particular actor follows an orthodox boxing stance, meaning that the left foot is in the front and the right foot is in the back. The height of the actor is 1.9m, and the actor is physically fit to perform the boxing motions of stepping and punching. The participation of the actor in this mocap session was completely voluntary.

Motion capture area

A typical boxing ring has a dimension of 6.5m x 6.5m, and it is a flat surface. Based on this, we selected the area seen in Figure 3.2 for our mocap session. This area is a flat surface on the Saarland Informatics campus, located in front of the German Research Center for Artificial Intelligence in Saarbrücken. The approximate dimensions of the capturing area are 4.5m x 12m. Thus, the capturing area has slightly different sizes but nearly the same area as a boxing ring. However, as we will see further, the actor is instructed to move only along the longest dimension (12m). This instruction is to obtain better trajectory information for the stepping action. Additionally, this outdoor area was chosen for the motion capturing as it is more suitable for the IMU sensors in the motion capturing suit.

Figure 3.2: The figure shows the 4.5m x 12m motion capturing area that was utilised for gathering the boxing mocap data.

Procedure

IMU-based systems need manual input of the body dimensions of the actor for calibration purposes. Thus, with the consent of the participant, the necessary information like height and distance between different joints in his body are measured. These values are entered in the Xsens MVN Animate software, followed by calibration tests where the actor is instructed to perform the T-pose and then to execute shadow boxing freely. These initial sessions are not recorded, as they are for calibrating the Xsens Awinda mocap system and also to allow the participant to get habituated to the suit and the capturing process.

The participant is then informed about the kind of actions to be performed. The general instruction is for the participant to freely go through the boxing motions using his natural boxing stance. However, we instruct the participant to cover particular actions in specific ways. These additional instructions are as follows:

1. **Move forward:** The participant is instructed to move forwards along the longest side of the motion capturing area while in the boxing stance.
2. **Move backward:** Similarly, the participant is instructed to move backwards in the boxing stance.
3. **Punch:** The participant is instructed to throw punches around the levels of the head, chest and abdomen of an imaginary opponent. The expectation is to obtain motion data for the following punches: jab, cross, lead hook, rear hook, lead uppercut and rear uppercut.

The instructions are not strict and are considered as guidelines to ensure that all the necessary motions are covered in the motion capturing process. During a recording trial, the participant can choose his action and the order of his actions freely. For example, he is allowed to punch while moving forward, rotate upon reaching the bounds of the capturing area, and in general, move about freely inside the capturing area.

The capturing process consisted of seven sessions. A standard round in a boxing match lasts for five minutes, but based on the skill level of our participant, each session was designed to be for around two minutes to ensure that the actor does not get fatigued. Moreover, if the participant is asked to perform motions continuously, it might lead to unnatural movements. Troje [60] uses habituation intervals of five minutes to solve this issue. Based on this, we ensure at least two minutes of cooldown time between each trial. Additionally, to verify the proper functioning of the mocap system, we start each trial with the participant in the T-pose. Once the calibration is confirmed, we instruct the participant to perform the boxing motions for the trial.

Post-processing

Once we have the mocap data, a post-processing tool is used on the Xsens MVN Animate software to correct foot skating problems as shown in [7]. We also utilise the high-quality export option and specify that the recording was on a flat surface in the software and finally obtain our boxing motion data in the Biovision Hierarchy format explained in[41]. The raw data was cropped to the relevant boxing mocap movements using Blender [11], a 3D modelling software tool. After all the post-processing, we obtain around 10mins of high-quality boxing mocap data.

3.2 Punch Annotation

In this section, we present the approach that was used to label the punches automatically. To enable such automatic labelling, we use a 3D modelling software called Blender [11]. In this section, we provide details about the punch labelling process.

The post-processed data from MVN Animate is loaded into the Blender software [11], where scripting is used to access the position of the bones of the virtual character. Then Algorithm 1 is used to identify the frame at which the arm of the virtual character is fully extended and this is followed by Algorithm 2 identifying the start and end frames of the punch. These algorithms are designed to identify all the different types of punches captured in the dataset. Our labelling approach determines the start, full extension and end frames of each punch and assigns appropriate labels at each frame.

Algorithm 1 is used to find the next frame at which the punch action is completed. To do this, we check whether the arm has reached the full extension and started moving backwards. Similarly, for the case of the uppercut-style punches, we check whether the hand has reached the maximum height and has begun moving backwards. The frame at which the backward motion has already started is used to obtain the next frame where the punch is completed.

Algorithm 1 This is used for detecting the next frame at which a punch is completed.

Input: $Hand, num\_frames, start$
Output: $NextPunchFrame, NextPunchFullExtension, HeightData$
Ensure: $Search\_Threshold \geq 0$ ▷ Manually set.

$Last\_Distance, Last\_Height \leftarrow 0$
$HeightData \leftarrow False$
for f in range($start, num\_frames$) **do**
  $Height\_Threshold \leftarrow Neck\_Pos$
  $Hand\_Height \leftarrow Hand\_Pos\_y$ ▷ "y" co-ordinate
  $Hand\_Distance \leftarrow \|(Shoulder\_Pos - Wrist\_Pos)\|$
  if $Hand\_Distance > Search\_Threshold$ **then**
    if $Hand\_Distance - Last\_Distance$ is negative **then**
      $NextPunchFrame \leftarrow f - 1$
      $NextPunchFullExtension \leftarrow Hand\_Distance$
      $HeightData \leftarrow False$
      return $NextPunchFrame, NextPunchFullExtension, HeightData$
    end if
  else if $Hand\_Height > Height\_Threshold$ **then**
    if $Hand\_Height - Last\_Height$ is negative **then**
      $NextPunchFrame \leftarrow f - 1$
      $NextPunchFullExtension \leftarrow Last\_Height$
      $HeightData \leftarrow True$
      return $NextPunchFrame, NextPunchFullExtension, HeightData$
    end if
  end if
  $Last\_Distance \leftarrow Hand\_Distance$
  $Last\_Height \leftarrow Hand\_Height$
end for

Algorithm 2 determines the start and end frames of the punch. This is done by looking forward and backwards in time from the frame at which the next punch was detected by Algorithm 1. The outward stretching punches are labelled from the frame where the magnitude of the difference between the shoulder and wrist positions exceeds a certain threshold called the $Punch\_Threshold$. For the case of the uppercuts, the labelling is started as soon as the wrist is above the height of the shoulder. In some cases, where the actor is getting warmed up or is tired, the punches are slow and at such edge cases we assigned a frame number at a distance of 25 frames from the frame at which the punch was identified by Algorithm 1. This number is chosen by manually verifying the number of frames taken for a punch and minimising the number of cases where the edge cases appear.

Algorithm 2 This is used for detecting the start or end frame of a punch.

Input: $Hand, NextPunchFrame, NextPunchFullExtension, HeightData, direction$
Output: $PunchStartOrEnd$
Require: $Punch\_Threshold > 0$ ▷ Manually set.
Ensure: $direction$ is 1 or -1. ▷ $1 \Rightarrow forward, -1 \Rightarrow reverse$

$Last\_Distance \leftarrow \|(Shoulder\_Pos - Wrist\_Pos)\|$
$End \leftarrow NextPunchFrame + (direction * 25)$ ▷ 25 is manually selected
$Step \leftarrow direction$
for f in range($NextPunchFrame, End, Step$) **do**
  $Last\_Distance \leftarrow \|(Shoulder\_Pos - Wrist\_Pos)\|$
  if not $HeightData$ **then**
    $Hand\_Distance \leftarrow \|(Shoulder\_Pos - Wrist\_Pos)\|$
    if $(NextPunchFullExtension - Hand\_Distance) \geq Punch\_Threshold$ **then**
      $PunchStartOrEnd \leftarrow f$
      return $PunchStartOrEnd$
    end if
  else if $HeightData$ **then**
    if $NextPunchFullExtension > Shoulder\_Pos\_y$ **then**
      $PunchStartOrEnd \leftarrow f$
      return $PunchStartOrEnd$
    end if
  end if
end for
$PunchStartOrEnd \leftarrow End$
return $PunchStartOrEnd$

A labelling scheme was adopted in which the frames from start to full extension (including full extension) are assigned the label "1", and the frames from full extension to the end of a punch are assigned "-1". Essentially, the forward motion of the arm, including the full arm extension, is assigned 1, and the backward motion is assigned -1. We treat all other frames as motion data that does not contain punch information and assign the label of 0. To set up this automatic labelling, manual identification of certain thresholds by verifying the generated labels against the mocap data was necessary. The best thresholds (see Algorithms 1 and 2) found for our boxing dataset are $Search\_Threshold = 0.04m$ and $Punch\_Threshold = 0.19m$ for both the left and the right arms. Note that we employed the $\mathbf{l^2}$ norm in Algorithms 1 and 2 and the thresholds mentioned earlier had to be

determined w.r.t. this norm. The results from this labelling scheme and simple manual inspection are utilised in the next section to analyse the newly prepared boxing dataset.

3.3 Data Analysis

In this section, we analyse the boxing dataset that was prepared. Then, we present the statistics for the stepping action and punching action separately. These statistics directly influence the parameters of the boxing controller (see Chapter 5) and are very relevant for various decisions throughout this thesis.

Note that our mocap system captured motion data at 60 Hz, which essentially means that each frame of our boxing data corresponds to approximately 0.017s of motion in the real world. However, in the following text, the statistics for stepping and punching are presented at the frame level as it is easier to follow for the readers.

Analysis of stepping action:

As explained earlier in Section 3.1 the character in the virtual world of the MVN Animate software can start skating along the floor. To overcome this issue, the software provides an automatic foot skating correction tool that was employed on our boxing data. After this application, we computed the average foot skating values for our entire dataset. The exact equation used is Equation (6.7) and it is presented in detail in Section 6.1.2, as it is an evaluation metric used for the experiments. The results of computing the foot skating in our dataset are presented in Table 3.1. These values are very low for our dataset as the proprietary foot skating correction algorithm in the MVN Animate software performs a good job.

Table 3.1: This table presents the average foot skating information computed over all frames in our boxing data.

Step type	Average foot skating $(cm/frame)$
right	0.058
left	0.171
overall	0.114

For the case of stepping action, a simple visual inspection was employed to determine the start and end frames of the steps in our boxing dataset. From this, the average duration of steps is computed over all the frames in our dataset. The results are shown in Table 3.2, and as per these results, the average left step duration is much lower than that of the right step. This difference is because of our mocap actor using the orthodox boxing stance in which the left foot is forward. Therefore, each step using the left foot is much shorter than the one with the right foot. This overall step duration is utilised to select the parameters for training our models in Chapter 6.

Table 3.2: This table presents the average number of frames per step, in our boxing data, for different configuration of steps.

Step type	Average number of frames
right	26.00
left	14.50
overall	20.25

Analysis of punching action:

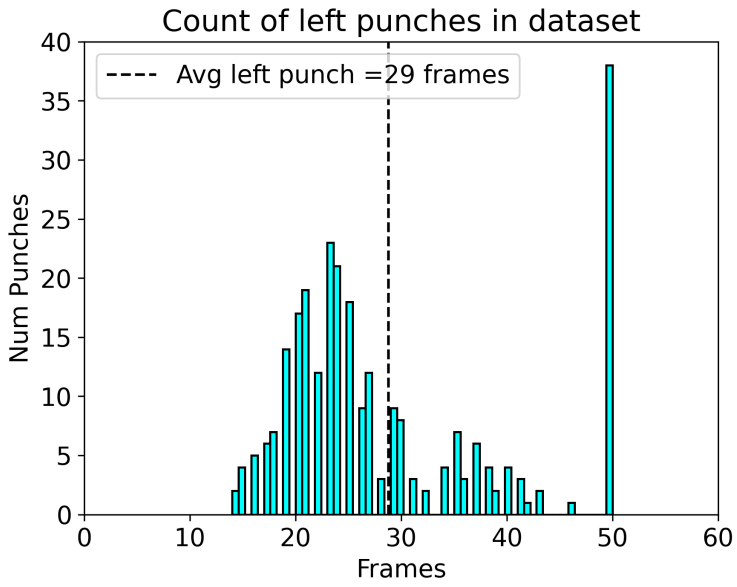
As our primary focus is the punching action, we invested more time in its analysis. The labels from the automatic punch labelling presented in Section 3.2 helped in computing the punch statistics. Based on this, the number of punches in the boxing dataset and their duration statistics were calculated. The analysis also considered the forward and backward motions of the arm separately within the punches.

The results of the punch analysis are presented in Table 3.3. As seen in the table, we have 451 punches in our boxing data. However, we have more left-handed punches in our data as the actor was allowed to box freely, and he preferred jabs using the left hand. But we still have a significant number of right-handed punches, and these punches are sufficient for training our models. The mean duration of these punches is utilised to set up parameters for the punch control experiments in Chapter 6. Additionally, the maximum punch duration computed here is used in selecting the parameters in the boxing controller system presented in Chapter 5.

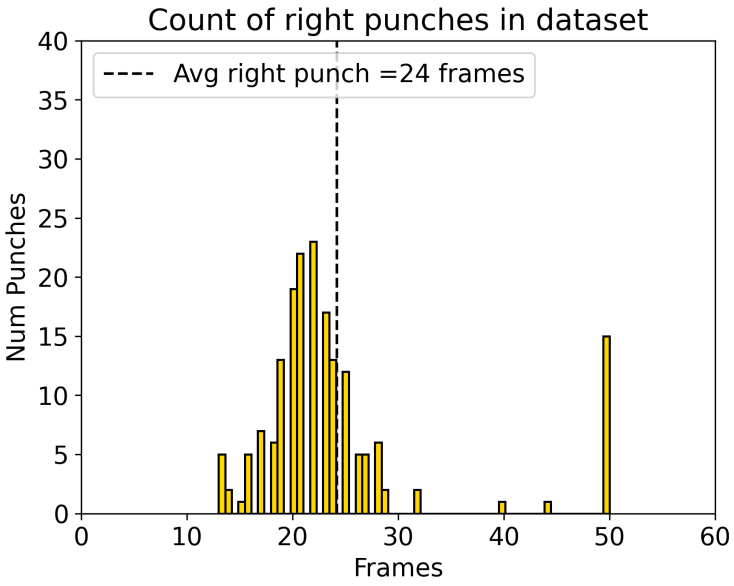
Table 3.3: This table presents the statistics for the all the punches in our boxing data.

Punch type	Number of punches	Minimum duration *(in frames)*	Mean duration *(in frames)*	Median duration *(in frames)*	Maximum duration *(in frames)*
right, forwards	182	7	13.71	12.5	26
right, backwards	182	5	10.48	9.0	24
left, forwards	269	3	15.45	14.0	26
right, backwards	269	6	13.35	10.0	24
right, full punch	182	13	24.19	22.0	50
left, full punch	269	14	28.80	25.0	50
overall	451	13	26.94	23.0	50

The punch labels were also utilised to generate the histogram plots in Figure 3.3. These plots show the distribution of the punch duration, separately for the left and right hands, in terms of frames. The cluster of punches with a duration of 50 frames stems from the limitation of the automatic labelling done by Algorithms 1 and 2. These spikes are seen for both the left and the right-handed punches in Figures 3.3a and 3.3b respectively. This spike is a direct result of the manually selected duration parameter of 25 frames in Algorithm 2. The particular value of 25 frames was selected as most of the recorded punches were of short duration as seen in Figure 3.3.



(a)



(b)

Figure 3.3: Histogram plots showing the distribution of the punch durations for the left hand Figure 3.3a and the right hand Figure 3.3b. 60 frames corresponds to 1s of motion.

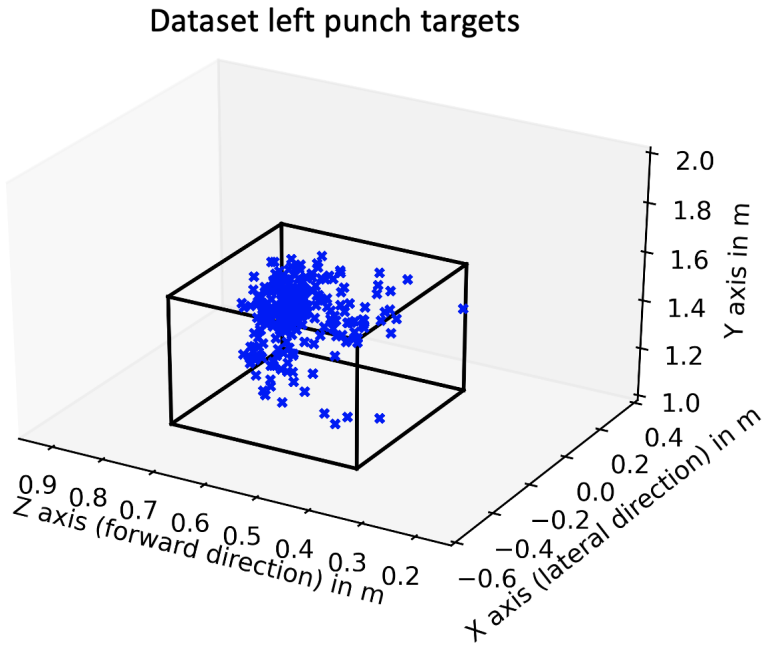
Visual inspection of the punches labelled as taking a duration of 50 frames reveals that these are also punches that are correctly detected by Algorithm 1. However, the actor is throwing these punches with lesser momentum compared to the others in the dataset. Such punches are found at the beginning of the recording corresponding to a trial or when the actor is fatigued. These outlier punches are primarily due to the actor still

getting habituated to consistently generating the requested actions. This is a habituation issue as explained by Troje [60]. One way to deal with these outlier punches would be by discarding all the punches taking a duration of 50 frames. However, our mocap data is already limited compared to the amount of data used currently in the literature (see Zhang et al. [63], Starke et al. [55, 56, 57]). Therefore, we decided to include these outlier punches as part of our dataset.

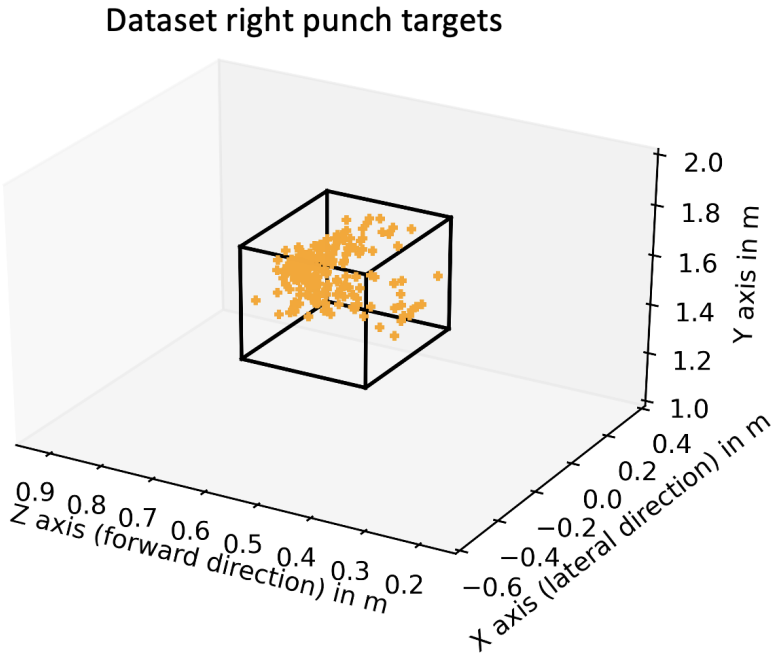
We also extracted the punch targets from the dataset using the associated labels. These targets are the wrist positions at the end of the full extension of the arm marked by Algorithm 1. These targets were then converted to the local space of the virtual character using Equation (4.12) (explained in Section 4.2). The resulting punch targets in the local space of the character are as seen in Figure 3.4. These figures are also indicating the bounds of the targets in the dataset with cubes in the plots. The important aspect to note here is that the punch targets are not spread out more inside the cube that demarcates the reach of the actor's punches. While this does not directly influence any of our design decisions, it will be an important factor in understanding the motion synthesis results from the models developed in this thesis (see Chapter 6).

Summary of boxing data analysis:

We use the details from the punch statistics and the target punches extracted from the dataset to develop the controller. Both the punch and the step statistics are used to determine feasible parameters of our system that we can evaluate. We will refer to the statistics for design decisions of our controller system in Chapter 5 and for selecting viable parameters for model evaluation in Chapter 6. However, in the next chapter, we will first present the MoE boxing model that is central to performing the mapping from the current motion state to the next one.



(a)



(b)

Figure 3.4: Target punch positions, in the local space of the virtual character, for the left hand (Figure 3.4a) and the right hand (Figure 3.4b). The cubes in the figures mark the bounds of the targets in the data. Note that the punch targets are not spread out inside the actor's punch bounds marked by the cube.

Chapter 4
Mixture of Experts Neural Model

In this chapter, we present information about the design and implementation of our MoE model for boxing motion synthesis. Our approach is inspired by the MANN model from Zhang et al. [63], which is an MoE-type model as explained in Section 2.3. However, a direct application of this model would not be suitable for boxing because the motion state representation used by Zhang et al. [63] was designed for quadruped locomotion. Therefore, we develop our own motion state representation. Accordingly, there were changes in the input and output layers of our MoE model when compared to the MANN model. In this chapter, we will discuss in Section 4.1 about how we extended the MoE model from [63] for boxing. While the model forms one necessary part, the other equally necessary part is the motion state representation used to control boxing actions. Therefore, this chapter also presents in Section 4.2 our boxing motion state representations that are used to train the MoE models.

4.1 Model Definition

We construct the network using an architecture similar to the one presented by Zhang et al. [63]. This architecture consists of a motion prediction network and a gating network. These two networks can together perform a regression from the current motion state to the next one, and this is achieved by combining the results computed by each of them. However, the combination process here is not trivial and the motivation behind utilising a novel combination process in the MoE-type MANN model is presented in Section 2.3. In this section, we focus on explaining the design of the model and the approach to achieve the non-trivial combination of the outputs from the motion prediction and gating networks.

The motion prediction network performs the mapping from the current motion state $\mathbf{x}$ to the next motion state $\mathbf{y}$. The gating network facilitates this mapping by dynamically computing blending coefficients ω. These coefficients are used to dynamically compute the expert weights $\boldsymbol{\alpha}$ of the motion prediction network. In order to achieve this, the

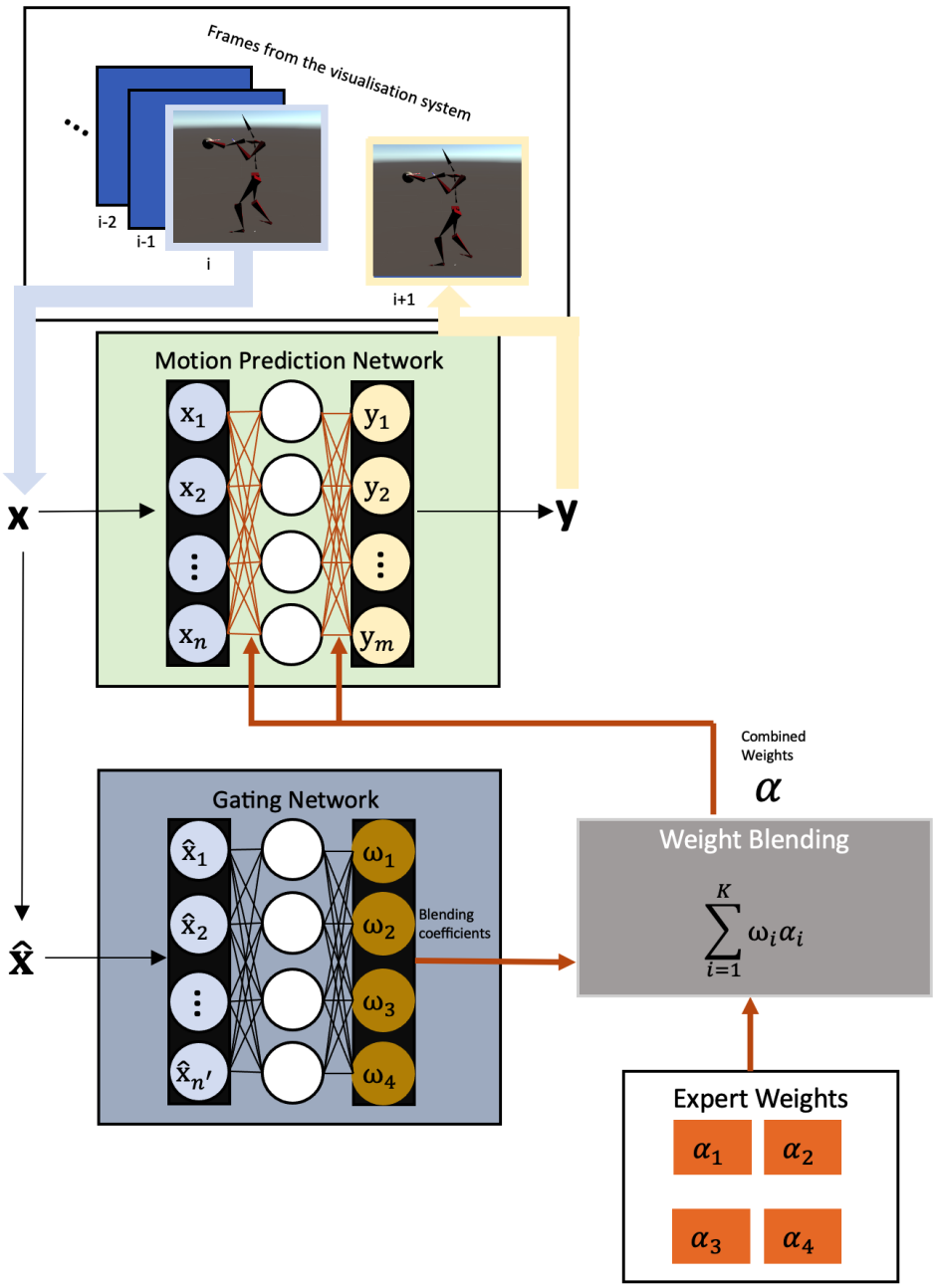


Figure 4.1: This figure shows the neural networks i.e. the motion prediction network and the gating network, in our MoE boxing model. The gating network uses a subset $\hat{\mathbf{x}}$ of the current boxing state $\mathbf{x}$ to predict a set of blending coefficients ω. These coefficients are used to dynamically compute the combination of K expert weights resulting in the current weights $\boldsymbol{\alpha}$ of the motion prediction network, which maps the current boxing state $\mathbf{x}$ to the next boxing state $\mathbf{y}$. For example, in this figure we have $K = 4$ expert weight sets.

gating network accepts a subset $\hat{\mathbf{x}}$ of the input motion state presented to the motion prediction network. Figure 4.1 visualizes this mapping process using our MoE boxing model.

As both of these models are fundamental units of the MoE boxing model developed in this thesis, we will present each of them in detail. First, we present the motion prediction network in Section 4.1.1, and then we cover the gating network in Section 4.1.2. The equations for the networks presented in this section are obtained from Zhang et al. [63], and they have been adapted for our boxing data and experiments.

4.1.1 Motion Prediction Network

When considering the motion prediction network independently, it is a simple feedforward neural network. It is designed to have three layers. The number of neurons in all of these three layers will be varied in our model based on the experiments presented in Chapter 6. However, coming to the functioning of the network, it receives the current state of the character **x** and maps this state to the next state **y**. We represent this mapping by $\mathbf{\Theta}(.)$ in the following way:

$$\mathbf{\Theta}(\mathbf{x}, \boldsymbol{\alpha}) = \boldsymbol{W_2}\mathrm{ELU}(\boldsymbol{W_1}\mathrm{ELU}(\boldsymbol{W_0}\boldsymbol{x} + \boldsymbol{b_0}) + \boldsymbol{b_1}) + \boldsymbol{b_2} \tag{4.1}$$

The above Equation (4.1) indicates that the motion prediction network needs a set of network parameters $\boldsymbol{\alpha}$ and the current motion state **x** as input. We defined the parameters $\boldsymbol{\alpha}$ of our motion prediction network as follows:

$$\boldsymbol{\alpha} = \{\boldsymbol{W}, \mathbf{b}\} \tag{4.2}$$

$$\boldsymbol{W} = \{\boldsymbol{W_0} \in \mathbb{R}^{\mathbf{h}\times\mathbf{n}}, \boldsymbol{W_1} \in \mathbb{R}^{\mathbf{h}\times\mathbf{h}}, \boldsymbol{W_2} \in \mathbb{R}^{\mathbf{m}\times\mathbf{h}}\} \tag{4.3}$$

$$\mathbf{b} = \{\mathbf{b_0} \in \mathbb{R}^{\mathbf{h}}, \mathbf{b_1} \in \mathbb{R}^{\mathbf{h}}, \mathbf{b_2} \in \mathbb{R}^{\mathbf{m}}\} \tag{4.4}$$

In Equation (4.3), $\boldsymbol{W}$ represents the weights of the network and in Equation (4.4), $\boldsymbol{b}$ represents the bias parameters in the network. In these equations, the length of the input vector is represented by $\mathbf{n}$, the length of the output vector by $\mathbf{m}$ and the number of hidden neurons by $\mathbf{h}$. In our experiments, we vary $\mathbf{n}$, $\mathbf{h}$ and $\mathbf{m}$, and more information is presented in Chapter 6.

As in the original MANN model [63], we also employ the exponential rectified linear unit as the activation function. Considering an input $\mathrm{z} \in \mathbb{R}$, this function is defined as follows:

$$\mathrm{ELU}(\mathrm{z}) = \max(\mathrm{z}, 0) + \exp(\min(\mathrm{z}, 0)) - 1 \tag{4.5}$$

The weights of the motion prediction network indicated in Equation (4.2) are computed by blending K expert weights $\boldsymbol{\beta} = \{\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, ..., \boldsymbol{\alpha}_K\}$. The different sets of weights from $\boldsymbol{\beta}$ are combined to produce the final set of weights for the current motion state input **x**. The final set of weights $\boldsymbol{\alpha}$ is computed as follows:

$$\boldsymbol{\alpha} = \sum_{i=1}^{K} \omega_i \boldsymbol{\alpha_i} \tag{4.6}$$

In the above Equation (4.6), $\omega = \{\omega_1, \omega_2, ..., \omega_K\}$ represents the set of K blending coefficients computed by the gating network. Note that number of expert weights and the number of blending coefficients must be equal to K. We experimented to ascertain the best value of K for our boxing motion data, and more information about this will be presented in Chapter 6. However, in the next section, we will present details about the gating network.

4.1.2 Gating Network

Similar to the motion prediction network, the gating network is also a simple feedforward neural network. It also consists of 3 layers. Note that the number of neurons in the input and output layers will be varied in our model based on the experiments covered in Chapter 6. However, the number of neurons in the hidden layer is fixed to 64. Coming to the functioning of the network, it receives a subset of the current state of the character $\hat{\mathbf{x}}$ and maps it to the blending coefficients ω. We represent this mapping by $\boldsymbol{\Omega}(.)$ in the following way:

$$\boldsymbol{\Omega}(\hat{\mathbf{x}}, \boldsymbol{\mu}) = \sigma(\hat{\boldsymbol{W}}_{\mathbf{2}}\mathrm{ELU}(\hat{\boldsymbol{W}}_{\mathbf{1}}\mathrm{ELU}(\hat{\boldsymbol{W}}_{\mathbf{0}}\hat{\boldsymbol{x}} + \boldsymbol{b}'_{\mathbf{0}}) + \boldsymbol{b}'_{\mathbf{1}}) + \boldsymbol{b}'_{\mathbf{2}}) \tag{4.7}$$

The above Equation (4.7) shows that the gating network needs a set of network parameters $\boldsymbol{\mu}$ and a subset of information $\hat{\mathbf{x}}$ from the current motion state. While this is similar to Equation (4.1) which was defined for the motion prediction network, we have an additional function $\boldsymbol{\sigma}$. This function in Equation (4.7), represents the softmax function to normalize the predictions such that they sum up to 1. This normalization is required for the linear blending given by Equation (4.6). Coming back to the parameters $\boldsymbol{\mu}$ of our gating network, we defined them as follows:

$$\boldsymbol{\mu} = \{\hat{\boldsymbol{W}}, \mathbf{b}'\} \tag{4.8}$$

$$\hat{\boldsymbol{W}} = \{\hat{\boldsymbol{W}}_{\mathbf{0}} \in \mathbb{R}^{\mathbf{h}' \times \mathbf{n}'}, \hat{\boldsymbol{W}}_{\mathbf{1}} \in \mathbb{R}^{\mathbf{h}' \times \mathbf{h}'}, \hat{\boldsymbol{W}}_{\mathbf{2}} \in \mathbb{R}^{K \times \mathbf{h}'}\} \tag{4.9}$$

$$\mathbf{b}' = \{\mathbf{b}'_{\mathbf{0}} \in \mathbb{R}^{\mathbf{h}'}, \mathbf{b}'_{\mathbf{1}} \in \mathbb{R}^{\mathbf{h}'}, \mathbf{b}'_{\mathbf{2}} \in \mathbb{R}^{K}\} \tag{4.10}$$

In Equation (4.9), $\hat{\boldsymbol{W}}$ represents the weights of the network and in Equation (4.10), $\mathbf{b}'$ represents the bias parameters in the network. In these equations, the length of the input vector is represented by $\mathbf{n}'$, the length of the output vector by K and the number of hidden neurons by $\mathbf{h}'$. In our case, we set $\mathbf{h}'$ to 64 as we found that the results looked sharper for our boxing data. This results in double the number of hidden neurons compared to the original MANN model [63]. Lastly, similar to the motion prediction network, the activation function used for the gating network is the exponential rectified linear unit given by Equation (4.5).

The motion prediction and gating networks together form the MoE model that we use to control boxing motions. In the following section, we will present the details regarding the training of this model.

4.1.3 Training Details

Our MoE model is working on the time series problem of transforming the current motion state $\mathbf{x}$ to the next motion state $\mathbf{y}$ (using $\hat{\mathbf{x}} \subset \mathbf{x}$ for the gating network). So our model works on the frame level of the data. To train our model to perform such a function, we first process the motion capture data (see Sections 3.1 and 3.2) and extract important information (presented in Section 4.2) to obtain the vectors $\mathbf{x}$, $\hat{\mathbf{x}}$ and $\mathbf{y}$ for each frame. These vectors are then stacked to obtain the data for training our models.

The training data, after stacking, will have the form: $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \dots]$, $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}, \dots]$ and $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_\mathbf{1}, \hat{\mathbf{x}}_\mathbf{2}, \dots]$. However, our boxing data is obtained from time-series information, and using $\mathbf{X}$ and $\mathbf{Y}$ directly without re-ordering the data points would lead the model to learn the time-series nature of our data from the ordering of the data pairs (for example, $(\mathbf{x_1}, \mathbf{y_1})$ is a data pair). Since we want the network to learn the time-series information from the variables within the motion states (see Section 4.2), the input and output data pairs from $\mathbf{X}$ and $\mathbf{Y}$ are randomised similar to the approach by Zhang et al. [63] (while the order in $\hat{\mathbf{X}}$ is always synchronised with $\mathbf{X}$ as $\hat{\mathbf{x}} \subset \mathbf{x}$). This randomisation is done w.r.t. the time domain of the mocap data, and it ensures that the motion states used for training is not in the order that they appear in the base mocap data. Therefore, the model learns the mapping from the current motion state to the next one, and the motion states inform the model about the time-series nature of our data.

The values in this training data are normalised by shifting and scaling by using their mean and standard deviation. In this thesis, the mean and standard deviation varies based on the different motion states considered in Chapter 6. However, in each case, the normalised data is used to train the model with the mean squared error loss function as in the original MANN model [63]. This loss is defined in the following equation:

$$Cost(\mathbf{X}, \mathbf{Y}; \boldsymbol{\beta}, \boldsymbol{\mu}) = \|\mathbf{Y} - \boldsymbol{\Theta}(\mathbf{X}, \boldsymbol{\Omega}(\hat{\mathbf{X}}; \boldsymbol{\mu}); \boldsymbol{\beta})\|_2^2 \tag{4.11}$$

In the training process, we employ the stochastic gradient descent algorithm AdamWR [38] with warm restarts as in [63], which already includes regularization. Therefore, AdamWR needs both the learning rate η and the weight decay rate λ. The learning rate was set to $\eta = 0.001$ after evaluation (see Section 6.2) and the weight decay rate was set as $\lambda = 10 * \lfloor length(\mathbf{X})/batchsize \rfloor$, where $length(\mathbf{X})$ represents the number of frames being considered for network training.

We consider a batch size of 32, with the training samples in each batch being selected in random order. Dropout [54] was applied with the retention probability as 0.2. Depending on the network parameters being considered, training takes around 30mins to 120mins on an NVIDIA Quadro RTX 8000 GPU consisting of 48 GB of video memory.

As observed in [63], a lower training loss or test loss does not necessarily imply that the model has learnt to produce a better quality of motion. Thus, instead of using a test set for verifying network performance, we employ visual inspection. Additionally, we also verify the training loss to confirm whether a better loss leads to better boxing motion (see Section 6.2 for more details).

We also test several input and output configurations for both motion prediction and gating networks. More details about these tests are provided in Chapter 6. However, in Section 4.2, we will present the exact variables used in the motion state representations that are used to train our MoE model.

4.2 Motion State Representation

Motion capture data is mapped to a virtual skeleton containing joints as shown in Figure 2.1a and it has an associated joint hierarchy as shown in Figure 2.1b. A common way of defining a pose for motion capture data is by using the rotation of the joints in their respective local coordinate space. Due to the joint hierarchy, a rotation applied to the parent joint will also affect its children, but vice versa does not apply. In the PFNN, Holden et al. [21] chose to work with joint positions instead of rotations. This change is because of the possibility of a neural network model to introduce errors in the produced output. Such an error might accumulate and lead the system to produce incorrect results. Additionally, the error in a joint such as the spine, which is higher up in the hierarchy, will introduce error in all its children, but an error in a joint such as the hand will affect only itself. Thus, to avoid such complexity, Holden et al. [21] used only the positional joint information in their motion state representation. However, Zhang et al. [63] found that the MANN performance was better when including the rotation along with the positional information of the joints. While we considered both these approaches, we concluded that it would be better to ensure that our model first worked well with positional information as that would streamline the focus of this master thesis.

The approach from Holden et al. [21] of using this type of motion state representation was motivated from motion matching [10], which in turn is based on the representation from the MoFi method [35]. Taking inspiration from the motion states in motion matching [10], which is prevalent in the interactive neural motion synthesis literature, we also utilise a similar representation. This representation is used to form both the network input $\mathbf{x}$ and the network output $\mathbf{y}$. To begin with, each motion state corresponds to one frame in the motion capture data. Each state contains information about not only the current frame being considered but also information about a fixed number of past and future frames. In our case, the information about the current frame consists of the punch action labels from Section 3.2, punch target positions from Section 3.3 and the position and velocity of all the joints in the local coordinate space of the skeleton.

The information about the past and future frames is referred to as the trajectory information. For one frame, the trajectory information consists of the ground projected position, velocity and directions of the hip joint and the position and velocity of the wrist joints. We evaluate different trajectory windows and the results of this evaluation are presented in Chapter 6.

All the variables corresponding to a frame of the data undergo a transformation called the root transformation (see Equations (4.12) and (4.13)), which converts all the variables to the local coordinate space of the skeleton and rotates it towards the forward-facing direction of the character. This transformation is utilised in [21, 63] to ensure that the motion manifold needed to be learned by the neural network is simpler.

The root transformation for obtaining the local joint position is represented by the following equation:

$$\mathrm{lp_i} = \mathrm{r_f} * (\mathrm{gp_i} - \mathrm{gp_0}|_{\mathrm{y}=0}) \tag{4.12}$$

In Equation (4.12), $\mathrm{lp_i} \in \mathbb{R}^3$ is the local joint position of a joint $\mathrm{i} \in \mathbb{J} = \{0, 1, \ldots, j\}$. In our case, $j = 21$ is the number of joints in our virtual skeleton shown in Figure 2.1a. $\mathrm{gp_i} \in \mathbb{R}^3$ is the global position (w.r.t. origin of the virtual world) of a joint i. As in Figure 2.1b, the hip joint is the parent of all the joints. Hence, the root transformation is computed w.r.t. the hip and $\mathrm{gp_0}$ represents the global position of the hip joint. However, the hip joint is

first projected to the ground by setting the y coordinate to 0. Lastly, the rotation of the hip joint at frame f is denoted by $\mathrm{r_f}$.

Similarly, the root transformation to obtain the local velocity and the local direction is presented by the following equation:

$$\mathrm{l_i} = \mathrm{r_f} * \mathrm{g_i} \tag{4.13}$$

In Equation (4.13), $\mathrm{l} \in \mathbb{R}^3$ represents the root transformation of the joint velocity or the joint direction and $\mathrm{g} \in \mathbb{R}^3$ represents the corresponding global velocity or direction. Similar to Equation (4.12), $\mathrm{i} \in \mathbb{J}$ represents a joint of the skeleton and $\mathrm{r_f}$ represents the rotation of the hip joint at frame f.

The root transformations defined above are utilised in the motion state representations (see Equations (4.14), (4.15) and (4.16)) used to train the motion prediction network and the gating network. The gating network input is a subset of the input used for the motion prediction network. Thus, the input and output of the motion prediction network essentially form the motion states for training our MoE model. Additionally, note that the gating network is used to predict the blending weights ω as explained in Section 4.1.2. Therefore, the motion state representations are utilised only for the inputs of the motion prediction and gating networks and also for the output of the motion prediction network. However, to provide a clear understanding of the motion states used in the MoE model, we present the exact inputs for the motion prediction and gating networks in Sections 4.2.1 and 4.2.2, respectively. Finally, the details of the motion prediction network output are presented in Section 4.2.3.

4.2.1 Motion Prediction Network Input

In Section 4.1.1, we saw that the motion prediction network is designed to do the mapping from the current motion state to the next one. To achieve this mapping, the current motion state fed into the network must carry sufficient information to describe not only the motion at the current frame but also the motion in a window of surrounding frames. The motion information in the window covers the past and future motions in relation to the current frame and is referred to as the trajectory information. A representation such as the one we consider was also presented by Holden et al. [21] for locomotion tasks using neural-based systems. However, in this thesis, we extend this representation to our boxing data to achieve neural-based interactive control of the boxing actions of punching and stepping. To achieve this, we introduce additional variables to handle punching action while the variables in the motion state required for stepping remain nearly the same as [21].

Therefore, the input to the motion prediction network $\mathbf{x}$ is designed to consider motion information representative of both the arm movements and the global translation of the character relative to the hip, which is referred to as the root joint. All the position, velocity and direction motion information must be transformed to the local space of the skeleton using the root transformation explained in Equations (4.12) and (4.13). Also, the trajectory window length in all the trajectory variables presented in the motion prediction network input is represented by w. Therefore, the resulting input vector $\mathbf{x}$, at frame $\mathbf{f}$, used by the motion prediction network is given by:

$$\mathbf{x_f} = \{\mathbf{t_f^{rp}}, \mathbf{t_f^{rv}}, \mathbf{t_f^{rd}}, \mathbf{t_f^{wp}}, \mathbf{t_f^{wv}}, \mathbf{a_{f-1}^{l}}, \mathbf{a_{f-1}^{pt}}, \mathbf{j_{f-1}^{p}}, \mathbf{j_{f-1}^{v}}\} \in \mathbb{R}^n \tag{4.14}$$

In the above Equation (4.14), we have the following information:

- **Root trajectory:** This information represents the motion related to the root of the character and it does not consider the y component in 3D space, meaning that it is two dimensional. This implies that the dimensions of the variables in this category is $w \times 2$. The different root trajectory variables are:
  - Position: $\mathbf{t_f^{rp}} \in \mathbb{R}^{w \times 2}$
  - Velocity: $\mathbf{t_f^{rv}} \in \mathbb{R}^{w \times 2}$
  - Direction: $\mathbf{t_f^{rd}} \in \mathbb{R}^{w \times 2}$
- **Wrist trajectory:** This information represents the motion related to the wrist of the character, and it also considers all the components in 3D space. Additionally, all the wrist trajectory variables below are collected for both the left and the right wrist joints. Thus, the dimensions of each trajectory variable collected in 3D space will be $2 \times w \times 3$. The various variables falling in this category are:
  - Position: $\mathbf{t_f^{wp}} \in \mathbb{R}^{2 \times w \times 3}$
  - Velocity: $\mathbf{t_f^{wv}} \in \mathbb{R}^{2 \times w \times 3}$
- **Punch information:** This consists of the labels for the punches from Section 3.2 and the target wrist positions at the end of the punch explained in Section 3.3. It is also collected for both the left and the right wrist joints. Thus, the punch information variables are:
  - Punch label: $\mathbf{a_{f-1}^{l}} \in \mathbb{L}^2$ where $\mathbb{L} = \{0, 1, -1\}$. This is obtained from the annotation process (see Section 3.2).
  - Punch target: $\mathbf{a_{f-1}^{pt}} \in \mathbb{R}^{2 \times 3}$. This is extracted from the wrist position at the full extension of the hand (as explained in Section 3.3).
- **Joint information:** This information is collected for all the $j = 21$ joints in the virtual skeleton and it consists of the following variables:
  - Positions of all joints: $\mathbf{j_{f-1}^{p}} \in \mathbb{R}^{j \times 3}$
  - Velocities of all joints: $\mathbf{j_{f-1}^{v}} \in \mathbb{R}^{j \times 3}$

Note that all of the above multidimensional variables must be flattened by placing all the elements in a single row resulting in single-dimensional variables. This gives the dimension n shown in Equation (4.14) which is also the number of neurons in the motion prediction network input as seen in the caption of Figure 4.1.

In the next section, we present the gating network input, which is a subset of the motion prediction network input.

4.2.2 Gating Network Input

As we saw in Section 4.1.2, the input vector for the gating network $\hat{\mathbf{x}}$ is a subset of the motion state $\mathbf{x}$, which is passed as the input to the motion prediction network. Therefore, the variables in the gating input are the same as the ones in $\mathbf{x}$ in Equation (4.14) or they are derived from the variables in $\mathbf{x}$. This gating input vector $\hat{\mathbf{x}}$ at frame $\mathbf{f}$ is given by:

$$\hat{\mathbf{x}}_\mathbf{f} = \{\mathbf{t}^{\mathbf{rp}}_\mathbf{f}, \mathbf{t}^{\mathbf{rd}}_\mathbf{f}, \mathbf{t}^{\mathbf{wv}}_\mathbf{f}, \mathbf{h}^{\mathbf{efv}}_{\mathbf{f-1}}, \mathbf{a}^{\mathbf{l}}_{\mathbf{f-1}}, \mathbf{b}^{\mathbf{efv}}_{\mathbf{f-1}}, \mathbf{b}^{\mathbf{efp}}_{\mathbf{f-1}}\} \in \mathbb{R}^{n'} \tag{4.15}$$

In the above Equation (4.15), we have the following information:

- **Root trajectory:**
  - Position: $\mathbf{t}^{\mathbf{rp}}_\mathbf{f} \in \mathbb{R}^{w\times 2}$
  - Direction: $\mathbf{t}^{\mathbf{rd}}_\mathbf{f} \in \mathbb{R}^{w\times 2}$
- **Wrist trajectory:**
  - Velocity: $\mathbf{t}^{\mathbf{wv}}_\mathbf{f} \in \mathbb{R}^{2\times w\times 3}$
- **Punch information:**
  - Wrist end effector velocities: $\mathbf{h}^{\mathbf{efv}}_{\mathbf{f-1}} \in \mathbb{R}^{\mathbf{2\times 3}}$ which represents the wrist velocities for both the left and the right hands. This results in a dimensionality of 2×3 for for velocity in 3D space.
  - Punch label: $\mathbf{a}^{\mathbf{l}}_{\mathbf{f-1}} \in \mathbb{L}^2$
- **Foot joints information:** This consists of the information for the foot end effector joints of the ankle and toe, for both the right and the left feet. This results in a dimensionality of $2 \times 2 \times 3$ for 3D points.
  - Velocity: $\mathbf{b}^{\mathbf{efv}}_{\mathbf{f-1}} \in \mathbb{R}^{2\times 2\times 3}$
  - Position: $\mathbf{b}^{\mathbf{efp}}_{\mathbf{f-1}} \in \mathbb{R}^{2\times 2\times 3}$

Note that as $\hat{\mathbf{x}} \subset \mathbf{x}$, most of the explanations from Equation (4.14) also applies to Equation (4.15). Firstly, the root transforms and flattening operation need to be applied to the above variables as well. Additionally, the trajectory window is represented by w. Lastly, unless explicitly specified, the above variables are the exact same ones from Equation (4.14) and the same explanation is applicable to them.

Along with the motion state inputs, the MoE model requires motion state outputs at the motion prediction network output layer. Such a motion state is presented in the next section.

4.2.3 Motion Prediction Network Output

As explained in Section 4.1.1, the motion prediction network performs a mapping from the current motion state to the next motion state and, therefore, enables our MoE model to synthesise the boxing motion in the next frame. In this section, we present information about the next motion state, which is the output vector expected as a prediction from the motion prediction network.

This output vector also contains trajectory information, but the key difference compared to Equation (4.14) is that the trajectory information here contains only the motion data corresponding to the future frames, and we indicate this by the trajectory window size of $\frac{w}{2}$. This output vector $\mathbf{y}$ at frame $\mathbf{f}$ is given by the following equation:

$$\mathbf{y}_\mathbf{f} = \{\mathbf{t}^{\mathbf{rp}}_{\mathbf{f+1}}, \mathbf{t}^{\mathbf{rv}}_{\mathbf{f+1}}, \mathbf{t}^{\mathbf{rd}}_{\mathbf{f+1}}, \mathbf{t}^{\mathbf{wp}}_{\mathbf{f+1}}, \mathbf{t}^{\mathbf{wv}}_{\mathbf{f+1}}, \mathbf{a}^{\mathbf{l}}_\mathbf{f}, \mathbf{a}^{\mathbf{pt}}_\mathbf{f}, \mathbf{q}^{\mathbf{p}}_\mathbf{f}, \mathbf{q}^{\mathbf{d}}_\mathbf{f}, \mathbf{b}^{\mathbf{c}}_{\mathbf{f-1}}, \mathbf{j}^{\mathbf{p}}_\mathbf{f}, \mathbf{j}^{\mathbf{v}}_\mathbf{f}\} \in \mathbb{R}^m \tag{4.16}$$

In the above Equation (4.16), we have the following information:

- **Root trajectory:**
  - Position: $\mathbf{t^{rp}_{f+1}} \in \mathbb{R}^{\frac{w}{2} \times 2}$
  - Velocity: $\mathbf{t^{rv}_{f+1}} \in \mathbb{R}^{\frac{w}{2} \times 2}$
  - Direction: $\mathbf{t^{rd}_{f+1}} \in \mathbb{R}^{\frac{w}{2} \times 2}$
- **Wrist trajectory:**
  - Position: $\mathbf{t^{wp}_{f+1}} \in \mathbb{R}^{2 \times \frac{w}{2} \times 3}$
  - Velocity: $\mathbf{t^{wv}_{f+1}} \in \mathbb{R}^{2 \times \frac{w}{2} \times 3}$
- **Punch information:**
  - Punch label: $\mathbf{a^{l}_{f}} \in \mathbb{L}^2$
  - Punch target position: $\mathbf{a^{pt}_{f}} \in \mathbb{R}^{2 \times 3}$
- **Root joint information:** The following information regarding the root joint consists of only the X and Z coordinates in 3D space.
  - $\mathbf{q^{p}_{f}} \in \mathbb{R}^2$ root position
  - $\mathbf{q^{d}_{f}} \in \mathbb{R}^2$ root direction
- **Foot joints information:**
  - Foot contact labels: $\mathbf{b^{c}_{f-1}} \in \mathbb{F}^4$ where $\mathbb{F} = \{0, 1\}$. These labels are generated for the ankle and toe joints on both the left and right feet which leads to 4 dimensions.
- **Joints information:**
  - Positions of all joints: $\mathbf{j^{p}_{f}} \in \mathbb{R}^{j \times 3}$
  - Velocities of all joints: $\mathbf{j^{v}_{f}} \in \mathbb{R}^{j \times 3}$

Note that, similar to $\mathbf{x}$, the root transforms and flattening operation explained for Equation (4.14) need to be applied to the above variables in $\mathbf{y}$ as well. Lastly, unless explicitly specified, the above variables are the exact same ones from Equation (4.14), and the same explanation is applicable to them. However, note that the variables in these equations are at different frames i.e. value of $\mathbf{f}$ might be different in the variables in Equations (4.14) and (4.16) to facilitate the visualisation to move from frame $\mathbf{f}$ to frame $\mathbf{f + 1}$ using our MoE model.

These motion state representations presented in Equations (4.14), (4.15) and (4.16) are inspired from the representations used by Holden et al. [21] and Zhang et al. [63]. As explained before, the PFNN developed by Holden et al. [21] does not use joint rotations to avoid the accumulation of error. However, the MANN model developed by Zhang et al. [63] makes use of joint rotations but does not explicitly consider phase labels. Our motion representation also does not explicitly consider the phase variable or joint rotations. So it is a combination of the two and is updated for controlling the punch action by including information related to the wrist joint movement. However, the dimensions of the trajectory information, even for the root joint, is different in our motion states compared to [21, 63]. This difference is because we perform several experiments

to determine the best duration of motion data needed in the trajectory for synthesising boxing motions. These experiments will be presented in Chapter 6.

Our MoE boxing model presented in Section 4.1 can perform the mapping from the current boxing state to the next one using the motion state representations in Section 4.2. To train multiple such models containing different amounts and sparsity of trajectory information (see Chapter 6 for more information), we developed separate units for data preparation and model training. While these units help train models that can perform the required mapping, the output from a model cannot be visualised directly and similarly, the input to the model cannot be obtained directly from the visualisation. Feedback is required from the output of the model to provide visualisation and interactive control. Additionally, user input must also be considered. Therefore, to achieve interactive control, several additional units were incorporated in a boxing controller system presented in the next chapter. Additionally, we also provide details about all the individual units that help in training and testing our models.

Chapter 5
Boxing Controller System

The MoE model we presented in the previous chapter performs a basic regression task from the current motion state to the next one. However, it cannot automatically ensure the feedback required to prepare the next input to the model. Moreover, it is also not capable of allowing a user to control the character in a virtual environment interactively. To achieve such feedback and allow user control, our model needs another system called the boxing controller. This section will present details about our boxing controller system, which allows for interactive control of boxing in a virtual environment.

Additionally, as we wanted to verify the performance of our model regarding several motion state and network parameters, we separated our software into several different units to achieve a smooth development and evaluation process. In the following Section 5.1, we will first see an overview of all the systems in our project and how they are beneficial. Subsequently, we will present the boxing controller system, which is comprised of the handler unit for handling the trained MoE model, a visualisation unit that also allows user inputs and a communication unit that ensures that the previous two units can share information.

5.1 Systems Overview

For small ML projects, the data preparation, model training and evaluation processes are all implemented in a single script, which is similar to the waterfall model in software design. However, with the motivation of evaluating the performance of our models in a better manner, we decided to separate the different processes involved in our implementation, leading to several interdependent units. A graphical interpretation of this setup is shown in Figure 5.1. As shown in the figure, we partition our implementation into the following units: motion state generator, neural model trainer, evaluator, model handler, server and a visualisation unit. The first two units prepare the motion states and train our MoE models, respectively. They are relatively more independent in terms of development as there are not many calls from one unit to another during execution. For example, the data preparation unit can be updated and tuned when the mocap data is available and similarly, the model trainer unit can be developed independently when

one set of motion states is available. This independence in development is beneficial as we prepare many sets of motion states and train several models that are evaluated by the evaluator unit (see Chapter 6 for evaluation results). The last three units are interdependent units as they must be executed together, and they collectively form the interactive boxing controller system. This system utilises the trained MoE model to synthesise motions that are sent to the visualisation unit through the server. The visualisation unit allows a user to view the synthesis results and input commands which are sent back to the controller through the server.

In the following paragraphs, we will present more details about the functions occurring within each unit and how they are set up to interact with other units of our implementation.

Motion state generator The neural models we develop cannot work on the motion data obtained from the mocap session directly. Once the boxing data is exported from the Xsens software, the motion state generator unit is used to prepare the motion states explained in Section 4.2. This unit takes the mocap data in the BVH format (explained in [41]) as input. Our unit can parse the motion data and generate the motion states while allowing the following two inputs from the experimenter: 1) trajectory window size and 2) frameskip which is the sparsity inside the trajectory window. For example, the user can specify that the trajectory windows for all the variables must contain 20 consecutive frames (i.e. no frameskip), which would correspond to a trajectory window that covers 0.167s of motion in the past, the current motion and 0.15s of motion in the future of our boxing data. The other available parameter is to allow for sparsity inside the trajectory window. For example, setting the sparsity parameter called frameskip to two and still maintaining 20 frames in the window would result in the trajectory window covering 0.33s of past motion, the current motion and 0.32s of future motion. Several such settings for both the trajectory window and the frameskip are tested in Chapter 6.

This unit has been developed using the virtual skeleton utilities presented by Herrmann et al. [17]. While their implementation allowed us to extract many of the variables required to generate the motion states, we needed to adapt their implementation for the case of boxing, also update the code to consider the punch labels from the annotation process described in Section 3.2 and provide the ability to input the experimentation parameters described above easily. This motion state generator unit is implemented in Python 3.7.

Neural model trainer The neural model trainer is the unit focused on training the MoE neural model that was explained in Section 4.1. This unit has an implementation of the MoE model, and it can take in the processed data from the motion state generator and train our model. Our MoE model is developed using Tensorflow 2.0 [1] and the exact parameters we used during the training process is presented in Section 4.1.3. This unit is designed to automatically set up the network and the training process based on any version of the motion states coming from the generator. The fundamental need of this unit is to separate the data preparation and model training as we need to consider several versions of the data and train several corresponding models that are evaluated in Chapter 6. This model training unit is also developed using Python 3.7.

Model handler The model handling unit is where the trained MoE model is placed. This unit 1) prepares the motion state to be passed as input to the model and 2) stores

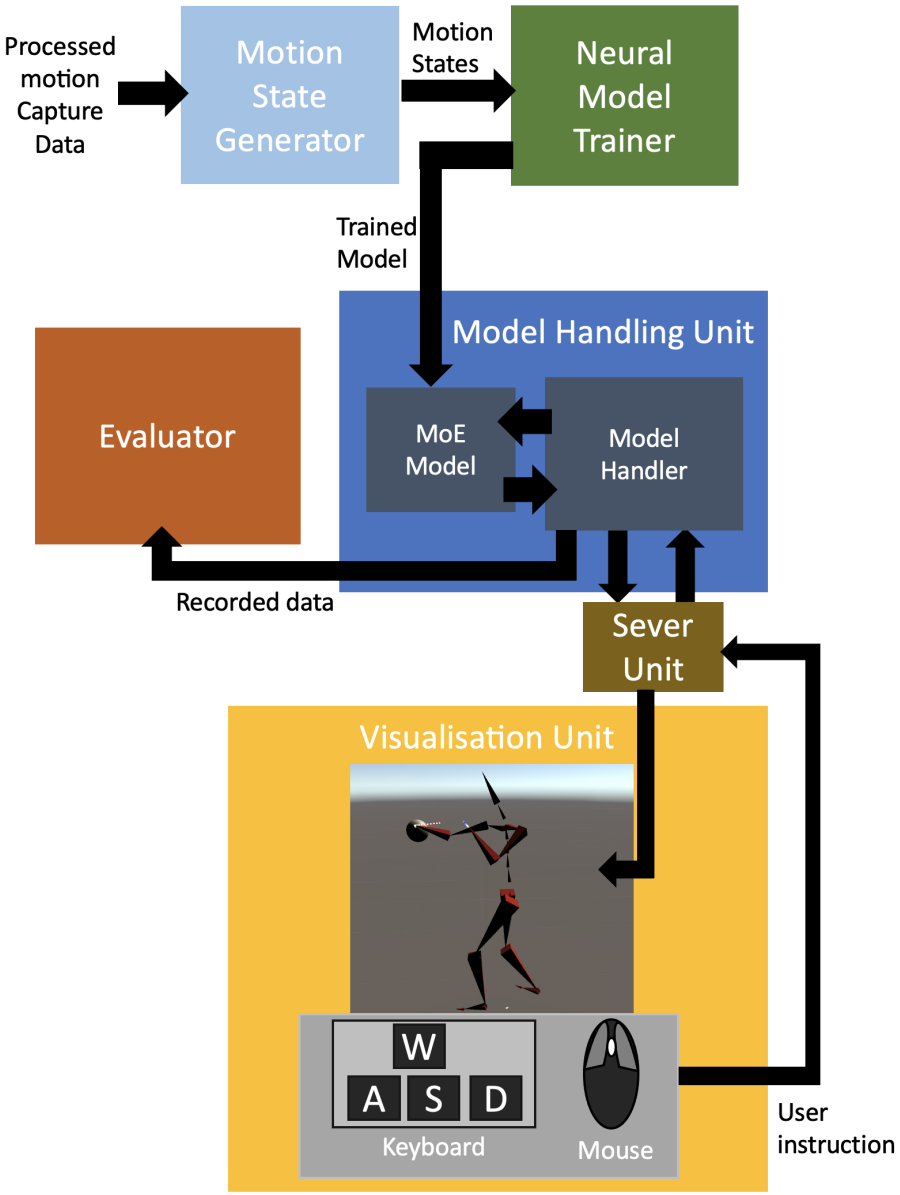


Figure 5.1: This figure shows an overview of all the developed units and how they are interacting with each other in our implementation. The visualisation unit also shows the interaction interfaces for providing user input.

within it the future motion information in the output motion state of the model. This unit needs a working communication setup with the visualisation unit that is enabled through the server unit. This model handling unit then communicates with the visualisation unit to obtain user input such as movement direction and punch target. These inputs are then merged with the information stored in the model handling unit to prepare the final version of the model input, enabling the synthesised motions to react to user input. The corresponding output is then stored in this model handling unit and used for the next

cycle. Lastly, this model handling unit is described as independent of gaming engines. This independence is because it can be hardcoded with virtual user inputs and used to synthesise motions without requiring any assistance from the visualisation unit, which is developed using a gaming engine.

Our handler was implemented using Python 3.7, and it was developed to cover many functionalities related to trajectory manipulation by considering the user input. We will present the different functionalities covered by the handler as a part of the boxing controller system in Section 5.2.

Visualisation unit Our visualisation unit is also another part of the boxing controller system. It allows for a user to see the synthesised motions and control them. The positional information of the joints coming from the handler cannot be visualised directly. Therefore, the visualisation unit uses the positional information to compute the rotations of the joints corresponding to the joint hierarchy and applies them to a virtual skeleton to visualise the results. The visualisation unit also allows the user to input movement directions or punch commands using a keyboard. Our visualisation unit is implemented in Unity Engine [59] using the C# programming language, and it is presented in more detail in Section 5.2.2.

Server The server is the unit that establishes communication between the model handler and the visualisation unit, and it is also a part of the boxing controller system. Many different endpoints are established with this server to pass information from the handler, such as the positional information of the joints, end of the punch action and test target positions used for evaluation (see Chapter 6). Similarly, it allows the visualisation unit to submit the user instruction corresponding to the punch or stepping actions. Additionally, the server also ensures that the information maintained in the handler and the user inputs are recorded at every frame of the visualisation, as this information will be used for evaluating our models. Lastly, the server unit is also responsible for resetting the pose of the character during the punch evaluation experiment. Our server unit is developed using the Flask package [47].

Evaluator The evaluator unit accesses the data stored by the server during an experiment and processes it to generate the results and the plots presented in Chapter 6. The metrics that are computed by this unit are explained in more detail in Section 6.1.

As we saw from the descriptions of the model handling, visualisation and server units, there is interdependence between them during the interactive synthesis process. Therefore, these three units together form the boxing controller system, which is explained in more detail in the following section.

5.2 Interactive Boxing Controller System

As explained earlier, the MoE models developed in this thesis are performing a regression task, and these models cannot work on the motion data directly. Therefore, a separate unit called the model handler must perform interpolation operations on the trajectory and prepare the data to be displayed by the visualisation unit. Additionally, the handler obtains the user input from the visualisation unit and blends it with the previous model prediction to prepare the current model input. Finally, the server unit enables the

communication between the model handler and the visualisation unit. These three units together form the interactive boxing controller system shown in Figure 5.2. In this section, we briefly discuss the importance of a controller system and the relevant related work. This discussion is followed by more details about the model handling unit in Section 5.2.1 and finally about the visualisation unit in Section 5.2.2. Note that we already presented an overview of the server unit in Section 5.1 and will not present it in further detail in this section, as it is a unit that is designed to facilitate communication only.

For the specific case of motion synthesis, the performance of the controller system is of equal importance as that of the neural model. Once the user is allowed to interact with the synthesis results through the controller system, the visualisation unit will call the model handler at every frame. Thus, an error in the trajectory information stored in the handler will lead to an accumulation of errors over time, and the interactive motion synthesis might become unstable or suffer in quality. As a result, the motion synthesis is poor even though the MoE model is capable of performing well. Hence, proper implementation of the boxing controller is required to analyse the performance of our interactive boxing motion synthesis.

The boxing controller implemented for this master thesis is similar to the controller proposed by Clavet [10] because the user input is integrated into the predictions of the motion synthesis model. Using a similar approach, Sprenger et al. [53] presented an implementation of a locomotion controller with a clear separation of the different functionalities occurring within a neural-based controller system. Therefore, our boxing controller system is inspired by the approach from Clavet [10] and implemented using the approach from Sprenger et al. [53]. While the controller from [53] was designed with a PFNN model for the specific case of controlling locomotion styles and actions, our controller is designed with an MoE model to handle the boxing actions of punching and stepping.

5.2.1 Model Handling Unit

The controller system must handle the user input coming from the visualisation unit and also utilise the predictions of the model to prepare its next input. In our controller, there are two types of user inputs: 1) direction input for stepping action, and 2) a specific punch target and the hand to be used for the punch action. These user inputs are merged with the trajectory predictions stored in the model handling unit.

This storage of the trajectory information is not just a trivial record of the model predictions as the handler is responsible for maintaining *all* the frames in the trajectory window, even when there is a frameskip used in the motion state generator as explained in Section 5.1. For example, when the trajectory window is of size 10 and the frameskip is two, the actual number of motion frames covered by the trajectory is 20, and the model handler must maintain all the 20 trajectory points in its storage.

The MoE model performs the mapping from the current motion state to the next one by considering only the keyframes (i.e. every alternate frame when frameskip is two), in the trajectory information found in the handler. However, to achieve motion synthesis at every frame rendered by the visualisation unit, the handler must **interpolate between the keyframes** and maintain all the frames in the trajectory window. While such usage of different frameskips in the motion states makes our controller system more complex, it is required for handling all the different versions of our MoE model presented in Chapter 6. Therefore, the handler is not only designed to manage different sizes of the trajectory

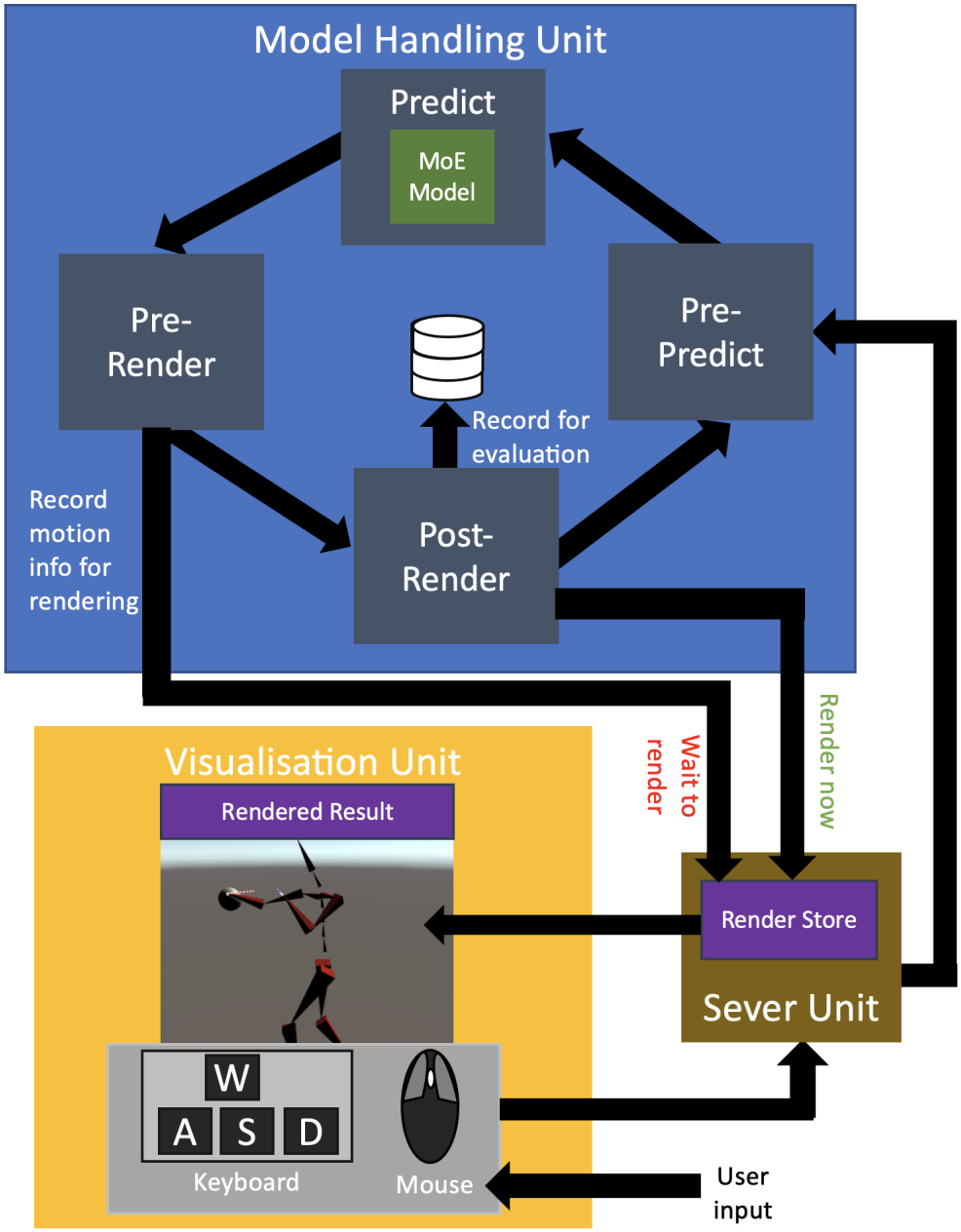


Figure 5.2: Shows the interactive boxing controller system with a simplified view of the different functions being applied inside its units of visualisation, server and model handler. This figure indicates that the user input is allowed at every frame, and the results are rendered after completing all the functions in the model handling unit.

window w but also designed to deal with different values of frameskip fs inside the trajectory window by applying interpolation operations similar to [53].

In addition to the post-prediction interpolation operation, the handler is also designed to **manage the user input by blending** with the stored information, before requesting the next prediction from the model. Essentially, the user input is blended with the predictions for the future trajectory positions of the root and wrist joints from the previous frame that are stored in the handler. In the blending operation utilised for interactive control of stepping, a virtual target velocity derived from the user input direction is blended with the future root trajectory and this blending operation is given by the following equation:

$$trp_i = trp_{i-1} + \left(1 - \left(\frac{(i - \frac{w}{2})}{\frac{w}{2}}\right)^{\tau}\right) \times (trp_i - trp_{i-1}) + \left(\frac{(i - \frac{w}{2})}{\frac{w}{2}}\right)^{\tau} \times v_{target} \quad (5.1)$$

In Equation (5.1), $\frac{w}{2} \leq i \leq w$ with w as the trajectory window size. Next, trp is the trajectory root position, τ is the blending bias and v_{target} is the virtual target velocity derived from the user input target direction as in [53].

Similarly, a blending operation is required to achieve interactive control of the punch action. To achieve this, the punch target position specified by the user is blended with the future wrist trajectory and the resulting operation is defined by the following equation:

$$thp_i = thp_{i-1} + ((pt - hp_i) \times st) \times i \quad (5.2)$$

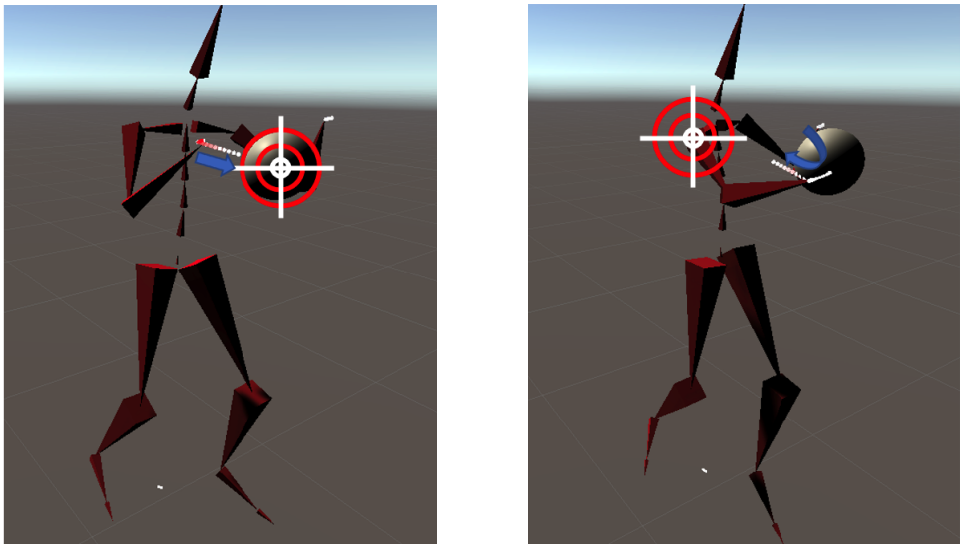
In Equation (5.2), $\frac{w}{2} \leq i \leq w$ with w as the trajectory window size. Next, thp is the trajectory wrist position, pt is the user-specified punch target, hp is the current wrist position, and st is the step size between trajectory points per frame. It was manually set to $st = 0.06m$ as it was appropriate for our punching actions. However, st can also be set up through the corresponding average value from the dataset, but this was not possible in our case due to the time constraints in this master thesis.

Other than the interpolation and blending operations, the model handler also needs to manage the **root transformation** explained in Equations (4.12) and (4.13). This transformation handling is required as the visualisation requires information in the global space of the virtual environment, and the network requires information in the local space of the virtual skeleton. Therefore, all the information stored in our handler is in the global space and can be visualised directly, but for obtaining the next motion state, the trajectory information is converted to the local space and the input motion state is prepared for the model.

The operations of interpolation, blending and root transformation that are applied in the boxing controller system are similar to the ones presented in [53]. However, all these operations are updated to manage variable trajectory window size w and frameskip fs in our system. For the specific case of interactive punch control, we added the additional blending operation given by Equation (5.2) to our boxing controller system. However, this operation needs to be managed further within the handler unit to synthesise the forward and backward motion of the arm in a punch action.

The model handler unit synthesises the forward movement of the arm during a punch by using the user-specified punch target as input. Subsequently, after the hand has reached the target, it must move backwards and send the character back into the rest pose. In such a condition, if the target is left to be at the user-specified location, then Equation (5.2) will be continually applied, and the synthesised result will forever remain in a position with an outstretched arm. Therefore, to avoid this problem and return the

character to the neutral pose, the handler uses the shoulder position as a virtual target to reach the rest pose and complete the punch. This **punch target management solution** implemented in the model handler is visualized in Figure 5.3. This method achieves realistic-looking results and retains responsivity. The handler also enforces conditions using the $\mathbf{l^2}$ norm to terminate the forward and backward movements when the wrist position is close enough to the target being considered.



(a) (b)

Figure 5.3: These images are from our visualisation unit and they are updated to depict the punch target management solution implemented in the model handler. As seen in Figure 5.3a, the handler utilises the user specified target that is marked by the sphere as the punch target in the forward motion. During the backward motion, the handler has reset the punch target to map to the shoulder joint of the character in Figure 5.3b. In each case, the current target is highlighted by the crosshair.

As we are designing an interactive boxing controller system, it also becomes essential to ensure that the controller is designed for **handling missed punches**. These punches refer to the instances in which the target is out of the range of our model, and the resulting punch is incapable of reaching it. In such a scenario, based on the statistics computed for our boxing data in Table 3.3, the model handler is set up to always retreat the outstretched hand after 25 frames of motion synthesis, even if the wrist position is not close to the user-specified target. Similarly, the retreat of the punch is also always terminated after 50 frames, regardless of the closeness to the shoulder. This termination process is visualised in Figure 5.4, which indicates that the hand is being retreated after 25 frames of motion synthesis.

As we have seen above, the model handler unit performs many functions that are key to ensuring that the generated motion is interactive and realistic. Therefore, we invested considerable effort in designing and implementing a sound model handler using the approach by Sprenger et al. [53], and our implementation approach is presented below.

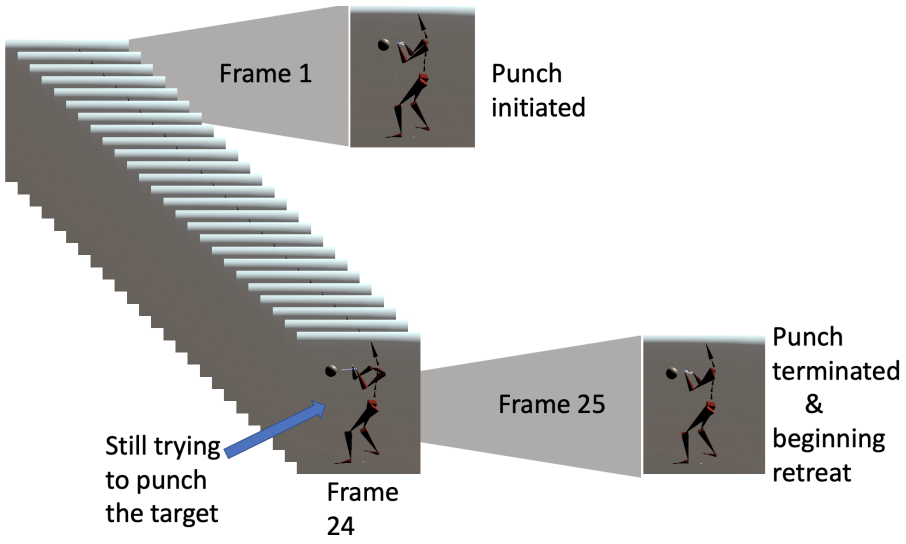


Figure 5.4: This is an image from the visualisation unit that is adapted to show the punch termination process. In this particular instance, the hand has started retreating after 25 frames of synthesis despite not having reached the target. Similar logic is applied to end the punch retreat after 50 frames of synthesis.

Implementation

The DL community actively utilises the Python programming language for research and development. Therefore, the support for model research is the best in this language, and most neural-based projects use it. As our motivation is to develop good neural models for boxing motion synthesis, we also develop our model and the associated model handling unit entirely in Python 3.7, which enables easier integration with the neural model. However, we implement our visualisation unit using the Unity engine [59] and this unit, and its implementation is discussed in more detail in Section 5.2.2. However, to achieve interactive control, both the model handling units and the visualisation unit need to perform several functions in tandem. Therefore, we use a server unit developed using the Flask framework [47] to enable communication between these units.

We established in the previous section that the model handling unit performs several necessary functions that are integral to synthesising high-quality motions. Additionally, we use this unit to test several MoE models in Chapter 6. Therefore, it is important to have a proper implementation of the model handling unit. To achieve this, we use the design from the locomotion controller presented by Sprenger et al. [53] as the foundation for the model handling unit developed in this thesis.

In this approach, the operations in the model handler are grouped into several functions called by the visualisation unit at each frame to render the movement of the character as shown in Figure 5.2. These different functions in the model handler are explained below:

1. **Pre-predict**: Upon obtaining the user input from the visualisation unit, the handler combines it with the trajectory information maintained within itself and prepares the motion state input for the motion prediction network explained in Section 4.2.1. As mentioned earlier, the trajectory information stored in the handler is in the

global space of the game engine, and this information needs to be mapped to the local space of the character to prepare the input motion state. Therefore, the pre-predict function is also responsible for applying the root transformations through Equations (4.12) and (4.13).

Additionally, the blending operations to update the trajectories based on the user inputs explained in Equations (5.1) and (5.2) are also applied here. Therefore, this function is utilised before the forward pass of the model to update the input vector with the command from the user. Finally, if there is a frameskip of higher than one, then this function must also ensure that the motion state will consist of only the corresponding keyframes from the trajectory maintained in the handler.

2. **Predict**: This step is where the handler passes the prepared motion state input through the MoE model to obtain the motion state output described in Section 4.2.3. The output motion state will be in the local space of the virtual character.

3. **Pre-render**: This function uses the predictions to update the joint position and velocity information in the handler. It also applies the inverse of the root transformations (Equations (4.12) and (4.13)) so that the predicted information in the local space of the skeleton is converted to the global space of the virtual environment. Finally, the transformed information is stored in the handler.

4. **Render**: This step corresponds to preparing the global space information stored in the handler and transforming it to a suitable version that can be used by the visualisation unit. Once this information is sent to the visualisation unit through the server, the synthesised motion is displayed, and the handler is contacted again with the next user input as explained in Section 5.2.2. Therefore, the render function in the handler corresponds only to the action of preparing the data to be sent to the visualisation unit. The data prepared for the visualisation unit is then stored in a temporary variable called the render store (in the server unit). The prepared data is then passed to the visualisation unit upon completion of all the five functions in the model handling unit as shown in Figure 5.2. However, since this step prepares the exact data that the game engine will render, it is referred to as the render function in the model handling unit.

5. **Post-render**: After the render step, the model handling unit will update the information stored within it to ensure that the trajectories will correspond to the frame that the visualisation unit will display. This is achieved by first parsing and gathering the different variables present in the output motion state and then updating the past trajectory points by stepping through the stored information. For updating the current point in the stored trajectory, it uses the corresponding root transformed prediction. Finally, the future trajectory points are updated using the root transformed versions of the corresponding variables in the prediction. Thus, all the root and wrist trajectory variables maintained inside the handler are updated in this function to match the information corresponding to the frame to be rendered by the visualisation unit (using the stored result from the above render function). When a frameskip is involved in the motion state, this function will also apply the interpolation operation described in Section 5.2.

   The interpolation operation and the other tasks occurring in this post-render function are explained better with an example. Therefore, when we come back to the case of having a frameskip of 2 in a trajectory window of size 10, the handler must maintain all the 20 trajectory points within it. The handler is at this state in the post-render function. At this junction, the handler will first step through

the current version of the 20 trajectory points with a step size of one (since the visualisation unit operates at every frame), and it updates the 10 trajectory points that correspond to the past motion in the trajectory. For the current motion, the handler updates the middle point ($\frac{20}{2} = 10$ in our example) of the trajectory window using the prediction from the model. Lastly, for the trajectory points corresponding to the future motion, the handler converts the four predicted trajectory points to nine trajectory points through interpolation and thus updates the entire trajectory window after a prediction is obtained from the model. This example also shows how the handler, and by extension the controller system, performs interpolation between the keyframes that the model operates on.

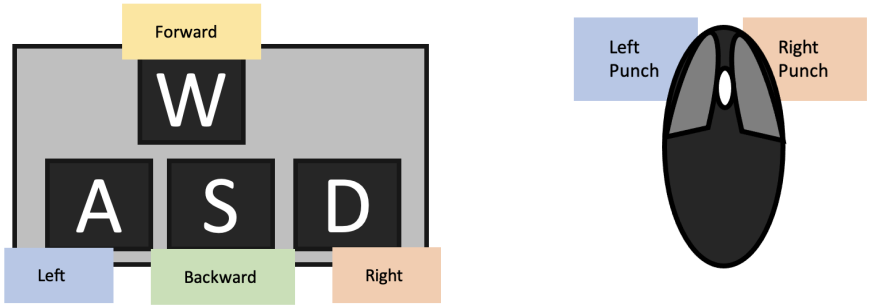
As seen in this section, the model handling unit performs many essential tasks in the controller system. Therefore, during implementation, it has been set up in an organised manner based on the different functions. This setup enables the smooth evaluation (see Chapter 6) of the multiple MoE models trained in this thesis. However, this handler alone cannot achieve interactive control with a user in the loop. On the other hand, the visualisation unit performs tasks that enable user interaction and are critical to our interactive boxing motion synthesis system. Therefore, this visualisation unit is presented in more detail in the next section.

5.2.2 Visualisation Unit

An essential part of interactive motion synthesis is the interface to view the synthesised motions and the interface through which the user can interact with the motion being synthesised. Without these interfaces, there cannot be much interaction, and there will not be a possibility to employ visual inspection to verify the quality of the motion synthesis. Therefore, the visualisation unit is a necessary part of the interactive boxing motion synthesis system.

The primary task of this unit is to **visualise the motions** predicted by the MoE model, and to do so, the visualisation unit has a virtual skeleton, on which the predicted motions can be applied. This unit will contact the model handler for the positional information to be applied to the skeleton before rendering each frame. The model handling unit prepares the positional information of the global joints of the virtual skeleton in the render function of the handler explained in Section 5.2.1. Upon the completion of execution of all the functions in the handler, the global joints information is sent to the visualisation unit as shown in Figure 5.2. Finally, the visualisation unit converts the positions to rotations using the Unity API and the generated rotations are then applied to all the joints in the virtual skeleton by going through them in the order of the joint hierarchy indicated in Figure 2.1b.

The other task in the visualisation unit is to provide an **interface for a user** to interact with the motions synthesised by our boxing controller system. To achieve this control, the visualisation unit allows the user to input the left and right punch commands through the left and right mouse buttons as indicated in Figure 5.5b. Similarly, the "WASD" keys from the keyboard are utilised to allow users to input the stepping directions as shown in Figure 5.5a. The punch target is the centre of the sphere shown in Figure 5.3. The visualisation unit determines the global position of this target in the virtual world and sends it to the model handler, along with which hand is supposed to be used to generate the punch.



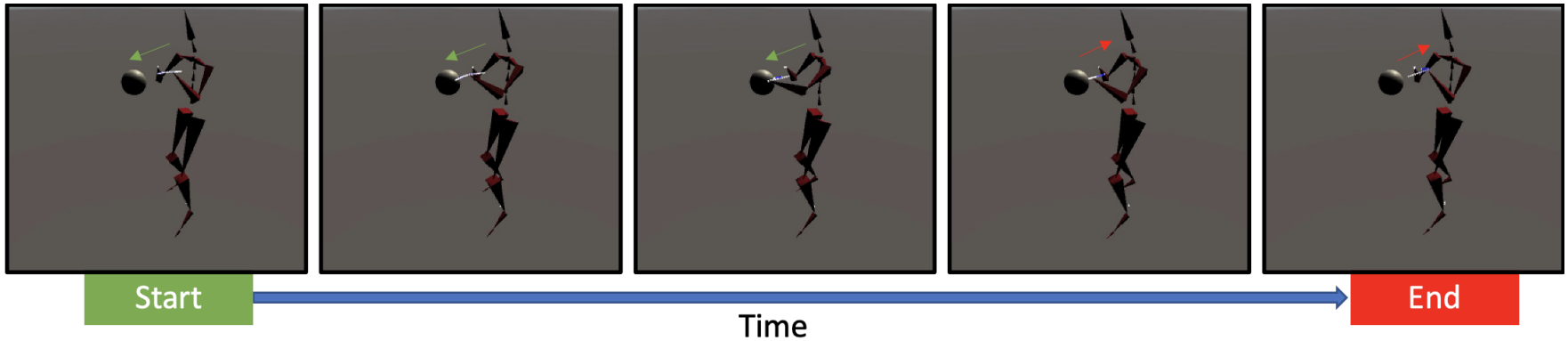
(a) (b)

Figure 5.5: The stepping control scheme uses the "WASD" keys from a keyboard as shown in Figure 5.5a and the punch control scheme uses the left and right mouse buttons for left and right punches, respectively, as seen in Figure 5.5b

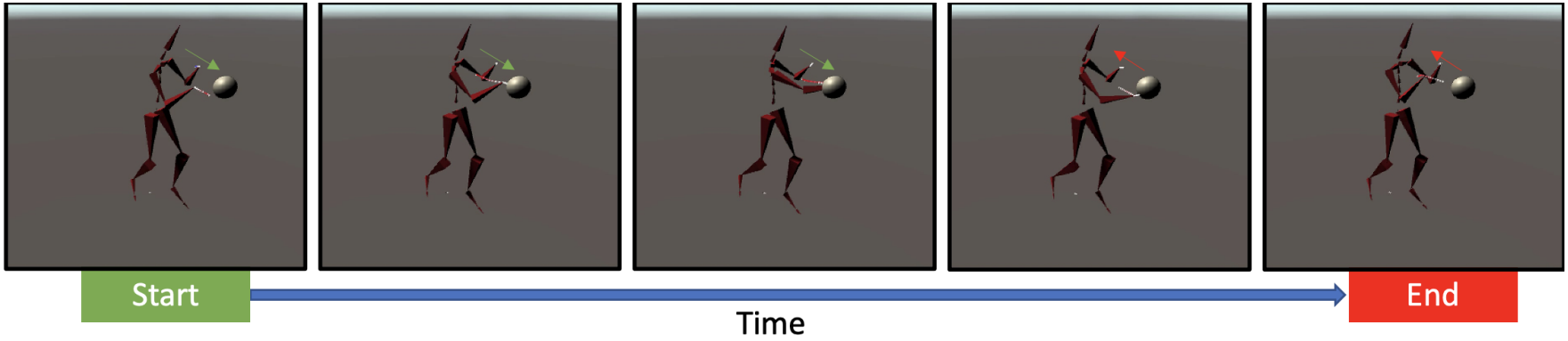
To evaluate the models, the visualisation unit **allows the punch targets to be moved automatically** without requiring any user input. This automatic movement is implemented in the interactive mode of the controller system, and this target manipulation is further explained in Chapter 6. However, due to the time limitation of the thesis, the visualisation unit does not currently offer the interactive movement of the punch targets for the user, and this can be developed in future work.

As seen in this section, the visualisation unit performs several essential functions for achieving interactive motion synthesis. This unit allows us to visually inspect the generated motions and manipulate the punch target positions to evaluate the model performance, which is presented in the next chapter.

To enable the readers to better understand the visual quality of the synthesised results, this chapter ends with several images from the visualisation unit that are displaying the motions synthesised in interactive control mode. First, Figures 5.6a and 5.6b show sequences of frames from the visualisation unit depicting the synthesised left and right punch actions respectively. Next, Figures 5.7a and 5.7b show the sequences of frames from the visualisation unit presenting the synthesised forward and backward stepping actions, respectively. Note that the sequences of images in these figures are for visualisation purposes only. The real motions occur over a longer duration, and we have sampled the frames in Figures 5.6a, 5.6b, 5.7a and 5.7b, from the longer time span to show the frames that are representative of the synthesised boxing motions.

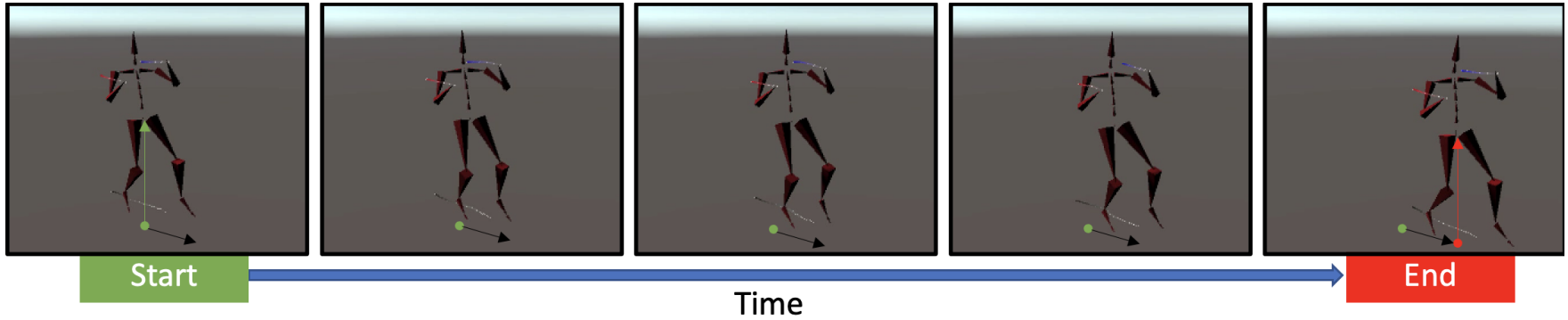


(a) The above sequence of images show a left punch.

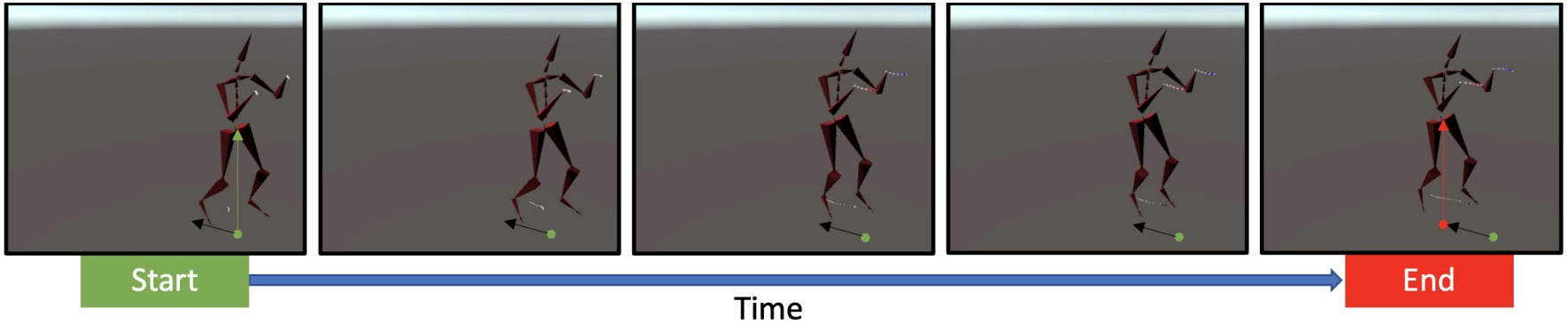


(b) The above sequence of images show a right punch.

Figure 5.6: The sequence of frames sampled from the synthesised results of a typical left punch in our system is shown in Figure 5.6a and similarly, the sequence of frames for a typical right punch is as in Figure 5.6b. The sphere represents the punch target and the green and red arrows in the figure indicate respectively the forward and backward motions of the hand in a synthesised punch.



(a) The above sequence of images indicate forward stepping.



(b) The above sequence of images indicate backward stepping.

Figure 5.7: The sequence of frames sampled from the synthesised results for the stepping forward action is shown in Figure 5.7a and similarly, the sequence of frames for the backward stepping action is as in Figure 5.7b.The black arrow indicates the user specified direction and the green and red circles indicate the position of the root bone corresponding to the start and end frames of the motion.

Chapter 6
Experiments

We utilise the boxing mocap dataset described in Chapter 3 to generate the motion state representations from Section 4.2 and train multiple versions of the MoE model described in Section 4.1. These models are placed in the boxing controller system, presented in the last chapter. In this chapter, the goal is to determine the best motion state representations for our MoE model. To achieve this goal, specific scenarios are set up and realised to evaluate the punching and stepping actions synthesised by our model. The synthesised motions for these scenarios are recorded by the controller system as shown in Figure 5.2. Once the rendering of all the scenarios is completed, these recordings are utilised by the evaluator unit from Section 5.1 to compute the evaluation metrics and performance plots that are used to verify the performance of our models for different scenarios and various system parameters.

In this chapter, the metrics used for evaluation and the reasoning behind them is explained in Section 6.1. Subsequently, we present the experiments conducted in this thesis with a detailed explanation of the associated experimental setups. Finally, each of these experiments are presented with an explanation of the results from the evaluation.

However, before going into the evaluation and experimental setup details, we present information about the parameters of our MoE boxing model that is common for all the experiments. A brief outline of our experiments is presented next as it will be helpful to understand the fixed model parameters and design decisions that are common for all the experiments.

The experiments conducted in this thesis to achieve better performance with our MoE boxing model are as explained below:

1. **Hyperparameter tuning experiment**: In this experiment, various models are trained to tune our MoE model to synthesise better motions. The particular hyperparameters in focus here are the number of hidden neurons in the motion prediction network, the learning rate for the model, and the number of gating experts.

2. **Punch control experiments**: This experiment determines the best performing models for the punch action. As the punching action is a critical part of boxing and as one of our primary goals is to achieve targeted control of motion synthesis (see

Section 1.1), we focused more on this action. In addition, there are also not many neural-based data-driven methods in the literature for synthesising better punches (we only know about [57]). Therefore, this experiment contains more information and is divided further into the following:

(a) *Trajectory sampling experiment*: Here, we change the variables corresponding to the wrist trajectory in the motion states to determine the performance of the model in relation to the number and the frequency of the past and the future motion samples in the trajectory variables. The former corresponds to the trajectory window w, and the latter corresponds to the frameskip fs in the trajectory window.

(b) *Ablation study*: This is a brief exploration of the performance of the model in synthesising punches when there are a varying number of wrist trajectory variables in the motion states.

3. **Stepping control experiment**: In this experiment, we are verifying the capability of our MoE model to perform the stepping action from one point to another in the boxing style. We train several models in relation to different root trajectory windows and frameskips within them. Therefore, this is also a *trajectory sampling experiment* applied to the stepping action.

For the hyperparameter tuning and trajectory sampling experiments conducted in this thesis, we keep the gating input fixed to the one shown in Equation (4.15). However, for the ablation study, the gating input varies according to the variables being considered in the motion states (see Section 6.3.3).

As mentioned earlier, many hyperparameters of our MoE boxing model are fixed based on the corresponding ones in MANN [63]. A detailed presentation of the parameters that were fixed during model training is presented in Section 4.1.3. However, we present them here briefly as they are a part of the setup for all the experiments.

Our MoE boxing model is designed using Tensorflow 2.0 [1] with the number of neurons in the hidden layer of the gating networks set to 64. The AdamWR [38] stochastic gradient descent algorithm is used for training with the weight decay parameter set as $\lambda = 10 * \lfloor length(\mathbf{X})/batchsize \rfloor$, where $length(\mathbf{X})$ represents the number of frames being considered for network training. The other parameter for AdamWR is the learning rate η which is tuned as part of our experiments (see Section 6.2). A batch size of 32, with each batch being sampled randomly, is used for training. Dropout [54] is applied with the retention probability as 0.2. Training takes around 30mins to 120mins on an NVIDIA Quadro RTX 8000 GPU, depending on the parameters considered in the experiments presented in this chapter. Similar to [63], we train the models presented in this chapter for 150 epochs as further training was not improving the visual quality of the synthesised motion. Finally, the entire boxing data is used for training all the models (see Section 6.1 for details). The settings mentioned here are constant throughout all the experiments and will not be reported again in the following sections.

In summary, the models are first tuned to produce better motions and then evaluated to verify the synthesis performance of the punch and stepping actions. During the evaluation, our motivation is to confirm whether the system can produce realistic-looking punches that a user can control by specifying targets. Similarly, we also want to verify if the system can produce realistic looking stepping action that can follow the direction commands from the user.

However, to understand the performance of the models, we first need to understand what the evaluation process is trying to achieve and what metrics are used to verify the performance. Therefore, we will present the details of the evaluation process in the next section.

6.1 Evaluation

In this section, we first look into what the evaluation intends to measure and then present the evaluation metrics that were used in the evaluator unit (see Section 5.1) to produce the tables and plots presented along with the experiments in the subsequent sections of this chapter.

6.1.1 Goals of Evaluation

In standard DL problems, the dataset is split into train and test sets and the models are trained on the former and tested on the latter. However, the use of a test set to evaluate the performance of generative models like our MoE model is not beneficial as Zhang et al. [63] found that better performance on the test set does not correspond with more natural-looking interactive motion synthesis when using MoE-type models. Therefore, the standard way of evaluation by comparison against a test set is not suitable for our models, and a more appropriate method needs to be determined.

On the one hand, interactive motion synthesis is a generative task wherein the stylistic quality and the correctness of the generated motions can be rated by the people interacting with the motions, and this is a valid approach used in the literature (for example, see [53]). On the other hand, since interactive motion synthesis using neural-based approaches are gaining more interest, it would be better if the research community could rely on standard metrics that can be utilised for comparison across different approaches. Therefore, we define metrics that are suitable for evaluating the punching and stepping actions. These metrics can also be extended to other targeted actions such as kicking.

For evaluating the punch action, we want to verify that the generated motion is smooth and does not have unexpected noise during the course of a punch. For example, in a regular punch action, the hand gets close to the target over the course of the action, reaches the target and then starts moving farther from it. We want to ensure that our evaluation metrics can ascertain the degree to which the models will perform these back and forth movements in relation to the target.

Similarly, to compare performance across multiple models, it would be beneficial to measure the accuracy of punches. However, accuracy can only confirm what percentage of punches reached the target. Thus, another measure is required to measure the amount of error in the accurate punches.

Finally, the feet also move during the execution of a punch as our mocap actor has taken a few steps while executing punches with more significant momentum. If the target is out of the reach of the model, then the synthesised result might float towards the target without stepping. Therefore, we need a metric that can measure the amount of floating during punching.

For the case of the stepping action, we would like to compare the ability of different models to synthesise motions capable of following the user input direction. Additionally, the model might have learned to translate the character in the global space without

performing any steps using the feet. This leads to the feet of the character floating along the ground, which might also be seen during the punching action. Therefore, a metric to measure the amount of floating during stepping is also required.

Note that since there is no test set requirement as determined by Zhang et al. [63], all the models in the following experiments sections (Sections 6.2, 6.3 and 6.4) are trained on the entire boxing dataset. Additionally, for the hyperparameter tuning experiments, we do not define any specific evaluation metrics as our focus in this thesis is in *interactive* motion synthesis and approaches specific to hyperparameter tuning fall into the core domain of ML. Therefore, we rely on the mean square error training loss plots and visual inspection of synthesised motions for the hyperparameter tuning experiments in Section 6.2.

After ascertaining how we would like to measure the performance of the models, we designed specific evaluation metrics in line with our evaluation goals, and they are presented in the next section. However, note that the synthesis results for all the models presented in this chapter are also verified by visual inspection as the goal of our motion synthesis is not only to be capable of producing accurate interactive synthesis but also to ensure realistic synthesis.

6.1.2 Evaluation Metrics

In this section we present the evaluation metrics that are utilised in the experiments (Sections 6.2, 6.3 and 6.4) to determine better performing models.

Punch Accuracy

The punch accuracy (PA) is a metric defined to determine the punching performance of our models when the wrist is closest to the user-specified target. We consider that the hand reaching inside a sphere, which has approximately the same dimensions as a human head, would be an accurate punch. Therefore, we define a sphere with a radius matching the average human head size and classify a punch as accurate when the hand is within this sphere and inaccurate otherwise. By consulting [26], it was found that the average head size of German males is 15cm. Therefore, we set the radius of the sphere as $p_{threshold} = 15cm$. To do this mathematically, the $\mathbf{l^2}$ norm is utilised as the sphere equation is similar to the norm equation. Using this, the accuracy of one punch is calculated as follows:

$$accuracy = \begin{cases} 1; & \mathbf{l^2}(x, y) < p_{threshold} \\ 0; & \mathbf{l^2}(x, y) \geq p_{threshold} \end{cases} \tag{6.1}$$

In the above equation Equation (6.1), the $\mathbf{l^2}$ norm is calculated between two 3D points x and y, where x represents the position of the wrist before the model handling unit starts retreating the hand and y represents the punch target position. The $\mathbf{l^2}$ norm between these two vectors is computed as follows:

$$\mathbf{l^2}(x, y) = \sqrt{\sum_{i=1}^{3}(x_i - y_i)^2} \tag{6.2}$$

Finally, the PA is calculated as the percentage of punches that were classified as accurate during an experiment. Therefore, for B accurate punches and Z number of total attempted punches, PA is given by:

$$\text{PA} = \frac{B}{Z} \times 100 \tag{6.3}$$

Note that we use the $\mathbf{l^2}$ norm to classify accurate punches as it is more intuitive to map the values from this norm to the bounds of a sphere in the virtual world. Hence, the $\mathbf{l^2}$ norm is not only used for evaluation but also in the model handling unit explained in Section 5.2.1. In the model handler, this norm is used to achieve the forward and backward motion of the hand during a punch as shown in Figure 5.3.

Punch Error

Punch error (PE) is the metric that has been defined to identify the punching performance of the model by measuring the error between the position of the hand and the target, right before the hand starts retreating to complete the punch. Additionally, since PA defined above can already give us a better understanding of the performance of the model over all the punches, this PE metric is designed to give us an understanding of how close the hand gets to the target for accurate punches. To compute this metric, we first calculate the mean squared error (MSE) between the wrist position $\mathbf{w}_n$ at the full extension of accurate punches B and the corresponding punch target $\mathbf{p}_n$, where $n \in \{1, 2, 3, ..., B\}$. Then PE is obtained as the average of the MSEs computed for all the B accurate punches.

To compute the PE, the MSE between two 3D points $\mathbf{x}$ and $\mathbf{y}$ is calculated as follows:

$$\text{MSE}(\mathbf{x}, \mathbf{y}) = \frac{1}{3}\sum_{i=1}^{3}(x_i - y_i)^2 \tag{6.4}$$

Using the above equation, PE is given by:

$$\text{PE}(\mathbf{W}, \mathbf{P}) = \frac{1}{B}\sum_{n=1}^{B} \text{MSE}(\mathbf{w}_n, \mathbf{p}_n) \tag{6.5}$$

where $\mathbf{W} \in \mathbb{R}^{B\times 3}$ is a matrix containing the wrist positions $\mathbf{w}_n \in \mathbb{R}^3$ of the B accurate punches and similarly, $\mathbf{P} \in \mathbb{R}^{B\times 3}$ is another matrix containing the punch target positions $\mathbf{p}_n \in \mathbb{R}^3$ of the B accurate punches.

This PE metric is designed to provide a better understanding of the punch performance in the punches that are classified as accurate. Therefore, a lower PE for model K compared to model L would imply that the target reaching capability in model K is better than that of model L.

Note that PA and PE are calculated for each hand separately. The results for each hand are then averaged to obtain the overall corresponding metric values presented in Section 6.3.

Stepping Path Error

Stepping path error (SPE) is the metric designed to calculate the degree to which the synthesised motion can follow the direction specified by the user while performing the stepping action. This metric is also computed using the MSE defined in Equation (6.4). For SPE calculation, the MSE is calculated between the ground projected root position $\mathbf{G}$ of the synthesised stepping action and the exact position to be followed $\mathbf{E}$. This MSE is calculated for every frame $f \in F = \{0, 1, ..., N\}$ in the evaluation where the character is stepping, and then SPE is obtained as the average of the computed MSEs. Therefore, SPE is given by:

$$\text{SPE}(\mathbf{G, E}) = \frac{1}{N}\sum_{f=1}^{N} \text{MSE}(\mathbf{g_f}, \mathbf{e_f}) \tag{6.6}$$

where $\mathbf{G} \in \mathbb{R}^{N\times 3}$ is a matrix containing the N ground projected root positions $\mathbf{g_f} \in \mathbb{R}^3$ and similarly, $\mathbf{E} \in \mathbb{R}^{N\times 3}$ is another matrix containing the N exact positions $\mathbf{e_f} \in \mathbb{R}^3$ to follow. Here, N represents the total number of frames where the stepping action is being performed.

Foot Skating Error

Motion synthesis systems are often unable to generate motions that can step correctly. Sometimes, the steps of the character do not involve lifting the feet away from the ground, which is referred to as foot skating. In this thesis, the amount of foot skating occurring during synthesis of both punching and stepping actions is represented by the foot skating error (FSE). This error is measured using the approach defined in [36]. However, our FSE measure is an average over both the feet l of the character, and the exact equation for measuring FSE is as follows:

$$\text{FSE} = \frac{1}{2}\sum_{l=1}^{2} d_l \left(2 - 2^{\left(\frac{h_l}{H}\right)}\right) \tag{6.7}$$

where d_l is the displacement of foot l and h_l is the average foot height from two consecutive poses during motion for foot l. H is a manually specified height threshold which we set to 3.5cm similar to [36]. Equation (6.7) is also used for calculating dataset foot skating scores in Table 3.1.

Note that SPE and FSE are calculated in each stepping direction separately. The results for each direction are then averaged to obtain the overall corresponding metric values presented in Sections 6.3 and 6.4.

The FSE metric is used in both Sections 6.3 and 6.4 for measuring the foot skating occurrences during punching and stepping, respectively. The PE and PA metrics are specific to the punch control experiment (Section 6.3) and are respectively used to measure the punch performance over all the punch frames and the performance at the frame where the hand is closest to the target. Finally, the SPE is used to measure the path following capability of the synthesised motions in the stepping control experiment (Section 6.4). However, before verifying the performance of the model using these metrics, its hyperparameters first need to be tuned. Therefore, the experiments that were performed to tune the model are presented in the next section.

6.2 Hyperparameter Tuning Experiments

As explained earlier, this experiment uses no specific setup as the model performance is not verified against a test set similar to the approach from Zhang et al. [63]. In addition, Zhang et al. [63] find that a lower training loss does not necessarily correlate with the quality of synthesised motion. Therefore, we also ensure that the synthesised motion is of high quality through visual inspection. However, for the parameters that we considered, we find that lower training loss did help in improving the visual quality of the synthesised motions for our boxing data.

In this experiment, while tuning a single variable, the remaining variables are fixed. The learning rate is fixed to 0.001, the number of hidden neurons in the motion prediction network is 512, and the number of gating experts is 6. Additionally, in these experiments, the trajectory window for both the stepping and punch control variables is set to 10 consecutive frames, i.e. window size w is 10 and frameskip fs is 1. The tuning experiments and the exact parameters we used are presented next. However, note that the sawtooth-like curves in the training loss plots presented in the tuning section (Figures 6.1, 6.2 and 6.3) correspond to the warm restarts being applied through the AdamWR stochastic gradient descent algorithm as explained in [38].

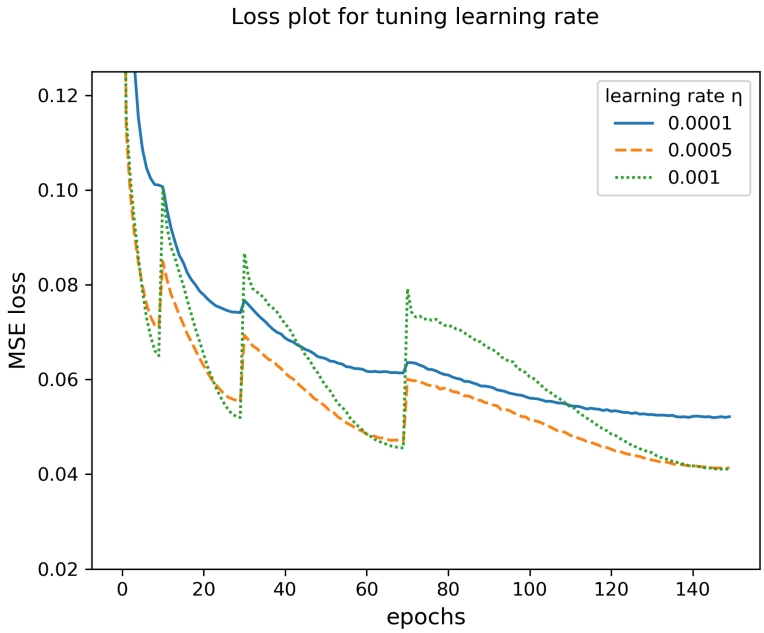


Figure 6.1: This figure shows the training loss curves for the learning rate tuning experiment. $\eta = 0.001$ leads to the lowest training loss.

Learning rate

The learning rate η is varied as one of $\{0.0005, 0.0001, 0.005, 0.001, 0.05\}$ in this experiment. The plots with the training loss corresponding to these values can be found in Figure 6.1. As per the training loss in the figure, $\eta = 0.001$ works best for our data, which

was also confirmed through visual inspection, as the resulting motions were sharper for η values of 0.001 and 0.0005. However, we choose $\eta = 0.001$ owing to the slightly lower training loss. Note that Figure 6.1 does not indicate loss curves for η values of 0.005 and 0.05 because these values cause the optimisation algorithm to land in a local minimum with a much higher training loss.

Number of gating experts

Similarly, we determine the best number of expert weights (K in Figure 4.1) to consider. These expert weights (combined using Equation (4.6)) are referred to as gating experts in this experiment. The number of gating expert parameters we considered for experimentation are $\{4, 6, 8\}$ and the loss plots are as in Figure 6.2. Zhang et al. [63] find that a higher number of experts results in improved motion quality for the quadruped data, but 4 experts are already sufficient. In our experiments, we find that 8 gating experts work well in terms of the training loss, but the visual quality of the results look similar. However, as we have many complicated and fast movements in our dataset, we decided to keep 8 gating experts as suggested by the loss and the finding mentioned above from Zhang et al. [63].

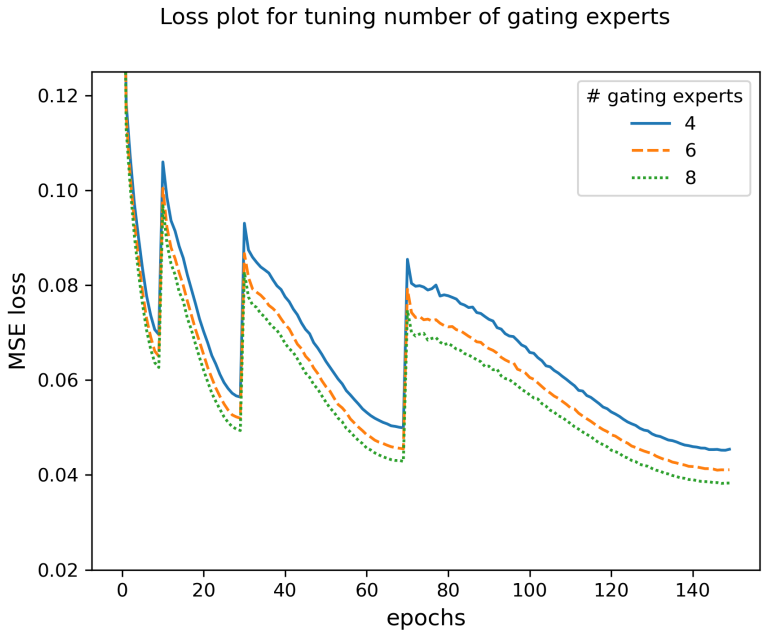


Figure 6.2: This figure shows the training loss curves for the tuning experiment for determining the best number of gating experts. Eight experts lead to the lowest training loss.

Number of hidden layer neurons in the motion prediction network

In the last tuning experiment, we identify the best number of hidden neurons in the motion prediction network. The parameters we considered here are $\{250, 275, 300, 512\}$.

The first three parameters are considered such that the number of hidden neurons are in between the number of input and output layer neurons in the motion prediction network, and the last parameter of 512 is as seen in [63]. The resulting loss plots are as seen in Figure 6.3. Considering the visual inspection and the training loss, we find that the best number of hidden neurons is 512 as in the original MANN model [63].

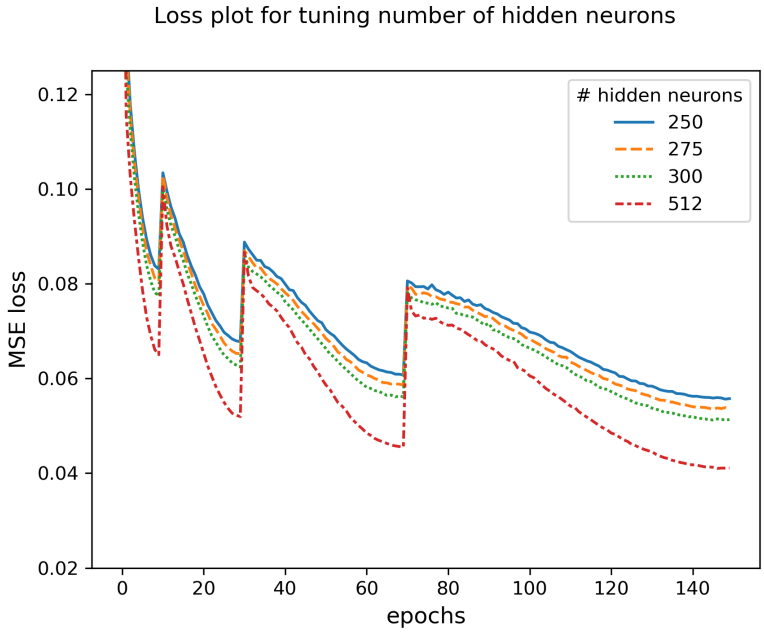


Figure 6.3: This figure shows the training loss curves for the tuning experiment to determine the number of hidden layer neurons in the motion prediction network. 512 hidden neurons leads to the lowest training loss.

As a result of the hyperparameter tuning experiments, the learning rate was fixed to $\eta = 0.001$, the number of gating experts to 8 and finally, the number of motion prediction network hidden layer neurons to 512 in all the models trained for the punch control and stepping control experiments presented in the following sections (Sections 6.3 and 6.4).

6.3 Punch Control Experiments

In this experiment, we analyse the performance of our MoE models during the synthesis of punch actions. In our boxing data, the punch actions occur over a short duration, and our models need to synthesise similar fast actions while also trying to reach for the user-specified target during a punch. Therefore, it is a complicated process for the model to handle. Additionally, there are not many neural-based data-driven approaches for punching action other than the very recent approach by Starke et al. [57]. However, the motivation for their work is different from our motivation (see Chapter 2). Regardless, there is a need to systematically study the performance of neural-based models for the punching action, which is covered in this section.

In this systematic analysis, we utilise the metrics that were defined in Section 6.1 to compare the performance across various models. As we are dealing only with the punching action in this experiment, we utilise the PA, PE and FSE for evaluating the model performance. Since the goal is also to achieve realistic motion, the synthesis results of the punch actions are also verified through visual inspection. Thus, this section provides information about the setup considered for the systematic analysis and presents the metrics computed during evaluation.

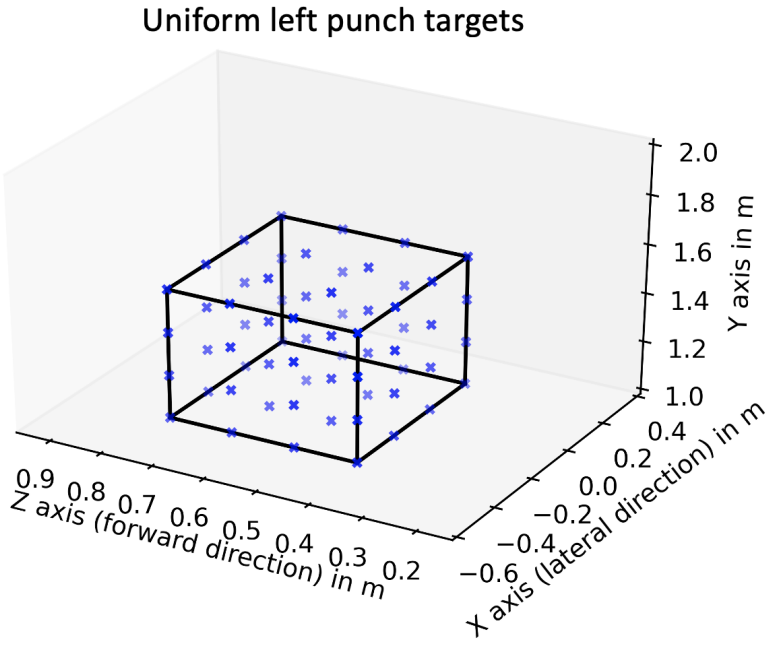
For this evaluation of punching, a specific testing scenario is developed by referring to the mocap recording results presented in Section 3.3. All the details about the testing scenario are presented in the next section. In addition, this experiment is subdivided into the trajectory sampling experiment and the ablation study experiment, which are presented in Sections 6.3.2 and 6.3.3 respectively. The former focuses on determining the duration of past and future motion information needed in the trajectory for improving model performance. The latter focuses on the kind of variables required in the motion states for similarly improving the model performance. More details about these experiments are presented after the experimental setup is described in the next section.

6.3.1 Experimental Setup

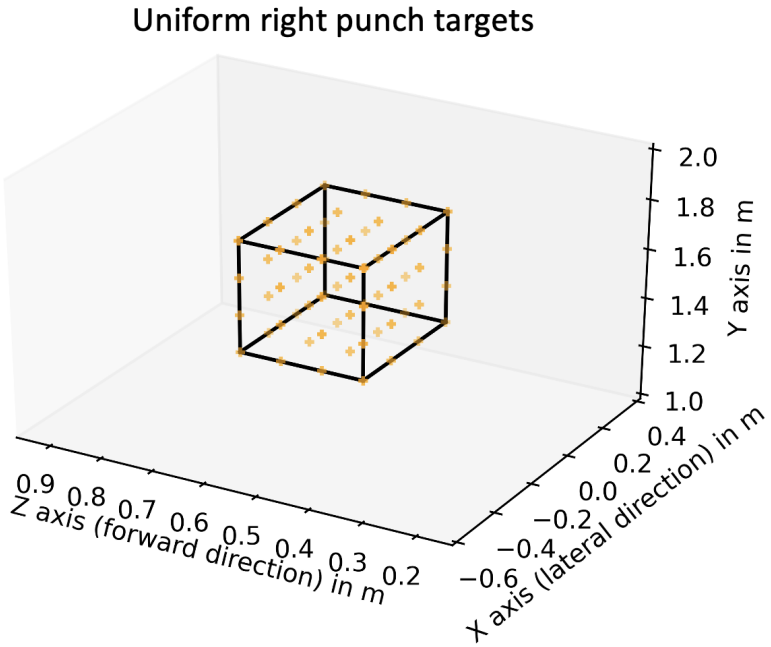
This thesis aims to allow interactive control of boxing motions through our MoE model. Since punching actions are a part of boxing, a suitable scenario is developed to verify the performance of our models in synthesising interactive punching. To achieve this, the punch target points from Figure 3.4 were used to determine the bounds of the punch targets in our data. A regular grid of 64 punch targets was spawned within the dataset bounds for each of the hand of the character. The resulting test targets generated using this method are as shown in Figure 6.4.

Note that the generated test targets are computed by determining the maximum and minimum extents in each of the three dimensions of the coordinate space of *all* the dataset targets (per hand), which also includes extreme points in the dataset targets that are far away from the cluster of targets, as seen in Figure 3.4. Thus, when comparing Figures 3.4 and 6.4, many of the generated targets are not overlapping with the dataset target cloud, and therefore, these test targets are not seen during training and are good targets for testing the models. In addition, the test targets have more extreme points than the train set, and these are deliberately included to test the limit of our MoE model while punching extreme targets not seen in training.

A visualisation of the character rendered by the visualisation unit along with the test targets is presented in Figure 6.5. As seen in the figure, the visualisation unit cycles through the generated test targets that are spawned in front of it and executes punches towards each one of the targets. To do this, the visualisation unit requests the model handler for the synthesised motions for punching these targets. Once the handler starts producing the punch action, the visualisation unit holds the punch target constant until the synthesised punch is completed. Thus, we have two different sets of 64 punch targets for each of the left and right hands, as indicated in Figure 6.4. The visualisation unit is set up to perform a left punch first and then a right punch. Using the model handler, the pose of the character is also reset to the starting pose after every punch is completed, and the starting pose is where the root is at the origin of the virtual world. The trajectories for the starting pose are extracted from a frame in the dataset where the character was idling. Finally, note that the trajectory window for the root variables (see Section 4.2) is permanently fixed to 10 frames in the following punch control experiments.



(a)



(b)

Figure 6.4: Generated target punch positions for evaluating the punch control in our system. Different set of targets are generated for the left hand as shown in Figure 6.4a and for the right hand as shown in Figure 6.4b. Each target set contains 64 targets, meaning that there are a total of 128 targets for both the hands.

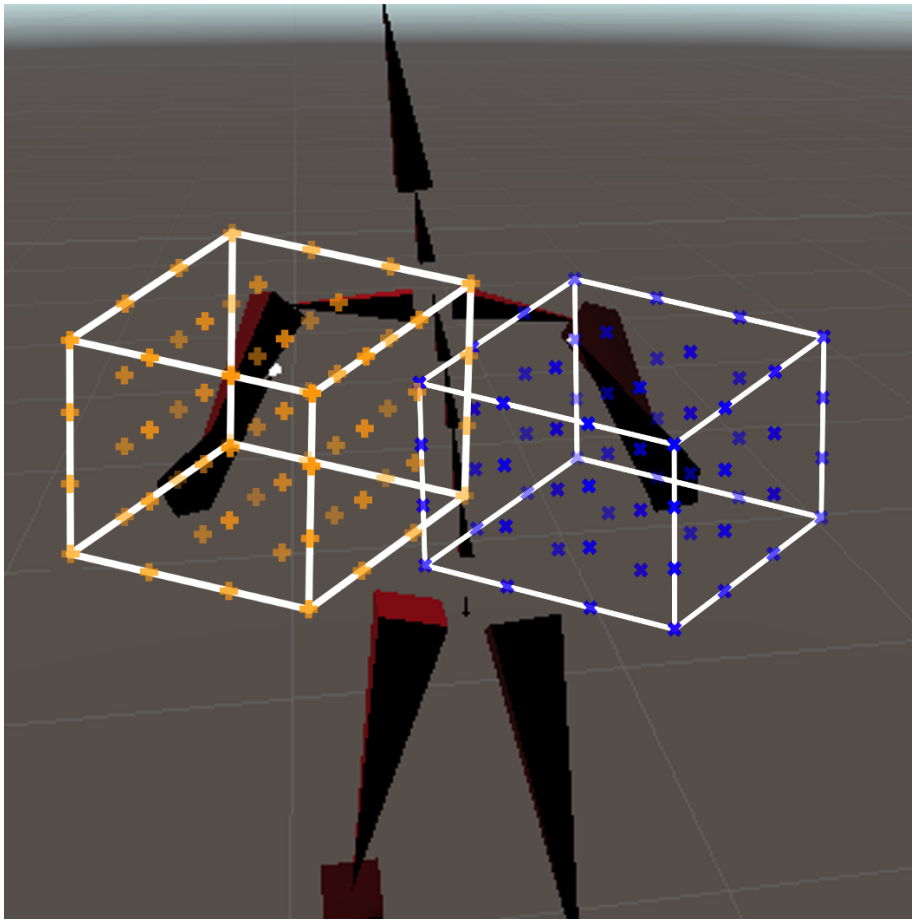


Figure 6.5: This is a visualisation of the evaluation punch targets superimposed on our character. Note that this image is for visualisation purpose only and the positions of the actual test targets in the virtual world may vary from this visualisation.

The experimental setup explained in this section applies to the evaluation of the ability of our MoE model to synthesise targeted punches in the trajectory sampling experiment presented in Section 6.3.2 and in the ablation study experiment presented in Section 6.3.3.

6.3.2 Trajectory Sampling Experiment

This experiment determines how many data points are required in the trajectory and at what rate these data points must be sampled to improve the model performance. This sampling experiment is performed for both the punching and the stepping actions. However, in this section, we present only the details corresponding to the trajectory sampling that affects the punch action and the experimental details about the sampling experiment conducted with stepping action is presented in Section 6.4.

Hypothesis: Boxing motions involve a variety of punches that occur in short bursts of a few frames, which means that a punch or a step takes place over a concise duration of

time. In our boxing data, the average punch is executed over approximately 27 frames (see Table 3.3) which corresponds to approximately 0.45s of motion in the real world. This is a significantly short amount of time. Based on this, the hypothesis in this experiment is that the trajectory sampling must coincide with the fast punch actions in the dataset to synthesise more accurate punches. Thus, the expectation here is to find that the trajectory window w and frameskip fs should lead to a trajectory window that covers motion similar to the average dataset punch duration and such a window will help our model synthesise accurate punches.

Experimental settings: To verify this hypothesis, we select several values of the trajectory window w and frameskip fs such that the trajectory window of the wrist variables in the motion states described in Section 4.2, covers motion information corresponding to approximately 0.07s to 0.9s of motion in the real world. These values ensure that the trajectory information tested covers all the motion durations that are under twice the average dataset punch duration (i.e. $2\times$0.45s). Using these parameter values to update the wrist trajectory variables, several versions of the MoE model are trained. Next, these models are deployed in the experimental setup described in Section 6.3.1, and visual inspection is employed to verify that the produced punching actions look realistic and adhere to the boxing style in the mocap data. The observation is that our interactive motion synthesis system can produce punches even for extreme targets that were not in the dataset. Also, as expected from neural networks, the model performs better on the targets close to the ones in the training set and fails to reach the other targets even when the model handling unit ensures that the trajectory is pointing to the target (see Section 5.2.1). More details about such missed punches are presented in Chapter 7.

Experimental Observations

The synthesised results for all the models trained in this experiment were verified to be visually plausible. Therefore, the next step is to employ the evaluation metrics explained in Section 6.3.1 and determine which trajectory parameters result in better targeted punching. The results of applying these metrics to verify the performance of the models trained on different wrist trajectory window size w and frameskip fs is presented in Table 6.1. For synthesising more accurate punches, PE must be low and PA must be high. As seen in the table, our MoE model performs more accurate punching (w.r.t. the PA metric) when the trajectory window size is $w = 14$ and frameskip is $fs = 3$ for the trajectory wrist variables in the motion states described in Section 4.2. These trajectory parameters result in an overall 42 frames (i.e. 0.7s) of motion information in the trajectory, i.e. this window covers 0.35s of past motion, one frame corresponding to current motion and 0.3s of future motion. This result is approximately around the average dataset punch duration, and this trajectory window of 0.7s is of much shorter duration compared to the trajectory window of 2s used by Holden et al. [21], Zhang et al. [63] for locomotion tasks.

However, note that, as Table 6.1 indicates, the FSE is quite high for this accurate model compared to models trained with $fs = 1$ or $fs = 2$. This increased FSE is related to the higher PA for the models with $fs = 3$ because these models are synthesising motions that are reaching and hitting the targets that are not close to the character. During this reach, there is more foot skating as the character takes a step or fails to take a step and floats to the target. On the contrary, the models with lower fs cannot synthesise motion that reaches many targets away from the character. Simultaneously, the synthesis also

Table 6.1: This table shows the results of the trajectory sampling experiment conducted for evaluating the punch control capability of our MoE models. The wrist trajectory parameters of trajectory window w and frameskip fs are varied to train several models and the corresponding evaluation metrics are calculated for both the hands and averaged to obtain the overall scores presented in this table.

w	fs	PE$_{right}$ (cm^2)	PE$_{left}$ (cm^2)	PE$_{overall}$ (cm^2)	PA$_{right}$ (%)	PA$_{left}$ (%)	PA$_{overall}$ (%)	FSE (cm)
4	1	47.16	42.50	44.83	15.62	25.00	20.31	**0.285**
6	1	44.72	42.58	43.65	20.31	37.50	28.91	0.305
8	1	43.03	39.46	41.24	21.88	35.94	28.91	0.332
10	1	45.84	41.74	43.79	25.00	40.62	32.81	0.336
12	1	43.16	41.23	42.19	26.56	42.19	34.38	0.403
14	1	42.98	41.15	42.06	26.56	48.44	37.50	0.341
16	1	42.66	40.36	41.51	21.88	31.25	26.56	0.418
18	1	41.47	39.67	40.57	26.56	34.38	30.47	0.378
20	1	43.88	43.13	43.50	21.88	35.94	28.91	0.404
12	2	45.06	36.56	40.81	18.75	56.25	37.50	0.321
14	2	35.98	**34.08**	**35.03**	15.62	57.81	36.72	0.320
16	2	**33.59**	36.93	35.26	23.44	54.69	39.06	0.335
18	2	43.11	36.91	40.01	32.81	53.12	42.97	0.350
20	2	43.07	39.87	41.47	28.12	39.06	33.59	0.804
10	3	45.98	36.88	41.43	34.38	56.25	45.31	0.549
14	**3**	44.98	43.40	44.19	**42.19**	**65.62**	**53.91**	0.667
16	3	41.95	44.37	43.16	32.81	48.44	40.62	0.777
18	3	38.42	46.16	42.29	37.50	48.44	42.97	0.846

does not move the character towards the target as much as in the more accurate models with $fs = 3$.

Another observation is that the model with trajectory window $w = 14$ and frameskip $fs = 2$ synthesises punches that reach points that are much closer to the target as indicated by the overall PE score in Table 6.1. However, we consider that the model trained on $w = 14$ and $fs = 3$ performs better targeted actions as it can synthesise motions that can punch accurately, but admittedly with a higher allowance for PE and a moderate allowance for FSE. Also note that the PE measure is calculated only for the accurate punches. Therefore, the model with lower PE does not result in the best targeted actions.

When considering only the models with frameskip $fs = 3$, the trajectory window size of $w = 14$ produces the best results, probably because the window covers the right amount of punching motion information for our MoE model to learn better mappings.

Finally, to verify that the punch action produced is smooth throughout the back and forth action involved in all the punches, the plot in Figure 6.6 is computed using the model trained with $w = 14$ and $fs = 3$. For this plot, the $\mathbf{l^2}$ norm between the wrist positions during a punch and the corresponding punch target is computed, and the norms corresponding to each punch are plotted. A smooth punch is expected to have a parabolic plot as the calculated $\mathbf{l^2}$ norm decreases when the character is reaching towards the target, reaches a minimum point and then starts increasing as the wrist

is being retreated from the target. The curves in Figure 6.6 are mostly parabolic and smooth. The smoothness of these curves indicates that the generated motions for the punch action do not involve much noise. Finally, Figure 6.6 also indicates the punch accuracy threshold $p_{threshold} = 15cm$ in Equation (6.1) that is used to classify the punches as accurate and this is indicated in Figure 6.6 by the green norm curves that are classified as accurate punches, and the red curves indicate missed punches. Additionally, this plot is also a good indicator of how the network predicted motion retrieves the hand to various different extents even when the model handler points the trajectory directly to the shoulder. These predictions are due to our MoE model trying to execute the punch action and reach targets that were not in the dataset. Such predictions also add variety to the synthesised motions.

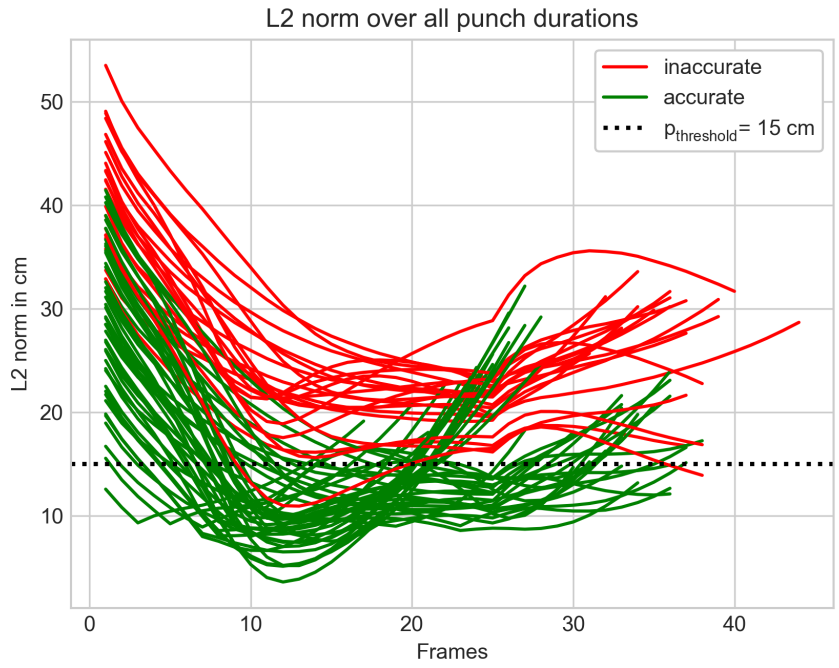


Figure 6.6: This plot shows how the $\mathbf{l^2}$ norm, computed between the wrist positions in a punch and the corresponding punch targets, varies over the course of the punch. Additionally, it also shows how the punch threshold $p_{threshold}$ is applied to classify punches as accurate using Equation (6.1). The curves in this figure are corresponding to the motions synthesised by the MoE model trained with $w = 14$ and $fs = 3$ for the wrist trajectory in the motion states.

Conclusion for wrist trajectory sampling experiment: The result of this experiment is that the punching action requires less duration of motion information in the wrist trajectory variables compared to [21, 63], which are approaches designed for locomotion tasks. Essentially, the trajectory variables covering 0.7s of motion results in more accurate punches. Since 0.7s is approximately around the dataset average punch duration of 0.45s (in comparison to 2s used in the literature), we can say that this experiment confirms our

hypothesis that the trajectory samples coinciding with the punch action statistics results in more accurate punches.

The corresponding best trajectory parameters of $w = 14$ and $fs = 3$ are utilised in the ablation study experiment presented in the next section, which is aimed at understanding whether all the wrist trajectory variables are required to achieve better performance with natural-looking results.

6.3.3 Ablation Study Experiment

In this experiment, the relative importance of the wrist trajectory variables is analysed in relation to the evaluation metrics and through visual inspection. The experimental setup remains the same as what is presented in Section 6.3.1.

Hypothesis: As indicated in the motion states presented in Section 4.2, the variables considered in this thesis to achieve targeted punching using our MoE model are the wrist trajectory positions, trajectory velocities, punch action label and punch target. The last two variables are required for the basic functioning of our MoE model, but the performance of the model with and without the trajectory variables can be verified. Note that, while designing the motion states in Section 4.2, we hypothesised that sufficient wrist trajectory information is required for controlling targeted actions, as neural networks most likely perform better when they are provided sufficient contextual information about the past and future motion. However, Freda [14] also found that including only the target position will not be sufficient for synthesising targeted control of sword motion using the PFNN model. Since our model is similar to the PFNN model (see Chapter 2), this observation is most likely to be applicable here as well. Therefore, in this experiment, our hypothesis is that all the trajectory variables considered in our motion states are necessary for achieving better targeted punches.

Experimental settings and observations

To test our hypothesis, three models are trained in this experiment, with the variables being considered as shown in the first column of Table 6.2. These models are evaluated through visual inspection first, and we find that the synthesised motions are realistic for all the cases. However, the model with no trajectory information fails to be responsive to the boxing controller system input as it generates many random punches in the direction of the punch targets. More details about this model are presented in Chapter 7. However, the models with the trajectory variables are more responsive to user inputs while still synthesising realistic motions.

Next, the evaluation metrics are computed for all the models considered, and the results are presented in Table 6.2. As expected, our MoE model, which is trained on all the wrist trajectory variables, provides the best punching performance in terms of more accurate punches as well as with lower FSE. The increased FSE for the other models is due to the character taking more steps towards the target instead of punching the target, which is most likely due to the underlying training data containing multiple punches where the mocap actor is stepping forward while punching.

Conclusion for the wrist trajectory ablation study: Thus, our experiment shows that all the trajectory variables considered in our motion states are essential for achieving

Table 6.2: This table shows the results of the ablation study experiment conducted for evaluating the punch control capability of our MoE models. The wrist trajectory variables are considered as shown in column one of the table and several models are trained with $w = 14$ and $fs = 3$ (when trajectory is being considered) and the results of the evaluation are presented in this table.

Wrist trajectory variables	PE_{right} (cm^2)	PE_{left} (cm^2)	$PE_{overall}$ (cm^2)	PA_{right} (%)	PA_{left} (%)	$PA_{overall}$ (%)	FSE (cm)
none	**42.28**	47.26	44.77	15.62	46.88	31.25	1.177
position	54.61	45.42	50.02	31.25	40.62	35.94	0.980
position & velocity	44.98	**43.40**	**44.19**	**42.19**	**65.62**	**53.91**	**0.667**

better targeted punches. This confirms our hypothesis that our neural network based MoE model needs all the trajectory information that we considered in the motion states for performing more accurate punches.

This concludes our brief ablation study, but it will be discussed further in Chapter 7. However, in the next section, we present details of the experimentation in relation to the stepping action.

6.4 Stepping Control Experiments

While punching is an important part of the action set in boxing, it also requires realistic and responsive stepping synthesis. In the specific case of this thesis, the stepping must be in the style of boxing captured in the data. Zhang et al. [63], Starke et al. [55, 56] have already analysed and improved neural-based interactive locomotion synthesis through MoE models. However, the style of movements involved in locomotion is not similar to the stepping action in boxing. Therefore, our MoE model needs to be tuned to produce natural-looking interactive stepping actions. To achieve this, we focus only on appropriately sampling the root trajectory variables in the motion states described in Section 4.2.

The models trained on several versions of the motion states updated for stepping are evaluated using the metrics defined in Section 6.1. However, a visual inspection is first applied to determine the models that produce realistic motions. To perform the visual inspection and to calculate the evaluation metrics, a specific scenario is defined to test the interactive synthesis of the stepping action. In the next section, we provide more details about the setup required to test the stepping action.

6.4.1 Experimental Setup

As described in Chapter 3, the mocap actor was instructed to execute the stepping action along the longest dimension of the mocap area. Accordingly, the resulting boxing data we gathered includes primarily forward stepping actions, followed by backward stepping actions and very few frames of lateral stepping. Thus, the stepping control experiment is classified into two: a) forward stepping control and b) reverse stepping control. On this basis, two separate scenarios are set up in our visualisation unit of the

boxing controller system. In the first scenario, the system will continuously request the controller to synthesise forward stepping, and in the second scenario, the system requests continuous synthesis of the backward stepping. Both of these scenarios are set up to synthesise 5s (i.e. 300 frames) of stepping action. Finally, note that the trajectory window for the wrist variables is always fixed to 10 frames in the following stepping control experiments.

Note that in both the forward and the backward stepping tests, the character starts at the origin and is expected to move along the z-axis. Thus, points along the z-axis are treated as the positions at which the character is expected to be while executing these actions.

In the following section, the above experimental setup is used to determine the best way to sample the root trajectory variables and improve the performance of our MoE models in the stepping action.

6.4.2 Trajectory Sampling Experiment

Hypothesis: As mentioned in Section 3.3, the stepping action in our dataset also occurs over a short duration. In our boxing data, an average step is executed over approximately 20 frames (see table 3.2) which corresponds to approximately 0.33s of motion in the real world. Thus, the stepping action is also a fast action in our boxing data. This fastness coupled with the boxing style of the stepping action shows that the trajectories utilised by Holden et al. [21] might not be applicable for stepping, as they considered long trajectory windows covering 2s of motion for their locomotion tasks. Thus, based on the dataset average step duration of 20 frames, we hypothesise that the stepping action would be synthesised better by our MoE model when a short trajectory window w and frameskip fs result in a trajectory covering motion similar to the average stepping duration.

Experimental settings and observations

Most of the approaches considering similar models and actions (for example, Holden et al. [21], Zhang et al. [63]) use trajectories that cover 2s of motion. Hence, we decided to test several trajectories that cover 0.3s to 2s of motion. Thus, based on the hypothesis and the literature, we trained several models in the experimental setup described in Section 6.4.1. First, we performed a visual inspection that verified the naturalness of the synthesised stepping actions. Probably as a consequence of the fast motions in our boxing data, the models that were trained on more than 1s of root trajectory information led to large predictions from the model and resulted in overflow during computation. These models are therefore excluded from further evaluation. However, the models trained with 0.3s to 1s of trajectory information synthesised good looking stepping actions, but the results are biased in terms of more natural-looking motion for the forward motion than for the backward motion. Essentially, forward stepping involves a good number of steps with little character floating, but the same is not true for backward stepping, and this is discussed further in Chapter 7.

For the models with natural-looking results, we compute the evaluation metrics of SPE and FSE. These metrics are computed for both the forward and backward stepping actions, and the results are averaged and presented in Table 6.3. From the visual inspection, we find that the model with root trajectory window $w = 10$ and frameskip $fs = 2$ produces the most realistic results for both the forward and the backward stepping. However, when considering the forward or backward stepping individually, the SPE

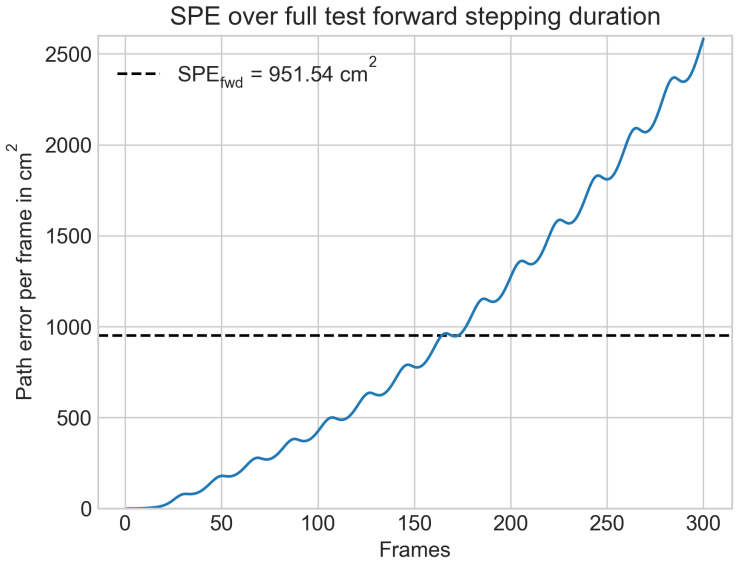
measures for this model are not the best as seen in Table 6.3. Contrarily, as indicated by the overall SPE, this model synthesises motions that are better capable of following the user-input in both the forward *and* backward directions. In the case of the FSE, the score corresponding to the backward stepping action for this model is the best, and the one corresponding to forward stepping is moderate. This leads the overall FSE to be the best and it indicates that the overall amount of foot skating in this model is considerably low. These metrics correspond to our visual inspection results and show that the model trained on motion states with root trajectory window $w = 10$ and $fs = 2$ performs well and results in the synthesis having a good number of steps and being capable of responding to both the forward and backward user inputs. Additionally, the trajectory window selected for this model also confirms our hypothesis, which was set up considering the fast nature of the stepping actions in our boxing data.

Table 6.3: This table shows the results of the trajectory sampling experiment conducted for evaluating the stepping control capability of our MoE models. The root trajectory parameters of trajectory window w and frameskip fs are varied to train several models and the corresponding evaluation metrics are calculated for both the forward and backward stepping actions and averaged to obtain the overall scores presented in this table.

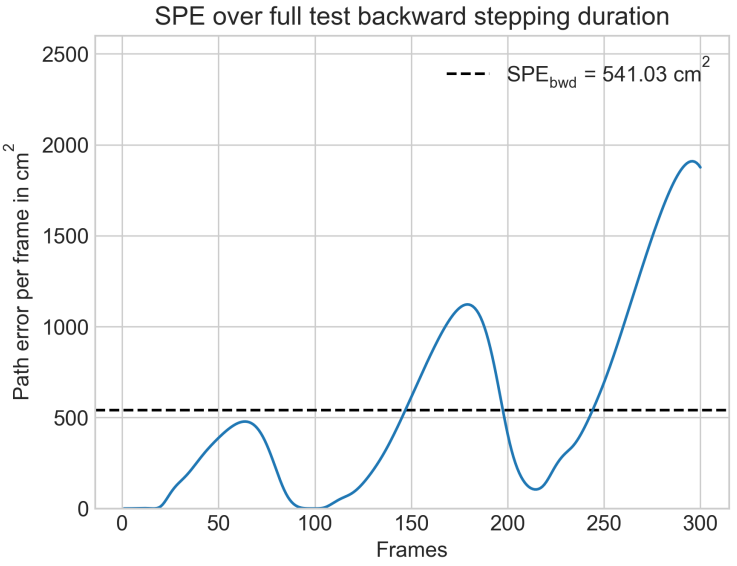
w	fs	SPE$_{fwd}$ (cm^2)	SPE$_{bwd}$ (cm^2)	SPE$_{overall}$ (cm^2)	FSE$_{fwd}$ (cm)	FSE$_{bwd}$ (cm)	FSE$_{overall}$ (cm)
10	**2**	951.54	541.03	**746.29**	0.77	**0.27**	**0.52**
12	2	1789.21	330.62	1059.91	0.89	0.51	0.70
14	2	5481.01	**65.27**	2773.14	0.93	0.59	0.76
16	2	5979.11	1199.40	3589.26	**0.62**	0.51	0.56
18	2	1373.73	4544.92	2959.33	1.19	0.75	0.97
10	3	2324.52	143.60	1234.06	1.55	1.01	1.28
8	4	490.94	8957.40	4724.17	0.92	1.8	1.36
14	4	**216.77**	6041.81	3129.29	2.19	3.0	2.59

The best model from the evaluation is also used to compute the plots in Figure 6.7. The Figures 6.7a and 6.7b are representing the MSE path error over the duration of 300s for the forward stepping and backward stepping actions, respectively, and the horizontal marking shows the average path error which is the SPE explained in Section 6.1.2. These plots indicate the biased nature of the motion synthesis as the MSE increases over time. The jagged nature of the curves in both Figures 6.7a and 6.7b indicates that the user input trajectory tries to pull the motion back towards the input direction. Finally, the fewer peaks and troughs in the curve associated with backward stepping imply that there are fewer steps in the backward stepping action compared to the forward stepping action, and this can be concluded as the MSE decreases when the character is floating along the ground and is being influenced more heavily to move in the direction of the user input.

Conclusion for the root trajectory sampling experiment: Many models were trained to improve the stepping action synthesis of our MoE model, and we found that natural-looking and responsive results are obtained when the root trajectory variables considered for training the model have nearly the same amount of motion information as the step duration in the boxing data. This confirms our hypothesis for the stepping control experiment that the stepping action in boxing requires short trajectories for synthesising



(a)



(b)

Figure 6.7: The path error for the forward stepping is shown in Figure 6.7a and similarly, the path error for the backward stepping is shown in Figure 6.7b. SPE from Equation (6.6) is shown for both forward (SPE$_{fwd}$) and backward (SPE$_{bwd}$) stepping actions. These plots are computed over the entire duration of the stepping experiments i.e. 300 frames.

better stepping motions as this boxing action is also non-periodic and fast, in comparison with walking action in locomotion tasks.

Thus, we conclude our presentation of the experiments conducted in this thesis with a brief summary of the experimental findings in the following section.

6.5 Brief Summary

In this chapter, we defined the metrics for evaluation, set up scenarios for verifying the stepping and punching capabilities of our MoE model and presented the results of the evaluation. The experiments are implemented using the boxing controller system from Chapter 5 and the evaluation metrics presented here are calculated using the evaluator unit (see Section 5.1).

The results from our evaluation showed that the trajectory for both the stepping and walking actions need to be in line with the motion duration of the corresponding actions in the dataset. Particularly for the case of punching action, we also showed that all the trajectory variables considered in our motion states (defined in Section 4.2) are essential for more accurate targeted punching.

Along with these results, many peculiar artefacts were observed during visual inspection of the synthesised motions, and more details are provided about these in the next chapter. We also present the most important conclusions from our experiments, the limitations of our system and the direction for future work.

Chapter 7
Discussion

In this chapter, the results of our interactive boxing synthesis approach are discussed with a focus on understanding why such synthesis results are obtained. Subsequently, we discuss the limitations of our approach and present details about where our MoE model and boxing controller system are lacking and how they can be improved. Next, we show how the approach presented in this thesis can be extended to improve the boxing motion synthesis. Finally, we conclude this thesis with information about our most important findings.

7.1 General discussion

In this thesis, the motivation is to apply modern neural models to synthesise boxing actions such as punching and stepping and adapt these models to allow real-time user control of the synthesised motions. To perform such interactive motion synthesis, a thorough literature survey is conducted to determine the prevalent motion synthesis approaches in the research domain. The finding from the survey is that a neural-based MoE model would be suitable for interactive synthesis. This is applicable for the non-periodic boxing actions of punching and stepping considered in this thesis as Zhang et al. [63] showed that MoE-type models are capable of synthesising such non-periodic actions (see Chapter 2).

In the pursuit of interactive boxing synthesis, it was determined that the boxing actions could be generated in a better way using our own mocap data that is covering all the motions that our MoE model should synthesise. Thus, a novel boxing dataset is prepared along with a system for automatic punch labelling. This data preparation is followed by designing boxing-specific motion states that can be used in neural model training. These motion states are used to train the MoE boxing model. Finally, for allowing user interaction, a boxing controller system is developed, which allows the users to control the motions synthesised by our MoE model.

Experimental findings

For training and testing several models, the software code used in this thesis is separated into several units, and consequently, these units help train and evaluate several models on various versions of the motion states. Several experimental scenarios are considered to verify the punching and stepping capability of the synthesised motions. By employing visual inspection and evaluation metrics that were designed specifically for boxing actions, we were able to confirm that our MoE model is capable of generating realistic and responsive punching and stepping actions. Such results are achieved despite having access to only 10mins of boxing mocap data, as presented in Section 3.3. The amount of mocap data varies from 30mins for training MANN [63] to 1hr for training the PFNN [21]. Similarly, the subsequent research on interactive synthesis using MoE-type models by Starke et al. [55, 56, 57] use a large amount of mocap data. However, our approach already produces responsive and realistic motions using significantly lesser data than these methods.

As explained in Section 2.3, our MoE model is based on the MANN model developed by Zhang et al. [63]. Initially, we tried to use the MANN model with the same parameters used in [63] and were successful in synthesising boxing motions. However, it was found through visual inspection that tuning the model further instead of using the typical model parameters from [63] helps produce sharper boxing motions. This finding is in line with standard practices in the DL domain where the hyperparameters of a model, shown to be successful on a certain dataset A, is fine-tuned when it is being adapted to a new dataset, and [46] further explains the importance of such tuning. Thus, when adapting neural models such as our MoE model to a new motion synthesis task, model tuning should be explored as it might improve the synthesis.

Our approach of synthesising targeted actions using neural-based models was also attempted by Freda [14] for controlling sword motions, and Freda found that using the neural-based model of PFNN did not produce good synthesis results. However, our neural model, which is an extension of the PFNN (see Chapter 2), can achieve targeted punch motion synthesis. In addition to the change in the network architecture, the main difference in our approach compared to Freda [14] is the addition of the wrist trajectory variables to the motion states, and this addition helps our model learn how the wrist needs to be moved towards the target. Furthermore, in our ablation study experiment focusing on punch control, it was also determined that the accuracy of the punches increases with the addition of more information relating to the trajectory of the wrist.

While improved scores across the evaluation metrics already show us that the trajectory variables are necessary for targeted punch control, this finding was reinforced further by visual inspection of the synthesised punches. This inspection in the ablation study experiment revealed that the model without any trajectory information predicts punches that cannot reach the target even though the synthesis moves the entire body of the character towards the target by a much larger extent than other models with trajectory variables. However, the foot skating error (FSE) metric computed for the model without trajectory variables is higher (see Table 6.2) as the synthesis results in the character floating towards the target when attempting to punch. While the model with the variables corresponding to only the trajectory positions performs more accurate punches than the one without any trajectory, we find that including the trajectory velocity variables further improves the punch accuracy while maintaining the realism of the synthesised motions. The foot skating error (FSE) for these models is also much lower as the models with

trajectory are focused on moving the wrist towards the target instead of the whole body. Hence, trajectory variables corresponding to the joint to be moved (towards a target) is required for neural-based targeted motion synthesis, as the model needs sufficient information to learn about the relationship between the joint and the target.

Including trajectory variables alone is not sufficient for better boxing motion synthesis. The motion data being represented by each point included in the trajectory should be representative of the motion being synthesised. Considering this, we trained several models on different versions of the motion states, which included all the trajectory variables from Section 4.2. In our trajectory sampling experiments, the time window covered by the trajectory and the possible distance between each sample in the window were analysed for both the punching and the stepping actions.

For determining the parameters to use for the trajectory sampling in the punch control experiment, we relied on the punch statistics as there are not many approaches in the motion synthesis literature focusing on targeted motion synthesis using neural models as explained in Chapter 2. The evaluation metrics computed for the trajectory sampling experiment (see Table 6.1) showed us that for the best targeted punch control using our MoE model, it requires a trajectory window covering 0.7s (with w=14 and fs=3 in Table 6.1) of both the past and the future motion. Our finding is significantly different from the trajectory window of 2s used by Starke et al. [57], which is a recently published approach for synthesising martial arts movements, including punching. Such a short trajectory window is suitable for our case as the punching actions in our boxing data are executed over a short duration of time.

Additionally, the trajectory sampling experiment for punch control also showed that the model with wrist trajectory covering 0.47s (with w=14 and fs=2 in Table 6.1) of motion produces punches that are reaching closer to the exact target point. However, the accuracy performance for this model is imbalanced w.r.t. left and right punches, with a significant bias towards the right hand, which shows that the trajectory information for the left and right hand might require different values for trajectory window w and frameskip fs. This might be due to the different velocities of the left and right punches in the underlying motion capture data. Thus, our result shows that tuning the trajectory sampling for each hand executing the punching action might lead to improved accuracy.

Similarly, we conducted a trajectory sampling experiment for synthesising the stepping action used to move from one point to another while boxing. Here, several trajectory sampling parameters were tested, including the parameters used by Holden et al. [21], which was the first approach to produce good interactive synthesis results for locomotion tasks using neural models. However, our MoE model was not able to produce stable synthesis with such trajectory samples, but trajectory sample windows that are more in line with the stepping action in the dataset led to good quality motion synthesis. We also found that the trajectory window that covered 0.33s of motion worked the best for stepping in both the forward and the backward directions. This trajectory duration coincides with the average step duration in our data (see table 3.2). Moreover, the trajectory sampling experiment was required to ensure better model performance in the backward stepping action as our mocap data contained a small amount of backward stepping. Despite this limitation in our data, our approach can produce good results in both directions when considering the right amount of trajectory information based on the dataset statistics of the stepping action.

Many of the directly relevant approaches in the interactive motion synthesis domain, like Holden et al. [21], Zhang et al. [63] and Starke et al. [55], utilise trajectory windows of 2s. However, while such a window might be suitable for locomotion-based motion synthesis, it might not hold up when considering different action sets like the one in this thesis. Therefore, as shown in this thesis, it is necessary to sample the trajectory variables in relation to the actions in the dataset being considered, and shorter windows are more beneficial for fast actions.

Post-prediction fixes in the boxing controller system

Along with the MoE model, a boxing controller system was implemented in this thesis which allows a user to interactively control the motions synthesised by our MoE model. Additionally, it is also responsible for tasks such as managing the targets to synthesise the back and forth movement of the hand in a punch (see Figure 5.3) and for terminating the punches that were missed (see Figure 5.4). Essentially, the controller system is an important unit for ensuring that our generative model is managed correctly to produce realistic motions and is applicable more when allowing the freedom of selecting a target that the synthesised motion must interact with.

Other than managing the punch target and terminating punches that miss the target, the controller can also intervene to rectify the predictions from the model. For example, our initial inspection of the synthesised motions indicated that the predicted trajectory points were not always precisely synchronised with the predicted joint positions, and this is referred to as **floating trajectory**. For instance, when considering the predicted trajectory wrist positions and the predicted position of the wrist from the variable containing predictions for all the joints, the expectation is to have the middle of the trajectory coincide exactly with the predicted wrist position. However, this was not the case with both the root and wrist trajectories, as the predicted trajectory points would be near the corresponding joint but would not coincide exactly. This trajectory floating issue was overcome using the model handling unit of the controller system by fixing the midpoint of the trajectory to the corresponding joint position.

In addition to this, we experienced an unexpected and peculiar effect in the synthesised boxing motions. Initially, the visual inspection of the synthesised motion revealed that our MoE model predicted punches even when the user did not specify any punch. This happened even when both the punch label (set to 0) *and* the punch target (set to $\vec{0}$) directed the model not to punch. This unexpected result is referred to as the **oscillation effect**, and it was the result of the network synthesising punches by continually predicting long, oscillating wrist trajectories for both the left and the right hands. Therefore, a post-prediction fix is needed in the model handling unit of the boxing controller system to prevent this issue. This fix is applied only when the character is not punching. In this fix, the input motion state is prepared with all the future trajectory points in the wrist trajectory being set to the current wrist position stored in the model handling unit of the controller system. Thus, the controller corrected input motion state curtails the model from producing the oscillation effect.

Both the trajectory floating and oscillation effect are trajectory-related issues that are continually passed on to the next input motion state by the controller. This controller feedback loop (see Figure 5.2) explains why these issues repeatedly occur during motion synthesis. However, the controller is just cycling through the predictions and feeding

them back to our MoE model, which implies that the predictions from the model are the source of these issues. Therefore, it is most likely that these issues are occurring due to the model not having fully learnt the relationship between the joint position and the trajectory position variables in the motion states. This inability of the model might be due to the small amount of boxing data used in the model training. However, the amount of training might not be an issue as our initial experiments also trained models further than 150 epochs and still came across these issues.

In addition to the above issues regarding punching, the model also faced issues synthesising the stepping action. The first issue is that the reverse stepping action was hard to synthesise due to the lack of sufficient data. Without any intervention from the controller system, this issue was solved by selecting suitable trajectory sampling parameters. In addition to the backward stepping issue, our MoE model has a significant **directional bias** towards the left or right directions while synthesising both the forward and the backward stepping actions. If the motion synthesis is continued for a longer duration, then the difference between the expected path and the path followed by the synthesis increases further, as shown in the Figure 6.7. This bias in the direction is a result of our boxing dataset containing a single actor. While it can probably be mitigated by reflecting the data, we chose not to do so as reflecting the data will add additional complexity for the model as it needs to distinguish between the orthodox and southpaw stances in boxing. However, this directional bias can be analysed and compensated for in the controller system, and the direction of the synthesised stepping motion can be rectified.

As can be seen from the above discussion, the MoE boxing model must be tuned and trained with motion states that are ensured to be representative of the actions in the dataset. We also show that the controller system can help in providing post-prediction fixes that can compensate for the errors and correct the biases in the predictions of our MoE boxing model.

As we are dealing with a neural generative model, our model has many biases and incorrect predictions, and additionally, the controller system adds another layer of complexity. While our approach is still capable of interactively synthesising boxing actions, it has certain shortcomings that are presented in the next section.

7.2 Limitations

Motion synthesis is, in general, a complicated task due to the high degrees of freedom in the movement of the joints in a human skeleton. Using a generative neural model to solve this task and allowing user input leads to complicated interactions between the developed units. While we can achieve the interactive synthesis of boxing actions by developing a controller system with such complicated interactions, our system and model have certain limitations that are discussed in this section.

In the previous section, we discussed about our MoE model being capable of producing boxing actions despite having been trained on a small duration of boxing data. While this is true, the final synthesis result is aided by the post-prediction fixes applied in the controller to overcome the floating trajectory and the oscillation effect. In addition, the controller can possibly be extended to overcome the directional bias in the stepping action as well. However, all these post-prediction fixes in the controller might not be

needed if we can ensure that the model can completely learn the relationship between the trajectory of a joint and the corresponding joint information in the variable representing the current frame information of all joints. This limitation in learning is most likely due to the model having insufficient data to learn this relationship, and it can be solved by considering more boxing data. However, a systematic study is also required in the motion synthesis literature to determine the amount of data to consider for neural-based approaches, as we show that boxing actions can be generated with 10mins of mocap data in comparison to 20hours used by Starke et al. [57].

Data acquisition issues

While increasing the amount of data might solve the issues described above, there are other issues in our data acquisition method presented in Section 3.1. With the motivation of capturing realistic boxing motions, the mocap actor was allowed to mostly shadow box freely during each capturing trial. Although instructions were given before our capturing process, they were insufficient and led to three clear issues in our boxing data.

First, the dataset punch targets shown in Figure 3.4 are mostly clustered together in the centre of the cube that demarcates the actual reach of the boxing actor. Training our MoE model on such target punches causes it to hyperfocus on the dataset target region. Therefore, Tables 6.1 and 6.2 show low punch accuracy (PA) scores for all our models that were tested using the uniform grid of test targets (see Figure 6.4) that are in the exact same space demarcated by the dataset target cube. Hence, the synthesis of targeted actions like punching can be improved by setting up a better capturing procedure that ensures the collection of motion corresponding to the entire region that is in the punch range of the actor.

Second, the boxing data collected for different stepping directions is imbalanced. There was no requirement placed on the mocap actor to ensure that the duration of forward stepping and backward stepping is equal, and this resulted in capturing imbalanced stepping data. Moreover, the backward stepping action is critical in real boxing to escape from the opponent's reach. As this was not explicitly considered for our capturing process, the resulting data is imbalanced. Thus, our MoE model synthesises forward stepping action better than the backward stepping action. This imbalance needs to be rectified, especially considering the significance of backward stepping in boxing. Including simple instructions for the actor to step forward and backward till the bounds of the capturing area will improve the boxing data. For example, the actor can be instructed to step forward from one edge of the capturing area to the opposite one and return to the starting point while employing the backward stepping action. However, the naturalness of backward stepping actions executed over long durations might be an issue as it is not common in boxing, but it is required for gathering good trajectory information for our MoE model. So backward stepping is, in general, a complicated action to consider.

Third, the boxing data is imbalanced in the extent of rotation allowed during stepping in both the forward and backward directions. This data influences the model to synthesise motions with a high degree of bias towards either the left or the right directions when synthesising the stepping action. To overcome this issue, the amount of stepping data must be increased so that the network can learn to synthesise stepping without direction bias. However, capturing is an expensive and time-consuming process. Therefore, a

simpler approach would be to ensure that the data has a good amount of stepping actions in at least each 45° rotation steps, and the capturing process can be improved by including specific markers on the capturing floor indicating the eight directions obtained from 45° rotation steps.

In addition to issues stemming from the instructions provided during our mocap session, the voluntary participant utilised in the data capture process only has a background in karate and is a beginner kickboxer. Thus, the motions recorded in this thesis are not necessarily showing the same skills and styles as a professional boxer.

For ensuring more natural boxing actions, we would ideally like to employ two professional boxers simultaneously for the mocap process, allowing each actor to fight in their distinct boxing style. However, this is not feasible with the IMU sensors due to the high risk of sensor destruction when the actors come into physical contact with each other while executing actions with high momentum. Thus, utilising optical sensors might be better for boxing, but any close interaction between the actors that lead to occlusions will cause problems for optical sensors. Hence, a combination of better capturing instructions, different capturing techniques and different boxing scenarios such as boxing with an opponent, boxing on a punching bag and shadow boxing would lead to better boxing data.

Data annotation issues

As explained in Section 3.3, the punch labels obtained from our annotation scheme result in a large number of punches that were labelled as being executed over 50 frames (see Figure 3.3). Upon visual inspection, it was revealed that these long-duration punches corresponded to the actor throwing punches with less momentum, mainly at the beginning of the capturing trials, and when he was fatigued. While this is an issue related to the data capture process, it is hard to control it during the capturing process, and such long duration punches will inevitably occur. Thus, when the annotation reveals such long duration punches, they should be dropped from the data as they are outlier data points. However, we decided to include them for training our models as our overall amount of boxing data was short. Therefore, it is better to ensure that a large amount of boxing data is captured even when the goal is to limit the amount of data used for training as there might be outliers in the data that need to be dropped.

Additionally, the automatic punch annotation scheme presented in Algorithms 1 and 2 was able to produce labels that enabled our models to produce sufficiently good synthesis results. While this is true, it might still be better to use manual annotation as it will be capable of handling edge-case punches in a better manner. Here, the edge case punches refer to the long duration punches. Our annotation scheme can be used as an initial step for obtaining labels that can be analysed, and if there are too many outliers, manual annotation can be utilised to verify the automatic labels. This would be a better approach as it will reduce the burden on the annotators and still ensure that high-quality labels are obtained for model training.

Issues related to the controller system

Currently, our mocap data contains rotations as the mocap actor was allowed to rotate upon reaching the bounds of the capturing area. Since our models are trained on this data, it is theoretically possible to synthesise rotation motions. Some experiments were conducted in this direction, but we were unable to obtain good synthesis results, primarily due to the time constraints of the thesis and limited amount of training data. However, with additional time and effort, the motion states for training our MoE model can be updated with variables containing rotations. Then, the associated controller system can be extended to allow for such rotations using the MoE model presented in this thesis. For such an extension, the controller system must handle user input rotation and update the future trajectory variables in the model handling unit.

Similarly, we experimented on adding a slight offset to the punch targets so that the punch goes slightly through the target and the wrist reaches closer to it. However, due to the time limitation of this thesis, we could not determine the best offsets and the direction to punch through. This feature can also be added to the controller, and it will possibly increase the punch accuracy. To add this feature, an experimenter must manually determine what the best offsets would be. However, an alternative approach would be to pull the dataset targets back by a small fixed amount and use these updated targets for training the models. The small fixed amount can be treated as the offset when using the model inside the controller system in interactive mode. In addition, we utilised the wrist positions in our motions states (see Section 4.2), which are not representative of the final end position of the hand of the mocap actor. Thus, using a better hand model could help in increasing the accuracy of the synthesised punches.

Lastly, even though our punch control synthesises realistic punches with a frameskip of two or higher in the trajectory window, the generated motions are not as good as in the case of no frameskip. This is primarily because the predicted trajectory is not as smooth as in the case of no frameskip. The controller system can probably assist here with a post-prediction fix to smooth the trajectory. An alternative would be to give more preference to the user input when merging it with the predicted trajectory in the model handling unit of the controller system. Therefore, it might be possible to improve the controller further and produce better punches when the frameskips are higher.

Based on the limitations of our approach and system presented in this section, we present the possibilities to extend our work in the next section.

7.3 Future work

The interactive boxing synthesis approach presented in this thesis shows that it is possible to achieve realistic and responsive results when using only 10mins of mocap data. However, there are certain limitations of our system stemming from the short duration of data. The latest approach from Starke et al. [57] uses 20 hours of mocap data to synthesise diverse mixed martial arts movements. To the best of our knowledge, there is currently no research work to systematically determine the amount of data required for motion synthesis. Our approach and evaluation method can be utilised to perform a study for determining the amount of boxing mocap data required for best synthesis results.

Our approach can be improved by extending the motion state representations by considering the idea of ego-centric and object-centric viewpoints from [55]. This will enable our model to synthesise realistic movements around the ropes forming the bounds of a boxing ring. To achieve this, the data capture process must be updated with markers on the capturing floor indicating the bounds of the boxing ring.

Starke et al. [56] have shown that it is possible to set up neural-based interactive synthesis for the scenario of one-on-one basketball games. Similarly, boxing is also a one-on-one sport, and our synthesis system can be adapted to achieve realistic boxing interactions with an opponent character. This can be achieved by considering the idea of opponent motion state information from [56] so that our MoE model can synthesise realistic actions in relation to an opponent character.

The above opponent motion state information can be adapted to synthesise boxing motions such as blocking and dodging using our MoE model. Since blocking is also a targeted action, it can probably be synthesised by including the opponent wrist trajectory and punch target in the motion state for the player character. Dodging can also be adapted similarly, but adapting both of these actions will require additional mocap data containing these actions.

In addition to synthesising additional boxing actions, our method can also be adapted to other targeted actions such as sword motion considered by Freda [14] and kicking considered by Starke et al. [57]. Additionally, the sword motion synthesis approach by Freda [14] can directly benefit from the findings of the ablation study experiment for punch control conducted in this thesis as the sword motion can also be improved by including the trajectories of the wrist and the sword in the motion state representations.

Currently, the visualisation unit works only on positional information of the joints. However, when we consider the different types of punches in boxing, the rotation of the wrist joints may be necessary. For example, the palm faces the ground while performing a jab and faces the sky while performing an uppercut. Therefore, we believe that working with rotations would benefit the naturalness of the motion synthesis further. Furthermore, there is already evidence from [63] that including joint rotations in the motion state representation improves the results even though they still use positions for visualisation. Based on this evidence, we hypothesise that a neural model designed to work on rotations can synthesise better and more realistic boxing motions.

Our data-driven approach ensures that the synthesised motion is similar to the underlying mocap data, but it does not ensure that the motions can respond to the laws of physics in the virtual environment. Therefore, the use of a simulation-based approach in conjunction with our MoE model would probably result in better motion synthesis. Bergamin et al. [5] already show that such a joint approach is possible as they use the data-driven approach of motion matching in conjunction with DRL to produce good simulation-based synthesis results. Therefore, it should be possible to extend our system in a similar manner.

As presented in this section, there are many ways in which this thesis can be extended to improve the synthesis of the current actions considered. Furthermore, it is also possible to extend our approach to include other actions from martial arts and improve the synthesis results by considering the laws of physics. Finally, in the next section, we conclude this thesis by summarising the most important findings.

7.4 Conclusion

Fast boxing actions such as punching and stepping carry a style inherent to the sport of boxing and, thus, are significantly different from the periodic motions involved in locomotion tasks. In this thesis, the approach designed using mixture of experts (MoE) neural models is shown to be capable of synthesising realistic and responsive boxing actions even when trained on relatively small amounts of mocap data.

To achieve such synthesis, a novel boxing dataset is prepared and converted into effective motion states designed particularly for stepping and punching. We experimentally determine that for sharper motion synthesis using our MoE model, it must be tuned well in relation to the boxing data. For achieving more accurate targeted punches: 1) the duration of past and future motion covered in the wrist trajectory variables of the motion states must be short, and 2) the motion states must include sufficient trajectory variables like position and velocity. In the case of the stepping action, our experiments show that effectively selecting the duration of past and future motion covered in the root trajectory variables results in responsive and realistic synthesis, even when there is an imbalance in the underlying forward and backward stepping actions in the mocap data. Essentially, for the high-quality synthesis of both the punching and the stepping actions, the trajectory duration must match the short duration of the fast actions in our boxing dataset.

Additionally, this thesis also shows that a sound boxing controller system is required for the punch target management that ensures the back and forth motion of the hand in a punch and for handling missed punches. The controller can also assist in overcoming model prediction issues such as floating trajectories, oscillation effect and possibly compensate for the learned directional biases while stepping.

Therefore, this thesis shows that a well-tuned MoE model trained on effective representations of the motion data can be incorporated in a well-designed controller system to provide realistic and responsive interactive boxing motion synthesis. Our approach and findings are very much relevant in the video game industry, and particularly, the neural-based targeted punching action implemented in this thesis can be used in video games to provide users more control, resulting in increased immersion in the game environment. Finally, our approach can be adapted further to interactively synthesise other targeted actions such as kicking and sword slashing.

Abbreviations

DL deep learning 2

DNN deep neural network 9

DRL deep reinforcement learning 9

ERD encoder recurrent decoder 13

FSE foot skating error 62

IMU inertial measurement unit 20

MANN mode adaptive neural network 4

ML machine learning 2

mocap motion capture 2

MoE mixture of experts 16

MoFi motion fields 1

MSE mean squared error 61

PA punch accuracy 60

PE punch error 61

PFNN phase function neural network 4

RL reinforcement learning 8

RNN recurrent neural network 9

SPE stepping path error 61

List of Figures

List of Tables

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016. arXiv:1603.04467. Retrieved from `https://arxiv.org/abs/1603.04467`.

[2] Kfir Aberman, Yijia Weng, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. Unpaired Motion Style Transfer from Video to Animation. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392469. URL `https://doi.org/10.1145/3386569.3392469`.

[3] André Antakli., Torsten Spieldenner., Dmitri Rubinstein., Daniel Spieldenner., Erik Herrmann., Janis Sprenger., and Ingo Zinnikus. Agent-based Web Supported Simulation of Human-robot Collaboration. In *Proceedings of the 15th International Conference on Web Information Systems and Technologies - WEBIST*, pages 88–99. INSTICC, SciTePress, 2019. ISBN 978-989-758-386-5. doi: 10.5220/0008163000880099. URL `https://www.researchgate.net/publication/336250239_Agent-based_Web_Supported_Simulation_of_Human-robot_Collaboration`.

[4] Tobias Baur, Ionut Damian, Patrick Gebhard, Kaska Porayska-Pomsta, and Elisabeth André. A Job Interview Simulation: Social Cue-Based Interaction with a Virtual Character. In *2013 International Conference on Social Computing*, pages 220–227, USA, 2013. IEEE Computer Society. ISBN 9780769551371. doi: 10.1109/SocialCom.2013.39. URL `https://ieeexplore.ieee.org/document/6693336`.

[5] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356536. URL `https://doi.org/10.1145/3355089.3356536`.

[6] Xsens Technologies B.V. MTw Awinda. Website of Xsens Technologies B.V., 2021. Retrieved November 22, 2021 from `https://www.xsens.com/products/mtw-awinda`, Last accessed November 22, 2021.

[7] Xsens Technologies B.V. *MVN Animate*. Xsens Technologies B.V., P.O. Box 559, 7500, Enschede, Netherlands, 2021. Retrieved November 22, 2021 from `https://www.xsens.com/products/mvn-animate`, Last accessed November 22, 2021.

[8] Andrea Bönsch, Jonathan Wendt, Heiko Overath, Özgür Gürerk, Christine Harbring, Christian Grund, Thomas Kittsteiner, and Torsten W. Kuhlen. Peers at work: Economic real-effort experiments in the presence of virtual co-workers. In *2017 IEEE Virtual Reality (VR)*, pages 301–302, 2017. doi: 10.1109/VR.2017.7892296. URL https://ieeexplore.ieee.org/document/7892296.

[9] Xiaobin Chang, Timothy M. Hospedales, and Tao Xiang. Multi-Level Factorisation Net for Person Re-Identification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, abs/1803.09132, 2018. URL http://arxiv.org/abs/1803.09132.

[10] Simon Clavet. Motion Matching and The Road to Next-Gen Animation. *In proc. of GDC 2016*, 2016. URL https://www.gdcvault.com/play/1023280/Motion-Matching-and-The-Road.

[11] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. Retrieved October 20, 2020 from http://www.blender.org, Last accessed November 22, 2021.

[12] Ubisoft Entertainment. *For Honor - Available now on PS4, Xbox One & PC | Ubisoft*. Ubisoft Entertainment, 625 Third Street, San Francisco, CA 94107, USA, 2021. Retrieved November 22, 2021 from https://www.ubisoft.com/en-gb/game/for-honor, Last accessed November 23, 2021.

[13] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent Network Models for Human Dynamics. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4346–4354, Los Alamitos, CA, USA, dec 2015. IEEE Computer Society. doi: 10.1109/ICCV.2015.494. URL https://doi.ieeecomputersociety.org/10.1109/ICCV.2015.494.

[14] Claudio Freda. Neural Network Based Motion Synthesis for Close Combat in Games. Master's thesis, Utrecht University, 2018. URL http://dspace.library.uu.nl/handle/1874/369857. Retrieved from Utrecht University Repository, Last accessed November 22, 2021.

[15] Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust Motion In-Betweening. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392480. URL https://doi.org/10.1145/3386569.3392480.

[16] Nicolas Manfred Otto Heess, TB Dhruva, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyun Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of Locomotion Behaviours in Rich Environments. *arXiv*, abs/1707.02286, 2017. arXiv preprint, Retrieved from https://arxiv.org/abs/1707.02286.

[17] Erik Herrmann, Han Du, Martin Manns, and Markus Mauer. *Data structures and utilities for skeleton animations.* German Research Center for Artificial Intelligence, D3 1, Campus D3 2, 66123 Saarbrücken, Germany. Retrieved June 29, 2020 from https://github.com/eherr/anim_utils, Last accessed November 22, 2021.

[18] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning Motion Manifolds with Convolutional Autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, SA '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450339308. doi: 10.1145/2820903.2820918. URL https://doi.org/10.1145/2820903.2820918.

[19] Daniel Holden, Jun Saito, and Taku Komura. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.*, 35(4), July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925975. URL https://doi.org/10.1145/2897824.2925975.

[20] Daniel Holden, Ikhsanul Habibie, Ikuo Kusajima, and Taku Komura. Fast Neural Style Transfer for Motion Data. *IEEE Comput. Graph. Appl.*, 37(4):42–49, January 2017. ISSN 0272-1716. doi: 10.1109/MCG.2017.3271464. URL https://doi.org/10.1109/MCG.2017.3271464.

[21] Daniel Holden, Taku Komura, and Jun Saito. Phase-Functioned Neural Networks for Character Control. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073663. URL https://doi.org/10.1145/3072959.3073663.

[22] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. Learned Motion Matching. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392440. URL https://doi.org/10.1145/3386569.3392440.

[23] Leslie Ikemoto, Okan Arikan, and David Forsyth. Generalizing Motion Edits with Gaussian Processes. *ACM Trans. Graph.*, 28(1), February 2009. ISSN 0730-0301. doi: 10.1145/1477926.1477927. URL https://doi.org/10.1145/1477926.1477927.

[24] Epic Games Inc. *The most powerful real-time 3D creation tool - Unreal Engine*. Epic Games, Inc., 620 Crossroads Blvd., Cary, NC 27518, USA, . Retrieved October 25, 2021 from https://www.unrealengine.com/en-US/, Last accessed November 22, 2021.

[25] Rockstar Games Inc. *Grand Theft Auto V*. Take-Two Interactive Games 622 Broadway New York, NY 10012, USA, . Retrieved October 25, 2021 from https://www.rockstargames.com/V/, Last accessed November 22, 2021.

[26] ISO/TR 7250-2:2010. Basic human body measurements for technological design — Part 2: Statistical summaries of body measurements from national populations. Standard, International Organization for Standardization, Geneva, CH, 2010. URL https://www.iso.org/standard/41249.html.

[27] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 03 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.1.79. URL https://doi.org/10.1162/neco.1991.3.1.79.

[28] Yifeng Jiang, Tom Van Wouwe, Friedl De Groote, and C. Karen Liu. Synthesis of Biologically Realistic Human Motion Using Joint Torque Actuation. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322966. URL https://doi.org/10.1145/3306346.3322966.

[29] Ioannis A Kakadiaris. Physics-Based Modeling, Analysis and Animation. *Technical Reports (CIS)*, page 274, 1993. URL https://repository.upenn.edu/cis_reports/274/.

[30] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion Graphs. *ACM Trans. Graph.*, 21(3):473–482, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566605. URL https://doi.org/10.1145/566654.566605.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[32] CMU Graphics Lab. Carnegie Mellon University - CMU Graphics Lab - motion capture library. Webpage of Carnegie Mellon University. Retrieved November 22, 2021 from http://mocap.cs.cmu.edu/, Last accessed November 22, 2021.

[33] Kyungho Lee, Seyoung Lee, and Jehee Lee. Interactive Character Animation by Learning Multi-Objective Control. *ACM Trans. Graph.*, 37(6), December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275071. URL https://doi.org/10.1145/3272127.3275071.

[34] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable Muscle-Actuated Human Simulation and Control. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322972. URL https://doi.org/10.1145/3306346.3322972.

[35] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. Motion Fields for Interactive Character Locomotion. *ACM Trans. Graph.*, 29(6), dec 2010. ISSN 0730-0301. doi: 10.1145/1882261.1866160. URL https://doi.org/10.1145/1882261.1866160.

[36] Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel Van De Panne. Character Controllers Using Motion VAEs. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392422. URL https://doi.org/10.1145/3386569.3392422.

[37] C. Karen Liu and Zoran Popović. Synthesis of Complex Dynamic Character Motion from Simple Animations. *ACM Trans. Graph.*, 21(3):408–416, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566596. URL https://doi.org/10.1145/566654.566596.

[38] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. arXiv preprint, Retrieved from https://arxiv.org/abs/1711.05101.

[39] Christian Mandery, Ömer Terlemez, Martin Do, Nikolaus Vahrenkamp, and Tamim Asfour. Unifying Representations and Large-Scale Whole-Body Motion Databases for Studying Human Motion. *IEEE Transactions on Robotics*, 32(4):796–809, 2016. URL https://ieeexplore.ieee.org/document/7506114.

[40] Matteo Menolotto, Sokratis Komaris, Salvatore Tedesco, Brendan O'Flynn, and Michael Walsh. Motion Capture Technology in Industrial Applications: A Systematic Review. *Sensors*, 20:5687, 10 2020. doi: 10.3390/s20195687. URL https://www.mdpi.com/1424-8220/20/19/5687.

[41] Maddock Meredith, Steve Maddock, et al. Motion capture file formats explained. *Department of Computer Science, University of Sheffield*, 211:241–244, 2001. URL https://staffwww.dcs.shef.ac.uk/people/S.Maddock/publications/MeredithMaddock2001_CS0111.pdf.

[42] Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. Learning Predict-and-Simulate Policies from Unorganized Human Motion Data. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356501. URL https://doi.org/10.1145/3355089.3356501.

[43] Monique Paulich, Martin Schepers, Nina Rudigkeit, and G. Bellusci. Xsens MTw Awinda: Miniature Wireless Inertial-Magnetic Motion Tracker for Highly Accurate 3D Kinematic Applications. 05 2018. doi: 10.13140/RG.2.2.23576.49929. White paper retrieved from https://www.xsens.com/hubfs/3446270/Downloads/Manuals/MTwAwinda_WhitePaper.pdf.

[44] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.*, 36(4):41:1–41:13, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073602. URL http://doi.acm.org/10.1145/3072959.3073602.

[45] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311. URL https://doi.org/10.1145/3197517.3201311.

[46] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *J. Mach. Learn. Res.*, 20(1): 1934–1965, jan 2019. ISSN 1532-4435. URL https://www.jmlr.org/papers/volume20/18-444/18-444.pdf.

[47] Armin Ronacher. Welcome to Flask — Flask Documentation (2.0.x). Webpage of The Pallets Projects, 2017. Retrieved February 2, 2021 from https://flask.palletsprojects.com/en/2.0.x/, Last accessed November 22, 2021.

[48] Steve Rotenberg. Lecture notes of CSE169, Computer Animation: Skeletons. Webpage. Delieverd on Septembre 01, 2018, https://cseweb.ucsd.edu/classes/wi20/cse169-a/readings/2-Skeleton.html, Last accessed November 12, 2021.

[49] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y. URL https://doi.org/10.1007/s11263-015-0816-y.

[50] Martin Schepers, Matteo Giuberti, and G. Bellusci. Xsens MVN: Consistent Tracking of Human Motion Using Inertial Sensing. 03 2018. doi: 10.13140/RG.2.2.22099.07205. White paper retrieved from https://www.researchgate.net/publication/324007368_Xsens_MVN_Consistent_Tracking_of_Human_Motion_Using_Inertial_Sensing.

[51] Yeongho Seol, Carol O'Sullivan, and Jehee Lee. Creature Features: Online Motion Puppetry for Non-Human Characters. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, page 213–221, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321327. doi: 10.1145/2485895.2485903. URL https://doi.org/10.1145/2485895.2485903.

[52] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

[53] Janis Sprenger, Han Du, Noshaba Cheema, Erik Herrmann, Klaus Fischer, and Philipp Slusallek. Learning a Continuous Control of Motion Style from Natural Examples. In *Motion, Interaction and Games*, MIG '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369947. doi: 10.1145/3359566.3360082. URL https://doi.org/10.1145/3359566.3360082.

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[55] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural State Machine for Character-Scene Interactions. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356505. URL https://doi.org/10.1145/3355089.3356505.

[56] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Trans. Graph.*, 39(4), jul 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392450. URL https://doi.org/10.1145/3386569.3392450.

[57] Sebastian Starke, Yiwei Zhao, Fabio Zinno, and Taku Komura. Neural Animation Layering for Synthesizing Martial Arts Movements. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459881. URL https://doi.org/10.1145/3450626.3459881.

[58] Pixar Animation Studios. Pixar's Animation Process. Webpage. Retrieved Novemeber 09, 2021 from http://www.animationmeat.com/pdf/featureanimation/pixarsanimationprocess.pdf, Last accessed November 22, 2021.

[59] Unity Technologies. *Unity Real-Time Development Platform | 3D, 2D VR I& AR Engine*. Unity Technologies, 30 Third Street San Francisco, CA 94103 USA. Retrieved February 2, 2021 from https://unity.com/, Last accessed November 22, 2021.

[60] Nikolaus F. Troje. Decomposing biological motion: A framework for analysis and synthesis of human gait patterns. *Journal of Vision*, 2(5):2–2, 09 2002. ISSN 1534-7362. doi: 10.1167/2.5.2. URL https://doi.org/10.1167/2.5.2.

[61] Xin Wang, Qiudi Chen, and Wanliang Wang. 3D Human Motion Editing and Synthesis: A Survey. *Computational and mathematical methods in medicine*, 2014: 104535, 06 2014. doi: 10.1155/2014/104535. URL https://www.hindawi.com/journals/cmmm/2014/104535/.

[62] Wenhao Yu, Greg Turk, and C. Karen Liu. Learning Symmetric and Low-Energy Locomotion. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201397. URL https://doi.org/10.1145/3197517.3201397.

[63] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. Graph.*, 37(4), jul 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201366. URL https://doi.org/10.1145/3197517.3201366.

[64] Yi Zhou, Zimo Li, Shuangjiu Xiao, Chong He, Zeng Huang, and Hao Li. Auto-Conditioned Recurrent Networks for Extended Complex Human Motion Synthesis. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r11Q2SlRW.