DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2022

A study of transfer learning on data-driven motion synthesis frameworks

Nuo Chen

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

Various research has shown the potential and robustness of deep learning-based approaches to synthesise novel motions of 3D characters in virtual environments, such as video games and films. The models are trained with the motion data that is bound to the respective character skeleton (rig). It inflicts a limitation on the scalability and the applicability of the models since they can only learn motions from one particular rig (domain) and produce motions in that domain only. Transfer learning techniques can be used to overcome this issue and allow the models to better adapt to other domains with limited data.

This work presents a study of three transfer learning techniques for the proposed Objective-driven motion generation model (OMG), which is a model for procedurally generating animations conditioned on positional and rotational objectives. Three transfer learning approaches for achieving rig-agnostic encoding (RAE) are proposed and experimented with: **Feature encoding (FE)**, **Feature clustering (FC)** and **Feature selection (FS)**, to improve the learning of the model on new domains with limited data.

All three approaches demonstrate significant improvement in both the performance and the visual quality of the generated animations, when compared to the vanilla performance. The empirical results indicate that the FE and the FC approaches yield better transferring quality than the FS approach. It is inconclusive which of them performs better, but the FE approach is more computationally efficient, which makes it the more favourable choice for real-time applications.

Keywords

Transfer learning, data-driven motion synthesis, objective-driven motion generation, rig-agnostic encoding, deep learning-based clustering model, procedural animation

Abstrakt

En studie av kunskapsöverföring på data-driven rörelse syntetiseringsramverk

Många studier har visat potentialen och robustheten av djupinlärningbaserade modeller för syntetisering av nya rörelse för 3D karaktärer i virtuell miljö, som datorspel och filmer.

Modellerna är tränade med rörelse data som är bunden till de respektive karaktärskeletten (rig). Det begränsar skalbarheten och tillämpningsmöjligheten av modellerna, eftersom de bara kan lära sig av data från en specifik rig (domän) och därmed bara kan generera animationer i den domänen. Kunskapsöverföringsteknik (transfer learning techniques) kan användas för att överkomma denna begränsning och underlättar anpassningen av modeller på nya domäner med begränsade data.

I denna avhandling presenteras en studie av tre kunskapsöverföringsmetoder för den föreslagna måldriven animationgenereringsnätverk (OMG), som är ett neural nätverk-baserad modell för att procedurellt generera animationer baserade på positionsmål och rotationsmål. Tre metoder för att uppnå rig-agnostisk kodning är presenterade och experimenterade: **Feature encoding (FE)**, **Feature clustering (FC)** and **Feature selection (FS)**, för att förbättra modellens lärande på nya domäner med begränsade data.

All tre metoderna visar signifikant förbättring på både prestandan och den visuella kvaliteten av de skapade animationerna, i jämförelse med den vanilla prestandan. De empiriska resultaten indikerar att både FE och FC metoderna ger bättre överföringskvalitet än FS metoden. Det går inte att avgöra vilken av de presterar bättre, men FE metoden är mer beräkningseffektiv, vilket är fördelaktigt för real-time applikationer.

Nyckelord

Kunskapsöverföring, data-driven rörelsesyntetisering, procedurell animation, mål-driven animation-genereringsmodel, rig-agnostisk-kodning, djupinlärningsbaserad klusteringsmodel

Acknowledgements

I would like to express my sincere gratitude to my supervisors Oskar Blom, Johan Tunkrans, Kristoffer Jonsson and many others at EA Digital Illusions CE AB (DICE), for their strong support and valuable advice throughout the course of this project. I also want to thank my supervisor and examiner Christopher Peters and Pawel Herman for their greatest assistance that improved my research and writing. At last, I want to thank Taqui Syed, Viktor Vitek and Viktor Meyer for their insightful and constructive feedback.

Authors

Nuo Chen
nuoc@kth.se
Stockholm, Sweden
Computer Science with specialisation in Data Science
KTH Royal Institute of Technology

Examiner

Pawel Herman
Stockholm, Sweden
KTH Royal Institute of Technology

Supervisor

Christopher Peters
Stockholm, Sweden
KTH Royal Institute of Technology

Contents

List of Figures

List of Tables

List of abbreviations

AE Autoencoder

C-model Clustering layer (RBF / VAE / DEC)

DEC Deep embedding clustering network

Dec Decoder

Enc Encoder

FC-OMG An OMG model with adopts the FC approach

FC Feature clustering

FE-OMG An OMG model that adopts the FE approach

FE Feature encoding

FS-OMG An OMG model with adopts the FS approach

FS Feature selection

LSTM Long short-term memory network

MLP Multi-layer perceptron network

MoE Mixture-of-experts network

MoGenNet Motion generation network (MoE, LSTM)

OMG Objective-driven motion generation network

RBF Radial-basis-function network

VAE Variational autoencoder

Chapter 1

Introduction

Animation is essentially moving pictures that are excellent at describing motions or phenomenon that changes over time. Computer animation refers to the computer-generated animation that brings the virtual characters to life in video games, cartoons and films. A character in this context refers to any animatable entity.

The traditional type of computer animation is the **interpolation-based animation** [19], which uses predefined key-frames to define the different states of the animation and let the computer generate the transitions between them. Hand-crafting the key-frames has been the conventional method for creating such animations, but the production is time-consuming and not very scalable.

The advancement in computer technologies has made **real-time animations** possible at a reasonable cost [20]. It allows the animators to procedurally animate the characters that react to the user inputs and interact with the environment, resulting in dynamic, natural-looking and environment-aware motions. **Deep learning-based motion synthesis systems** are capable of learning from large and high-dimensional motion datasets, producing high quality and complex animations while having fast execution time and low memory footprints [26]. Figure 1.1 illustrates the architecture of a deep learning-based motion synthesis framework by Starke et al. [26].

As with every other data-driven approach, these models require a large motion dataset for training, and the data is mostly motion captured animations which are expensive and time-consuming to produce. The interface of the models is dependent on the configuration of the character skeleton (rig), on which the motion data is recorded. Thus, the models can only produce animations for that particular rig.

Transfer learning techniques in the form of **rig agnostic encoding (RAE)** methods are a possible solution to overcome this issue and enable the models to learn and generate motions for new rigs with limited motion data. This work investigates the possibility of

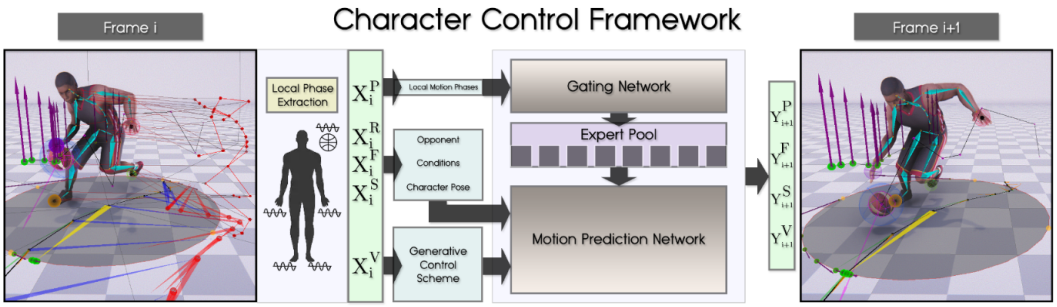


Figure 1.1: The architecture of the MoE-based motion synthesis framework by Starke et al. [26]

three RAE approaches for the proposed **Objective-driven motion generation model (OMG)**, to gain insights into how transfer learning can be applied to motion synthesis frameworks for improving the learning with limited data.

1.1 Problem

The main problem is that the trained motion synthesis framework such as the Mixture-of-Experts (MoE) based framework by Starke et al. [26] and the Variational autoencoder-based framework by Zinno et al. [15], cannot be applied to new character rigs (domain) due to the different configurations.

New instances must be trained from the ground which requires large datasets to achieve comparable performance. Acquiring large datasets is generally expensive and should be avoided. Transfer learning is a concept in machine learning about transferring the knowledge in a trained model to another model in another domain. It should enable the trained synthesis frameworks to be reused for new rigs to some extent and improve the learning.

There is little research in transfer learning of motion synthesis frameworks, hence the main research problem is concerned with investigating the suitability and performance of different transfer learning techniques within motion synthesis. This work focuses on three transfer learning techniques: feature encoding (FE), feature clustering (FC) and feature selection (FS). This choice is motivated in Section 2.2.5. The transfer learning techniques are implemented and experimented with the proposed OMG model.

> **Main research question.** Which transfer learning implementation of OMG achieves the highest generation performance and transferring quality on new domains with limited training and data?

The transferring quality refers to the generation performance improvement of the transferred models, and the generation performance is evaluated by the reconstruction error and the other metrics described in Section 3.5.1.

The following sub-questions investigate several neural network candidates for constructing OMG. These networks are chosen based on the previous work in data-driven motion synthesis as described in Section 2.4.

> **RQ.1** Which of the two neural networks: Mixture-of-experts (MoE) and Long short-term memory (LSTM) is better suited for OMG, in terms of generation performance and complexity?
>
> **RQ.2** For the FC approach, which of the three neural networks: Radial-basis-function (RBF), Variational autoencoder (VAE) and Deep embedding clustering (DEC) achieves the highest transferring quality and generation performance?

RQ.1 investigates two model candidates for the motion generation module in OMG, where both are fundamentally different neural networks and heavily researched in this field [9, 26]. RQ.2 investigates three model candidates for the rig-agnostic encoding part in OMG to implement the feature clustering approach. These models are particularly interesting because they explore the different ways of clustering data points, as explained in Sections 2.3.3,2.3.5,2.3.4.

1.2 Delimitations

This work only focuses on bipedal, humanoid character rigs, though the results should be applicable to all other characters rigs such as quadruped regardless of the configuration.

The motion data that is used for training and testing is from the same category, where the character reaches both hands to the targets. Diverse motion types are desirable for simulating real-world scenarios, but due to the limited time and computing power, this work only focuses on transfer learning over different domains (rigs) with the same fixed task (motion type).

The experiments only measure the offline performance of OMG models, because it is easier and more controllable. The goal application area is the runtime environment, thus testing the online performance of OMG models with the RAE approaches is necessary for future work.

1.3 Outline

- **Chapter 2** presents the important background information about computer animation, deep learning and data-driven motion synthesis frameworks.
- **Chapter 3** presents the research process, the design of OMG and the RAE approaches. It provides a detailed description of the implementation, the experiments, the evaluation method and the model specifications.
- **Chapter 4** presents the obtained results from the conducted experiments, including generation performance of the models and the transferring quality of the RAE approaches.
- **Chapter 5** discusses the research and the results with regards to the research questions, transferability, ethical aspects, social impacts, limitations and future work.
- **Chapter 6** presents the conclusion of this work.

Chapter 2

Background

In this chapter, the relevant information for understanding computer animation, transfer learning, deep learning-based motion synthesising frameworks are presented. Section 2.1 provides a brief introduction to the different types of computer animation. Section 2.2 gives an overview of deep learning. Section 2.4 presents the previous work on data-driven motion synthesis frameworks.

2.1 Computer animation

Technology drives innovation. With the advancement of computer technologies and digital tools, new possibilities for making animations have been introduced. Three-dimensional (3D) animations started to appear at SIGGRAPH, and the pursue of realistic 3D animations started to gain popularity among the game developers and the filmmakers such as Lucasfilm (Pixar) [19]. The digitally modelled 3D characters can be animated by rotating the bones manually, or using projections on live actors or using physical rules and algorithms.

2.1.1 Interpolation-based animation

Interpolation-based animation is the most common type of computer animation [19], where the animator specifies a set of key poses (keyframes) that altogether define the motion. The character is animated by interpolating between the keyframes. Various interpolation techniques can be applied to create unique and stylish animations.

Many of the early computer animation systems were keyframe systems, that provide the animators with total control over the motion and the expected appearance. Handcrafting the keyframes required immense manual labour and knowledge to create

natural and realistic animations. The generated clips in this work are sequences of keyframes. The clips are played by interpolating between the keyframes.

2.1.2 Kinematic linkage

In computer animation, a character rig is an interactive interface for controlling the character skeleton, which the animators can manipulate to animate the character. It consists of a hierarchy of objects that represent the components of the rig, usually in the form of joints or bones. Kinematic linkage refers to dependencies between the objects. It describes the relation of the objects to their parents, which simulates the rigid connection between the biological joints in a human. An example of the rig hierarchy in the tree structure is demonstrated in Figure 2.1

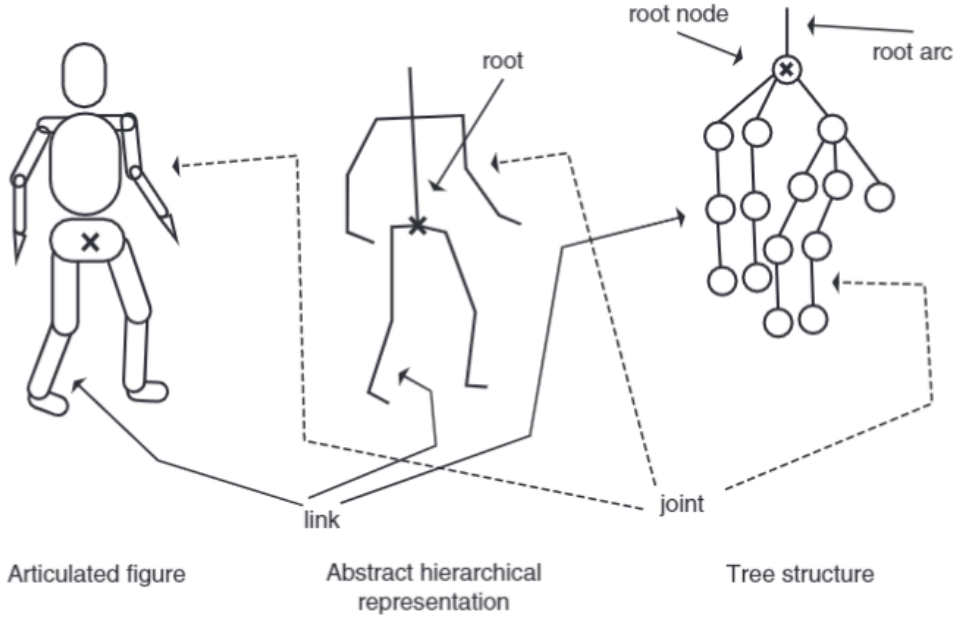


Figure 2.1: Example of a rig structure [21]

The hierarchy can be represented by a tree structure, where the root node is the root of the rig that is often at the pelvis. The nodes are the joints and the links are the rigid connections between the joints. The leaf nodes are the end effectors. Each path from the root to a leaf node is a kinematic chain. Each node contains information both in the joint space (joint angles) and in the Euclidean space (joint positions and rotations).

2.1.3 Forward- and inverse kinematics

Forward- and inverse kinematics are two mathematical processes that calculate the joint parameters with respect to the kinematic linkage. Forward kinematics traverse

through the hierarchy in a depth-first pattern, and propagate the changes of the parent nodes to end effectors, in the joint space. Conversely, inverse kinematics calculate the corresponding joint parameters of all parent nodes, from the end effectors' parameters in Euclidean space [21]. These two processes are essential building blocks for robotics and digital animation tools.

The degree of freedom is the number of directions, in which a joint is allowed to rotate. Total degrees of freedom describe the complexity of the character rig. The higher the degrees of freedom the more possible solutions exist for an inverse kinematic problem.

Analytical solutions are possible for simple systems. For more complex systems, the Jacobian method can be used. By iteratively solving the Jacobian matrix of changes of end effector parameters with respect to the changes of the joint parameters.

However, this numerical solution is not guaranteed to be natural-looking. Further control terms can be added to constrain the solution space or bias the solution towards some specific ideal joint angles. Another approach for solving inverse kinematic problems is by using cyclic coordinate descent [21]. It chooses the best value for each joint sequentially, from the outermost inwards.

Another iterative approach to solving inverse kinematic problems is the Evolutionary strategy-based solver as described in Section 2.4.4, which is used in this work to generate motion data.

2.1.4 Motion capture (mo-cap)

Motion capture techniques are widely used in film and game industries, for creating physically correct, natural-looking and detailed animations. While it is possible to achieve the same quality using a keyframe system with kinematic solvers, it is much faster to record real actors and project the recorded motions onto virtual characters for further processing.

In the game industry, for established game studios, it is the primary source for creating realistic and expressive animations. The mo-caps are further processed and cleaned using keyframe systems, and stored in a motion database. An animation system is usually a state machine, where each state corresponds to an animation clip in the database. When transiting from one state to another, a blending of the two corresponding clips is played.

Kinematic solvers can be applied on top of the animation system, to generate secondary animations that adjust the poses according to certain conditions, such as fitting the feet to the terrain. Most of the recent research in this area use mo-caps for the training of the frameworks.

2.2 Deep learning and transfer learning

Machine learning (ML) is about finding patterns in data and transforming them into useful knowledge for tasks such as prediction and classification of new data [2]. Deep learning (DL) is a subcategory in ML that utilises deep artificial neural networks (ANN) to efficiently capture complex patterns in large and high-dimensional datasets.

2.2.1 Basic concepts

A ML model can be expressed as a function $f(x;\theta)$ that takes the n-dimensional input vector $x \in \mathbb{R}^n$, with a set of parameters θ. Learning of a ML model refers to the optimisation of the function given some dataset $D = \{x_0, x_1, ..., x_M\}$.

There are generally three types of machine learning: *supervised learning*, *unsupervised learning* and *reinforcement learning*. The first type is the learning with labelled data $D = \{X, Y\}$, which is to find the optimal parameters θ^* such that the error between the $f(X;\theta^*)$ and Y is minimised, where Y is the labels (target outputs). Unsupervised learning focuses on recognising the relationships between the data points, to extract latent information about the dataset. Reinforcement learning is about learning the optimal action policy that maximises some reward function, conditioning on the observations of some state [2].

A neural network consists of an input- and an output layer, and some hidden layers in-between them. The feed-forward layer is the most simple layer type in ANN. It performs the following operation:

$$h = \phi(w \cdot x + b)$$

where h is the layer output, w and b are the learnable network parameters: weights and biases. $\phi(\cdot)$ is the activation function that is usually non-linear. A visual example is illustrated in Figure 2.2. For example, Rectified Linear Unit (ReLU) and Exponential Linear Unit (ELU) are two activation functions:

$$\text{ReLU}(x) = max(0, x)$$

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x <= 0 \end{cases}$$

Optimisation of the network implies updating the parameters iteratively to reduce some loss function $\mathcal{L}(y_p, y_o)$, where $y_p \in \mathbb{R}^y$ is the predicted output and $y_o \in \mathbb{R}^y$ is the

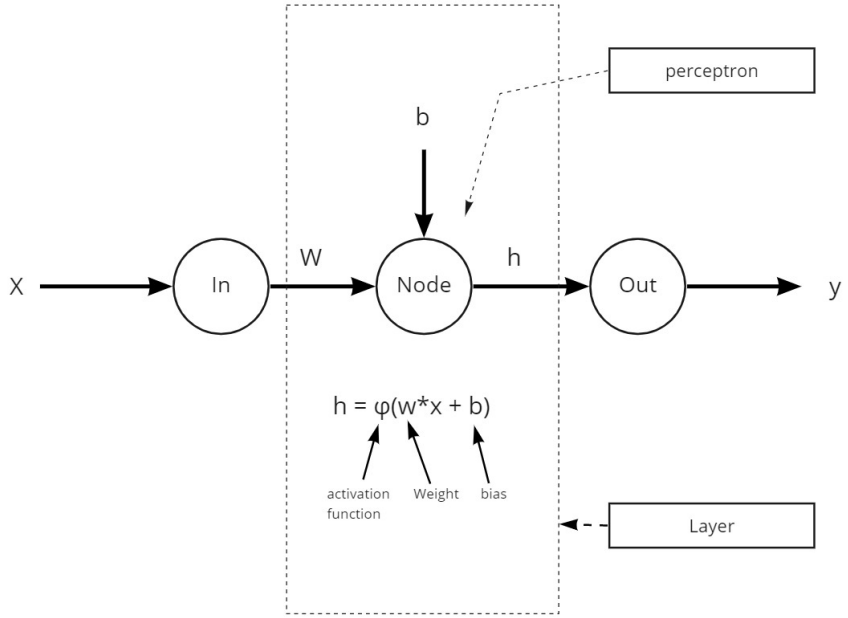


Figure 2.2: Example of feed-forward network operation with a single layer and a single hidden node.

target output. A common loss function is mean squared error (MSE):

$$MSE(y_p, y_o) = \frac{1}{M}\sum_{i=0}^{M}(y_p^{(i)} - y_o^{(i)})^2$$

where the superscript denotes the ith sample in the dataset. The gradients of the loss function with respect to each of the parameters, $\{ \frac{\partial MSE}{\partial w}, \frac{\partial MSE}{\partial b} \}$ are calculated and propagated back through the network to compute the gradients for each layer. The gradients are then used to update the parameters at each layer:

$$\Delta w = -\eta \frac{\partial MSE}{\partial w}$$
$$\Delta b = -\eta \frac{\partial MSE}{\partial b}$$

where η is the learning rate that scales the update step. This optimisation process is called gradient descent and it is based on the MSE of the entire dataset, which can be burdensome for large datasets. A more computational efficient approach is called stochastic gradient descent (SGD), which processes the dataset in batches and computes gradients for each batch individually. One full iteration over all batches is called an epoch. The models in this work are optimised using SGD.

2.2.2 Training, validation and testing

To train a neural network, a common practice is to partition the dataset into 3 separate sets: training set, validation set and test set. The training set is used for optimising the network. The validation set is used for monitoring the learning of the network. The validation error reflects the performance of the network on unseen data. If this error increases after some epochs while the training error is steadily decreasing, it implies the network is overly adapted to the training data which results in worse generalisation performance. This behaviour is called overfitting. Conversely, if the optimisation fails to decrease the validation error, the network is said to be underfitting. The test set is used to access the performance of the network after the training is finished.

2.2.3 Autoencoder (AE)

Autoencoder is an auto-associative deep learning model that learns efficient data encoding in an unsupervised manner [1]. On the high level, the model is a function $f(x;\theta) = x$, that maps any input sample to the same input sample in the same space, as displayed in Figure 2.3.

Internally, the model can be decomposed into two modules: an encoder (Enc$(\cdot)$) and a decoder (Dec$(\cdot)$). The encoder encodes the input sample to some latent representation (z) and the decoder decodes it back to the original input sample. A bottleneck can be formed by specifying the encoder to output a low dimensional z. In this way, the encoder is forced to learn to extract important features from the input samples. Thus, autoencoders can be used for feature extraction and dimensionality reduction. The latter is a common practice to handle the curse of dimensionality, which is a critical problem in ML. It refers to the high dimensionality of the data that makes learning difficult.

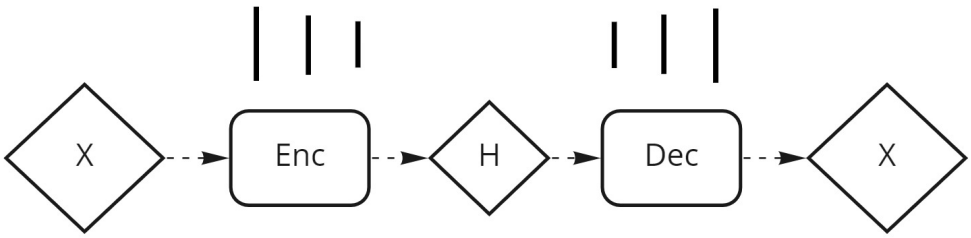


Figure 2.3: Visualisation of an autoencoder.

2.2.4 Autoregressive model (AR)

An autoregressive model is a type of deep learning model that predicts future values based on its own previous predictions [18]. Autoregression is a regression of the variable against itself. An AR model of order p can be expressed as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t$$

where ε is a stochastic term, c is a constant term and ϕ_i is a parameter. AR models are suitable for time-series predictions and sequence data processing. In this work, all the models are AR models of order 1.

2.2.5 Transfer learning

Transfer learning has started gaining popularity in machine learning. It aims to improve the performance of target learners on target domains by transferring the knowledge in previously trained models. In this way, the amount of training data in the target domains, that is required to achieve comparable performance, can be reduced. Collecting a large amount of quality data is generally considered an expensive and time-consuming process.

According to Zhuang et al:s comprehensive survey [30], the definition of transfer learning is given as the following:

> ***Definition 1.*** (Domain) *A domain D is composed of two parts, i.e. a feature space $\mathcal{X}$ and a marginal distribution $P(\mathcal{X})$. In other words, $D = \{\mathcal{X}, P(\mathcal{X})\}$. And the symbol $\mathcal{X}$ denotes an instance set, which is defined as $\mathcal{X} = \{x|x_i \in \mathcal{X}, i = 1, ..., n\}$*
>
> ***Definition 2.*** (Task) *A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a decision function f, i.e. $\mathcal{T} = \{\mathcal{Y}, f\}$.* The decision function f is an implicit one, which is expected to be learned from the sample data.
>
> ***Definition 3.*** (Transfer learning) *Given some/an observation(s) corresponding to $m^S \in \mathcal{N}^+$ source domain(s) and task(s) (i.e. $\{D_{S_i}, \mathcal{T}_{S_i}|i = 1, ..., m^S\}$), and some/an observation(s) about $m^T \in \mathcal{N}^+$ target domain(s) and task(s) (i.e. $\{D_{T_i}, \mathcal{T}_{T_i}|i = 1, ..., m^T\}$), transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions f_{T_j} ($j = 1, ..., m^T$) on the target domain(s)*

If $m^S = 1$, it is called single-source transfer learning, otherwise, it is called multi-source transfer learning. This work only focuses on the former type of transfer learning.

There are many strategies and techniques for transfer learning. The two primary strategies are data transformation and model control [30]. The former focuses on transforming the features in the target domains, such that the distribution differences between the source domains and the target domains are reduced.

There are three types of feature transformation approaches: feature augmentation feature reduction and feature alignment. This work focuses on the feature reduction approach because it also mitigates the notorious curse of dimensionality problems in machine learning. The approach can be further divided into three categories: feature clustering, feature selection and feature encoding.

Feature clustering is about finding abstract feature representations of the original features. By utilising clustering techniques, the relationship of the samples to the clusters can be used as input or augmented features for performing the task(s). Dai et al. proposed an unsupervised clustering approach that is called Self-Taught Clustering (STC) [4]. It assumes that the source domain and the target domain share the same feature clusters in their common feature space.

Feature selection is used to extract pivot features, which are the features that share the same behaviour in the different domains.

Feature encoding is another method for feature extraction that produces an abstract representation of the samples, using autoencoders as described in Section 2.2.3.

2.3 Common neural network models

This subsection presents the relevant neural network models that are used in this work.

2.3.1 Multi-layer perceptron (MLP)

Multi-layer perceptron (MLP), or feedforward neural network, is the simplest and basic neural network model. It consists of feedforward layers, where every node in a layer is connected to all the nodes in the next layer. A basic MLP network of L hidden layers that are parameterised by weights and biases can be expressed as the following [2]:

$$f(x; W, B) = \phi_{L-1}(...(\phi_1(w_1 \cdot \phi_0(w_0 \cdot x + b_0) + b_1)))$$

where $\theta = (W = \{w_0, w_1, ..., w_{L-1}\}, B = \{b_0, b_1, ..., b_{L-1}\})$. MLP is generally good for regression tasks, or classification tasks by attaching a softmax layer to the output layer: $\text{softmax}(z) = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}}$ for $i = 1, ..., y$ and $z \in \mathbb{R}^y$ and y is the output dimension.

2.3.2 Convolutional neural network (CNN)

Convolutional neural network (CNN) is commonly applied to computer vision-related tasks due to its space invariant property. At each layer, a convolution filter is applied to the input features and provide translation equivalent responses known as feature maps [13]. CNN can extract locality-preserving latent features, which makes it robust at performing image-related tasks.

2.3.3 Radial-basis function (RBF)

Radial-basis function is a linear combination of C basis functions (kernels) $\phi_j(\cdot)$, where $j = 1, 2, ..., C$, that are only dependent on the radial distance from the respective centre μ_j [2], such that:

$$\text{RBF}(x) = \begin{pmatrix} \phi(||x - \mu_0||) \\ \phi(||x - \mu_1||) \\ ... \\ \phi(||x - \mu_C||) \end{pmatrix}^T$$

Common basis functions are:

- Gaussian basis function: $\phi(\alpha) = exp(-1 \cdot \alpha^2)$ where $\alpha = \frac{||x-\mu_j||}{\sigma}$ or $\alpha = \sigma||x - \mu_j||$.
- Quadratic function: $\phi(\alpha) = \alpha^2$.
- Spline function: $\phi(\alpha) = \alpha^2 \cdot ln(\alpha + 1)$

A RBF layer with Gaussian kernel is parametrised by $\theta = (\sigma, \mu)$ where $\sigma = \{\sigma_0, \sigma_1, ..., \sigma_C\}, \mu = \{\mu_0, \mu_1, ..., \mu_C\}$, can be expressed as the following:

$$f(x; \theta) = \sum_{i=1}^{C} \sigma_i \phi(||x - \mu_i||)$$

RBF is similar to MLP, but its capacity is limited by the number of kernels. The selection of the kernel centres is critical for RBF networks. K-means clustering based on Euclidean distance can be applied to the data points prior to training, to specify the kernel centres that no longer coincide with the training samples. Another approach is to let the centres be randomly initialised, by using a subset of data points as the centres or sampled from a normal distribution. The centres are part of the learnable parameters and get optimised during the training of the network.

2.3.4 Deep embedding clustering (DEC)

Deep embedding clustering (DEC) is an unsupervised deep learning-based clustering model proposed by Xie et al. [28]. It clusters the data points by simultaneously learning a set of C cluster centres $\{ \mu_0, \mu_1, ..., \mu_C \}$ in the latent feature space $\mathbb{R}^k$, and the parameters θ of the MLP that maps the data point $x_i \in \mathbb{R}^n$ to $z_i \in \mathbb{R}^k$, where $k < n$ and $i = 1, 2, ..., M$.

DEC consists of two modules, an autoencoder based on MLP that learns the mapping function, and a clustering model that assigns the data points in the latent space to the C clusters. The clustering model performs a soft assignment that uses Students' t-distribution as the kernel function for measuring the similarity of the embedded points z_i to the respective cluster centres μ_j:

$$q_{ij} = \frac{(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum[(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}]}, \quad \text{where } j = 1, 2, ..., C$$

DEC is optimised by minimising the reconstruction error produced by the autoencoder and the Kullback-Leibler (KL) divergence of the soft assignment distribution to the auxiliary target distribution:

$$p_{ij} = \frac{{q_{ij}}^2/f_i}{\sum_j {q_{ij}}^2/f_i}, \quad \text{where}$$
$$f_i = \sum_i q_{ij}$$

2.3.5 Variational autoencoder (VAE)

Variational autoencoder is a probabilistic variant of autoencoder. Instead of mapping data points to latent representations directly, the encoder produces a set of parameters that describe the latent distribution of the data points [2]. The embeddings can be sampled from this distribution that can be used for the decoder to reconstruct the input sample.

Distribution of the encoder function can be described as:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}, \quad \text{where } p(x) = \int p(x|z)p(z)dz$$

where z is the embedding of the input sample x. $p(x)$ is a normalisation constant and usually an intractable distribution, making direct inference impossible. Instead, variational inference can be used to approximate $p(z|x)$ by construction another

distribution $q(z|x)$ that is tractable. Optimising q implies minimising the KL divergence:

$$KL(q||p) = \sum_x q(x) log \frac{q(x)}{p(x)}$$

- $KL(q||p) \geq 0$ for all q, p.

- $KL(q||p) = 0$ if and only if $q = p$

The optimisation is achieved by raising the evidence to lower bound (ELBO):

$$\begin{aligned} Let \quad & p = p(z|x), \quad q = q(z|x), \quad Z = p(x) \\ & J(q) = \sum_x q \; log(\frac{q}{\tilde{p}}) \\ & J(q) = \sum_x q \; log(\frac{q}{p}) - log(Z) \\ & J(q) = KL(q||p) - log(Z) \end{aligned}$$

where $\tilde{p}$ is the unnormalised $p(z|x)$. Since $KL(q||p) \geq 0$, after rearranging the terms:

$$\begin{aligned} log(Z) = KL(q||p) - J(q) \geq -J(q) \\ log(Z) \geq \mathbb{E}_q[log(\tilde{p}) - log(q)] \end{aligned}$$

$-J(q)$ is the ELBO. By maximising this term, $KL(q||p)$ is "squeezed" between $-J(q)$ and $log(Z)$, since Z is a constant term. By mean-field theory, q can be decomposed into a set of fully factored distributions: $q = \prod_{i=1}^{C} q_i$, where each q_i is parameterised by θ_i. MLP are commonly used to produce a set of distribution parameters $f(x; \theta_{MLP}) = \theta_q, \quad$ where $\theta_q = \{\theta_1, \theta_2, ..., \theta_k\}$. The latent representation $z_i \in \mathbb{R}^k$ can then be sampled from the joint distribution. A common practice is to set the target distribution $\tilde{p}$ to be normal distributed: $\tilde{p} \sim \mathcal{N}(0, 1)$, which also acts as regularisation on the learning.

Similarily, the distribution of the decoder can be expressed as $p(x|z)$ and is usually approximated using MLP. The loss function of VAE is formulated as the following:

$$\mathcal{L} = \mathbb{E}_q[log(p(x|z)] - \mathbb{E}_q[log(\tilde{p}) - log(q)]$$

where the first term is the reconstruction likelihood and the second term is approximation error to the target distribution.

2.3.6 Mixture-of-Experts (MoE)

Mixture-of-Experts (MoE) consists of a fully conditional mixture model where the blending coefficients, that is produced by a gating function, and the component densities, also called experts are conditional on some input variables [3]. It is used in a variety of contexts including regression, classification and clustering.

Regression aims to recognise the relationship of an observed random variable $Y \in \mathbb{R}^y$ given a covariate sample $x \in \mathbb{R}^n$ by learning the conditional density function $p(Y|X = x)$. The univariate mixture of the regression model assumes that the observed pairs (x, y) are generated from Q regression functions that are governed by a hidden categorical random variable Z, indicating the component from which each observation is generated. A nonlinear regression model can be decomposed into a weighted sum of Q regression components:

$$f_k(y|x;\theta) = \sum_{i=1}^{k} \pi_i f_i(y|x,\theta_i)$$

where $\pi_i = P(Z = i)$ represents the non-negative blending coefficients such that $\sum_{i=1}^{K} \pi_i = 1$. In MoE, they are modelled as a function of some covariates, generally implemented using a softmax function:

$$\pi_i(x;\theta) = P(Z = i|x;\theta) = \frac{exp(\theta_i^T x)}{\sum_{i=1}^{k} exp(\theta_i^T x)}$$

2.3.7 Long short-term memory (LSTM)

Long short-term memory is a recurrent neural network that contains feedback connections from the output nodes back to the input nodes [10]. It is widely used for sequential data processing, such as time series prediction, natural language processing and audio processing. An LSTM layer has a cell, an input gate, an output gate and a forget gate. With these cells, it can memorise information from earlier inputs, making LSTM

robust to longer sequence data. The operations in the LSTM layer are the following:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f + h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i + h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o + h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c + h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

where

$$
\begin{aligned}
x_t \in \mathbb{R}^d : &\quad \text{input vector to the LSTM unit} \\
f_t \in \mathbb{R}^h : &\quad \text{forget gate's activation vector} \\
i_t \in \mathbb{R}^h : &\quad \text{input/update gate's activation vector} \\
o_t \in \mathbb{R}^h : &\quad \text{output gate's activation vector} \\
h_t \in \mathbb{R}^h : &\quad \text{hidden state vector} \\
\tilde{c}_t \in \mathbb{R}^h : &\quad \text{cell input activation vector} \\
c_t \in \mathbb{R}^h : &\quad \text{cell state vector} \\
W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h : &\quad \text{weight matrices and bias vector}
\end{aligned}
$$

2.3.8 Least Square Generative Adversarial Network (LSGAN)

GAN is a special type of generative model, that consists of two modules: a generator and a discriminator. The generator is trained to generate samples like the observed samples x_i in the datasets. The discriminator is trained to differentiate the generated samples from the real samples, adding extra error to the generator and forcing it to learn better. Least-square GAN is a variant of GAN, that used distances instead of probabilities for evaluating the samples [16]. Let $G(\cdot)$ be the generator and $D(\cdot)$ be the discriminator, the following loss terms are computed and used to optimise the models:

$$
\begin{aligned}
\mathcal{L}_D &= \frac{1}{2}\mathbb{E}_{y \sim p_{data}(y)}[(D(y) - 1)^2] + \frac{1}{2}\mathbb{E}_{x \sim p_x(x)}[D(G(x))^2] \\
\mathcal{L}_G &= \frac{1}{2}\mathbb{E}_{x \sim p_x(x)}[(D(G(x)) - 1)^2]
\end{aligned}
$$

2.4 Related work

This section presents the related work in data-driven motion synthesis models.

2.4.1 Kernel-based approaches

Principal component analysis (PCA) has been successfully applied for reducing the dimensionality of full-body motion vectors, but the low dimensional latent space has issues capturing a wide range of movements. Kernel-based approaches have been proposed to overcome this issue. RBF and Gaussian process (GP) are two common kernel-based models for learning different types of locomotion [12]. RBF is tested as a component in OMG in this work.

2.4.2 Time series models

Time series models are the models that predict the future pose of the character based on the current and the past poses, conditioned on some control signals. Various neural network models have been proposed for this task, such as conditional Restricted Boltzmann Machine (cRBM) [27] and Encoder-Recurrent-Decoder model (ERD) [5]. Although these models are more robust and efficient than the kernel-based models, they suffer from drifting issues, where the generated motion comes off from the motion manifold and converges to a generic pose.

LSTM-based approaches are a natural choice for motion prediction, due to their capability of handling long sequence data. ERD is a model that incorporates an LSTM model with an autoencoder, that is capable of generating mo-cap quality animation, body pose labelling and body pose forecasting in videos. Harvey et al. proposed an LSTM-based model (TR-LSTM [9]) for generating transitions between temporally-sparse keyframes, by introducing two novel additive embeddings: *time-to-arrival embedding* and *scheduled target noise embedding*. The LSTM models are difficult to tune and suffers heavily from bias-variance tradeoff, they are also less ideal for runtime generation tasks due to their low responsiveness caused by their high dimensional internal memory state [26].

Holden et al. proposed a novel neural network structure called Phase-Functioned neural network (PFNN) [12], where the weights of the motion generation network (MoGen) is computed by Catmull-Rom spline function conditioned on a phase variable. The phase variable tells the progression of the motion and is defined by the foot contact pattern of bipedal locomotion. The introduction of the phase variable improved the quality of the generated motion that can adapt to different terrains.

Based on PFNN, Starke et al. proposed a framework called Mode-Adaptive neural network (MANN) for quadruped motion synthesis [29]. MANN replaces the original phase function with a gating function, transforming the model into an MoE model. The gating function is conditioned on the feet velocities, action labels and target velocity.

MANN is capable of generating believable quadruped locomotion with different gaits, adapting to different geometry and does not suffer from foot sliding issues.

Based on MANN, Starke et al. proposed two enhanced versions that are capable of generating character-scene interacting animations (Neural State Machine, NSM [25]), and complex multi-contact character-object animations (MoE with local motion phase, LMP-MoE [26]). The latter is capable of animating bipedal characters to play basketball, with various movement and interaction modes, by introducing a local motion phase variable. Unlike PFNN, where the locomotion is governed by one single global phase variable, LMP-MoE computes local motion phases for each key-bone (feet, hands and ball) based on the contact transitions. With the local motion phase variable, the network can generate both cyclic and acyclic motions where the body parts are moving at different and inconsistent velocities and frequencies.

Zinno et al. proposed a VAE-based framework for synthesising bipedal locomotion, referred to as Motion VAE (MVAE [15]). It combines MoE with an autoregressive conditional VAE to predict the next pose given the current pose. Furthermore, the model can learn the distribution of next-pose predictions. This latent distribution is treated as the action space for their reinforcement learning-based animation controller, that combined with MVAE is capable of generating goal-driven motions. The architecture of the system is illustrated in Figure 2.4.

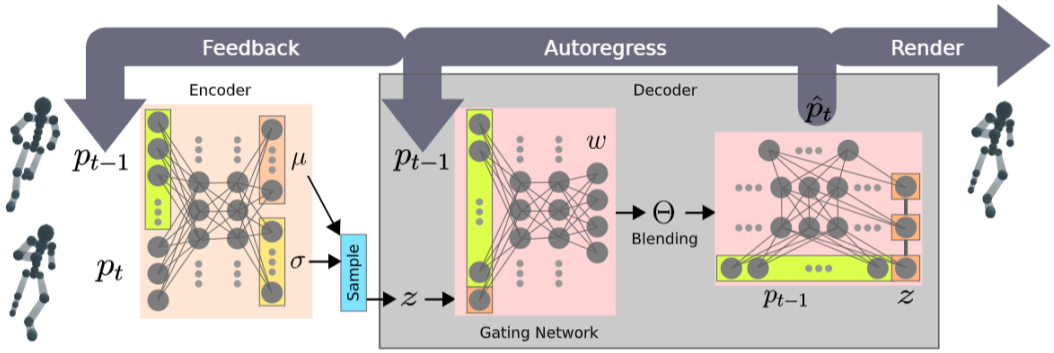


Figure 2.4: The architecture of Motion VAE framework by Zinno et al. [15]

These LSTM- and MoE-based time series models are studied in this work because they are versatile and relatively easy to implement. They can be implemented in both offline and online fashion and the generation process only depends on the input variables. Unlike the physically-based animation systems, where the animation quality is also governed by the underlying physics engine.

2.4.3 Physically-based animation systems

Physically-based animation systems generate kinematic animations that obey the physical constraints or animations that are produced by applying forces (torques) to the character in a physically-based environment. A forward dynamic approach is proposed by Hodgins et al. [11], which computes and applies joint torques to the body, to generate realistic motion. Peng et al. proposed a deep reinforcement learning framework called DeepMimic [22], that is capable of learning mo-cap data in a physically-based environment. These systems are not experimented with in this work due to the time constraint and complexity.

2.4.4 Evolutionary strategy-based IK solver

Starke et al. has proposed an evolutionary strategy-based IK solver that can solve fully constrained generic inverse kinematics with multiple end effectors and goal objectives [24]. The solver uses a combination of evolutionary strategy optimisation technique and swarm optimisation, together with limited-memory-Broyden-Fletcher-Goldfarb-Shanno for gradient-based optimisation on inverse kinematic problems. The algorithm is fast, robust to avoid suboptimal extrema and capable of producing accurate solutions that satisfy biological constraints in real-time. The solver is also developed into a plugin that is available on Unity Asset Store, which is used for creating the motion data for this research.

Chapter 3

Methods

The purpose of this work is to study and investigate the possibilities of transfer learning on deep learning-based motion synthesis frameworks. This chapter presents the design and implementation of OMG (Sections 3.1, 3.6), the transfer learning techniques (Section 3.2), the data collection (Section 3.3), the experiments and the evaluation method (Section 3.5).

The research process is divided into the five stages that are summarised and illustrated in Figure 3.1. Table 3.1 presents the various symbols that are used in this chapter.

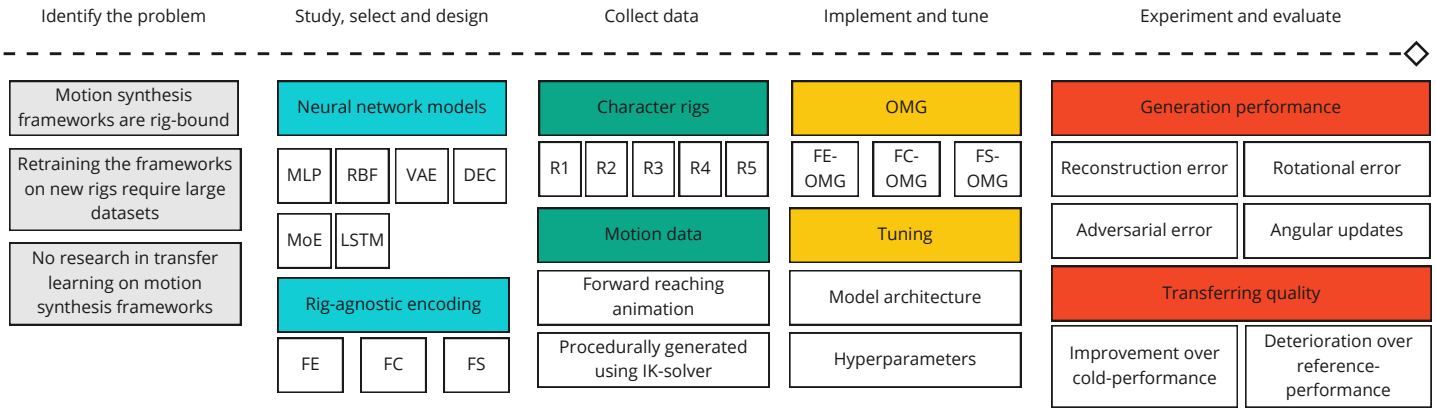


Figure 3.1: The research process with the five stages: 1) understanding and formulating the problem; 2) selecting and designing the models and the transfer learning approaches; 3) collecting training data; 4) implementing and tuning the models and the approaches; 5) conducting the experiments and evaluating the results.

Symbol	Explanation
X, Y	Input and output variables
x, y	Input and output vectors
n	Input dimension
M	Dataset size
K	Number of key joints
X^S	Input pose features
X^V	Input control features
X^P	Input local motion phase
w, b	Weights, biases
μ, σ	Cluster centres (mean), scalars (standard deviation)
$\mathcal{L}$	Loss
z	Embedding
C	Number of clusters / distribution factors
ϕ	Activation function, basis function
ϕ	Activation function, basis function
Ψ	Output from the clustering layer (C-model)
θ	Network parameters
Θ	Local motion phase
k	Number of experts in MoE model

Table 3.1: The symbols that are used in the description of the systems.

3.1 Objective-driven Motion Generation network (OMG)

A motion synthesis framework that predicts the next pose based on the current pose and some objective function is proposed and referred to as Objective-driven motion generation network (OMG). It is based on LMP-MoE and MVAE, with some objective cost as the condition. The objective cost is a quantity that needs to be minimised by the framework, guiding it to generate desired poses.

For simplicity, the objective functions are selected to be the Euclidean distance between the key joints (hands, feet) and their target positions, and the angular difference to the target rotations. It is technically an MoE-based IK solver, approximating the behaviour of the Evolution strategy-based IK solver, described in Section 2.4.4. By using temporally sparse targets, OMG can be effectively used as TR-LSTM.

Figure 3.2 visualises one architecture of OMG. It is essentially an ensemble of four modules, where each of them is a neural network. There is an autoencoder (AE) for encoding and decoding the character pose data, an encoder for the control variable data, the main network (MoGenNet) for predicting the next pose and feature updates, and potentially a clustering layer for the feature clustering approach.

The local motion phase can be computed from the contact patterns of the key joints and their target positions along the timeline. The MoGenNet can be either MoE or LSTM. Starke et al. have presented a comparison of their MoE-based framework with the LSTM equivalent, and concluded that MoE performs better with fewer animation artefacts, such as foot sliding and pose instability, and achieved higher responsiveness [25, 26, 29].

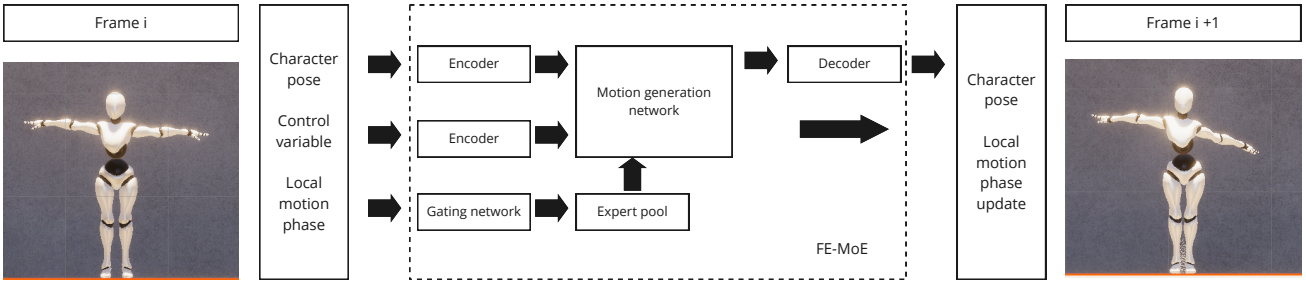


Figure 3.2: The overview of the architecture of FE-OMG with MoE as the motion generation module. The local motion phase variable X_i^P is fed to the gating network to produce the weights for the main network. The pose data X_i^S and the control variable X_i^V are encoded by the respective encoder and fed to the main network to produce the next frame.

3.2 Rig-agnostic encoding (RAE)

Intuitively, the feature reduction techniques, in the data transformation category, are suitable candidates for this work, since the target task remains the same as the source task. In this work, these techniques are used to transform the data points to a more abstract representation in the shared feature space across different domains and achieve rig-agnostic encoding on the pose data.

The three feature reduction approaches for achieving RAE are the following:

1. **Feature encoding (FE)**, by integrating an autoencoder (AE) with MoGenNet, the pose data can be encoded into compact embeddings, which are directly fed to the MoGenNet.

2. **Feature clustering (FC)**, by applying unsupervised clustering on the pose embeddings in the latent feature space, it can obtain the distance vectors of the embeddings to the cluster centres. The distance vectors can be used as inputs (FC-IN) or auxiliary features to MoGenNet (FC-CAT). The learnt cluster centres can be regarded as pivot poses in the latent space, that capture the distribution of the pose

domain which is defined by the motion data. In this work, RBF, VAE and DEC are implemented and tested as the clustering layer.

3. **Feature selection (FS)**, by simply using a fixed subset of features for all domains, the same framework can be technically reused across the domains. More specifically, by using only 6 key joints (feet, hands, spine and head), poses from all the domains can be represented in pose vectors of the same length. The approach also greatly reduces the dimensionality of the data. To compute the full-resolution pose from the 6-joints output poses, kinematic solvers can be used. In this work, for facilitating the comparison between the approaches, the MoGenNet is trained to upsample and output full-resolution poses.

The RAE approaches are illustrated in Figure 3.3.

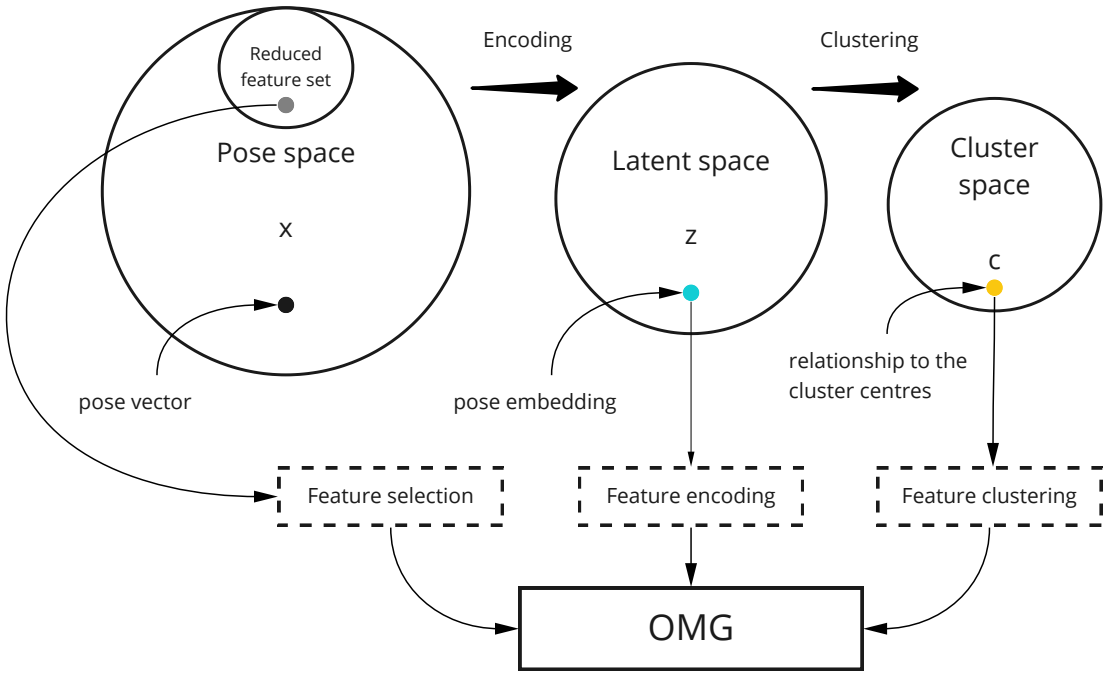


Figure 3.3: The illustration of the three RAE approaches, where the points are data points in the respective space. The figure displays the transformation of the pose vectors to achieve rig-agnostic encoding.

For transferring the trained models to other domains, the AE can be swapped with a new one that is pre-trained on the new domain, with the same internal configuration. When using feature clustering, the clustering layer is also transferred to the new domain, for obtaining the relationship of the pose embeddings from the new domain to the cluster centres, that represent the previous pose domain.

FE-OMG, FC-OMG and FS-OMG denote the OMG models that adopt the respective RAE technique.

3.3 Data collection

For experimenting with transfer learning, multiple bipedal humanoid character rigs and motion data on each of the rigs are required. The type of motion data is the same and fixed for all rigs.

3.3.1 Character rigs

It requires a set of character rigs that share different configurations including the number of joints, the topology and the proportion. The experiments involve multiple repetitions of training various models on each rig, thus the experiment time is linearly proportional to the number of rigs. Due to the time constraint, having a large number of rigs is infeasible. Instead, five rigs are collected from various sources that represent real-world scenarios in the game industry, which is the main application area of the framework.

1. (R1) A character rig from Adobe Mixamo. It is a clean and industry-standard character rig provided by Adobe for games and films. It is available for all creators and free to use in commercial projects.

2. (R2) A free bipedal monster rig from Unity Asset Store. It represents the type of rigs that would be used by indie studios with limited resources and funding.

3. (R3) The default character rig from Unity. It is the default rig that comes with Unity Engine and is accessible for all developers.

4. (R4) The character rig that is used in Neural State Machine (NSM) research project by Sebastian et al [25].

5. (R5) A character rig from DICE. It is currently being used in their ongoing projects. This rig represents the character rigs that are created by animators from established game studios to fit their specific needs, and it is used for mo-cap animations.

The specification of the five character rigs are the following:

heightRig	From	Joints	DOF	Average bone length (m)
R1	Adobe Mixamo	31	71	0.175
R2	Unity Asset Store	31	71	0.191
R3	Unity	29	65	0.189
R4	NSM repository	24	60	0.210
R5	DICE	32	74	0.169

3.3.2 Motion data

Since OMG is objective-driven, using IK-solver to procedurally animate the characters to create motion data is a feasible choice. This method is time-efficient and scalable. It also ensures full control over the generated motion data for all the rigs.

Motion sequence creation

The motion sequences are created in Unity using the proprietary Bio-IK plugin, described in Section 2.4.4. The positional and rotational objectives are used to drive the end effectors (hands) towards the target positions and the target rotations. All the clips have a length of 300 frames (5 seconds at 60 frames per second), where each frame contains the parameters of the joints, the parameters of the targets and the respective cost.

The following clips are created for every rig:

For every reaching motion, the targets for both hands are spawned around the character. The Bio-IK plugin drives the hands towards the objectives. When the distance between a hand and its target is less than 0.1 m, it is considered as contacted. The hand stays at that position for 5 frames, before the next target spawns.

The above process repeats (1 / 2 / 3) times for each clip, in which the targets are discreetly spawned on the surface of a cylinder with the rig pivot as the centre. The height of the cylinder is 0.7 m and the radius is (0.35 m / 0.7 m). The cylinder is divided into 5 levels with 8 spawn points on the perimeter of each level, evenly distributed with $\frac{2\pi}{8}$ degrees between them, as illustrated in Figure 3.4.

For the single repetition case, 40 clips are created with the targets spawning on every spawn point (with/without) random rotation. After the targets are reached, they respawn on the initial positions, guiding the hands back to the default positions and orientations.

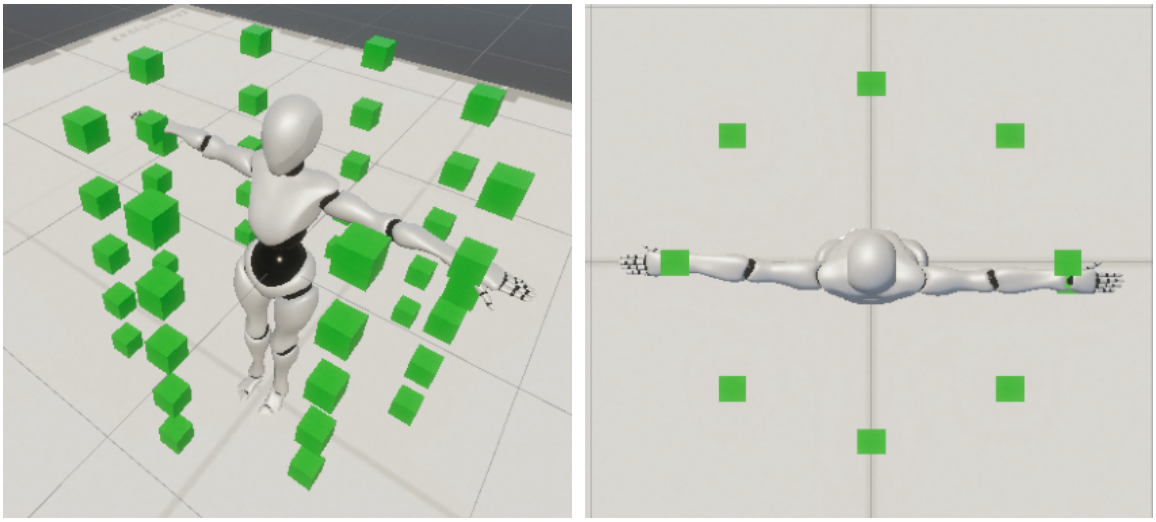


Figure 3.4: Spawn points of the targets for creating motion data

For the multiple repetition case, there are (2) spawn modes. In both modes, 40 clips are created where the spawn points are randomly selected, but in mode 1, adjacent spawn points for the targets are sampled and in mode 2, the opposite spawn points are selected such that the angle difference between them is π.

The sequences are generated using Bio-IK with the following parameters:

Property	Value
Generations	3
Individuals	120
Elites	2
Motion type	Realistic
Maximum velocity	5
Maximum acceleration	5

The specification of the generation procedures of motion data is the following:

Spawn points	Number of repetitions	Random rotation	Spawn radius (m)	Spawn mode
40	1	False/True	0.7/0.35	-
40	2	False/True	0.7/0.35	adj./opp.
40	3	False/True	0.7/0.35	adj./opp.

where adj. denotes the spawn mode that the adjacent spawn points are selected, and opp. denotes the mode that the opposite spawn points are selected. In total, there are 960

motion clips in a dataset. There is a dataset for each rig, resulting in 5 datasets of 4800 clips in total.

Data features

The OMG models are primarily based on LMP-MoE and MVAE, thus the data features for training models are similarly formatted. The following features are included in the dataset:

- **Character pose** $X_i^S = \{p_i, r_i, v_i\}$ represents the pose of the character with J number of joints at the current frame i. $p_i \in \mathbb{R}^{3J}$ is the joint positions; $r_i \in \mathbb{R}^{6J}$ is the joint rotations in the form of a pair of the Cartesian forward and up vectors; $v_i \in \mathbb{R}^{3J}$ is the joint linear velocity.

- **Control variables** $X_i^V = \{T_i^p, T_i^r, C_i^p, C_i^r\}$ represents the control signals used to drive the prediction of the future poses. $T_i^p \in \mathbb{R}^{3K}$ is the target positions, where K is the number of key joints; $T_i^r \in \mathbb{R}^{6K}$ is the target rotation; $C_i^p \in \mathbb{R}^{3K}$ is the position cost for each key joint; $C_i^r \in \mathbb{R}^{3K}$ is the rotation cost.

- **Local motion phase** $X_i^P = \Theta_i \in \mathbb{R}^{2K}$ is the local motion phase vector for each key joint. The details of the phase vector is described in Section 3.3.2.

All the features are transformed to the rig root space.

Local motion phase

The local motion phase describes the progression of the motion for each joint individually, based on the contact pattern with the target. It contains information about the timing and the speed of the movement. The feature is computed in the same way as originally described in the LMP-MoE paper [26] but without the optimisation process.

Let $G(t)$ be the block function of a key joint, where $G(t) = 1$ if contact and $G(t) = 0$ otherwise. $G(t)$ is first normalised in a sliding window $W = 30$ frames centered at the frame t:

$$G_{norm}(t) = \frac{G(t) - \mu_{G_W}}{\sigma_{G_W}} \tag{3.1}$$

where $G_{norm}(\cdot)$ is the normalised block function, μ_{G_W}, σ_{G_W} are the mean and the standard deviation of $G(t)$ in the window G_W. The sliding window is padded with edge values and if $\sigma_{G_W} = 0$, then it is set to 1. The Butterworth low-pass filter $\mathcal{B}(\cdot)$ of order 3 with the

threshold = 0.1 is applied to th e block function.

$$G_{\mathcal{B}} = \mathcal{B}(G_{norm}) \tag{3.2}$$

The local motion phase vector is constructed as the following:

$$\Theta = \begin{pmatrix} \Delta_{sin} \cdot sin(G_{\mathcal{B}}) \\ \Delta_{cos} \cdot cos(G_{\mathcal{B}}) \end{pmatrix} \tag{3.3}$$

where $\Delta_{sin} = sin(G_{\mathcal{B}}(t)) - sin(G_{\mathcal{B}}(t-1)), \Delta_{cos} = cos(G_{\mathcal{B}}(t)) - cos(G_{\mathcal{B}}(t-1))$. The outcome of the different steps in the computation is plotted in Figure 3.5.

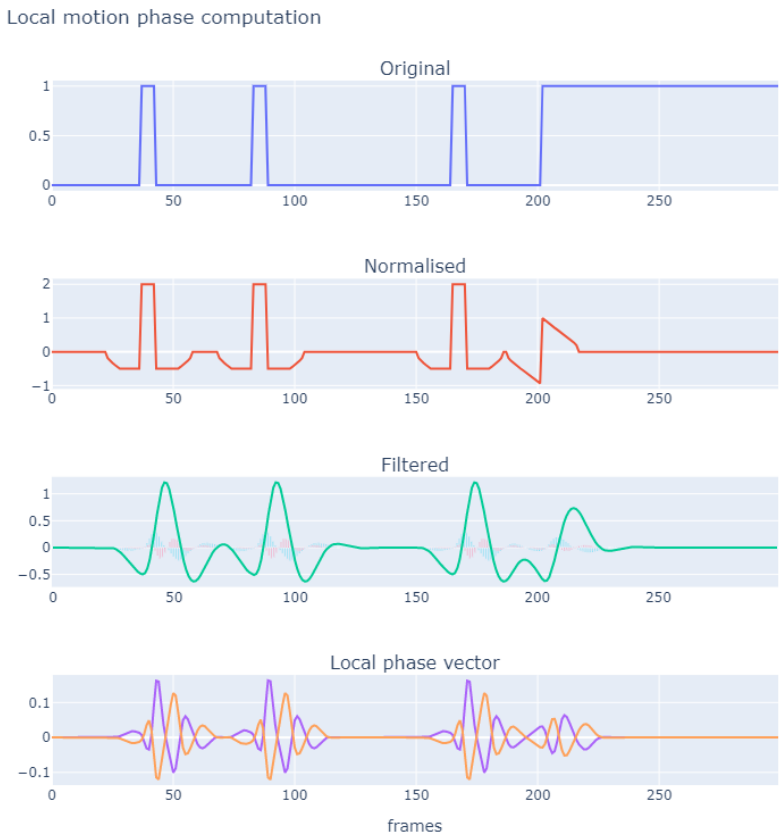


Figure 3.5: The outcome of the different steps in the computation of local motion phase.

Objectives

The objectives are the target position and the target rotation. The objective cost $\{C^p, C^r\}$ for a key joint j, in the form of vectors, is the conditioning features that guide the prediction of future state variables of the joint. C^p is the Euclidean distance from the

joint position to the target position with respect to the root coordinates. C^r are the angular distance between the joint and the target {forward, up} vectors.

$$C^p = T^p(j) - p(j) \tag{3.4}$$

$$C^r = \frac{1}{\pi} cos^{-1}(\frac{T^r(j) \cdot r(j)}{||T^r(j)|| \cdot ||r(j)||}) \tag{3.5}$$

3.4 Implementation and tuning of the models

The created motion data are exported from Unity as JSON files, which are parsed and extracted to Numpy [1] arrays and stored as bzip2-compressed binary files.

The models are implemented in Python using Pytorch [2] and Pytorch-Lightning [3]. The implementation of the models are based on MANN, NSM, LMP-MoE, MVAE and TR-LSTM [9, 25, 26, 29]

The implemented models are tested with a small subset of the dataset, to verify the implementation. Ensuring that the reconstruction errors are optimised during the training, and the models are capable of generating correct animations. The hyperparameters such as the number of layers, the layer sizes and the learning rates are tuned using Ray Tune [4] with ASHA scheduler and a grid search algorithm.

3.4.1 Network training

An input vector of OMG is X_i = $\{X_i^P, X_i^S, X_i^V\}$, where each item is described in Section 3.3.2. An input sample (clip) is a matrix of shape $299 \times n$, where n is the length of X_i.

$$X = \begin{pmatrix} X_0 \\ X_1 \\ ... \\ X_{299} \end{pmatrix}$$

For training, the input samples are stacked into a three-dimensional matrix for each batch with the size=32. The matrix has the shape $32 \times 299 \times n$. It is later decomposed into sequences of length 13, forming a new matrix with shape $736 \times 13 \times n$. The sequences are processed frame-wise, where the final input matrix to the network at each training step

[1] https://numpy.org/
[2] https://pytorch.org/
[3] https://www.pytorchlightning.ai/
[4] https://docs.ray.io/en/latest/tune/index.html

has the shape $736 \times 1 \times n$. The output vector is $Y_i = \{Y_i^P, Y_i^S, Y_i^V\}$, sharing the same length as X_i.

All modules are trained in the end-to-end fashion, with AdamW optimiser with weight decay rate 0.01 and $\beta = \{0.9, 0.999\}$. The layer weights are initialised using the Xavier method [7]. The learning rate is set to $1.0 \cdot 10^{-4}$ with the decay rate 0.9 every 80 training steps. The teacher forcing technique is applied, with a uniform probability, the output of OMG is fed back to the network as input for the next frame (autoregression). The autoregression probability is set to zero at the beginning and is gradually incremented to one.

The following loss terms for optimising the network are computed during training:

$$
\begin{aligned}
\mathcal{L}_{\text{recon}} &= \text{MSE}(Y_i, X_i) \\
\mathcal{L}_r &= \text{MSE}(y_i^r, x_i^r) \\
\mathcal{L}_{\text{adv}} &= \text{D}(X_i^S) \\
\mathcal{L}_{\text{KL}} &= \text{KL}(X_i^S) \quad \text{if VAE or DEC is used} \\
\mathcal{L}_{\text{ortho}} &= 0.1 \cdot \mathbb{E}(|\mu\mu^T - I|_1) \quad \text{if RBF is used}
\end{aligned}
$$

where only $\mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{ortho}}$ is used for optimising OMG.

3.5 Experimentation and evaluation

The experiments are designed to test the generation performance of the models and the transferring quality of the RAE approaches. More specifically, to investigate how much the performance is improved by warm starting the OMG models on the new domains with limited training and data. These models are referred to as the transferred models.

Generation performance is assessed using four metrics: reconstruction error, adversarial error, rotation error and root mean square of the angular updates, which are described in Section 3.5.1. The performance of the models that are fully trained is referred to as the reference performance.

Transferring quality refers to the generation performance of the transferred models, in comparison to their reference-level. The cold-started models are initialised with random parameters. The warm-started models are initialised with the parameters from the trained models in a different domain. For warm-started models, there are two training strategies: *frozen* and *trained*. The former freezes the inherited clustering layer and the MoGenNet module, and only allows the AE to be trained. This provides insights into the

transferring capability of the RAE approaches. The latter allows all modules to be trained, which experiments with the improvement headroom of the models.

All the experiments are run on the local machine with the following specifications:

- OS: Arch Linux with the linux kernel version 5.12.5
- CPU: AMD Ryzen 9 3900 @ 4.2 GHz
- GPU: Nvidia GTX 1080 Ti
- RAM: 32 GB @ 3200 MHz

3.5.1 Evaluation methods

The collected performance data from the experiments are processed and compared. The quality and the correctness of the generated animation clips from the models are visualised and observed.

Performance evaluation Metrics

The following metrics are measured for assessing the performance of the models:

- **Reconstruction error,** $\mathcal{L}_{\text{recon}}$, the mean squared error of the generated output against the target output. It provides an overview of the generation performance.

$$\mathcal{L}_{\text{recon}} = \text{MSE}(Y_i, X_i) \tag{3.6}$$

- **Rotational error,** $\mathcal{L}_{\text{r}}$, the mean squared rotation error of the predicted joint rotations and the target joint rotations. It is the rotation proportion in the reconstruction error.

$$\mathcal{L}_{\text{r}} = \text{MSE}(y^r, x^r) \tag{3.7}$$

- **Adversarial error,** $\mathcal{L}_{\text{adv}}$, the adversarial error produced by the LSGAN discriminator. It tells the likelihood of the generated pose is sampled from the true data set. In practice, it indicates the semantic correctness of the generated poses, meaning if the joints are correctly placed in relation to each other and not falling apart.

$$\mathcal{L}_{\text{adv}} = \frac{1}{2}\mathbb{E}[(\mathbf{D}(Y^S) - 1)^2] \tag{3.8}$$

- **Root mean square roughness,** $\mathcal{L}_{RMS}$, it is the root mean square of the joint rotation angular updates. It is a similar concept as the sum of delta rotation angles that is used in the LMP-MoE paper [26], but more sensitive to the large spikes in the curves [6].

$$\mathcal{L}_{RMS} = \sqrt{\sum_{i=0}^{299} \mathbb{E}_{\text{joints}}[r_i^2]} \tag{3.9}$$

The following properties are used for quantifying the complexity of the models:

- **Number of learnable parameters,** it is a commonly used metric for quantifying the complexity of a neural network. It is the main factor for determining the number of float-point operations (FLOPs) that are needed for a forward pass.

- **Memory footprints,** it includes the size of the model as a sum of all parameters and the size of all the internal buffers.

- **Inference time on CPU,** it only provides a preliminary look at the execution time of the models, as they are not optimised for the production environment. Furthermore, the time is measured on the CPU to avoid potential synchronisation issues and the unpredicted internal data transportation to GPU.

For every comparison of multiple models, the null hypothesis is tested using the Kruskal Wallis H test [14], with a significance level at 95%. Each comparison is performed independently on every domain $\{R_1, R_2, ..., R_5\}$. The calculated p-values are corrected using Dunn–Šidák method [23] to counteract the multiple comparisons problem. The highest p-value is used to reject or accept the hypothesis.

$$\text{Let } H_0 : M_1 = M_1 = ... = M_k \text{ denote the null hypothesis} \tag{3.10}$$

$$\text{where } M = M_1, M_2, ..., M_k \text{ is the set of models to be compared.} \tag{3.11}$$

$$\text{Reject } H_0 \text{ when:} \tag{3.12}$$

$$\max_{\{R_i \in R\}} \frac{1 - (1-\alpha)^{\frac{1}{k}}}{\alpha} \cdot Kruskal(M_1(R_i), M_2(R_i), ..., M_k(R_i)) < \alpha \tag{3.13}$$

$$\text{where } \alpha = \text{ family-wise error rate } = 0.05 \tag{3.14}$$

3.6 OMG implementation details

The section presents the architectures and the internal operations of all the implemented neural networks, and the three variants of OMG models.

3.6.1 Component implementations

Autoencoder $\mathbf{AE}(\cdot)$

It consists of an encoder $\text{Enc}(\cdot)$ and a decoder$\text{Dec}(\cdot)$ with the same configuration, where both are implemented as three-layer MLP with following network operation:

$$\text{Enc}(x; W, B) = \text{Dec}(x, W, B) = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{3.15}$$

where ELU is the Exponential linear unit activation function; $W = \{w_1, w_2, w_3\}$ and $B = \{b_1, b_2, b_3\}$ are the learnable parameters of the network.

LSGAN discriminator, $\mathbf{D}(\cdot)$

It is a temporal convolutional discriminator that learns the distribution of the pose data from the true dataset and provides an adversarial loss $\mathcal{L}_{\text{adv}}$ that tells the probability of generated pose is being sampled from the real pose distribution.

$$L1 = \text{BatchNorm}(\text{MaxPool}(\text{Conv}(x, w_1, b_1))) \tag{3.16}$$

$$L2 = \text{BatchNorm}(\text{MaxPool}(\text{Conv}(L1, w_2, b_2))) \tag{3.17}$$

$$L3 = w_3(\text{Flatten}(L2)) + b_3 \tag{3.18}$$

$$D(x; W, B) = L3(L2(L1(x))) \tag{3.19}$$

where BatchNorm(·) is a batch normalisation layer, MaxPool(·) is a max-pooling layer, Conv(·) is a 2D convolutional layer and $\text{Flatten}(\cdot)$ is the flatten operation.

RBF clustering layer, $\mathbf{RBF}(\cdot)$

It uses Gaussian basis function $\phi(\cdot)$ for clustering the pose embeddings.

$$\text{RBF}(x; \mu, \sigma) = \phi(\sigma||x - \mu||) \tag{3.20}$$

With a orthogonality loss $\mathcal{L}_{ortho}$ as the regularisation term:

$$\mathcal{L}_{ortho} = 0.1 \cdot \mathbb{E}(|\mu\mu^T - I|_1) \tag{3.21}$$

VAE layer, VAE$(\cdot)$

It consists of two modules $\{\mu(\cdot), \sigma(\cdot)\}$ that outputs the parameters of C Gaussian distributions. The KL-divergence of the output distributions to the unit Gaussian distribution is used as a regularisation term.

$$\text{VAE}(x; W, B) = \{\mu = w_1 \cdot x + b_1, \sigma = w_2 \cdot x + b_2\} \tag{3.22}$$

The reparameterisation trick [15] is implemented to sample the latent vector from the computed distributions:

$$z = \mu + \sigma \cdot \varepsilon, \quad \text{where } \varepsilon \sim \mathcal{N}(0, 1) \tag{3.23}$$

With a KL-loss $\mathcal{L}_{\text{KL}}$ of the C Gaussian distributions to the normal distribution $\mathcal{N}(0, 1)$, as the regularisation term:

$$\mathcal{L}_{\text{KL}} = \beta * \mathbb{E}[-\frac{1}{2}\sum(1 + \sigma - \mu^2 - e^{\sigma})] \tag{3.24}$$

where the β is a weighing term (based on β-VAE) that is set to 0.2.

DEC layer, DEC$(\cdot)$

It outputs a probability vector of the given input vector belonging to the respective cluster centres. The KL-divergence of the output distribution to the auxiliary distribution is used as a regularisation term.

$$\text{DEC}(x; \mu) = \frac{(1 + ||x - \mu||^2)^{-\frac{1}{2}}}{\sum[(1 + ||x - \mu||^2)^{-\frac{1}{2}}]} \tag{3.25}$$

With a KL-loss $\mathcal{L}_{\text{KL}} = \text{KL}(q||p)$ of the output distribution q to the auxiliary distribution p as described in Section 2.3.4.

MoE network, MoE$(\cdot)$

It consists of a gating network $\Omega(\cdot)$ with k experts and a main network $N(\cdot)$. Both networks are implemented as three-layer MLP, where the main network has k sets of parameters

which are blended using the coefficients from the gating network. The gating network is conditioned on the local motion phase variable Θ_i and produces the blending coefficients for blending the parameters on each layer. The main network takes pose embedding z_i and cost embedding c_i as input $x = [z_i, c_i]$, and outputs local motion phase update $\Delta\Theta$, next pose embedding z_{i+1} and next cost c_{i+1}, where i denotes the ith frame.

$$\Omega(\Theta; W, B) = U = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{3.26}$$

$$N(x; W, B, U) = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{3.27}$$

$$\text{where } w_l = \sum_{e=1}^{K} W_i^{(e)} \cdot U^{(e)}, \quad l = 1, 2, 3 \tag{3.28}$$

where e denotes the e-th expert.

LSTM network, LSTM$(\cdot)$

It is a network with 4 LSTM-layers. It a many-to-many type of recurrent architecture that takes local motion phase Θ_i, pose embedding z_i and cost embedding c_i as input and outputs local motion phase update $\Delta\Theta$, next pose embedding z_{i+1} and next cost c_{i+1}. The operation details are provided in Section 2.3.7.

3.6.2 OMG implementations

OMG is a complete framework for predicting the next pose given the current pose, the objectives and local motion phase. It consists of four modules: AE$(\cdot)$ is the autoencoder for encoding and decoding the pose data; C$(\cdot)$ is the clustering layer; CostEnc$(\cdot)$ is a three-layer MLP for encoding the cost data; MoGenNet$(\cdot)$ is the main generation network. The framework is visualised in Figure 3.2 with MoE as the motion generation network.

FE-OMG

$(x;\theta)$ It consists of only AE and MoGenNet and implements the feature encoding technique. The pose encoding z is directly used as input to MoGenNet. The architecture is visualised in Figure 3.6.

$$\begin{aligned} X_i^P, X_i^S, X_i^V &= \text{slice}(X_i) \\ z_i &= \text{Enc}(X_i^S) \\ c &= \text{CostEnc}(X_i^V) \\ Y_{i+1} &= \text{MoGenNet}(X_i^P, z_i \oplus c) \\ \Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} &= \text{slice}(Y_{i+1}) \\ Y_{i+1}^P &= \text{update}(X_i^P, \Delta\Theta) \\ Y_{i+1}^S &= \text{Dec}(z_{i+1}) \\ Y_{i+1}^V &= \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\ Y_{i+1} &= Y_{i+1}^P \oplus Y_{i+1}^S \oplus Y_{i+1}^V \end{aligned} \tag{3.29}$$

The exact models are denoted as **AE+MoE** and **AE+LSTM**.

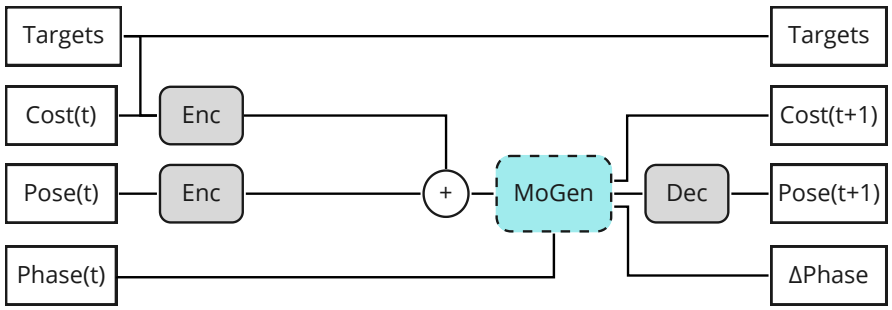


Figure 3.6: Architecture diagram of FE-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P and t is the frame index. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder and MoGenNet is the motion generation network (MoE / LSTM)

FC-IN-OMG

$(x;\theta)$ It implements the feature clustering technique. It uses a clustering layer to obtain the cluster information Ψ of the pose embeddings, which is used as input to MoGenNet. Figure 3.7 displays the architecture diagram.

$$\begin{aligned}
X_i^P, X_i^S, X_i^V &= \text{slice}(X_i) \\
z_i &= \text{Enc}(X_i^S) \\
\Psi &= \text{C}(z_i) \\
c &= \text{CostEnc}(X_i^V) \\
Y_{i+1} &= \text{MoGenNet}(X_i^P, \Psi \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} &= \text{slice}(Y_{i+1}) \\
Y_{i+1}^P &= \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S &= \text{Dec}(z_{i+1}) \\
Y_{i+1}^V &= \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} &= Y_{i+1}^p \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned} \tag{3.30}$$

The exact models are denoted as **RBF-IN+(MoGenNet)**, **VAE-IN+(MoGenNet)** and **DEC-IN+(MoGenNet)**.

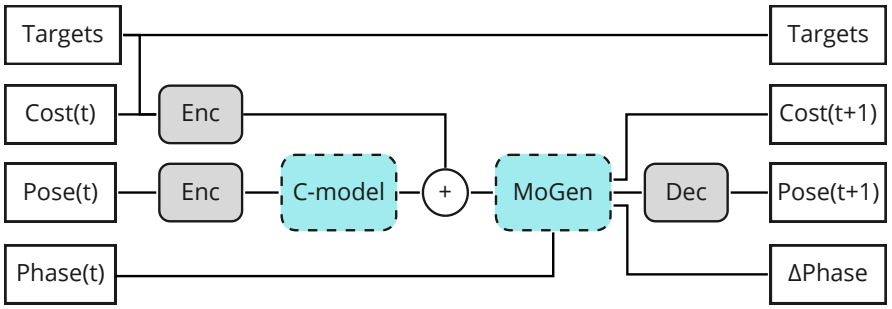


Figure 3.7: Architecture diagram of FC-IN-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder, C-model denotes the clustering layer (RBF/VAE/DEC) and MoGenNet is the motion generation network (MoE / LSTM)

FC-CAT-OMG

$(x;\theta)$ It is the other architecture of FC-OMG, where it uses Ψ as auxiliary features, that are concatenated with the pose embedding z before feeding to MoGenNet. The network is illustrated in Figure 3.8.

$$\begin{aligned}
X_i^P, X_i^S, X_i^V &= \text{slice}(X_i) \\
z_i &= \text{Enc}(X_i^S) \\
\Psi &= \text{C}(z_i) \\
c &= \text{CostEnc}(X_i^V) \\
Y_{i+1} &= \text{MoGenNet}(X_i^P, z_i \oplus \Psi \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} &= \text{slice}(Y_{i+1}) \\
Y_{i+1}^P &= \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S &= \text{Dec}(z_{i+1}) \\
Y_{i+1}^V &= \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} &= Y_{i+1}^P \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned} \tag{3.31}$$

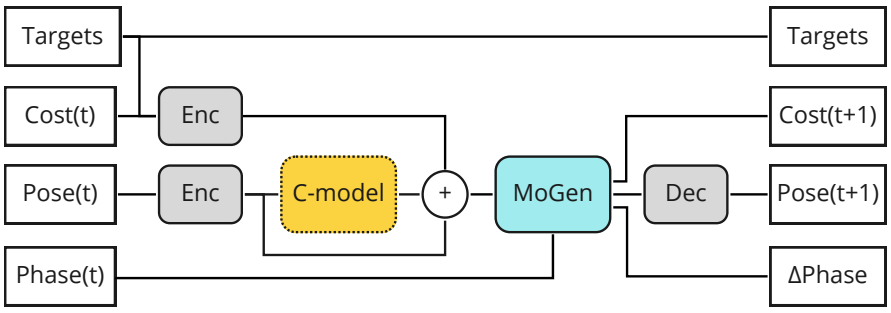


Figure 3.8: Architecture diagram of FC-CAT-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder, C-model denotes the clustering layer (RBF/VAE/DEC) and MoGenNet is the motion generation network (MoE / LSTM)

where $\oplus$ denotes concatenate operation, slice$(\cdot)$ is the slicing operation splitting the vector into components on axis 1 (columns). update$(\cdot)$ is the local motion phase update function:

$$\text{update}(X_i^p, \Delta\Theta) = 0.9 \cdot X_i^p + 0.1 \cdot \Delta\Theta \tag{3.32}$$

The exact models are denoted as **RBF-CAT + (MoGenNet)**, **VAE-CAT + (MoGenNet)** and **DEC-CAT + (MoGenNet)**.

FS-OMG

$(x;\theta)$ It refers to the FE-OMG and the FC-OMG models that implement the feature selection technique. The input pose vectors to the framework only contain information about 6 joints (feet, hands, spine and head) and the output vectors contain full-resolution pose information. **FS-CAT-OMG** and **FS-IN-OMG** correspond to the FC-OMG variants.

3.6.3 Model configurations

The following configuration of the respective network is used for the experiments.

Model	Input dimension	Output dimension	hidden dimension/kernel size
Enc	IN	256	256
Dec	256	OUT	256
D	256	1	3×3
RBF	256	128	-
DEC	256	128	-
VAE	256	128,128	-
MoE	8, 256/384	8,256,24	256
LSTM	8, 256/384	8,256,24	256
CostEnc	60	128	128

where IN and OUT are the dimension of pose vector.

The special parameters of certain models are summarised in the table below:

Model	Parameter	Value
RBF	C	128
VAE	C	128
DEC	C	128
MoE	k	4
MoE	Gate size	128
LSTM	Layers	4

Chapter 4

Results

4.1 Generation performance of OMG models

Experiment

Train the AE and its LSGAN discriminator on all domains (R1-R5) for 200 epochs. Incorporate the AE and the LSGAN into the respective OMG models: FE-OMG, FC-OMG and FS-OMG. Train the OMG models on all domains for 51 epochs, with the autoregression probability starting at 0 and incremented with 0.5 every 20 epochs.

80% of the dataset is used as training data, 10% is used for validation and 20% is used for testing (including the validation set). During the testing, for each of the test samples, the four performance metrics described in Section 3.5.1, are measured and recorded.

Results

The experiment produces a set of trained OMG models with reference performance data on all domains. The results demonstrate the level of performance that is expected from the models that are fully trained with the whole dataset. The performance data of the models are compared to determine which OMG implementation achieves the highest generation performance.

The results in this chapter are presented in the form of comparisons between models or RAE approaches. The percentage difference in each of the metrics between the baseline model(s) and the target models are visualised in the graphs. The corrected p-values (P) from the null hypothesis tests of the models (baseline vs target, target vs target), as described in Section 3.5.1, are listed in the tables, where $P \geq 0.05$ are bold-annotated. A p-value is provided for every statement made in this section. If the value is less than 0.001 and also presented in a table, then "($P < .001$)" is used.

This chapter only presents comparisons in the reconstruction error (Re. error) and the root mean square roughness (RMS) because they are the most interesting and informative metrics.

The rotation error (Rot. error) is included in Re. error, hence it is omitted to make the graphs and the tables less cluttered. There are not much adversarial error (Adv. error) differences between the models, as illustrated in Fig. 4.1, which could be a result of the training schema as discussed in Section 5.3. Despite the statistical analysis of all models rejecting the null hypothesis ($P_{Adv} = 2.88\text{e}{-17}$), the post hoc analysis reveals that most of the differences between AE+MoE and FC+MoE models are statistically insignificant, as presented in Table. 4.1. The statistical test of AE+MoE and all FC-CAT+MoE models shows $P_{Adv} = .980$, hence the adversarial errors are also omitted in this chapter. The list of the complete result numbers can be found in Appendix B.

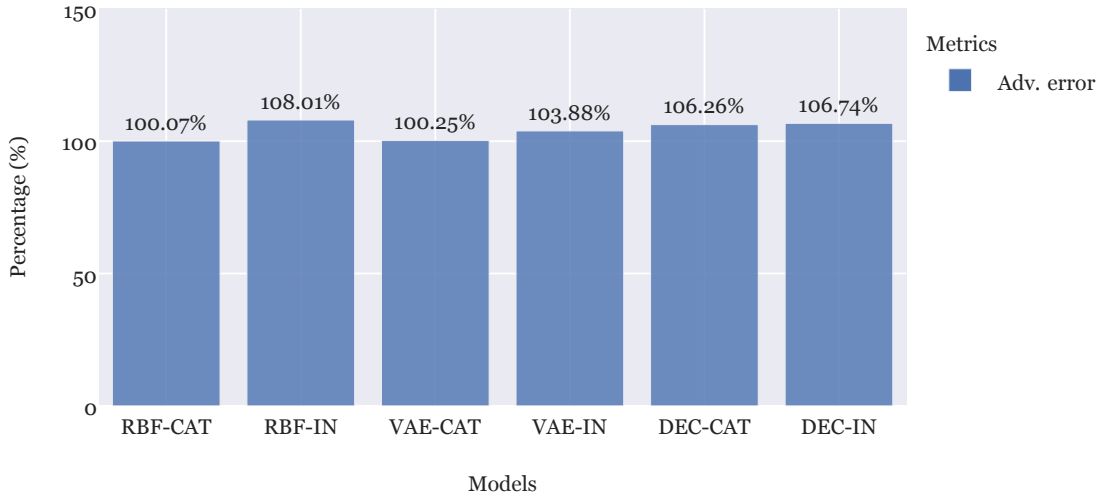


Figure 4.1: Percentage differences in Adv. error between AE+MoE (target) and the FC+MoE models (baselines). Each bar illustrates how much higher Adv. error does the target model have when compared to the baseline models. The differences are all single-digits. Only DEC-IN+MoE model shows statistically significant difference when compared with the target model. ($P < .001$).

Table 4.1: p-values of Fig. 4.1

	RBF-CAT	RBF-IN	VAE-CAT	VAE-IN	DEC-CAT	DEC-IN
Adv. error	**9.16e-01**	**8.65e-02**	**9.35e-01**	**7.67e-01**	**3.93e-01**	6.06e-07

The p-values of the model comparisons (FE vs FC). It shows that most of the differences between the FC models and the FE model are not significant.

MoE vs LSTM

This section presents the comparison of MoE and LSTM as the MoGenNet for OMG. Performing the statistical analysis shows that all models are not equal ($P_{Re} = 6.43\text{e}{-105}$, $P_{RMS} = 1.37\text{e}{-102}$).

Fig. 4.2 displays the comparison of the OMG+MoE models against their LSTM counterparts. For instance, the AE bars illustrate the differences in Re. error and RMS of AE+MoE compared to AE+LSTM, where the former achieves approximately 68% lower Re. error ($P < .001$), and 41% lower RMS ($P < .001$). The p-values of all comparisons are presented in Table 4.2.

The results indicate that LSTM is the worse candidate as MoGenNet for OMG. From here and on, only the MoE variants are considered and presented. In the following sections, the notations 'FE', 'AE' and 'AE+MoE' are used interchangeably to denote the AE+MoE model using the FE approach.

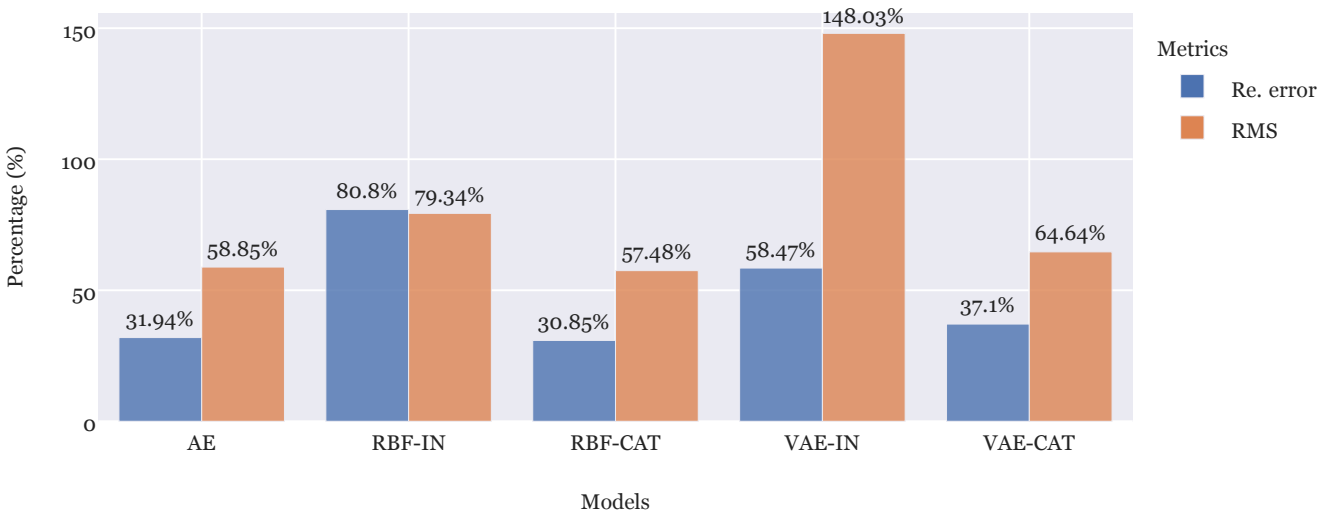


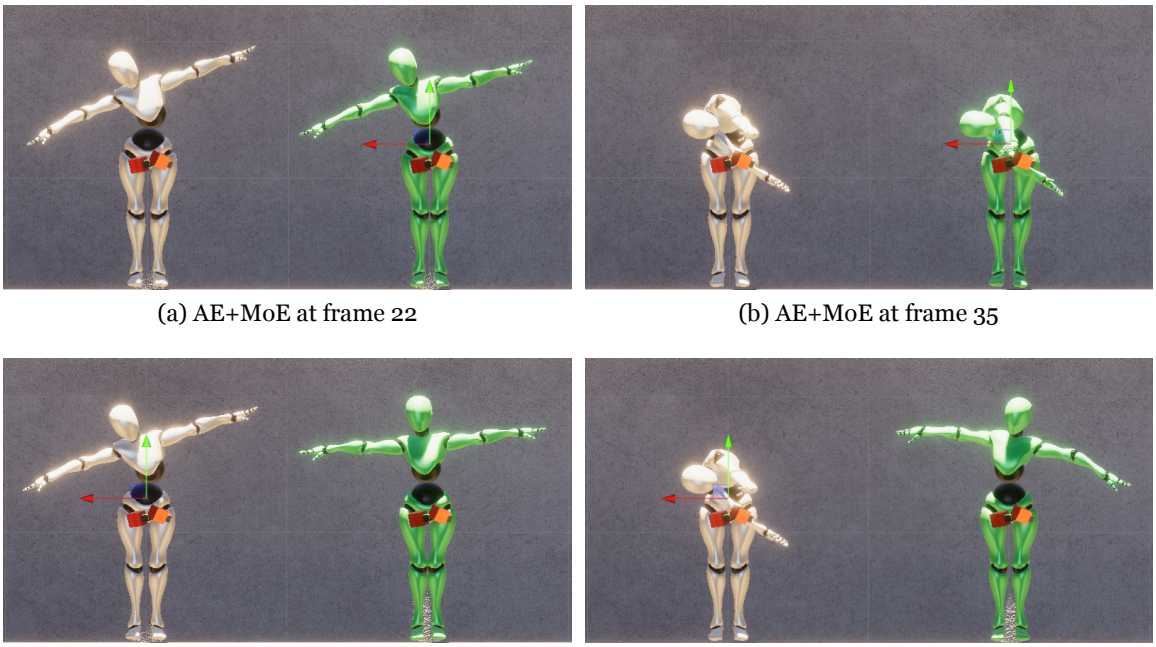
Figure 4.2: Percentage differences in Re. error and RMS when comparing the OMG+MoE models (targets) against their LSTM counterparts (baselines). Each bar illustrate how much lower errors the target model achieves in the respective metric. For instance, The Re. error of AE+MoE is 32% of the Re. error of AE+LSTM. A lower bar (< 100%) indicates that the target model is better and vice versa. The results show that LSTM is worse than MoE for all models. The advantage of VAE-IN+LSTM in RMS is not statistically significant ($P = .170$).

Table 4.2: p-values of Fig. 4.2

	AE	RBF-IN	RBF-CAT	VAE-IN	VAE-CAT
Re. error	2.73e-41	1.57e-02	5.53e-49	9.28e-12	4.85e-36
RMS	6.12e-06	3.94e-04	7.72e-21	**1.70e-01**	2.48e-11

The p-values of the model comparisons (MoE vs LSTM). Most of the performance differences between the OMG+MoE models and the OMG+LSTM models are significant, except for the RMS of the VAE-IN models.

Fig. 4.3 demonstrates the generated poses (green) from AE+MoE and AE+LSTM and the target poses (white), at two different frames. At the frame 35, AE+MoE managed to generate the correct pose while AE+LSTM failed.



(a) AE+MoE at frame 22

(b) AE+MoE at frame 35

(c) AE+LSTM at frame 22

(d) AE+LSTM at frame 35

Figure 4.3: The screenshots of the generated poses (the green character) by AE+MoE and AE+LSTM, compared against the target poses (the white characeter). AE+LSTM failed to generated correct post at the frame 35.

FC-IN vs FC-CAT

Fig. 4.4 visualises the comparison between the FC-CAT models against the FC-IN models. The models are not all equal ($P_{Re} = 1.76\text{e}{-8}$, $P_{RMS} = 7.46\text{e}{-7}$). The results indicate that FC-CAT models are generally better than FC-IN models. RBF-CAT has approximately 73% lower Re. error ($P < .001$) and 27% lower RMS ($P < .001$) than RBF-IN. VAE-CAT

achieves roughly 57% lower Re. error ($P < .001$) and 61% lower RMS ($P < .001$). Only the DEC variant shows insignificant differences in Re. error ($P = .586$), but DEC-CAT has lower RMS than DEC-IN ($P < .001$). Table 4.3 contains the p-values of all comparisons. Some samples of the generated animations are included in Appendix A.

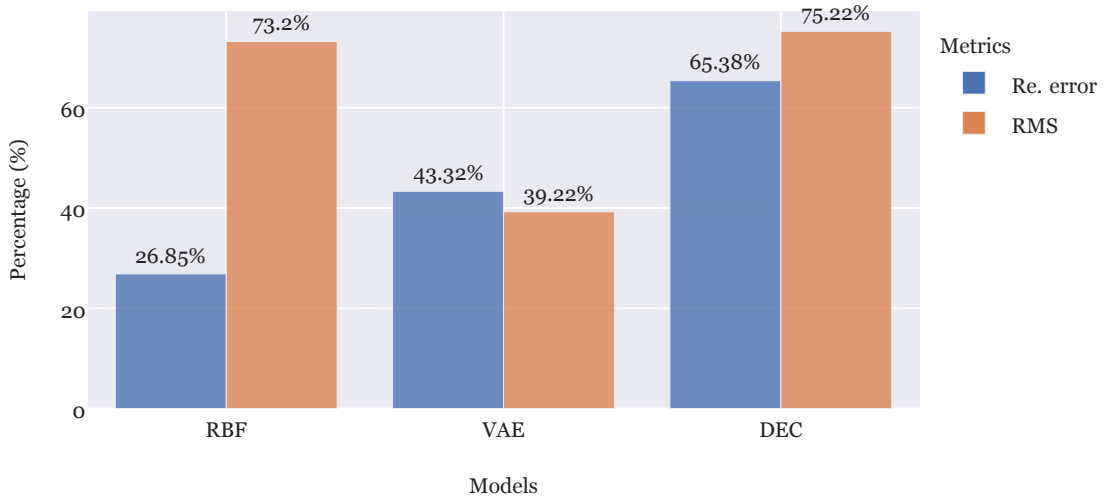


Figure 4.4: Percentage differences in Re. error and RMS when comparing the FC-CAT models (targets) against the corresponding FC-IN models (baselines). Each bar illustrate how much lower errors the target model achieves in the respective metric. The results show that FC-CAT is significantly better than FC-IN for RBF and VAE.

Table 4.3: p-values of Fig. 4.4

	RBF	VAE	DEC
Re. error	2.95e-55	8.45e-32	**5.86e-01**
RMS	1.61e-10	4.10e-63	3.95e-08

The p-values of the model comparisons (FC-IN vs FC-CAT). Most of the differences are significant except for the Re. error of the DEC models.

FE vs FC-CAT

Fig. 4.5 presents the comparison between the FC-CAT models against the FE model (AE+MoE). Both RBF-CAT and VAE-CAT perform similarly as AE+MoE. The statistical analysis on all models shows $P_{Re} = .003$ and $P_{RMS} = .136$, implying that there are no significant differences in RMS between the models. Table 4.4 presents the p-values of the comparisons and most of them are not statistically significant. The only significant

findings are that DEC-CAT has roughly 2 times larger Re. error than the other models ($P < .001$).

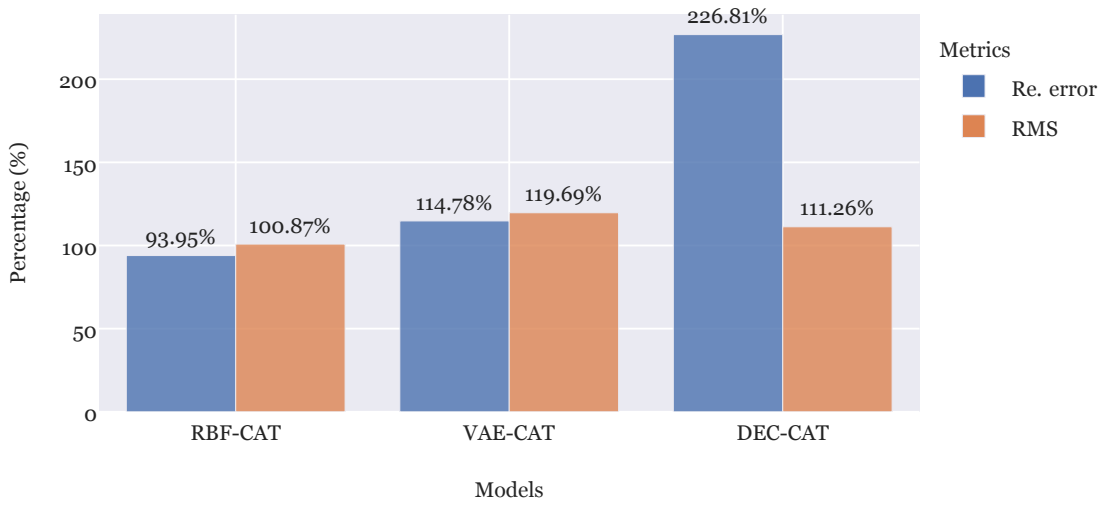


Figure 4.5: Percentage differences in Re. error and RMS when comparing the FC-CAT models (targets) against the FE model (baseline). Each bar illustrate how much lower errors the target model achieves in the respective metric. It shows that DEC-CAT is the worst model in terms of Re. error.

Table 4.4: p-values of Fig. 4.5

Re. / RMS	AE	RBF-CAT	VAE-CAT	DEC-CAT
AE	\	2.04e-02	**6.16e-01**	4.89e-13
RBF-CAT	**9.71e-01**	\	**2.61e-01**	3.20e-24
VAE-CAT	**9.46e-02**	**7.54e-02**	\	1.42e-09
DEC-CAT	**9.11e-01**	3.67e-03	9.30e-03	\

The p-values of the model comparisons (FE vs FC-CAT), The upper matrix contains the p-values of testing Re. errors of the models. The lower matrix are for RMS. The comparison in Fig. 4.5 corresponds to the first row and column of this table. The results indicate that most of the RMS differences between the models are insignificant.

Summary

The MoE-based OMG models are capable of learning from the datasets and generating correct animations with good quality.

In response to RQ.1, the results show that MoE is the better choice as the MoGenNet for OMG models. It has demonstrated statistically significant numerical advantages over the LSTM variants ($P_{Re} = 6.43\text{e}{-105}$, $P_{RMS} = 1.37\text{e}{-102}$), while also being able to generate more correct animation poses. This observation aligns with the results that are reported by Starke et al. [26, 29]. The explanation is that the LSTM models fail to capture the different motion variants, hence converging to some generic responses irrespective of the control signals.

Comparing the two architectures of FC-OMG models shows that the RBF-CAT and VAE-CAT are remarkably superior to their FC-IN counterparts ($P_{Re} < .001, P_{RMS} < .001$). It is expected since the input to the MoGenNet in FC-CAT models is the extracted features from the encoder and the clustering vector from the clustering layer. They have a higher capacity and complexity than their FC-IN counterparts.

The RMS differences between FC-CAT-OMG and FE-OMG are mostly statistically insignificant, as seen in Table 4.4. DEC-CAT is considerably worse than the other models in terms of Re. error ($P < .001$). In terms of animation quality, both the FE model and the RBF-CAT model can produce near-identical animations as the target animations. Although the VAE-CAT model achieves similar numerical results, it produces higher RMS which translate to a visible amount of jittering in the generated animations. It might be explained by the variational sampling mechanism in the VAE layer.

4.2 Transferring quality

Experiment

Train the OMG models on R2-R5 respectively for 31 epochs, with the autoregression probability incremented with 0.5 every 10 epochs. Only 16% of the dataset (20% of the training set) is used for training, 10% is used for validation and 84% is used for testing (including the validation set and the remainder of the training set). The experiment is performed for each of the training strategies:

- **Raw:** The model parameters are randomly initialized using the Xavier initialisation [7].
- **Frozen:** The models are warm-started by loading the parameters from the pre-trained instances from the experiment in Section 4.1, that are trained on R_1. The *frozen* strategy is applied, where only the AE is allowed to be optimised during the training.
- **Trained:** Same procedure as **Frozen** but applies the *trained* strategy, where all modules are optimised during the training.

Results

The experiment produces a set of transferred OMG models with the performance data on the domains (R2-R5). The experiment **Raw** demonstrates the vanilla performance of the models that are trained with limited training and data. **Frozen** aims to provide insight into the transfer learning capability of the RAE approaches. **Trained** demonstrates the highest possible performance of the models under these training conditions. The results are compared to determine which RAE approach achieves the highest transferring quality.

This section presents the comparison of the transferred models against their reference counterparts. The percentage differences are visualised in the graphs and the p-values of the statistical analyses of the models are presented in the tables. The actual performance values can be found in Appendix B.2.

FE-OMG

Fig. 4.6 presents the comparison of the transferred performance of FE-OMG against its reference performance. There are statistically significant differences between the models ($P_{Re} = 1.48\text{e}{-67}, P_{RMS} = 1.44\text{e}{-6}$). The **Raw** Re. error is approximately 350% higher than the reference Re. error ($P < .001$). Using the warm-starting technique with the *frozen* strategy decreases the Re. error to only being 168% higher than the reference Re. error ($P < .001$). The improvement in RMS is not significant as seen in Table 4.5. Unexpectedly, using the *trained* strategy does not yield any significant improvement over the *frozen* strategy, in terms of Re. error ($P = .844$).

Fig. 4.7 visualises the comparison of **Frozen** and **Trained** against **Raw**. It confirms that the Re. error is almost halved when warm-starting the model ($P < .001$).

Figure 4.6: Percentage differences in Re. error and RMS when comparing the transferred FE model (target) against the reference model (baseline). Each bar illustrate how much higher errors the target model achieves in the respective metric. It shows that when cold-started, the model achieves roughly 350% higher Re. error. When warm-started, the Re. error is only 170% higher than the reference value.

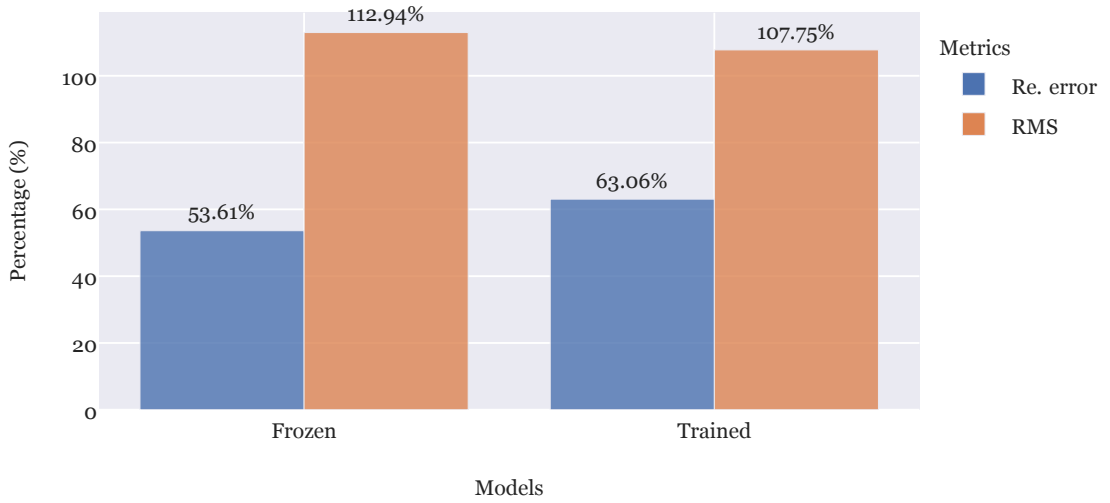


Figure 4.7: Percentage differences in Re. error and RMS when comparing **Frozen** and **Trained** (targets) against **Raw** (baseline), for the FE model. Each bar illustrate how much lower errors the target model achieves in the respective metric. Warm-starting the model with the *frozen* strategy reduces the Re. error by approximately 50%.

Table 4.5: p-values of Fig. 4.6 and 4.7

	REF vs Raw	REF vs F.	REF vs T.	Raw vs F.	Raw vs T.	F. vs T.
Re. error	2.49e-75	1.50e-11	1.90e-13	3.90e-62	1.17e-12	**8.44e-01**
RMS	2.40e-03	**3.15e-01**	**3.74e-01**	1.76e-03	**2.09e-01**	1.12e-02

The p-values of the performance comparison of FE. REF denotes the reference performance, F the **Frozen** performance and T the **Trained** performance. The RMS differences between the reference FE model and the transferred models are neglectable. The Re. error difference between **Frozen** and **Trained** is also insignificant.

FC-OMG: RBF

Fig. 4.8 displays the comparison of the transferred RBF-IN and RBF-CAT against their reference counterparts. The null hypothesis that all models are equal is rejected with $P_{Re} = 7.29\text{e}{-75}, P_{RMS} = 2.87\text{e}{-3}$. Surprisingly, the RBF-IN models demonstrate near reference-level performance, even when cold-started ($P < .001$). Similar to the FE model, The **Raw** RBF-CAT model has approximately 350% higher Re. error, which is reduced by roughly 40% if warm-staring the model. The *trained* strategy does not yield any statistically significant improvement, as seen in Table 4.6.

Fig. 4.9 highlights the comparison of the warm-started RBF-CAT and RBF-IN models against their cold-started equivalent. The figure shows that only warm-starting the RBF-IN model with the *trained* strategy yields a statistically significant reduction in Re. error ($P_{Re} = .005$), whereas it is greatly reduced on the RBF-CAT model with the *frozen* strategy ($P_{Re} < .001$).

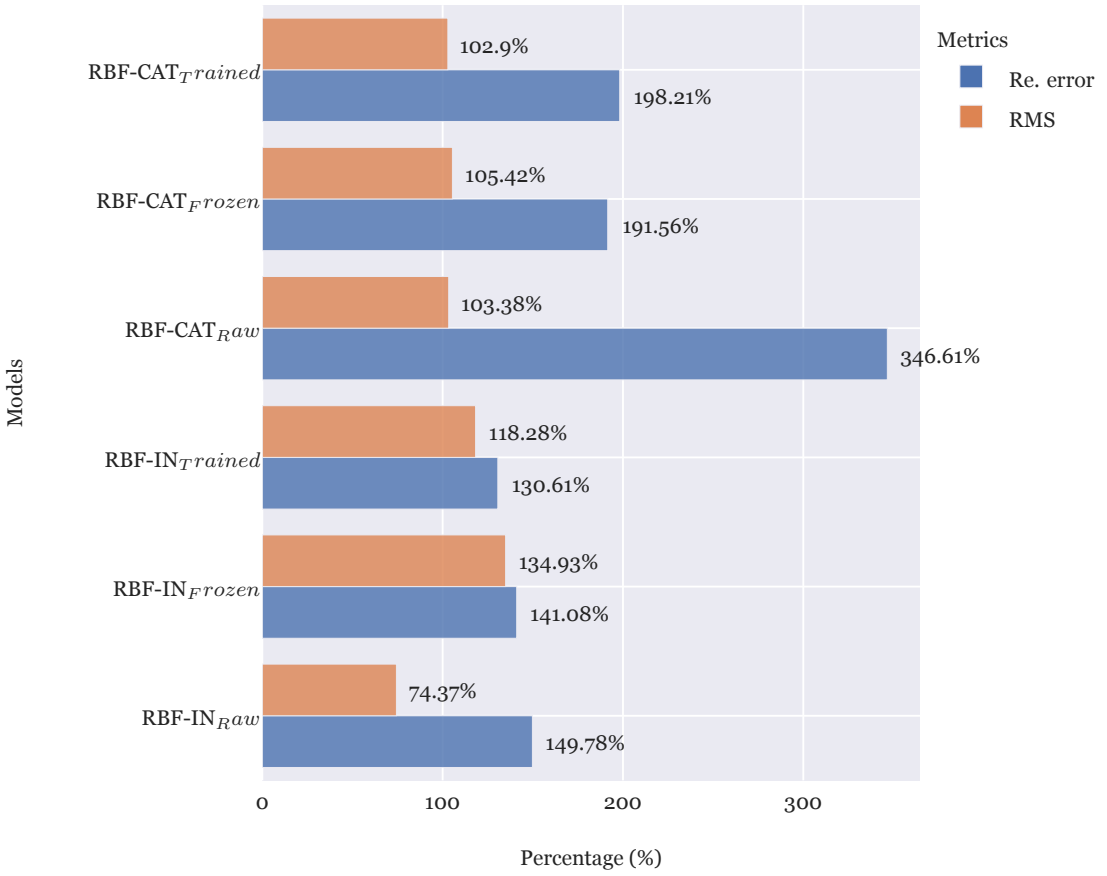


Figure 4.8: Percentage differences in Re. error and RMS when comparing the transferred RBF models (target) against the reference models (baseline). Each bar illustrate how much higher errors the target model achieves in the respective metric. It shows that the RBF-IN model achieves near reference-level performance irrespective of the training strategy. The RBF-CAT model demonstrates similar pattern as the FE model, with the Re. error reduced by roughly 40% when warm-starting the model with the *frozen* strategy.

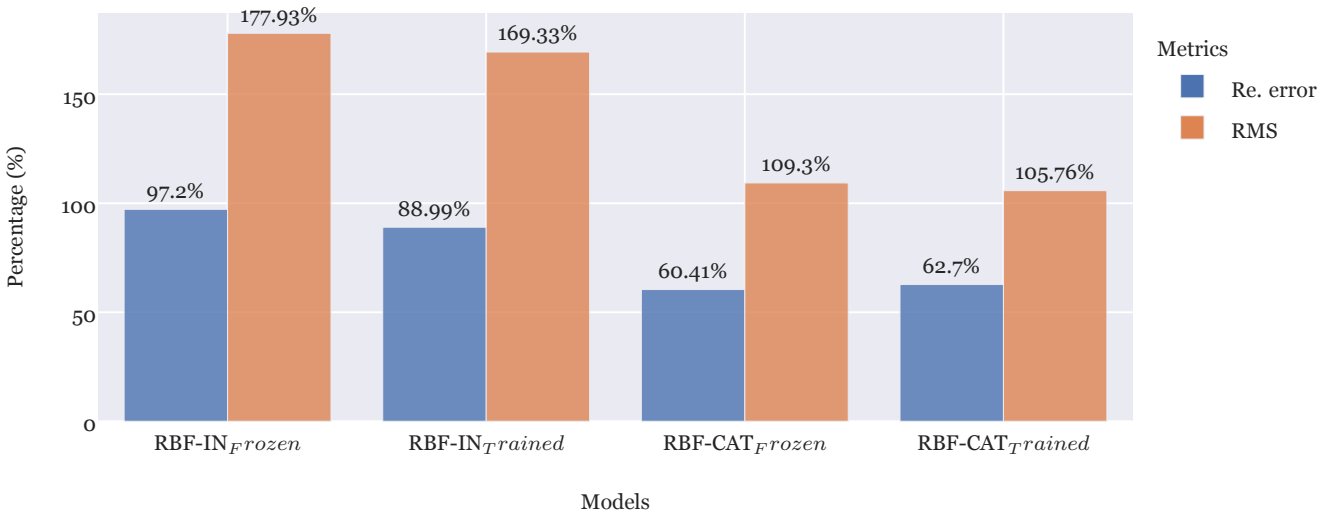


Figure 4.9: Percentage differences in Re. error and RMS when comparing **Frozen** and **Trained** (targets) against **Raw** (baseline), for the RBF-IN and RBF-CAT models. Each bar illustrate how much lower errors the target model achieves in the respective metric. Warm-starting the model reduces the Re. error by approximately 40%.

Table 4.6: p-values of Fig. 4.8 and 4.9

	REF	IN_RAW	IN_F	IN_T	CAT_RAW	CAT_F	CAT_T
REF	\	2.33e-29	1.02e-06	1.40e-06	3.65e-54	2.70e-12	1.80e-02
IN_RAW	1.29e-50	\	**9.268e-01**	4.814e-03	2.275e-47	1.975e-151	9.892e-157
IN_F	**6.70e-01**	1.057e-14	\	**5.444e-01**	2.377e-27	5.731e-128	4.989e-133
IN_T	**1.33e-01**	9.567e-40	**4.567e-01**	\	9.680e-22	2.195e-141	6.122e-148
CAT_RAW	8.78e-41	**5.280e-01**	2.229e-21	4.144e-59	\	2.462e-57	1.401e-57
CAT_F	1.32e-14	**1.167e-01**	4.728e-33	1.741e-66	1.000e-03	\	**1.284e-01**
CAT_T	**8.10e-02**	**1.993e-01**	7.670e-24	1.349e-62	**4.072e-01**	**6.821e-01**	\

The p-values of the performance comparison of FC: RBF. REF denotes the reference performance, IN_RAW denotes the **Raw** RBF-IN performance, CAT_T the **Trained** RBF-CAT performance et cetera. The upper matrix contains the p-values of testing Re. errors of the models. The lower matrix are for RMS. The differences between **Frozen** and **Trained** are insignificant for both RBF-IN and RBF-CAT.

FC-OMG: VAE

Fig. 4.10 and 4.11 visualise the transferring quality of VAE-IN and VAE-CAT. The statistical analysis indicate that all models are not equal with $P_{Re} = 8.20\text{e}{-71}, P_{RMS} = 8.42\text{e}{-138}$. Unlike the RBF models and the FE model, using the *trained* strategy further reduces RMS of VAE-CAT ($P < .001$). By warm-starting the model, it reduces Re. error for both VAE-IN and VAE-CAT. **Trained** VAE-IN achieves similar Re. error reduction as **Frozen** VAE-CAT. Table 4.7 presents the p-values of the performance comparison.

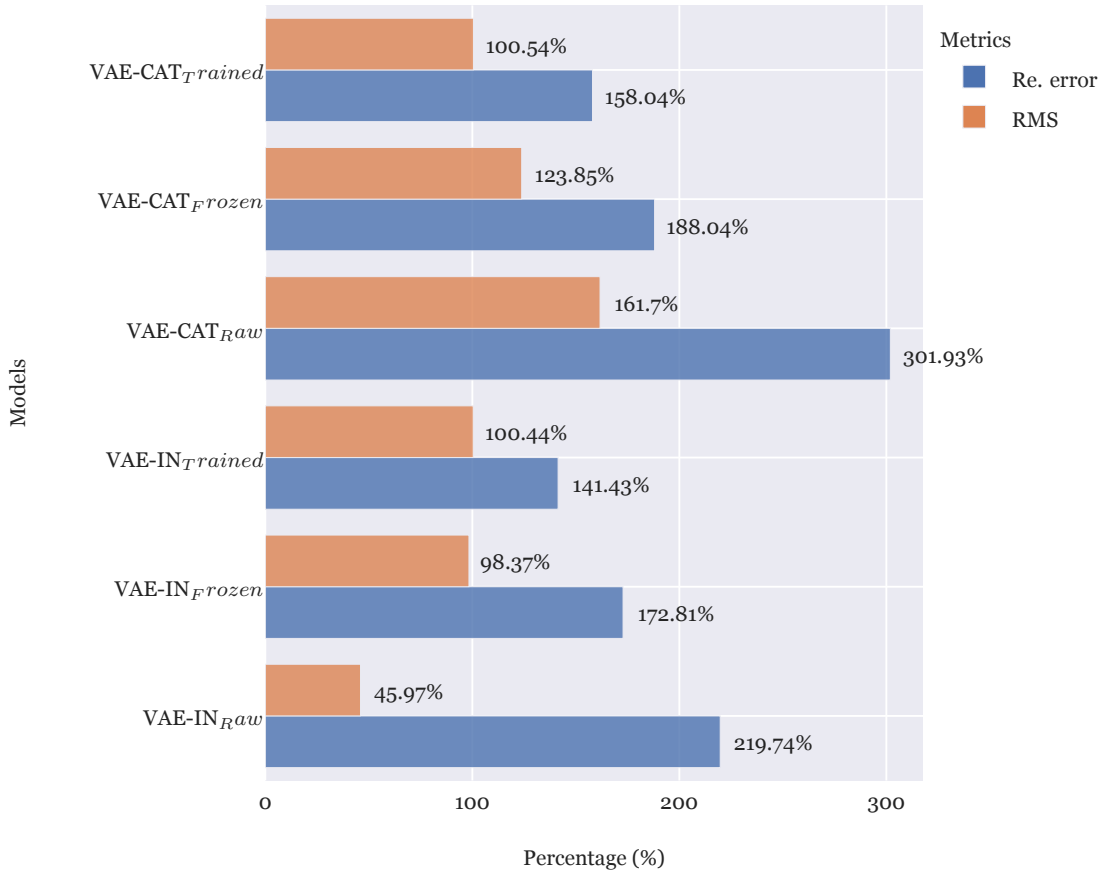


Figure 4.10: Percentage differences in Re. error and RMS when comparing the transferred VAE models (target) against the reference models (baseline). Each bar illustrate how much higher errors the target model achieves in the respective metric. It shows that Re. error is reduced for both variants when warm-started, and VAE-IN manages to reach closer to its reference-level performance than VAE-CAT ($P < .001$).

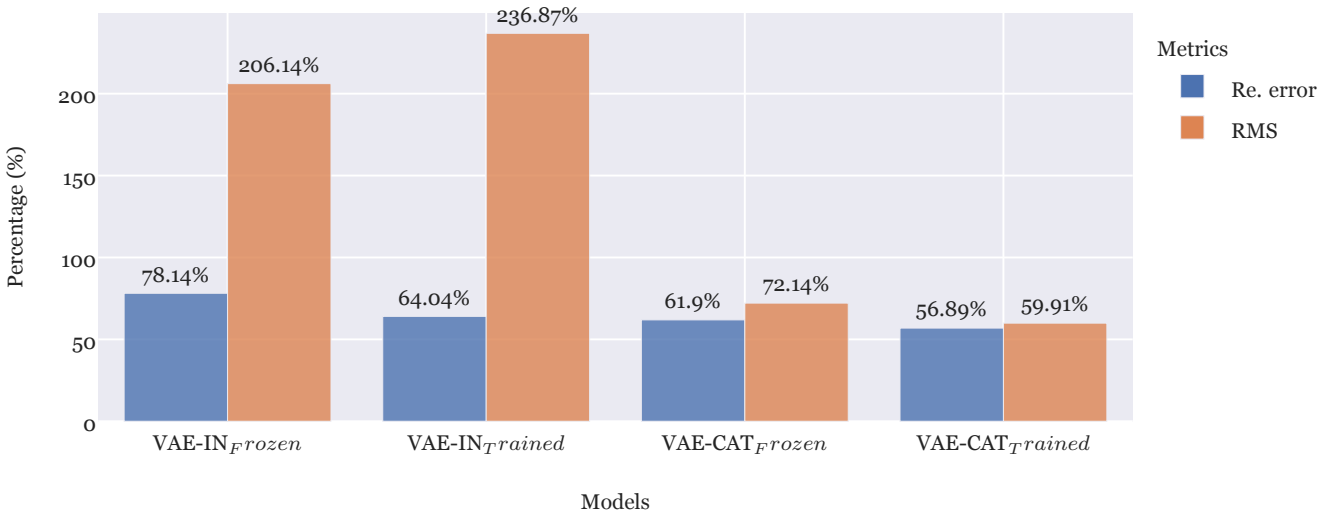


Figure 4.11: Percentage differences in Re. error and RMS when comparing **Frozen** and **Trained** (targets) against **Raw** (baseline), for the VAE-IN and VAE-CAT models. Each bar illustrate how much lower errors the target model achieves in the respective metric. Warm-starting the model reduces the Re. error by approximately 40% for both variants. The *trained* strategy does not yield any further reduction in Re. error ($P_{IN} = .995$, $P_{CAT} = .628$).

Table 4.7: p-values of Fig. 4.10 and 4.11

	REF	IN_RAW	IN_F	IN_T	CAT_RAW	CAT_F	CAT_T
REF	\	1.95e-09	1.64e-04	1.95e-02	5.96e-85	8.17e-16	6.31e-34
IN_RAW	2.67e-06	\	9.56e-03	3.64e-30	3.61e-72	1.87e-183	1.29e-165
IN_F	**9.84e-01**	1.06e-17	\	**9.95e-02**	2.03e-14	6.24e-120	5.39e-97
IN_T	6.16e-04	7.66e-132	4.31e-21	\	**7.64e-01**	3.45e-97	1.20e-102
CAT_RAW	**5.45e-01**	9.41e-44	2.23e-43	3.44e-14	\	1.77e-42	1.69e-42
CAT_F	**8.78e-01**	3.91e-03	5.44e-135	6.32e-224	4.54e-33	\	**6.28e-02**
CAT_T	**1.46e-01**	5.92e-11	7.46e-164	1.07e-257	7.97e-120	3.22e-08	\

The p-values of the performance comparison of FC: VAE. REF denotes the reference performance, IN_RAW denotes the **Raw** VAE-IN performance, CAT_T the **Trained** VAE-CAT performance et cetera. The Re. error differences between **Frozen** and **Trained** are insignificant for both VAE-IN and VAE-CAT.

FC-OMG: DEC

Fig. 4.12 and 4.13 displays the transferring quality of DEC-IN and DEC-CAT. The statistical analysis indicate that all models are not equal with $P_{Re} = 2e{-}72, P_{RMS} = 1.26e{-}35$. Unlike the previous two FC models, warm-staring does not yield any statistically significant reduction in Re. error for DEC-IN ($P = .183$). It is even increased for DEC-CAT ($P < .001$). The p-values of the comparisons can be found in Table 4.8.

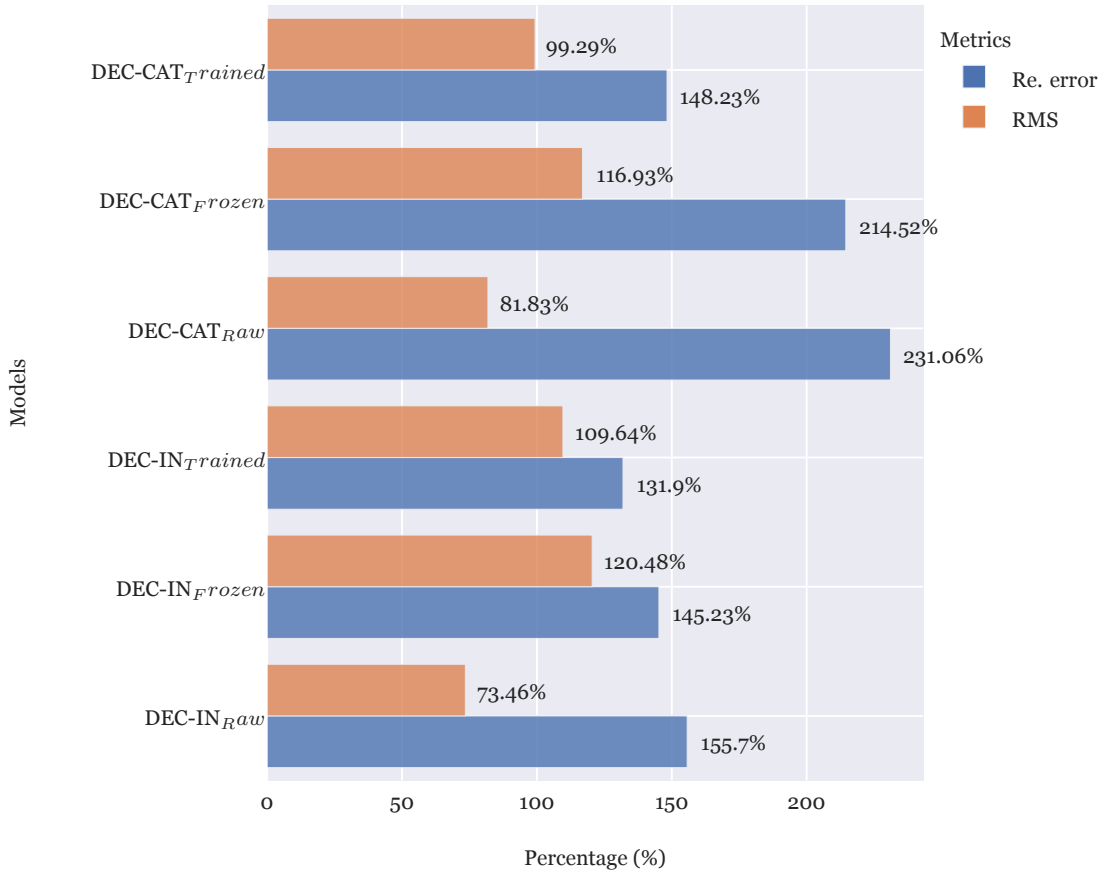


Figure 4.12: Percentage differences in Re. error and RMS when comparing the transferred DEC models (target) against the reference models (baseline). Each bar illustrate how much higher errors the target model achieves in the respective metric. It shows that warm-starting the models does not yield any significant reduction in Re. error, whether statistically or numerically.

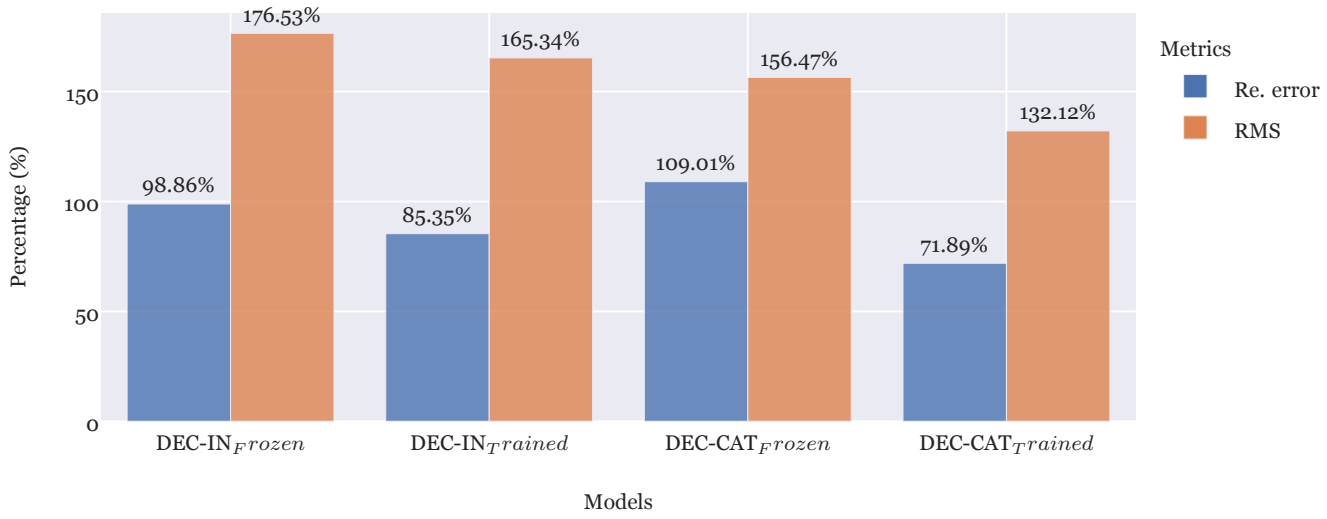


Figure 4.13: Percentage differences in Re. error and RMS when comparing **Frozen** and **Trained** (targets) against **Raw** (baseline), for the DEC-IN and DEC-CAT models. Each bar illustrate how much lower errors the target model achieves in the respective metric. Warm-starting the model does not yield any significant reduction in Re. error for **Frozen**. The *trained* strategy does reduce the Re. error and RMS for both variants ($P < .001$).

Table 4.8: p-values of Fig. 4.12 and 4.13

	REF	IN_RAW	IN_F	IN_T	CAT_RAW	CAT_F	CAT_T
REF	\	9.40e-09	7.85e-06	2.46e-02	1.21e-05	**6.95e-02**	4.37e-05
IN_RAW	1.80e-14	\	**1.83e-01**	5.84e-07	**5.32e-02**	4.57e-17	4.00e-78
IN_F	8.24e-05	1.01e-12	\	4.30e-04	**3.33e-01**	3.44e-08	9.27e-68
IN_T	**9.92e-01**	4.65e-31	1.48e-04	\	6.86e-06	**6.54e-01**	1.93e-40
CAT_RAW	**7.85e-01**	3.74e-03	4.08e-39	1.88e-24	\	2.88e-12	3.69e-33
CAT_F	9.50e-03	8.40e-08	**8.04e-01**	3.73e-08	5.64e-35	\	6.96e-18
CAT_T	**9.93e-01**	**7.07e-02**	2.70e-08	7.52e-25	2.29e-15	1.20e-05	\

The p-values of the performance comparison of FC: DEC. REF denotes the reference performance, IN_RAW denotes the **Raw** DEC-IN performance, CAT_T the **Trained** DEC-CAT performance Et cetera. The differences between **Frozen** and **Trained** are statistically significant for both DEC-IN and DEC-CAT.

FS-OMG

This section presents the transferring quality of the models using the FS approach. FS-CAT models are trained with the reduced feature set and reduced data set from R2 - R5. As input, the models take a pose of only 6 joints and output a full-resolution pose for the

next frame. Fig. 4.14 visualises how the warm-started models perform when compared to the reference performance of the FE model. The **Raw** performance data is not reported due to missing data. The statistical analysis indicate that all models are not equal with $P_{Re} = 4.43\text{e}{-56}, P_{RMS} = 2.18\text{e}{-55}$. The p-values of all comparisons can be found in Table 4.9.

Using the *trained* strategy does not yield any statistically significant reduction in Re. error for all models except VAE-CAT and DEC-CAT ($P < .001$). Fig. 4.15 shows the screenshots of the frame 35 of the generated clip by the models. Only RBF-CAT has managed to produce a somewhat correct pose at that specific frame.

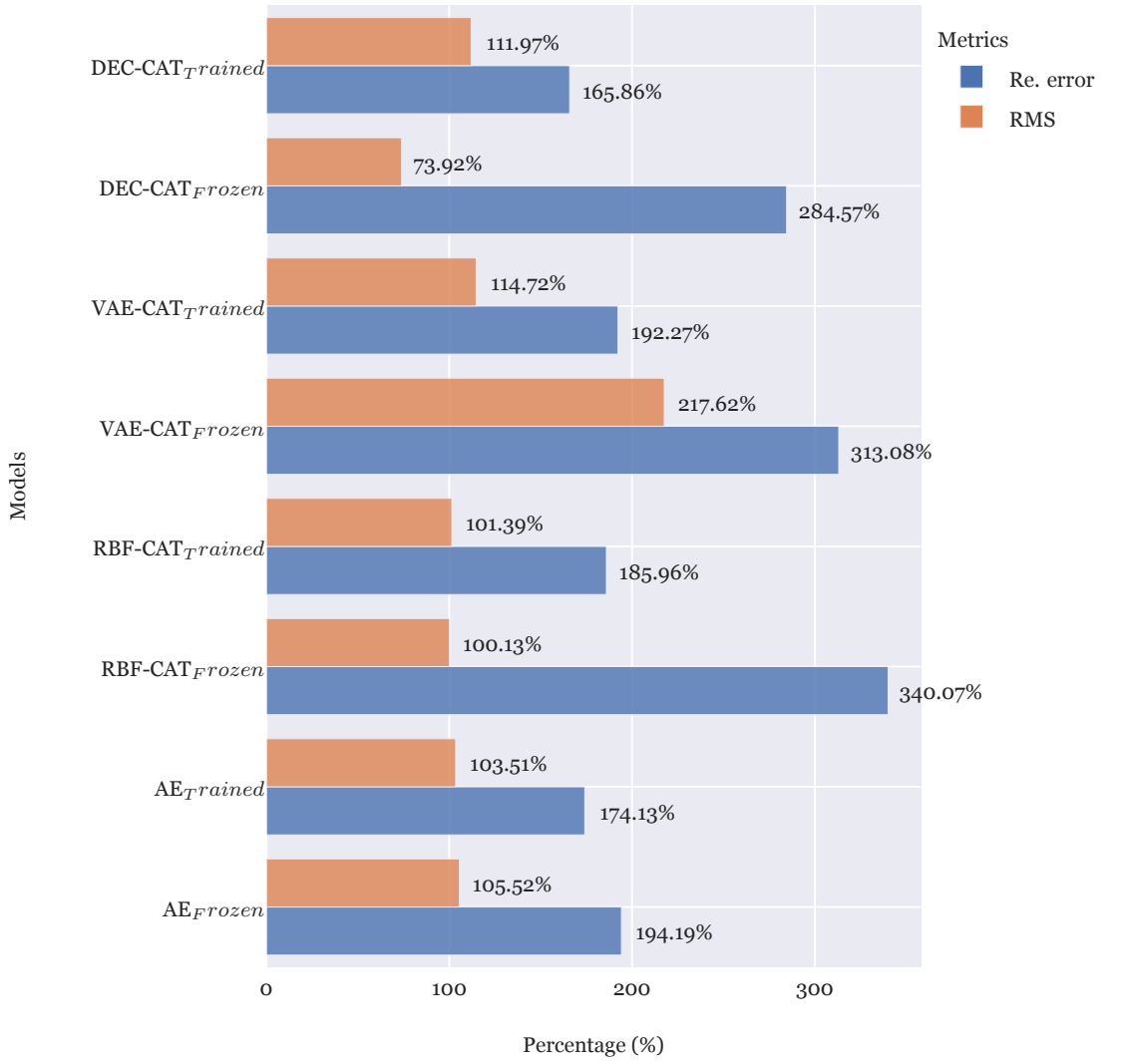


Figure 4.14: Percentage differences in Re. error and RMS when comparing the transferred FS models (target) against the reference models (baseline). Each bar illustrate how much higher errors the target model achieves in the respective metric.

Table 4.9: p-values of Fig. 4.14

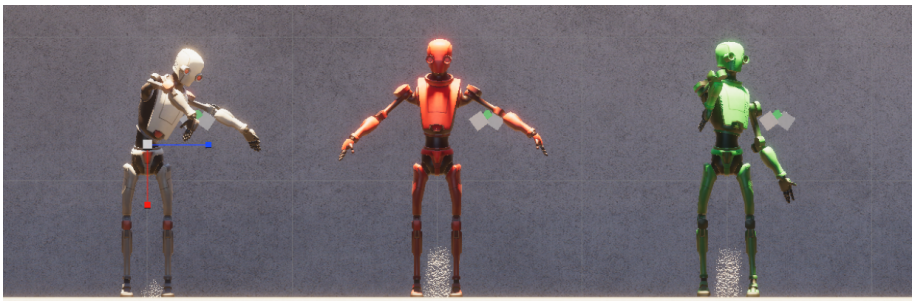
	REF_FE	AE_F	AE_T	RBF_F	RBF_T	VAE_F	VAE_T	DEC_F	DEC_T
REF_FE	\	6.52e-17	1.75e-14	3.60e-81	4.94e-40	1.75e-59	2.07e-15	1.01e-07	5.76e-11
AE_F	**2.99e-01**	\	**3.93e-01**	6.37e-25	**6.79e-02**	1.16e-34	**1.82e-01**	2.59e-162	2.11e-37
AE_T	**9.53e-02**	**2.97e-01**	\	3.15e-03	**7.89e-02**	7.95e-07	4.35e-08	1.35e-153	2.06e-17
RBF_F	**8.86e-01**	**7.75e-01**	**2.58e-01**	\	**5.55e-01**	**7.27e-01**	1.51e-21	2.43e-68	**7.89e-01**
RBF_T	**8.89e-01**	**6.73e-01**	**1.86e-01**	**3.80e-01**	\	**2.24e-01**	**3.47e-01**	3.10e-145	1.46e-07
VAE_F	4.53e-25	6.58e-120	1.70e-123	3.49e-139	3.76e-154	\	3.86e-31	6.50e-61	**2.26e-01**
VAE_T	**6.63e-01**	3.97e-22	4.99e-32	1.46e-39	5.14e-38	1.69e-59	\	5.30e-156	1.11e-24
DEC_F	**8.96e-01**	**3.81e-01**	1.11e-07	1.12e-04	1.35e-02	9.29e-192	7.85e-10	\	6.66e-83
DEC_T	4.22e-02	1.69e-07	1.74e-04	1.43e-07	4.63e-09	4.85e-79	5.06e-04	3.27e-39	\

The p-values of the performance comparison of FS. REF_FE denotes the reference FE performance, AE_F denotes the **Frozen** AE+MoE performance using the FS approach, RBF_T denotes the **Trained** RBF-CAT performance Et cetera. The differences between **Frozen** and **Trained** are statistically insignificant for all models except VAE-CAT and DEC-CAT.

(a) AE+MoE at frame 35

(b) RBF-CAT-MoE at frame 35

(c) VAE-CAT-MoE at frame 35



(d) DEC-CAT-MoE at frame 35

Figure 4.15: The sample frame from the generated clip using AE+MoE, RBF/VAE/DEC-CAT-OMG models using only 6-joints pose data as input. From left to right, it shows the target pose (white), the pose from the warm-started model with the *frozen* strategy (red) and the pose from the warm-started model with the *trained* strategy (green). It shows that only the RBF variant can produce a more accurate pose at that frame.

Summary

The results show that all the transferred models fail to generalise over the test set when cold-started. The models with FC-IN architecture has achieved closer to its reference performance than their FC-CAT counterparts, possibly due to having smaller latent dimension (128 vs 384).

Warm-starting the models with the pre-trained parameters from their reference model, which is trained on a previous domain, shows significant improvement over the cold-started models in reducing the Re. error for all FE and FC models ($P < .05$), using the *trained* strategy. The RMS is also reduced for the VAE-CAT and DEC-CAT models ($P < .001$), whereas for the FE and RBF-CAT models, the differences are insignificant as seen in Table 4.5 and 4.6.

With the *frozen* strategy, the Re.errors are greatly reduced for the FE, RBF-CAT and VAE-CAT models ($P < .05$). It suggests that MoGenNet and the clustering layer that are trained on another rig can be applied on new rigs without further tuning. Only the input and output layers need to be tuned to adapt to the new topology and size of the new rig. However, using the *trained* strategy does not yield further statistically significant error reductions. This implies possible overfitting when the training data is insufficient to train models of this complexity. Moreover, it might also be caused by catastrophic forgetting [17], when training with the new data corrupts the knowledge the models have learnt from the previous domain.

Using the FS approach, where the models are only exposed to pose data of 6 joints, the transferring quality of the models is not far from the other two approaches, as visualised in Fig. 4.16. The differences are either statistically insignificant or numerically insignificant. The statistical test results between the approaches can be found in Table 4.11. It indicates that the models can achieve comparable performance with the low-dimension data. Moreover, it suggests that OMG models are also capable of upsampling poses with only 6 joints to full resolution poses. This is especially interesting for real-time applications because it reduces the required bandwidth and inference time. However, these numerical results do not translate well into the animation quality and correctness. It means the models might be overfitted and the performance metrics do represent the true performance.

4.3 Model complexity

The number of parameters, the memory footprint and the inference time on CPU of the models is presented in Table 4.10. These properties are relevant in determining which model is more efficient while having good performance.

The inference times are measured when generating a motion clip on R1 and averaged over the frames. It shows that the MoE module is responsible for most of the complexity of the framework. The LSTM module has roughly three times more parameters than the MoE counterpart while being roughly 20% faster ($P = .012$).

FC-IN-OMG models are smaller and faster than AE+MoE, despite having an extra clustering layer. This is because the input pose vectors to the MoE module in FC-IN-OMG models are the cluster information of dimension 128, whereas the input pose vectors in AE+MoE are the pose embeddings of dimension 256. FC-IN-OMG models are roughly 25% faster than FC-CAT-OMG models ($P = 2.32\text{e}{-4}$).

Name	Parameters	Memory (MB)	Inference time (ms)
AE	0.45 M	1.82	11
AE+MoE	2.94 M	11.78	233
AE+LSTM	8.80 M	35.22	194
RBF-CAT+MoE	3.65 M	14.6	287
RBF-IN+MoE	2.71 M	10.87	210
VAE-CAT+MoE	3.68 M	14.73	281
VAE-IN+MoE	2.74 M	11	203
DEC-CAT+MoE	3.65 M	14.6	289
DEC-IN+MoE	2.71 M	10.87	214

Table 4.10: The complexity properties of the most relevant models.

4.4 FE vs FC-CAT vs FS-CAT

Fig. 4.16 presents the performance comparison of the FE, FC-CAT and FS-CAT models against the reference FE model which is the closest equivalent to LMP-MoE by Starke et al. [26]. The transferred models are trained with limited training and data, using one of the transfer learning approaches with *trained* strategy. The statistical analysis indicate that all models are not equal with $P_{Re} = 4.87\text{e}{-151}, P_{RMS} = 1.55\text{e}{-85}$. The other p-values can be found in Table 4.11.

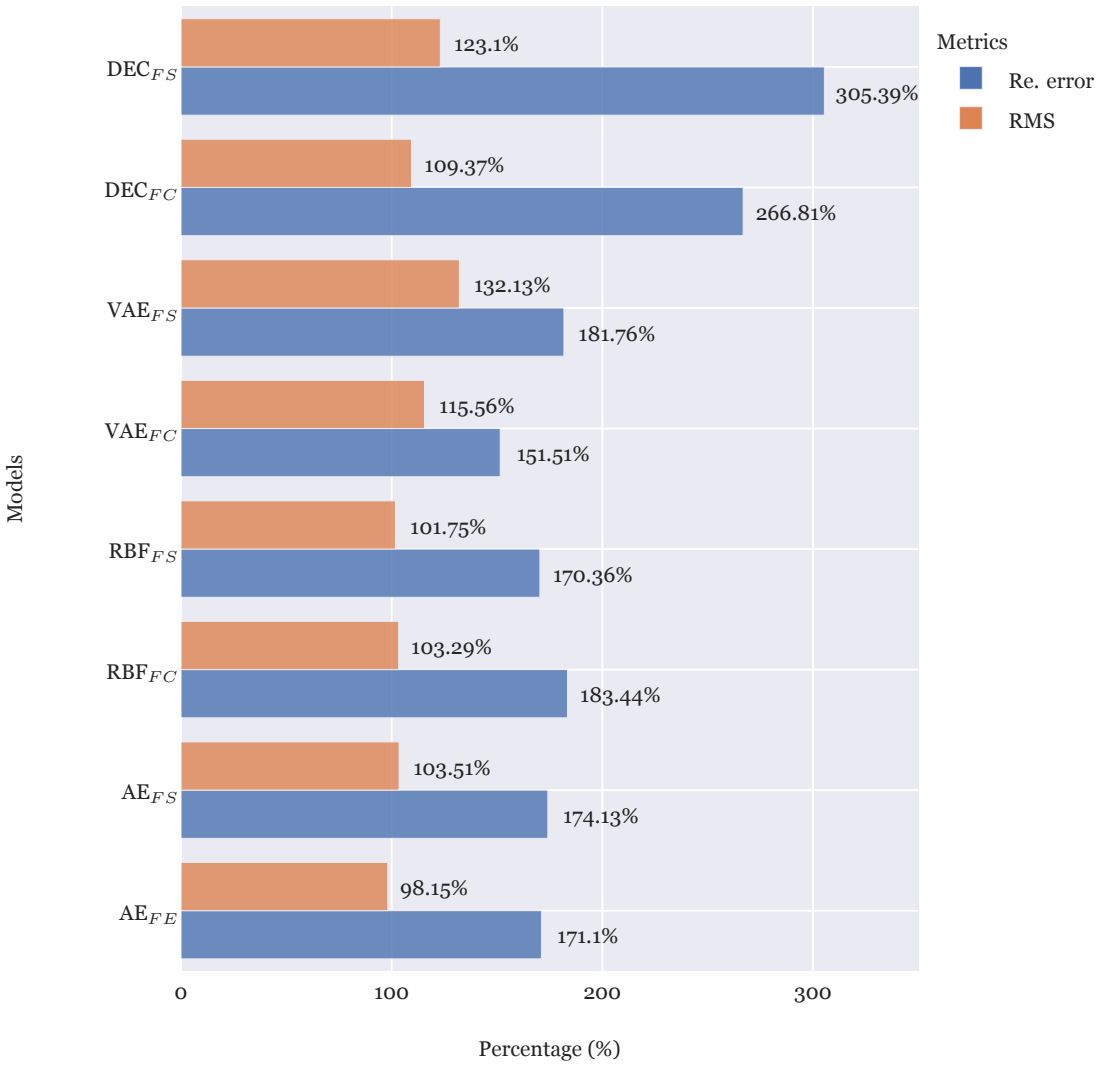


Figure 4.16: Percentage differences in Re. error and RMS when comparing the transferred FE, FC-CAT and FS-CAT models (target) against the reference FE models (baseline). Each bar illustrate how much higher errors the target model achieves in the respective metric. The DEC-CAT model is the worst, in terms of Re. error ($P < .001$). The VAE-CAT model has highest RMS when compared to the other models (FC/FS) ($P < .001$).

Interestingly, the FS models have achieved similar results as the models that are trained with full-resolution poses. However, they do not translate into similar quality of the generated animations.

The AE, RBF-CAT and VAE-CAT models achieve similar Re. error using the FE/FC approach, whether statistically or numerically, whereas the VAE-CAT model has the highest RMS among them ($P < .001$). It results in more visible jittering in the generated animations, which is likely caused by the internal sampling mechanism. AE and RBF perform similarly in all aspects, hence it is inconclusive which performs better. However,

the FE approach with AE+MoE has 20% fewer parameters, making it less computational complex, which is favourable for real-time applications.

Table 4.11: p-values of Fig. 4.16

	REF_FE	AE_FE	RBF_FC	VAE_FC	DEC_FC	AE_FS	RBF_FS	VAE_FS	DEC_FS
REF_FE	\	6.52e-17	3.60e-81	1.75e-59	1.01e-07	1.75e-14	4.94e-40	2.07e-15	5.76e-11
AE_FE	**2.99e-01**	\	3.85e-12	**1.92e-01**	1.94e-03	**4.90e-01**	1.46e-02	1.05e-07	1.65e-23
RBF_FC	**8.86e-01**	**5.97e-01**	\	4.84e-02	3.96e-32	5.05e-14	4.83e-08	**8.91e-01**	2.00e-26
VAE_FC	4.53e-25	4.67e-27	9.93e-15	\	6.09e-35	5.78e-07	2.30e-10	3.63e-02	7.00e-30
DEC_FC	**8.96e-01**	1.80e-02	2.83e-02	4.33e-03	\	1.82e-02	3.22e-03	5.01e-30	**3.23e-01**
AE_FS	**9.53e-02**	**8.23e-01**	**8.87e-01**	3.58e-09	**6.39e-01**	\	**7.89e-02**	4.35e-08	2.06e-17
RBF_FS	**8.89e-01**	**3.86e-01**	**7.61e-02**	5.17e-24	**8.58e-01**	**1.86e-01**	\	**3.47e-01**	1.46e-07
VAE_FS	**6.63e-01**	1.86e-59	1.51e-43	**7.70e-01**	2.01e-27	4.99e-32	5.14e-38	\	1.11e-24
DEC_FS	4.22e-02	8.65e-15	9.74e-04	**1.53e-01**	1.31e-03	1.74e-04	4.63e-09	5.06e-04	\

The p-values of the performance comparison between the FE, FC and FS models . REF_FE denotes the reference FE performance, AE_FE denotes the AE+MoE performance using the FE approach, RBF_FC denotes the RBF-CAT performance using the FC approach, VAE_FS denotes the VAE-CAT performance using the FS approach Et cetera.

Chapter 5

Discussion

5.1 Implication

This work aims to investigate the effect of the three transfer learning techniques within objective-driven motion synthesis, to enable and facilitate adaptation of the framework on new domains. The results imply that OMG can be trained to predict poses in the future frames given the current pose, positional and rotation objectives and a local motion phase. They align with the findings in MANN, NSM and LMP-MoE [25, 26, 29]. They also confirm that MoE is better-suited than LSTM even in the static context, confirming the findings in LMP-MoE where the animation is generated in real-time against dynamic targets.

FE-OMG and FC-OMG can generate near-identical motion clips as the ground truth. FE, FC and FS have demonstrated the capability of improving the generation performance of the OMG models on new domains with limited data and training. The performance improvement from the *frozen* strategy implies that the MoGenNet in OMG operates on the latent feature space that, despite being trained on a previous domain, still works on the new domains to some degree without further training. It demonstrates the possibility of rig-agnostic encoding, where only the AE needs to be adapted to the new rigs.

The key contribution of this work is to provide insights into how the RAE techniques could improve the learning of the other state-of-art MoE-based frameworks, such as MANN, NSM, LMP-MoE, MVAE [15, 25, 26, 29], on new domains. These frameworks can be theoretically reduced to OMG because they share the same internal generation process, where the parameters of the main network are generated by a gating network that is conditioned on local motion phases. The main network is conditioned on pose data, condition features and control signals. It outputs pose and feature updates for the next frame. The architecture of OMG ensures that the RAE techniques can be implemented in

the other state-of-art frameworks in a similar manner. The results of FE-OMG also imply that these MoE-based frameworks already possess this feature encoding transferring capability.

5.2 Benefits, ethical aspects and sustainability

Transfer learning is designed specifically to reduce the effort and the work needed to apply the machine learning models to new domains or tasks. From developers' perspective, it reduces time, cost and computing power needed for acquiring data sets and training models. Allowing the developers to deploy the models on new domains easier and faster. For example, by utilising a pre-trained model or publicly available dataset, the required motion-captured clips that a studio needs for their project can be reduced. This reduces staff costs, actor costs, motion capture studio rental cost and energy usage from the equipment, the facility and the computers. Moreover, it also reduces gas emissions and other carbon footprints produced by the staff travelling to the studio.

The RAE techniques would also make deep learning-based motion synthesis frameworks a more sustainable and scalable option in the video game industry, by allowing the developers to reuse the existing animation dataset to create animations for new characters. During the covid-19 pandemic, there have been gathering restrictions in some countries that may inflict difficulties for studios to perform motion capturing tasks. It would have severely impacted the development schedule and affected the business plans. Being able to reuse existing datasets would have mitigated the problem if a similar situation occurs again.

Ultimately, the RAE techniques can also enable the models to learn from motion data that are captured on different rigs. Making them available and affordable for smaller studios or indie developers by using publicly available animation datasets, such as Ubisoft's LAFAN1 dataset [1]. However, this particular dataset is under a non-commercial license. It is an example of an ethical concern that the studios may use these non-commercial datasets to generate animations for their commercial projects, and it will be difficult to notice this in the end products.

This work is completely open and transparent. The source code is available on the Github repository [2]. The procedures and the experiments are explained in such a manner that the work can be replicated and reproduced by anyone with access to a computer. The data is procedurally generated using the proprietary Bio-IK plugin, available for

[1] https://github.com/ubisoft/ubisoft-laforge-animation-dataset
[2] https://github.com/Neroro64/Deep-learning-based-rig-agnostic-encoding

purchase on Unity Asset Store. The generation procedure is described in Section 3.3.2. The character rigs are under public copyright licences that permit use for educational purposes. The rig R5 is a proprietary property owned by DICE. The allowance for using the rig in the research and disclosing its specifications has been obtained.

5.3 Limitations and future work

As suggested in Section 4.2, the four evaluation metrics fail to represent the true generation performance of the models. A lower reconstruction error does not necessarily translate to a better quality of the generated animation. A better metric could be Normalised-power-spectrum similarity (NPSS) [8], because it compares the animation sequences in their power spectrum which is time-invariant. The animation quality is a rather subjective matter and difficult to access quantitatively. The models might generate connected, cohesive poses that fail to reach the objectives or disconnected poses that can reach the objectives. A set of tests could be designed to access the various properties of the generated motion clips to quantify the overall animation quality.

The adversarial discriminator is only trained together with the autoencoder, and not with the OMG models in the experiments, due to the time constraint. It makes the adversarial loss not depict the true quality of generated poses. It is observed that the generated poses with disconnected joints usually yield higher adversarial losses, but it is not verified.

The rotational error is computed as the mean squared error between the two orientations, which is not good as it only performs a value-wise comparison of the orientation vectors, that are not guaranteed to be orthonormal. It would have been better to compute the angular differences between the orientation matrices.

Due to the time constraint, the training is heavily limited when performing the experiments. Each model is trained for only $\frac{1}{3}$ number of epochs of what was initially planned. For the same reason, the same set of hyperparameters is used for all models regardless of the RAE approach, resulting in FS-OMG models sharing the same complexity as the other models, despite only having only 6 joints in the pose representation.

In the other motion synthesis frameworks, the condition features and the control signals are sampled from a window of frames centred at the current frame, but in this work, only the features existing in the current frame are considered and used. By including

the features that exist in a longer period of frames or using multiple frames as input to the framework, it would improve the generation performance of the models.

For future work, it would be interesting to further experiment and test probabilistic models with feature clustering and feature selection approaches. As described in Section 4.4, they might have a better transferring capability of generalising over different domains. An improvement would be to decompose the animation database into a set of key poses, that can be used to initialise the cluster centres. It is also necessary to test with a variety of topologically different rigs and different types of motion data. Moreover, it is interesting to experiment with transfer learning between different motion types (tasks) or different motion styles.

Chapter 6

Conclusion

In this work, three rig-agnostic encoding (RAE) approaches are presented: feature encoding (FE), feature clustering (FC) and feature selection (FS), for knowledge transferring of the proposed OMG model. It can generate the pose for the next frame based on the positional and the rotational objectives and the local motion phase variable.

The purpose is to investigate how the RAE approaches can improve the learning of OMG on new domains (character rigs) with limited training and data, by warm-starting the model with the parameters from the pre-trained instance on a previous domain. The results show that MoE performs better than LSTM at predicting new poses for OMG. RBF is better than VAE and DEC as the clustering layer for the FC approach. Both FE and FC can improve the generation performance, wherein AE+MoE and RBF-CAT+MoE are the two most successful models. It is inconclusive which of them performs better, but AE+MoE is less computational complex, hence it is considered to be the better choice. FS does achieve similar numerical results but fails to generate correct animations.

In conclusion, all three RAE approaches can be used for transfer learning of OMG, where the FE approach is considered to be the best of them due to its simplicity and overall performance.

Bibliography

[1] Bank, Dor, Koenigstein, Noam, and Giryes, Raja. “Autoencoders”. In: *arXiv* (2020). eprint: 2003.05991.

[2] Bishop, Christopher M. *Pattern recognition and Machine learning*. Springer Science, 2209.

[3] Chamroukhi, F. “Robust mixture of experts modeling using the t distribution”. In: *Neural Networks* 79 (2016), pp. 20–36. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2016.03.002. eprint: 1701.07429.

[4] Dai, Wenyuan, Yang, Qiang, Xue, Gui-Rong, and Yu, Yong. “Self-taught clustering”. In: *Proceedings of the 25th international conference on Machine learning - ICML ’08* (2008), pp. 200–207. DOI: 10.1145/1390156.1390182.

[5] Fragkiadaki, Katerina, Levine, Sergey, Felsen, Panna, and Malik, Jitendra. “Recurrent Network Models for Human Dynamics”. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 4346–4354. DOI: 10.1109/iccv.2015.494.

[6] Gadelmawla, E.S., Koura, M.M., Maksoud, T.M.A., Elewa, I.M., and Soliman, H.H. “Roughness parameters”. In: *Journal of Materials Processing Technology* 123.1 (2002), pp. 133–145. ISSN: 0924-0136. DOI: 10.1016/s0924-0136(02)00060-2.

[7] Glorot, Xavier and Bengio, Yoshua. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[8] Gopalakrishnan, Anand, Mali, Ankur, Kifer, Dan, Giles, C Lee, and Ororbia, Alexander G. “A Neural Temporal Model for Human Motion Prediction”. In: *arXiv* (2018). eprint: 1809.03036.

[9] Harvey, Félix G., Yurick, Mike, Nowrouzezahrai, Derek, and Pal, Christopher. “Robust motion in-betweening”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), 60:1–60:12. ISSN: 0730-0301. DOI: 10.1145/3386569.3392480.

[10] Hochreiter, Sepp and Schmidhuber, Jrgen. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[11] Hodgins, Jessica K., Wooten, Wayne L., Brogan, David C., and O’Brien, James F. “Animating Human Athletics”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 71–78. ISBN: 0897917014. DOI: 10.1145/218380.218414.

[12] Holden, Daniel, Komura, Taku, and Saito, Jun. “Phase-functioned neural networks for character control”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/3072959.3073663.

[13] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[14] “Kruskal-Wallis Test”. In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 288–290. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_216.

[15] Ling, Hung Yu, Zinno, Fabio, Cheng, George, and Panne, Michiel Van De. “Character controllers using motion VAEs”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), 40:1–40:12. ISSN: 0730-0301. DOI: 10.1145/3386569.3392422.

[16] Mao, Xudong, Li, Qing, Xie, Haoran, Lau, Raymond Y K, Wang, Zhen, and Smolley, Stephen Paul. “Least Squares Generative Adversarial Networks”. In: *arXiv* (2016). eprint: 1611.04076.

[17] McCloskey,
Michael and Cohen, Neal J. “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem”. In: *Psychology of Learning and Motivation* 24 (1989), pp. 109–165. ISSN: 0079-7421. DOI: 10.1016/s0079-7421(08)60536-8.

[18] Mills, T.C. and Mills, T.C. *Time Series Techniques for Economists*. Cambridge University Press, 1990. ISBN: 9780521405744.

[19] Parent, Rick. “Computer Animation (Third Edition)”. In: (2012), pp. 111–160. DOI: 10.1016/b978-0-12-415842-9.00004-6.

[20] Parent, Rick. “Computer Animation (Third Edition)”. In: (2012), p. 1. DOI: 10.1016/b978-0-12-415842-9.00007-1.

[21] Parent, Rick. “Computer Animation (Third Edition)”. In: (2012), pp. 161–185. DOI: 10.1016/b978-0-12-415842-9.00005-8.

[22] Peng, Xue Bin, Abbeel, Pieter, Levine, Sergey, and Panne, Michiel van de. “DeepMimic”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3197517.3201311. eprint: 1804.02717.

[23] Šidák, Zbyněk. “Rectangular Confidence Regions for the Means of Multivariate Normal Distributions”. In: *Journal of the American Statistical Association* 62.318 (2012), pp. 626–633. ISSN: 0162-1459. DOI: 10.1080/01621459.1967.10482935.

[24] Starke, Sebastian, Hendrich, Norman, and Zhang, Jianwei. “Memetic Evolution for Generic Full-Body Inverse Kinematics in Robotics and Animation”. In: *IEEE Transactions on Evolutionary Computation* 23.3 (2019), pp. 406–420. ISSN: 1089-778X. DOI: 10.1109/tevc.2018.2867601.

[25] Starke, Sebastian, Zhang, He, Komura, Taku, and Saito, Jun. “Neural state machine for character-scene interactions”. In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3355089.3356505.

[26] Starke, Sebastian, Zhao, Yiwei, Komura, Taku, and Zaman, Kazi. “Local motion phases for learning multi-contact character movements”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), 54:1–54:13. ISSN: 0730-0301. DOI: 10.1145/3386569.3392450.

[27] Taylor, Graham W, Hinton, Geoffrey E, and Roweis, Sam. “Modeling Human Motion Using Binary Latent Variables”. In: *Advances in Neural Information Processing Systems*. Ed. by B. Schölkopf, J. Platt, and T. Hoffman. Vol. 19. MIT Press, 2007.

[28] Xie, Junyuan, Girshick, Ross, and Farhadi, Ali. “Unsupervised Deep Embedding for Clustering Analysis”. In: *arXiv* (2015). eprint: 1511.06335.

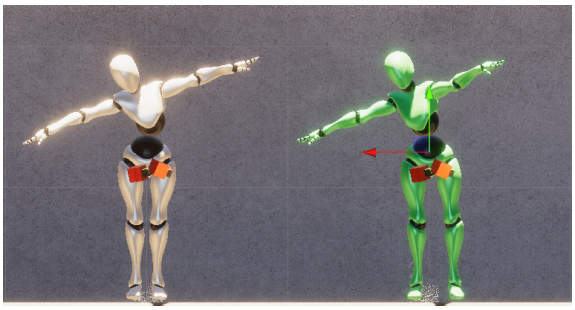
[29] Zhang, He, Starke, Sebastian, Komura, Taku, and Saito, Jun. “Mode-adaptive neural networks for quadruped motion control”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/3197517.3201366.

[30] Zhuang, Fuzhen et al. “A Comprehensive Survey on Transfer Learning”. In: *arXiv* (2019). eprint: 1911.02685.

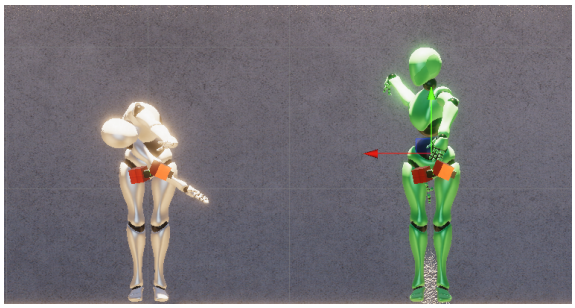
Appendix - Contents

Appendix A

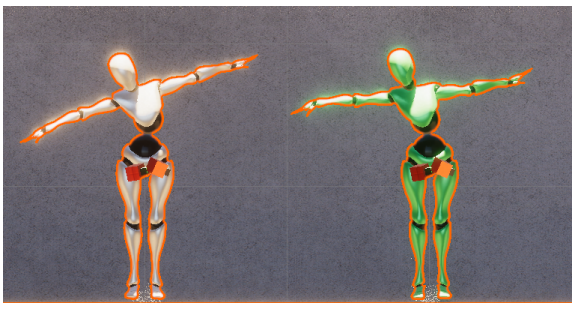
The screenshots of the generated animation clips by the reference FC-OMG models



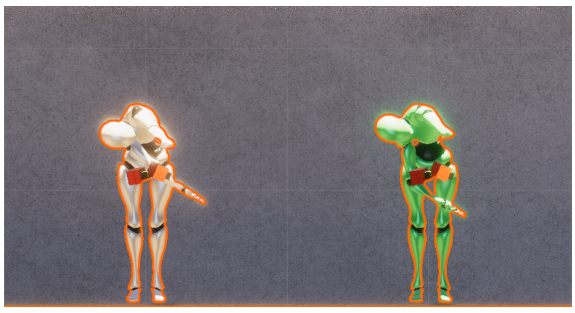
(a) RBF-IN+MoE at frame 22



(b) RBF-IN+MoE at frame 35

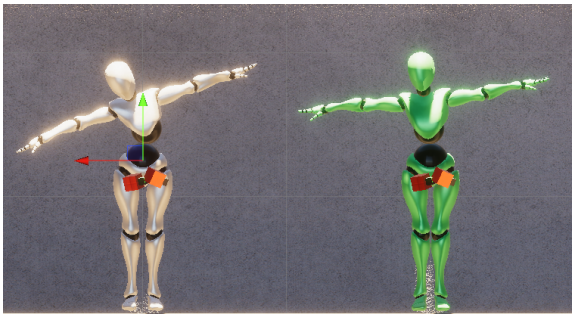


(c) RBF-CAT+MoE at frame 22



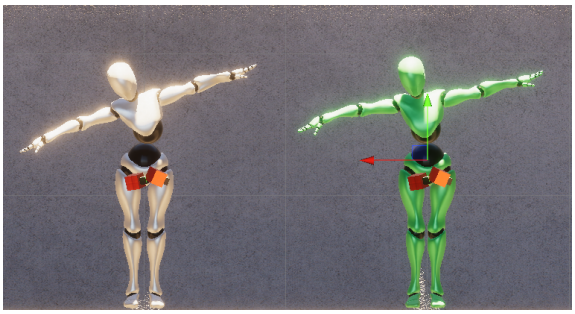
(d) RBF-CAT+MoE at frame 35

Figure A.1: The screenshots of the generated clip (the green character) by RBF-IN+MoE and RBF-CAT+MoE, compared against the target clip (the white character).

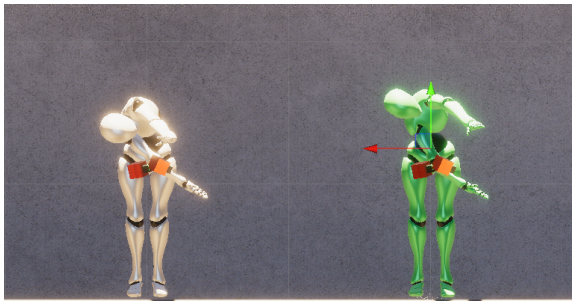


(a) VAE-IN+MoE at frame 22

(b) VAE-IN+MoE at frame 35

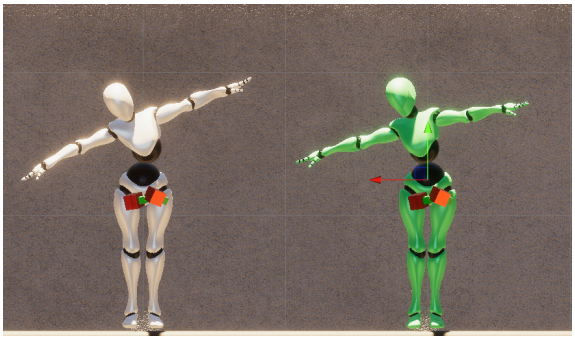


(c) VAE-CAT+MoE at frame 22

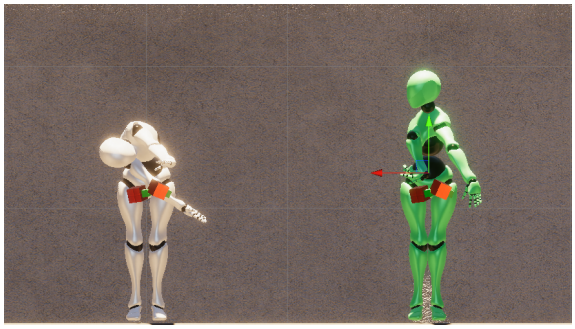


(d) VAE-CAT+MoE at frame 35

Figure A.2: The screenshots of the generated clip (the green character) by VAE-IN+MoE and VAE-CAT+MoE, compared against the target clip (the white character).

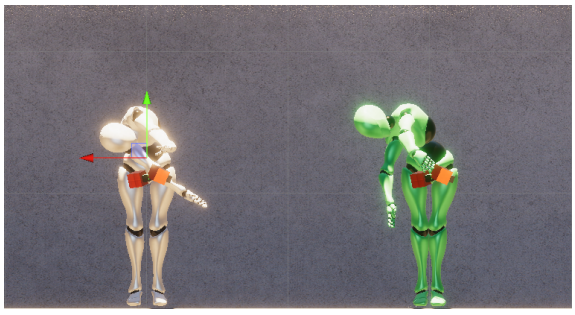


(a) DEC-IN+MoE at frame 22



(b) DEC-IN+MoE at frame 35

(c) DEC-CAT+MoE at frame 22



(d) DEC-CAT+MoE at frame 35

Figure A.3: The screenshots of the generated clip (the green character) by DEC-IN+MoE and DEC-CAT+MoE, compared against the target clip (the white character).

Appendix B

Raw result

B.1 Reference performance values

	Re. error	Adv. error	Rot. error	RMS
AE+MoE_R1	0.017	0.132	8.185	0.052
AE+MoE_R2	0.027	0.151	4.153	0.061
AE+MoE_R3	0.021	0.293	6.124	0.061
AE+MoE_R4	0.014	0.145	2.536	0.045
AE+MoE_R5	0.022	0.169	2.836	0.048
AE+LSTM_R1	0.063	0.060	10.699	0.099
AE+LSTM_R2	0.076	0.172	5.346	0.125
AE+LSTM_R3	0.070	0.296	7.922	0.114
AE+LSTM_R4	0.053	0.151	3.734	0.085
AE+LSTM_R5	0.075	0.175	4.072	0.084
RBF-IN+MoE_R1	0.078	0.076	10.587	0.071
RBF-IN+MoE_R2	0.085	0.170	5.299	0.089
RBF-IN+MoE_R3	0.076	0.359	7.923	0.102
RBF-IN+MoE_R4	0.059	0.148	3.965	0.059
RBF-IN+MoE_R5	0.072	0.172	3.958	0.067
RBF-IN+LSTM_R1	0.101	0.047	11.078	0.087
RBF-IN+LSTM_R2	0.112	0.175	5.402	0.126
RBF-IN+LSTM_R3	0.095	0.329	8.293	0.151
RBF-IN+LSTM_R4	0.074	0.150	4.299	0.080
RBF-IN+LSTM_R5	0.086	0.193	4.274	0.095
RBF-CAT+MoE_R1	0.015	0.137	8.129	0.051
RBF-CAT+MoE_R2	0.023	0.151	4.181	0.061
RBF-CAT+MoE_R3	0.023	0.290	6.424	0.062
RBF-CAT+MoE_R4	0.016	0.146	2.563	0.047

RBF-CAT+MoE_R5	0.015	0.166	2.527	0.046
RBF-CAT+LSTM_R1	0.065	0.062	10.707	0.092
RBF-CAT+LSTM_R2	0.072	0.167	5.046	0.109
RBF-CAT+LSTM_R3	0.066	0.290	7.610	0.123
RBF-CAT+LSTM_R4	0.053	0.149	3.808	0.091
RBF-CAT+LSTM_R5	0.065	0.173	3.499	0.085
VAE-IN+MoE_R1	0.049	0.127	9.574	0.152
VAE-IN+MoE_R2	0.056	0.160	5.509	0.162
VAE-IN+MoE_R3	0.056	0.262	7.724	0.191
VAE-IN+MoE_R4	0.044	0.145	4.105	0.157
VAE-IN+MoE_R5	0.053	0.164	3.621	0.145
VAE-IN+LSTM_R1	0.095	0.057	10.979	0.128
VAE-IN+LSTM_R2	0.108	0.173	5.204	0.173
VAE-IN+LSTM_R3	0.096	0.344	8.096	0.173
VAE-IN+LSTM_R4	0.079	0.155	4.528	0.089
VAE-IN+LSTM_R5	0.090	0.188	4.104	0.111
VAE-CAT+MoE_R1	0.021	0.136	8.303	0.061
VAE-CAT+MoE_R2	0.016	0.152	4.073	0.063
VAE-CAT+MoE_R3	0.024	0.290	6.299	0.071
VAE-CAT+MoE_R4	0.025	0.146	2.902	0.060
VAE-CAT+MoE_R5	0.021	0.165	2.925	0.055
VAE-CAT+LSTM_R1	0.065	0.061	10.581	0.111
VAE-CAT+LSTM_R2	0.067	0.168	5.139	0.122
VAE-CAT+LSTM_R3	0.069	0.300	7.573	0.124
VAE-CAT+LSTM_R4	0.050	0.150	3.637	0.085
VAE-CAT+LSTM_R5	0.065	0.176	3.577	0.097
DEC-IN+MoE_R1	0.077	0.079	10.052	0.073
DEC-IN+MoE_R2	0.089	0.167	5.810	0.104
DEC-IN+MoE_R3	0.074	0.341	7.460	0.103
DEC-IN+MoE_R4	0.060	0.152	4.247	0.063
DEC-IN+MoE_R5	0.073	0.179	3.926	0.070
DEC-CAT+MoE_R1	0.041	0.098	9.348	0.057
DEC-CAT+MoE_R2	0.051	0.158	5.366	0.071
DEC-CAT+MoE_R3	0.030	0.285	6.186	0.067
DEC-CAT+MoE_R4	0.034	0.148	3.725	0.051
DEC-CAT+MoE_R5	0.073	0.175	3.844	0.053

Table B.1: The reference results of the models on the respective domain.

B.2 Transferred performance values

	Re. error	Adv. error	Rot. error	RMS
AE+MoE_RAW_R2	0.071	0.170	5.762	0.066
AE+MoE_RAW_R3	0.069	0.348	8.378	0.072
AE+MoE_RAW_R4	0.046	0.151	3.772	0.042
AE+MoE_RAW_R5	0.071	0.187	4.125	0.041
AE+MoE_F_R2	0.039	0.156	4.612	0.069
AE+MoE_F_R3	0.028	0.316	6.410	0.063
AE+MoE_F_R4	0.033	0.154	3.308	0.051
AE+MoE_F_R5	0.034	0.170	3.182	0.049
AE+MoE_T_R2	0.053	0.163	4.862	0.064
AE+MoE_T_R3	0.027	0.305	6.298	0.064
AE+MoE_T_R4	0.040	0.152	3.388	0.048
AE+MoE_T_R5	0.030	0.171	3.167	0.046
RBF-IN+MoE_RAW_R2	0.132	0.177	5.675	0.061
RBF-IN+MoE_RAW_R3	0.102	0.457	8.053	0.075
RBF-IN+MoE_RAW_R4	0.083	0.162	4.865	0.052
RBF-IN+MoE_RAW_R5	0.097	0.211	4.534	0.036
RBF-IN+MoE_F_R2	0.135	0.187	6.083	0.089
RBF-IN+MoE_F_R3	0.098	0.343	7.997	0.154
RBF-IN+MoE_F_R4	0.083	0.162	4.748	0.061
RBF-IN+MoE_F_R5	0.084	0.185	4.439	0.078
RBF-IN+MoE_T_R2	0.115	0.168	5.443	0.095
RBF-IN+MoE_T_R3	0.084	0.372	8.154	0.118
RBF-IN+MoE_T_R4	0.076	0.155	4.619	0.067
RBF-IN+MoE_T_R5	0.084	0.182	4.349	0.076
RBF-CAT+MoE_RAW_R2	0.058	0.164	5.703	0.059
RBF-CAT+MoE_RAW_R3	0.066	0.370	8.178	0.068
RBF-CAT+MoE_RAW_R4	0.049	0.152	4.164	0.050
RBF-CAT+MoE_RAW_R5	0.058	0.181	3.867	0.048
RBF-CAT+MoE_F_R2	0.038	0.159	4.752	0.071
RBF-CAT+MoE_F_R3	0.031	0.331	6.720	0.072
RBF-CAT+MoE_F_R4	0.034	0.148	3.374	0.048
RBF-CAT+MoE_F_R5	0.034	0.169	3.209	0.051
RBF-CAT+MoE_T_R2	0.042	0.158	5.211	0.065
RBF-CAT+MoE_T_R3	0.040	0.337	6.841	0.066
RBF-CAT+MoE_T_R4	0.025	0.151	3.222	0.050
RBF-CAT+MoE_T_R5	0.034	0.169	3.234	0.050
VAE-IN+MoE_RAW_R2	0.126	0.179	6.008	0.106

VAE-IN+MoE_RAW_R3	0.109	0.429	8.711	0.101
VAE-IN+MoE_RAW_R4	0.086	0.154	4.836	0.062
VAE-IN+MoE_RAW_R5	0.100	0.206	4.432	0.043
VAE-IN+MoE_F_R2	0.086	0.170	5.877	0.125
VAE-IN+MoE_F_R3	0.086	0.319	8.122	0.192
VAE-IN+MoE_F_R4	0.079	0.159	4.859	0.136
VAE-IN+MoE_F_R5	0.064	0.183	4.337	0.118
VAE-IN+MoE_T_R2	0.064	0.162	5.519	0.154
VAE-IN+MoE_T_R3	0.068	0.316	7.838	0.148
VAE-IN+MoE_T_R4	0.053	0.150	4.425	0.183
VAE-IN+MoE_T_R5	0.067	0.177	4.282	0.131
VAE-CAT+MoE_RAW_R2	0.065	0.163	5.421	0.167
VAE-CAT+MoE_RAW_R3	0.060	0.317	7.289	0.123
VAE-CAT+MoE_RAW_R4	0.044	0.150	4.008	0.082
VAE-CAT+MoE_RAW_R5	0.053	0.179	3.673	0.093
VAE-CAT+MoE_F_R2	0.038	0.157	4.678	0.083
VAE-CAT+MoE_F_R3	0.031	0.368	6.543	0.094
VAE-CAT+MoE_F_R4	0.033	0.153	3.597	0.070
VAE-CAT+MoE_F_R5	0.031	0.168	3.104	0.066
VAE-CAT+MoE_T_R2	0.040	0.156	4.633	0.078
VAE-CAT+MoE_T_R3	0.026	0.302	5.849	0.073
VAE-CAT+MoE_T_R4	0.023	0.148	2.979	0.056
VAE-CAT+MoE_T_R5	0.034	0.172	3.188	0.058
DEC-IN+MoE_RAW_R2	0.141	0.193	5.839	0.062
DEC-IN+MoE_RAW_R3	0.104	0.437	8.511	0.086
DEC-IN+MoE_RAW_R4	0.087	0.167	5.082	0.051
DEC-IN+MoE_RAW_R5	0.098	0.206	4.278	0.031
DEC-IN+MoE_F_R2	0.150	0.197	6.489	0.090
DEC-IN+MoE_F_R3	0.098	0.339	7.977	0.139
DEC-IN+MoE_F_R4	0.086	0.161	4.898	0.057
DEC-IN+MoE_F_R5	0.087	0.185	4.408	0.080
DEC-IN+MoE_T_R2	0.119	0.181	5.701	0.087
DEC-IN+MoE_T_R3	0.088	0.362	7.631	0.116
DEC-IN+MoE_T_R4	0.078	0.157	4.677	0.064
DEC-IN+MoE_T_R5	0.079	0.177	4.049	0.075
DEC-CAT+MoE_RAW_R2	0.078	0.173	6.061	0.073
DEC-CAT+MoE_RAW_R3	0.075	0.347	8.093	0.064
DEC-CAT+MoE_RAW_R4	0.054	0.151	3.901	0.045
DEC-CAT+MoE_RAW_R5	0.091	0.205	4.316	0.035
DEC-CAT+MoE_F_R2	0.091	0.175	6.268	0.131

DEC-CAT+MoE_F_R3	0.086	0.469	8.047	0.091
DEC-CAT+MoE_F_R4	0.069	0.157	4.847	0.058
DEC-CAT+MoE_F_R5	0.062	0.179	4.323	0.056
DEC-CAT+MoE_T_R2	0.061	0.164	5.795	0.102
DEC-CAT+MoE_T_R3	0.057	0.350	7.040	0.073
DEC-CAT+MoE_T_R4	0.038	0.155	3.910	0.052
DEC-CAT+MoE_T_R5	0.048	0.173	3.515	0.050

Table B.2: The transferred results of the models on the respective domain.

	Re. error	Adv. error	Rot. error	RMS
AE+MoE_F_R2	0.046	0.160	6.419	0.070
AE+MoE_F_R3	0.037	0.315	3.277	0.071
AE+MoE_F_R4	0.038	0.152	4.691	0.050
AE+MoE_F_R5	0.033	0.163	3.890	0.049
AE+MoE_T_R2	0.054	0.158	5.836	0.065
AE+MoE_T_R3	0.032	0.295	3.204	0.066
AE+MoE_T_R4	0.037	0.149	4.825	0.051
AE+MoE_T_R5	0.029	0.162	4.974	0.049
RBF-CAT+MoE_F_R2	0.058	0.164	6.027	0.066
RBF-CAT+MoE_F_R3	0.066	0.359	3.995	0.063
RBF-CAT+MoE_F_R4	0.052	0.158	4.549	0.050
RBF-CAT+MoE_F_R5	0.052	0.176	5.520	0.049
RBF-CAT+MoE_T_R2	0.057	0.160	4.749	0.068
RBF-CAT+MoE_T_R3	0.035	0.316	3.338	0.067
RBF-CAT+MoE_T_R4	0.029	0.150	3.813	0.048
RBF-CAT+MoE_T_R5	0.029	0.167	4.239	0.048
VAE-CAT+MoE_F_R2	0.060	0.164	5.748	0.187
VAE-CAT+MoE_F_R3	0.056	0.317	4.432	0.193
VAE-CAT+MoE_F_R4	0.052	0.150	5.723	0.075
VAE-CAT+MoE_F_R5	0.054	0.173	5.090	0.133
VAE-CAT+MoE_T_R2	0.047	0.160	6.279	0.078
VAE-CAT+MoE_T_R3	0.035	0.309	3.222	0.085
VAE-CAT+MoE_T_R4	0.031	0.149	4.466	0.061
VAE-CAT+MoE_T_R5	0.035	0.169	4.478	0.067
DEC-CAT+MoE_F_R2	0.106	0.178	6.528	0.071
DEC-CAT+MoE_F_R3	0.102	0.412	4.238	0.060
DEC-CAT+MoE_F_R4	0.085	0.155	5.476	0.041
DEC-CAT+MoE_F_R5	0.095	0.190	5.400	0.029
DEC-CAT+MoE_T_R2	0.064	0.166	6.302	0.091

DEC-CAT+MoE_T_R3	0.066	0.367	3.733	0.086
DEC-CAT+MoE_T_R4	0.053	0.158	4.161	0.055
DEC-CAT+MoE_T_R5	0.047	0.175	5.149	0.055

Table B.3: The transferred results of the FS-OMG models on the respective domain.