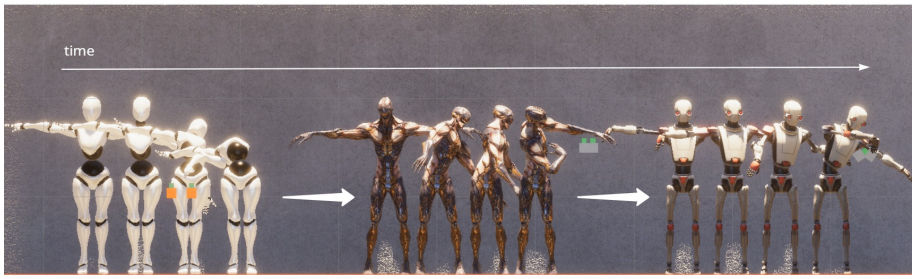
DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021



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

KTH Master Thesis

Nuo Chen

KTH ROYAL INSTITUTE OF TECHNOLOGY
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 thesis 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 the three approaches demonstrate significant improvement in both the numerical errors 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 achieve better performance than the FS approach, in terms of the generation performance and the transferring quality. The FC approach is slightly better than FE but the latter approach is less computational complex, which is favourable 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

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 karaktärskelett (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 modellen måldriven animationgenereringsnätverk (OMG), som är ett neural nätverk-baserad modell för att procedurellt generera animationer baserade på position och rotation mål. 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 de numeriska felen och den visuella kvaliteten av de skapade animationerna, i jämförelse med den vanilla prestanda. De empiriska resultaten indikerar att både FE och FC metoderna presterar bättre än FS metoden, med avseende på genereringsprestandan och överferingspotentialen. FC metoden presterar en aning bättre än FE metoden, men FE metoden är mindre beräkningskomplex, vilket är fördelaktigt för real-time applikationer.

Nyckelord

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

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

Nomenclature

Acronyms

AE Autoencoder

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

DEC Deep embedding clustering network

Dec Decoder

Enc Encoder

FC-CAT-OMG OMG with AE, C-model and MoGen, where Ψ is concatenated with z before feeding to MoGen

FC-IN-OMG OMG with AE, C-model and MoGen, where Ψ is used as input to MoGen

FE-OMG OMG with only AE and MoGen

LSTM Long short-term memory network

MLP Multi-layer perceptron network

MoE Mixture-of-experts network

MoGen Motion generation network (MoE, LSTM)

OMG Objective-driven motion generation network

RBF Radial-basis-function network

VAE Variational autoencoder

Data related symbols

K Number of key joints

M Dataset size

n Input dimension

X, Y Input and output variables

x, y Input and output vectors

X^P Input local motion phase

X^S Input pose features

X^V Input control features

Neural network related symbols

$\mathcal{L}$ Loss

μ, σ cluster centres (mean), scalars (standard deviation)

ϕ activation function, basis function

Ψ	Output from the clustering layer (C-model)
Θ	Local motion phase
θ	Network parameters
C	Number of clusters / distribution factors
k	Number of experts in MoE model
w, b	Weights, biases
z	Embedding

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 **interpolation-based animation**, 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 animation** possible at a reasonable cost. 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. Figure 1.1 illustrates the architecture of a deep learning-based motion synthesis framework by Starke et al. [Sta+20].

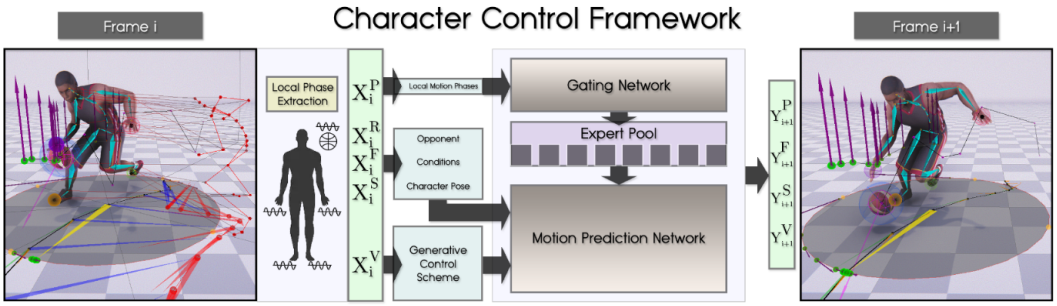


Figure 1.1: The architecture of the MoE-based motion synthesis framework by Starke et al. [Sta+20]

As with every other data-driven approach, these models require a large motion dataset for training, and the data are 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 on new rigs with limited motion data. This project investigates the possibility of three RAE approaches for the proposed **Objective-driven motion generation model (OMG)**, with the goal 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 pre-trained motion synthesis framework such as the Mixture-of-Experts (MoE) based framework by Starke et al. [Sta+20] and the Variational autoencoder-based framework by Zinno et al. [Lin+20], cannot be applied on new character rigs (domain) due to the different configurations, thus new instances must be trained from the ground which requires a large dataset to achieve comparable performance. Acquiring multiple large datasets is generally expensive and time-consuming and should be avoided. Transfer learning is a concept in machine learning about transferring knowledge in a trained model to another model in another domain. It should enable the pre-trained synthesis frameworks to be reused for new rigs to some extent and improve the learning. The hypothesis of this research is formulated as the following:

> H_1**.** Transfer learning techniques can be applied to motion synthesis frameworks for transferring knowledge from the pre-trained instances, to improve the learning of new instances on new domains.

However, there is little research in this particular area, hence the main research question is rather general and formulated as the following:

> **Main research question.** What transfer learning techniques can be applied to motion synthesis frameworks, for improving the learning of the frameworks on new domains and the quality of the synthesised motions? Which of the techniques achieves better transferring quality and generation performance on the new instances with limited training and data?

The transferring quality refers to the generation performance improvement of the transferred models, and the generation performance refers to the reconstruction error and other metrics described in Section 4.4.1.

Experimenting directly with the motion synthesis frameworks from the previous work is infeasible. An alternative framework is proposed and is referred to as Objective-driven motion generation model (OMG). The following sub-questions investigate several deep learning models for OMG to understand which model is better.

> **RQ.1** Which of the two neural networks: Mixture-of-experts (MoE) and Long short-term memory (LSTM) is better suited for Objective-driven motion generation model (OMG), in terms of generation performance and complexity?

> **RQ.2** For feature clustering, which is a transfer learning technique, which of the three neural networks: Radial-basis-function (RBF), Variational autoencoder (VAE) and Deep embedding clustering (DEC) achieves better transferring quality and generation performance with respect to the model complexity?

RQ.1 investigates two model candidates for the motion generation module in OMG, where both are popular neural networks and have been heavily researched in the related area. RQ.2 investigates three model candidates for the rig-agnostic encoding part in OMG with the feature clustering approach.

1.2 Methodology

This project is problem-solving research that follows an experimental design with a deductive approach using quantitative methods. The literature study is conducted to gain a deeper understanding of the topics such as procedural animation, transfer learning and data-driven motion synthesis, to identify and formulate the problem.

Three rig-agnostic encoding (RAE) approaches are defined and a set of neural networks are selected for each of the approaches. The data involves multiple character rigs and motion data for each of the rigs. The rigs are obtained from various sources that represent real-world scenarios in the game industry. The motion data are procedurally generated using an IK-solver for every rig.

An alternative (OMG) to the motion synthesis framework from the previous work is designed. The neural networks are implemented and tested with the created motion data, to find a working architecture and tune the hyperparameters. A set of experiments are designed to test the generation performance of the models and the transferring quality of the RAE approaches, based on the available data, computing power and time. The experiments are carried out on the local machine.

The quantitative results are evaluated using statistical analysis methods for testing the null-hypothesis, to confirm the statistical significance of the improvements. The visual quality of the generated animations are observed and discussed.

1.3 Benefits and Sustainability

By enabling the motion generation networks to learn and produce animations across the rigs, it reduces the required amount of data and the training , saving a great amount of computational resources and time. Furthermore, it opens up new possibilities to train

models for generating animations on characters, where no motion data are available, by using pre-trained models on similar domains.

1.4 Commissioned work

The project is sponsored and supported by EA Digital Illusion CE AB, a Swedish game studio that is behind the first-person shooter game series Battlefield and Star Wars: Battlefront. The studio provides the hardware and the mo-cap data for developing and training the networks. The supervisors from the company provide guidance, advice and help throughout the project.

1.5 Delimitations

This project only focuses on bipedal, humanoid character rigs, though the results should apply to all other characters rigs such as quadruped regardless of the configuration. Since the networks are treating the joints as individual data points and not considering the dependencies between them.

The motion generation networks that are implemented and tested in this project are modified versions of the original models from the previous work, to cope with the created motion data and to fit the time scope. Ideally, to maximise the transferability of the results, the RAE approaches should also be tested on the original models with the original data.

All the motion data for training and testing are from the same category, namely forward reaching animations, where the character reaches both hands to the targets. Diverse motion types are desirable for representing real-world scenarios, but due to the limited time and computing power, this project only focuses on transfer learning over different domains (rigs) with the same fixed task (motion type).

The project only measures the offline performance of OMG, as it is easier and more controllable. The application area of the research is the runtime environment, thus testing the online performance of OMG with the RAE approaches is necessary for future work.

1.6 Outline

Section 2 presents the important background information about computer animation, deep learning and data-driven motion synthesis frameworks.

Section 3 presents the research process and the methodology of the research, and explains the various design choices made in the project.

Section 4 presents the implementations of the models and the RAE approaches, the specifications of the data, the experiments and the evaluation details.

Section 5 presents the obtained results from the conducted experiments, including generation performance of the models, the transferring quality and the statistical analysis.

Section 6 discusses the results in terms of validity and transferability, and how the results answer the research questions. Moreover, the limitations and the future work possibilities are presented in the section.

2 Theoretical background

In this section, the relevant information for understanding computer animation, transfer learning, deep learning-based frameworks for synthesising novel animations are presented. Section 2.2 gives an brief 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 unlocked. 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) [Par12a]. 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 [Par12a], 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 project 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 or a creature. An example of rig hierarchy in the tree structure is demonstrated in Figure 2.1

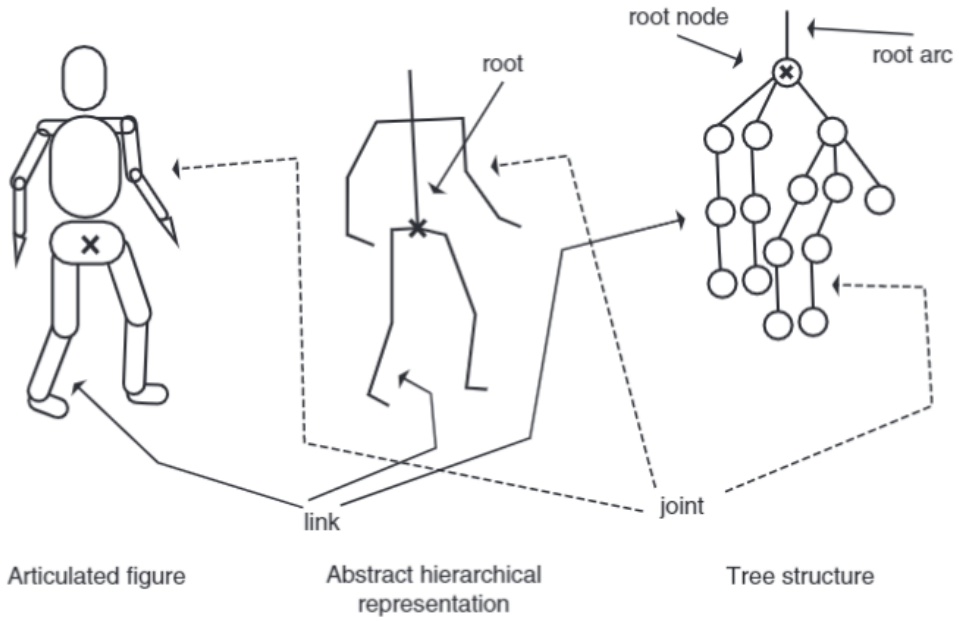


Figure 2.1: Example of a rig structure [Par12b]

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 kinematic traverses through the hierarchy in a depth-first pattern, and propagates the changes of the parent nodes to end effectors, in the joint space. Conversely, inverse kinematics calculates the corresponding joint parameters of all parent nodes, from the end effectors' parameters in Euclidean space [Par12b]. 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 to solving inverse kinematic problems is by using cyclic coordinate descent. It chooses the best value for each joint sequentially, from the outermost inwards, that would get the end effectors to the target positions.

Another iterative approach to solving inverse kinematic problems is the Evolutionary strategy-based solver as described in Section 2.4.5, which is used in this project 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, 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 motion-captured animation for the training of the frameworks.

For this project, DICE sponsored a couple of their motion captured vaulting and weapon reloading animations. However, those clips are not used because the animations are only for their rig.

2.2 Deep learning and transfer learning

Machine learning (ML) is about finding patterns in data and transform them into useful knowledge for tasks such as prediction and classification of new data [Bis09]. 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). The second type is learning that focuses on recognising the relationships between the data points, to extract latent information about the dataset. The third type is the learning of some optimal action policy that maximises some reward function, conditioning on the observations of some state.

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:

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

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

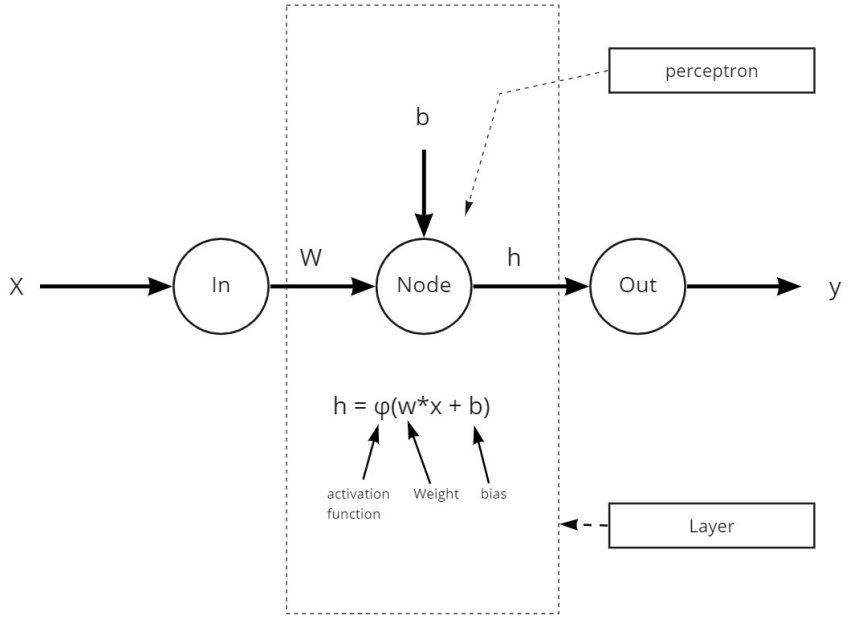


Figure 2.2: Example of feed-forward network operation with a single layer and a single hidden node.

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 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.

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 indicates 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 measure 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 [BKG20]. 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 ($\text{Enc}(\cdot)$) and a decoder ($\text{Dec}(\cdot)$). 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.

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 [HA]. Autoregression is a regression of the variable against itself. An AR model of order p can be expressed as:

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

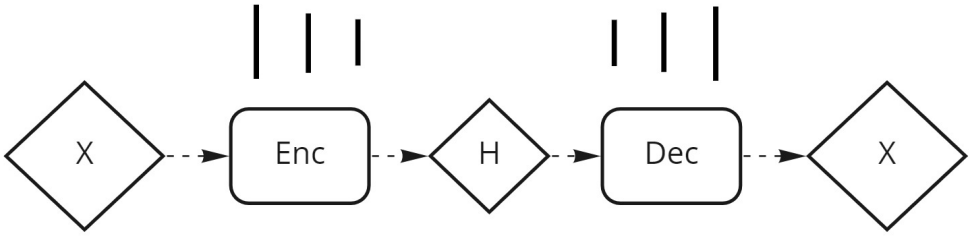


Figure 2.3: Visualisation of an autoencoder.

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.

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 still an expensive and time-consuming process.

According to Zhuang et al:s comprehensive survey [Zhu+19], 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 project only focuses on the former type of transfer learning.

There are many strategies and techniques for transfer learning. The two primary strategies data transformation and model control [Zhu+19]. 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 project focuses on the feature reduction approach, which 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 on the features or the instances, the relationship of the instances 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) [Dai+08]. It assumes that the source domain and the target domain share the same feature clusters in their common feature space. STC co-clusters the source domain and the target domain instances simultaneously.

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

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

2.3 Common neural network models

This subsection presents the relevant neural network models for this project.

2.3.1 Multi-layer perceptron (MLP)

Multi-layer perceptron (MLP), or feedfoward neural network, is the simplest and basic neural network model. It consists of feedfoward 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 [Bis09]:

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

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 [KSH17]. 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 [Bis09], 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. [XGF15]. 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 [Bis09]. 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:

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

$-J(q)$ 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 [Cha16]. It is used in a variety of context including regression, classification and clustering.

The aim of regression is 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 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 [HS97]. 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 special type of generative models, that consists of two modules: a generator and a discriminator. The generator is trained to generates 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 [Mao+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 [HKS17].

2.4.2 Motion alignment models

A straightforward approach to synthesise motion using mo-caps is to align the motion sequences of the same motion type along the timeline and blend them with some blending coefficients computed by the model. Motion matching is a framework for synthesising motion by blending segments of different motion sequences in the database [Hol+20], that satisfy the constraints and the conditions. The optimal segments are obtained by computing the k-nearest neighbour at runtime. Min et al. proposed a model called motion graph++ [MC12], where the motion sequences in the database are represented by functional PCA, and the optimal segments are obtained through maximum a posteriori (MAP).

2.4.3 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) [THR07] and Encoder-Recurrent-Decoder model (ERD) [Fra+15]. 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 [Har+20]) 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 [Sta+20].

Holden et al. proposed a novel neural network structure called Phase-Functioned neural network (PFNN) [HKS17], 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 [Zha+18]. 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 feet 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 [Sta+19]), and complex multi-contact character-object animations (MoE with local motion phase, LMP-MoE [Sta+20]). 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 [Lin+20]). 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.

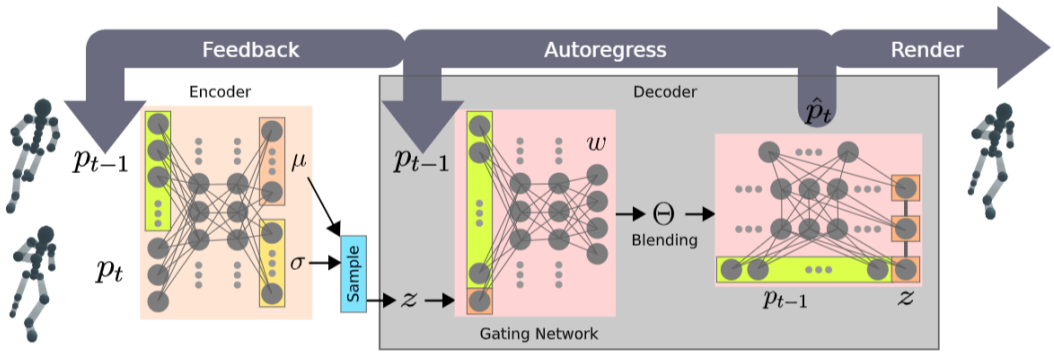


Figure 2.4: The architecture of Motion VAE framework by Zinno et al. [Lin+20]

2.4.4 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. [Hod+95], which computes and applies joint torques to the body, to generate realistic motion. Peng et al. proposed a deep reinforcement learning framework called DeepMimic [Pen+18], that is capable of learning mo-cap data in a physically-based environment. These systems are not experimented with in this project due to the time constraint and complexity.

2.4.5 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 [SHZ19]. 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.

3 Methodology

In this section, the methodology and the processes are presented and explained in detail. The research design is presented in Section 3.1, which covers all scientific components such as the research approach, the research method and the research strategy. Section3.3 presents the process of selecting and designing Objective-driven motion generation model (OMG). Section 3.4 describes the process of acquiring the data needed to train the models. Section 3.5 explains the implementation and the tuning processes.

The research is divided into five stages that are visualised in Figure 3.1

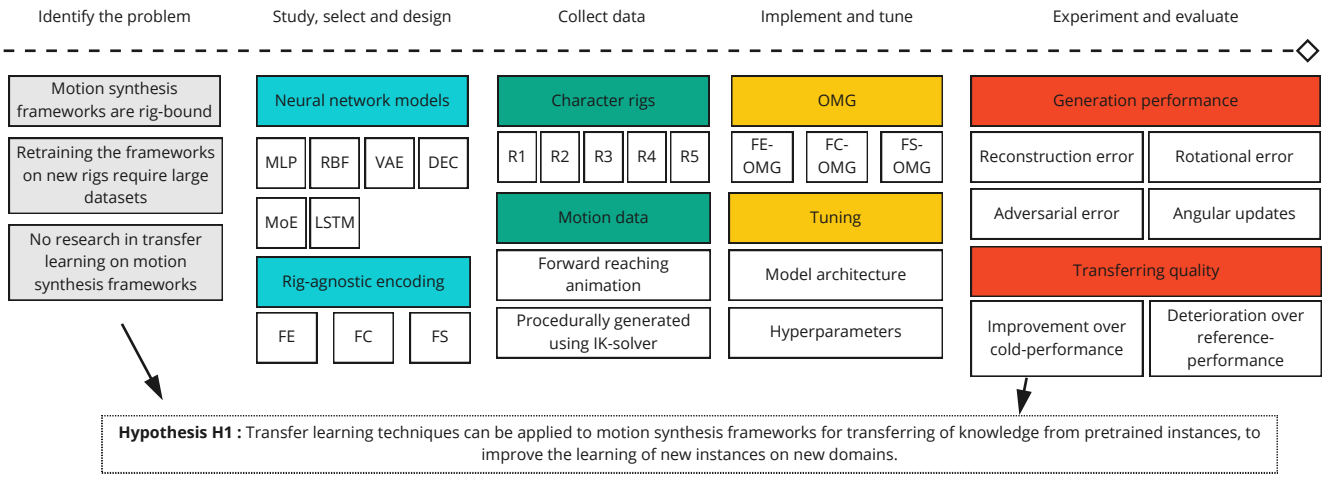


Figure 3.1: The research process with the five stages: 1) understanding and formulating the problem; 2) selecting and designing the models and the transfer learning approaches; 3) collecting training data; 4) implementing and tuning the models and the approaches; 5) conducting the experiments and evaluating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

3.1 Research overview

This is problem-solving research following the deductive approach and the experimental design and relies on qualitative and quantitative methods for evaluation of the results.

The purpose of this research is to study different neural network models for Objective-driven motion generation model (OMG), transfer learning techniques, and the dynamics between them, hence the experimental design and the deductive approach are a natural choice for constructing the research. Another dominant factor of this choice is the difficulty of proving and guarantee the correctness and the validity of the models in a theoretical manner, due to the complex internals of neural networks and the manifold

nature of the datasets. Thus using the empirical results from the experiments that study the relationships between the variables and approaches, is a more time-efficient approach to demonstrate and validate the effect of the various models and techniques.

The experiments are designed to test the selected transfer learning techniques on the implemented OMG. The results are evaluated both qualitatively by observing the generated motions, and quantitatively by measuring the errors and the accuracy of the models. The following qualities of the research are assured:

1. **Validity**, the quantitative results are validated using statistical analysis methods for confirming the statistical significance of the improvements.

2. **Reliability**, all the experiments are conducted on the same local machine. By using a fixed random generator seed (2021), the randomness introduced in the system is controlled and guarantees consistency throughout the experiments.

3. **Replicability**, all details of the implementations and the data generation method is presented in this thesis, making the research easily replicable for other researchers. Moreover, the source code of the project is available on GitHub [Che].

4. **Ethics**, the implementation of the neural networks are based on published work, which is correctly referenced in the thesis. The character rigs are all available and free to use for educational and commercial purposes, except the rig received from DICE. The DICE rig is thus kept private and not included in the GitHub repository. All the motion data are original and created using the Bio-IK plugin in Unity 2.4.5.

5. **Transferability**, is the most important quality of this research. It describes whether the findings of this research applies to other relevant frameworks. This quality is discussed in Section 6.3.

3.2 Problem formulation

The problem is initially an extension to one of the future work directions described in the MANN paper [Zha+18]. The direction is to extend MANN for motion retargeting tasks, making MANN capable of producing motions for quadrupeds of different size and morphology. Since the motion capture of the quadruped is a challenging task, and the motions for different body sizes might not be always available. Starke et al. also mentioned that it would be interesting to investigate feature transfer type of models to transfer the various gaits and movements from one animal to another animal [Zha+18].

Collecting mo-caps of different animals with distinct gaits and styles is difficult and out of scope for this project.

Using MANN to animate other quadrupeds is only possible if the rigs share the same configuration, especially the number of joints since the input and the output layers of the model are dependent on the pose vectors, which in turn are dependent on the rig configuration. This limitation also applies to many other motion synthesis frameworks. The closest research is Neural Kinematic Network (NKN) for motion retargeting, by Villegas et al. [Vil+18]. It is based on a regular Recurrent Neural Network (RNN) that projects the input pose from one character to other characters with different skeleton bone length, which is the only difference between the rigs. NKN is only for motion retargeting and cannot generate novel motions, and it does not work on rigs with a different number of joints.

The problem of this research is identified as investigating and finding methods for overcoming this limitation and making the motion synthesis frameworks such as MANN function on other character rigs with different configurations. The problem falls under the transfer learning domain and is formulated as the following based on the definition of transfer learning described in Section 2.2.5:

- Given a **motion generation network** (MoGen) that is trained with **motion data** from **bipedal rig 1** (source domain D_S) to generate certain type of motion such as locomotion (source task T_S), what **transfer learning techniques** can be used to make MoGen produce same type of motion (target task $T_T = T_S$) on **bipedal rig 2** (target domain D_T) of different configuration, including different number of joints, with smaller training dataset?

3.3 Model selection

Ideally, the transfer learning techniques should be tested directly on the most recent motion synthesis frameworks such as LMP-MoE [2.4.3], MVAE [2.4.3] or TR-LSTM [2.4.3]. Fortunately, these models are well-documented and explained in the papers, and the code including the datasets are all available on GitHub, facilitating the replication of the models. However, since the focus is on transfer learning techniques, motion data of the same type on multiple rigs are required, and these models are all designed for different rigs and conditioned on different control signals. In addition, for easier and consistent comparison between models and approaches, only the offline performance are considered, hence using joystick inputs as the control signal, as with LMP-MoE, is not an

ideal choice to drive the motion prediction. MVAE uses a reinforcement learning-based controller, which would add unnecessary complexity and uncertainty to this project. TR-LSTM uses predefined keyframes as conditions for pose prediction, which suits the scope of this project but it would imply completely discard any possibility of using the model in a real-time environment in the future. Thus, a modified motion generation network is necessary.

3.3.1 Objective-driven Motion Generation network (OMG)

A 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 essentially LMP-MoE with some cost as the condition, which is computed from some objective function. The objective function should return a quantity that needs to be minimised. For instance, it could be a function of the resultant force acting on the rigid body, making the framework to learn physics-based behaviours such as ragdoll physics.

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. By using temporally sparse targets, OMG can be effectively used as TR-LSTM. OMG can also be used in real-time applications by specifying the targets at runtime.

The local motion phase can be computed from the contact patterns of the key joints and their target positions along the timeline. The main network (MoGen) can be either MoE or LSTM. Starke et al. did compare their MoE-based frameworks with the LSTM counterpart, and concluded that MoE performs better with fewer animation artefacts such as foot sliding and pose instability and achieved higher responsiveness [Zha+18; Sta+19; Sta+20]. However, the models were tested in a real-time environment with user inputs as the control signal, hence it is interesting to test whether the conclusion holds in the offline environment with fixed objectives as the control signal.

3.3.2 Transfer learning techniques

There are two main approaches in transfer learning, data transformation and model manipulation. Intuitively, the feature reduction techniques in the data transformation category are suitable candidates for this project. Since the target task remains the same as

the source task, thus the MoGen should not be altered. Using feature reduction techniques to transform the data points to a more abstract representation in the shared feature space, reduces the complexity of the data and the domain differences. The transformation is referred to as rig-agnostic encoding in this project.

3.3.3 Rig-agnostic encoding

Three approaches for achieving rig-agnostic encoding are presented as the following:

1. **Feature encoding (FE)**, by integrating an autoencoder (AE) with MoGen, the pose data can be encoded into compact embeddings, which are fed to the MoGen. MoGen outputs the pose embeddings in the same latent feature space, which is then fed to the decoder to construct the pose vector in the original space.

2. **Feature clustering (FC)**, unsupervised clustering on the pose embeddings in the latent feature space can be performed by inserting an extra layer in between the AE and MoGen. The relationship of the pose embeddings and the cluster centres can be used as inputs (FC-IN) or auxiliary features to MoGen (FC-CAT), similar to the STC model as described in Section 2.2.5. The cluster centres can be considered as specific poses in the latent space and they should be selected to best capture the distribution of the pose domain. In this project, RBF, VAE and DEC are tested as the clustering layer.

    - RBF operates on the radial distance between the embeddings and the C clusters. It learns the centres $\{\mu_1, \mu_2, ..., \mu_C\}$ and the range of the centres $\{\sigma_1, \sigma_2, ..., \sigma_C\}$, and outputs a vector of radial distances to the respective centres given a pose embedding. An orthogonality loss term is added to regularise the clustering, promoting the layer to find orthogonal centres.

    - VAE learns the distribution of the latent pose domain, as factorised by C Gaussian distributions. The layer learns C set of Gaussian parameters $\{(\mu_1, \sigma_1), ..., (\mu_C, \sigma_C)\}$, and outputs a latent vector sampled from these distributions.

    - DEC is similar to RBF that learns a set of cluster centres but it uses Students' t-distribution for measuring similarity between the pose embeddings and the cluster centres. It outputs a vector of probabilities belonging to the respective cluster.

3. **Feature selection (FS)**, since the blocking issue of directly reusing the networks on other domains is the difference in the input and output dimensions, it can be avoided by simply using a fixed subset of features for all domains. More specifically, by using only 6 key joints (feet, hands, spine and head) to represent the pose, it can represent the poses from all domains in pose vectors of the same length, and also greatly reduces the dimensionality of the data. Kinematic solvers can then be used to compute the parameters of other joints for the full-body pose, which adds extra complexity. For simplicity, MoGen is trained to generate the full-body pose in the next frame given the pose embedding from the 6 key joints.

The RAE approaches as illustrated in Figure 3.2.

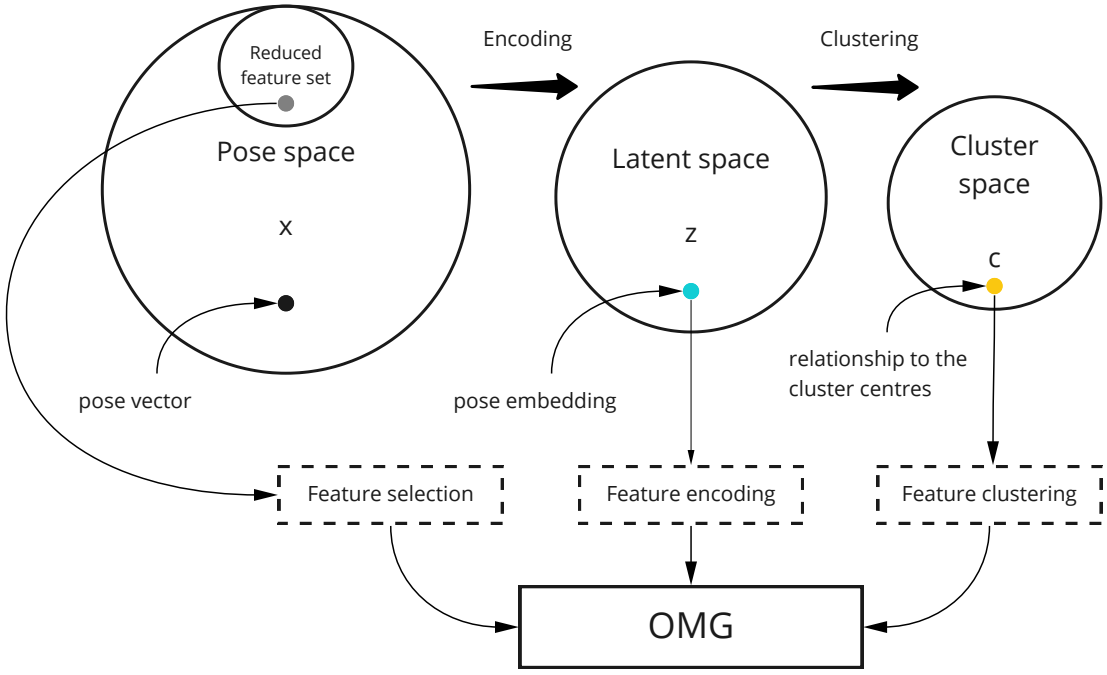


Figure 3.2: The illustration of the three RAE approaches, where the points are data points in the respective space. The figure displays the transformation of the pose vectors to achieve rig-agnostic encoding.

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 configuration except for the input and output layers. When using feature clustering, the clustering layer is also transferred to the new domain, to obtain the relationship of the new pose embeddings to the cluster centres that represent the previous domain.

3.4 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.

3.4.1 Character rigs

It requires a set of character rigs that share different configurations including the number of joints, the topology and the proportion. The more rigs and the more variation in the rigs, the better generalisation of the results. 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 [Sta+19].

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.

3.4.2 Motion data

Creating motion data using motion capture is not feasible due to limited resources and time. Since OMG is objective-driven, using IK-solver to procedurally animate the characters to create motion data is a reasonable choice. This method is time-efficient and scalable, and it ensures full control over the generated motion data for all the rigs.

The motion sequences are created in Unity using the Bio-IK described in Section 2.4.5. 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 which is going to reach the hands towards the targets. 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 root of the rig 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 3.3.

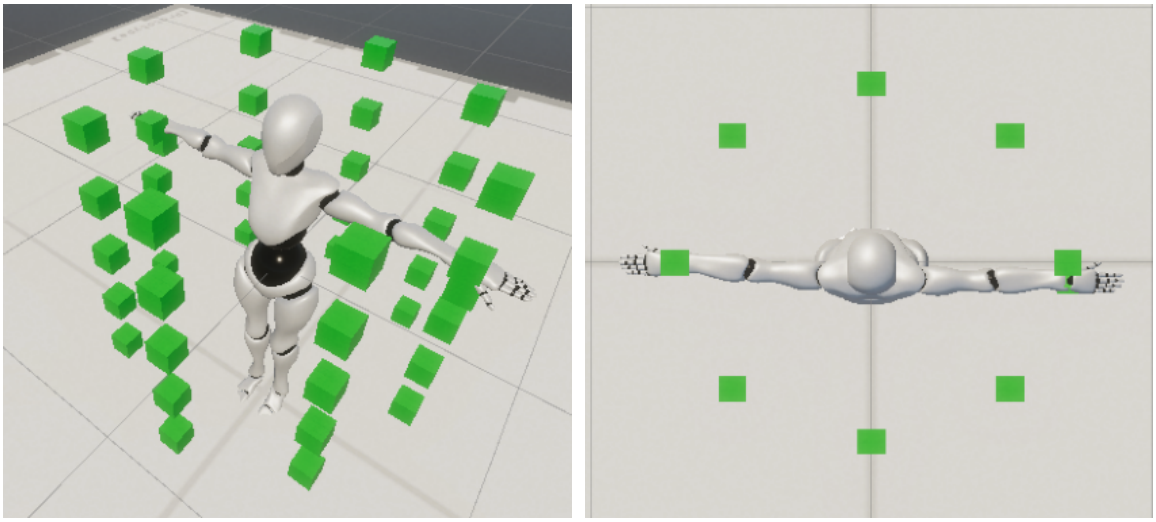


Figure 3.3

For a 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.

For 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.

In total, there are 960 sequences created, which corresponds to 288 000 frames (80 minutes) of motion data.

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

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

3.5 Implementation and tuning of the models

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

The models are implemented in Python using Pytorch and Pytorch-Lightning. The implementation of the models are based on MANN, NSM, LMP-MoE, MVAE and TR-LSTM [Zha+18; Sta+19; Sta+20; Har+20]

The implemented models are tested with a small subset of the dataset, to ensure the correctness of the implementations, that is, 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 with ASHA scheduler and a grid search algorithm.

3.6 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: R2, R3, R4 and R5, by loading the parameters of OMG that is fully trained on the domain R1, with limited training and data.

Generation performance is quantified by the four metrics: the reconstruction error, the adversarial error, the rotational error and the sum of angular updates, which are described in Section 4.4.1. Transferring quality refers to the improvement of the generation

performance (warm-performance) over the performance from the cold-started instances (cold-performance), that is, the new instances with randomly initialised parameters and they are only sparingly trained with the limited data. Performance improvement in this context refers to error reduction. Furthermore, the performance deterioration over the performance from the reference instances (reference-performance), which are fully trained with all the training data, is also computed and compared.

For each of the transferred (warm-started) models, two training strategies are tested: *frozen (F)* and *trained (T)*. The former freezes the inherited clustering layer and the MoGen module, and only allow the AE of the OMG to be trained. This provides insights into the transferring capability of the three RAE approaches and the models. The latter allows all modules to be trained, which demonstrate the improvement headroom of the warm-started 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

3.6.1 Evaluation methods

The collected performance data from the experiments are compared, the improvements over cold-performance perf_I and the deterioration over the reference-performance perf_D are computed and compared between the models and approaches.

$$\text{perf}_I(OMG) = \frac{\mathcal{L}_{OMG_1} - \mathcal{L}_{OMG_2}}{\mathcal{L}_{OMG_1}} \tag{1}$$

$$\text{perf}_D(OMG) = \frac{\mathcal{L}_{OMG_0} - \mathcal{L}_{OMG_2}}{\mathcal{L}_{OMG_0}} \tag{2}$$

$$\tag{3}$$

where $\mathcal{L}_{OMG_0}$ is the reference-performance of the model OMG, $\mathcal{L}_{OMG_1}$ is the cold-performance and $\mathcal{L}_{OMG_2}$ is the warm-performance.

The quality and the correctness of the generated animation clips from the models are visualised and observed. The statistical significance of the improvements is verified using Independent two-sample T-test and Mann-Whitney U test [Bis09]. If the p values

from both tests are below 0.05, which corresponds to a 95% confidence level, the null hypothesis H_0 is rejected.

> H_0. There are no statistical significant difference between the cold-performance and the warm-performance.

In detail, the reconstruction error of every sample in the test set from a warm-started model is tested against the reconstruction errors of a cold-started model. The exact same test set is used for both models.

4 Implementation

This section presents the specifications of the generated motion data, the features that represent the sequences, the model implementation details and the evaluation metrics. Section 4.1.2 described the features that are used to train the models, and how they are computed. Section 4.2 presents the implementation details and the parameters of the models. Section 4.4.1 presents the metrics used to quantify the generation performance of the models.

4.1 Data

The motion data that is used for training and testing the models, is the reaching animation clips that are generated for each of the five character rigs. In each clip, the character reaches forward with both hands to the targets, minimising the distance to the target position and the target rotation. 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.

4.1.1 Character rig specification

The specification of the five character rigs are the following:

Rig	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

4.1.2 Data features

The OMG models are primarily based on LMP-MoE [2.4.3] and MVAE [2.4.3], 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; $Tc_i^p \in \mathbb{R}^{3K}$ is the position cost for each key joint; $Tc_i^r \in \mathbb{R}^{3K}$ 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 4.1.3.

All the features are transformed to the rig root space.

4.1.3 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 [Sta+20] 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:

$$G_{norm}(t) = \frac{G(t) - \mu_{G_W}}{\sigma_{G_W}} \tag{4}$$

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.

$$G_{\mathcal{B}} = \mathcal{B}(G_{norm}) \tag{5}$$

The local motion phase vector is constructed as the following:

$$\Theta = \begin{pmatrix} \Delta_{sin} \cdot sin(G_{\mathcal{B}}) \\ \Delta_{cos} \cdot cos(G_{\mathcal{B}}) \end{pmatrix} \tag{6}$$

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 4.1.

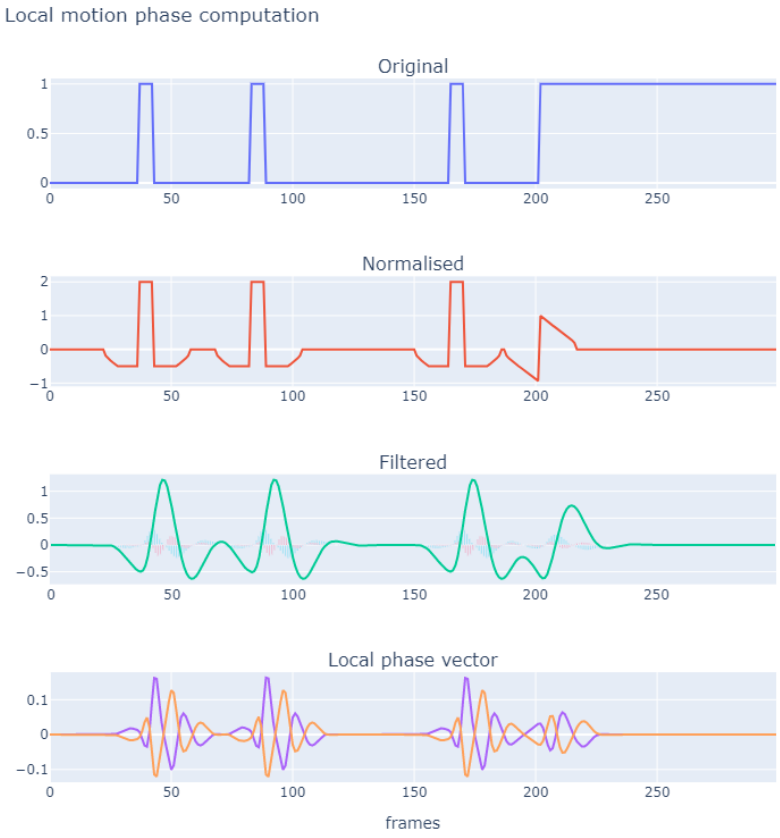


Figure 4.1: The outcome of the different steps in the computation of local motion phase.

4.1.4 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.

$$C^p = T^p(j) - p(j) \tag{7}$$

$$C^r = \frac{1}{\pi} cos^{-1}(\frac{T^r(j) \cdot r(j)}{||T^r(j)|| \cdot ||r(j)||}) \tag{8}$$

4.2 Models

The following neural network models are implemented:

1. **Autoencoder [2.2.3], AE**$(\cdot)$ with an encoder Enc$(\cdot)$ and a decoderDec$(\cdot)$ with the same architecture, where both are implemented as three-layer MLP [2.3.1] with following network operation:

$$\text{Enc}(x; W, B) = \text{Dec}(x, W, B) = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{9}$$

   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.

2. **LSGAN discriminator [2.3.8], D**$(\cdot)$, 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.

$$L1 = \text{BatchNorm}(\text{MaxPool}(\text{Conv}(x, w_1, b_1))) \tag{10}$$

$$L2 = \text{BatchNorm}(\text{MaxPool}(\text{Conv}(L1, w_2, b_2))) \tag{11}$$

$$L3 = w_3(\text{Flatten}(L2)) + b_3 \tag{12}$$

$$D(x; W, B) = L3(L2(L1(x))) \tag{13}$$

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

3. **RBF clustering layer [2.3.3], RBF**$(\cdot)$ with Gaussian basis function $\phi(\cdot)$ for clustering the pose embeddings.

$$\text{RBF}(x; \mu, \sigma) = \phi(\sigma||x - \mu||) \tag{14}$$

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

$$\mathcal{L}_{ortho} = 0.1 \cdot \mathbb{E}(|\mu\mu^T - I|_1) \tag{15}$$

4. **VAE layer [2.3.5], VAE**$(\cdot)$ with 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.

$$\text{VAE}(x; W, B) = \{\mu = w_1 \cdot x + b_1, \sigma = w_2 \cdot x + b_2\} \tag{16}$$

The reparameterisation trick [Lin+20] is implemented to sample the latent vector from the computed distributions:

$$z = \mu + \sigma \cdot \varepsilon, \quad \text{where } \varepsilon \sim \mathcal{N}(0, 1) \tag{17}$$

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:

$$\mathcal{L}_{\text{KL}} = \beta * \mathbb{E}[-\frac{1}{2}\sum(1 + \sigma - \mu^2 - e^{\sigma})] \tag{18}$$

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

5. **DEC layer [2.3.4], DEC**$(\cdot)$ that 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.

$$\text{DEC}(x; \mu) = \frac{(1 + ||x - \mu||^2)^{-\frac{1}{2}}}{\sum[(1 + ||x - \mu||^2)^{-\frac{1}{2}}]} \tag{19}$$

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.

6. **MoE network [2.3.6] MoE**$(\cdot)$ with k experts is used for motion generation with a gating network $\Omega(\cdot)$ 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.

$$\Omega(\Theta; W, B) = U = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{20}$$

$$N(x; W, B, U) = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{21}$$

$$\text{where } w_l = \sum_{e=1}^{K} W_i^{(e)} \cdot U^{(e)}, \quad l = 1, 2, 3 \tag{22}$$

where e denotes the e-th expert.

7. **LSTM network [2.3.7] LSTM**$(\cdot)$ 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.

8. **Objective-driven motion generation network OMG**$(\cdot)$ is the complete framework for generating motion given the objectives. 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; MoGen$(\cdot)$ is the main generation network (either MoE or LSTM). The framework is visualised in Figure 4.2 with MoE as the motion generation network.

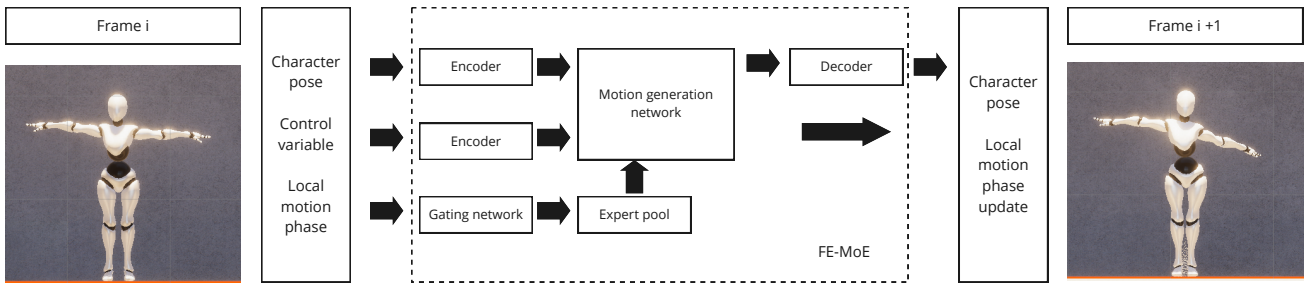


Figure 4.2: The overview of the architecture of FE-OMG with MoE for the motion generation module. The local motion phase variable X_i^P is fed to the gating network to produce the weights for the main network. The pose data X_i^S and the control variable X_i^V are encoded by the respective encoder and fed to the main network to produce the next frame.

The following are three variants of OMG:

- **FE-OMG**$(x; \theta)$ is a framework consisting of only AE and MoGen. It implements the feature encoding technique. The pose encoding z is directly

used as input to MoGen. The architecture is visualised in Figure 4.3.

$$
\begin{aligned}
X_i^P, X_i^S, X_i^V &= \text{slice}(X_i) \\
z_i &= \text{Enc}(X_i^S) \\
c &= \text{CostEnc}(X_i^V) \\
Y_{i+1} &= \text{MoGen}(X_i^P, z_i \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} &= \text{slice}(Y_{i+1}) \\
Y_{i+1}^P &= \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S &= \text{Dec}(z_{i+1}) \\
Y_{i+1}^V &= \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} &= Y_{i+1}^P \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned}
\tag{23}
$$

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

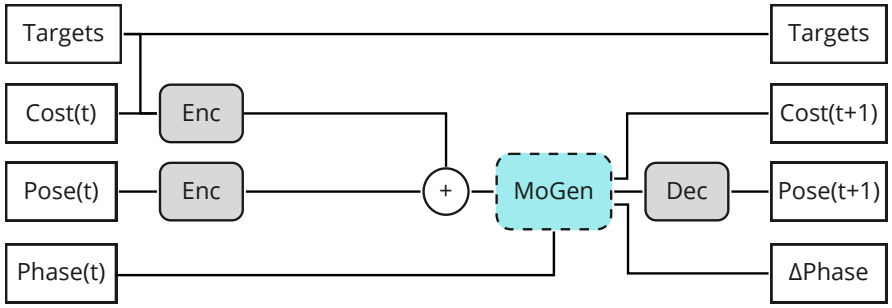


Figure 4.3: Architecture diagram of FE-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P and t is the frame index. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder and MoGen is the motion generation network (MoE / LSTM)

- **FC-IN-OMG**$(x; \theta)$ is a framework that implements the feature clustering technique. It uses a clustering layer (either RBF or VAE or DEC) to obtain the cluster information Ψ of the pose embeddings, which is used as input to

MoGen. Figure 4.4 displays the architecture diagram.

$$
\begin{aligned}
X_i^P, X_i^S, X_i^V &= \text{slice}(X_i) \\
z_i &= \text{Enc}(X_i^S) \\
\Psi &= \text{C}(z_i) \\
c &= \text{CostEnc}(X_i^V) \\
Y_{i+1} &= \text{MoGen}(X_i^P, \Psi \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} &= \text{slice}(Y_{i+1}) \\
Y_{i+1}^P &= \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S &= \text{Dec}(z_{i+1}) \\
Y_{i+1}^V &= \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} &= Y_{i+1}^p \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned}
\tag{24}
$$

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

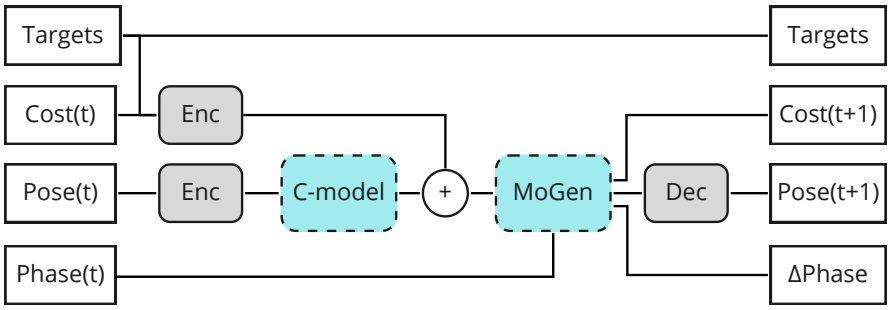


Figure 4.4: Architecture diagram of FC-IN-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder, C-model denotes the clustering layer (RBF/VAE/DEC) and MoGen is the motion generation network (MoE / LSTM)

- **FC-CAT-OMG**$(x;\theta)$ is similar to FC-IN-OMG, but using Ψ as auxiliary features, that are concatenated with the pose embedding z before feeding to

MoGen. The network is illustrated in Figure 4.5.

$$
\begin{aligned}
X_i^P, X_i^S, X_i^V &= \text{slice}(X_i) \\
z_i &= \text{Enc}(X_i^S) \\
\Psi &= \text{C}(z_i) \\
c &= \text{CostEnc}(X_i^V) \\
Y_{i+1} &= \text{MoGen}(X_i^P, z_i \oplus \Psi \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} &= \text{slice}(Y_{i+1}) \\
Y_{i+1}^P &= \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S &= \text{Dec}(z_{i+1}) \\
Y_{i+1}^V &= \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} &= Y_{i+1}^P \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned}
\tag{25}
$$

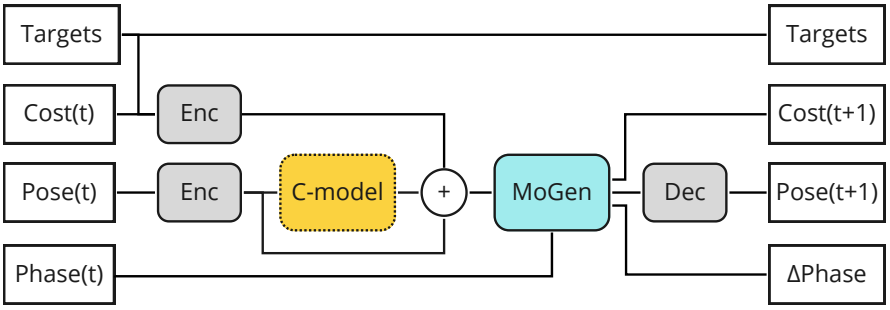


Figure 4.5: Architecture diagram of FC-CAT-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder, C-model denotes the clustering layer (RBF/VAE/DEC) and MoGen 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:

$$
\text{update}(X_i^p, \Delta\Theta) = 0.9 \cdot X_i^p + 0.1 \cdot \Delta\Theta \tag{26}
$$

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

- **FS-OMG**$(x; \theta)$ refers to FE-OMG but implements 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, but operating this reduced feature set.

4.2.1 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

4.3 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 4.1.2. 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 $32 \times 299 \times n$. The matrix 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 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 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 0 at the beginning and is gradually incremented to 1.

The following loss terms 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.

4.4 Experiments

The following experiments are conducted:

1. **(E0)** Train the autoencoder and its LSGAN discriminator on all domains (R1-R5) for 200 epochs.

   Expected results: a set of trained autoencoders $\{\text{AE}_{R1}, \text{AE}_{R2}, ..., \text{AE}_{R5}\}$, the reference performance data of the autoencoders and the reference adversarial losses from the discriminator.

2. **(E1)** Train OMGs (FE-OMGs, FC-OMGs, FS-OMGs) on all domains (R1-R5) 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).

Expected results: a set of trained OMGs with the reference performance data on all the domains. Demonstrating the level of performance that is expected if the models are trained with the full training dataset.

3. **(E2)** Train the OMGs 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).

   Expected results: the cold-performance data of the models on all domains (R2-R5) with limited training and data.

4. **(E3)** Same procedure as E2 but warm start the models by loading the parameters from the pre-trained instances OMG_{R1} from E1. The *"frozen"* training strategy is applied.

   Expected results: the warm-start performance data of the models on all domains (R2-R5) with the knowledge in the pre-trained instances on R1 transferred. The clustering layer and the MoGen are frozen to better demonstrate the effect of the RAE approaches.

5. **(E4)** Same procedure as E3 but the *"trained"* training strategy is applied.

   Expected results: the performance data showing how much the performance can be improved by allowing all modules to be trained.

The variables that are experimented in each of the experiments, are the following:

- **MoGen module**: MoE, LSTM
- **Clustering layer**: RBF, VAE, DEC
- **Feature set**: full resolution pose data, reduced resolution (6-joints) pose data.

4.4.1 Metrics

The following metrics are measured for quantifying 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.

$$\mathcal{L}_{\text{recon}} = \text{MSE}(Y_i, X_i) \tag{27}$$

- **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. It is highlighted because it explains the most of variances in the performance differences, as observed in the empirical results.

$$\mathcal{L}_{\text{r}} = \text{MSE}(y^r, x^r) \tag{28}$$

- **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.

$$\mathcal{L}_{\text{adv}} = \frac{1}{2}\mathbb{E}[(\text{D}(Y^S) - 1)^2] \tag{29}$$

- **Sum of angular updates,** $\mathcal{L}_{\Delta R}$, it is the the absolute rotation differences in radian, averaged over all joints and summed over all frames in the generated clips. It indicates the smoothness of the generated animations as described in the LMP-MoE paper [Sta+20].

$$\mathcal{L}_{\Delta R} = \sum_{i=0}^{299} \mathbb{E}_{\text{joints}}[cos^{-1}(\frac{r_i \cdot r_{i+1}}{||r_i||||r_{i+1}||})] \tag{30}$$

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 and it is the main factor 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 CPU to avoid potential synchronisation issues and the unpredicted internal data transportation to GPU.

5 Evaluation

This section presents the main results from the conducted experiments [4.4]. Section 5.1 presents the properties of the most relevant models. Section 5.2 presents the generation performance of the models from Experiment E1. Section 5.3 presents the transferring quality using the three RAE approaches, from Experiments E2, E3 and E4. Section 5.4 presents the final comparison of the best performing models for the respective RAE approach.

5.1 Model property

The number of parameters, the memory footprint and the inference time on CPU of the respective model is presented in Table 5.1. The reported values in the table are measured on the implemented models for R1. It shows that the MoE module is responsible for the major complexity of the framework and the LSTM module is roughly three times larger than the MoE counterpart. It is worth noting that the FC-IN-OMG models are smaller and faster than AE+MoE, despite having an extra 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. The complete list of all models can be found in Appendix A.

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 5.1: The complexity properties of the most relevant models.

5.2 Generation performance

In this subsection, the generation performance of the implemented models is presented. It demonstrates the capabilities of the models if fully trained with all the training data and it sets the upper bound of the transferred quality.

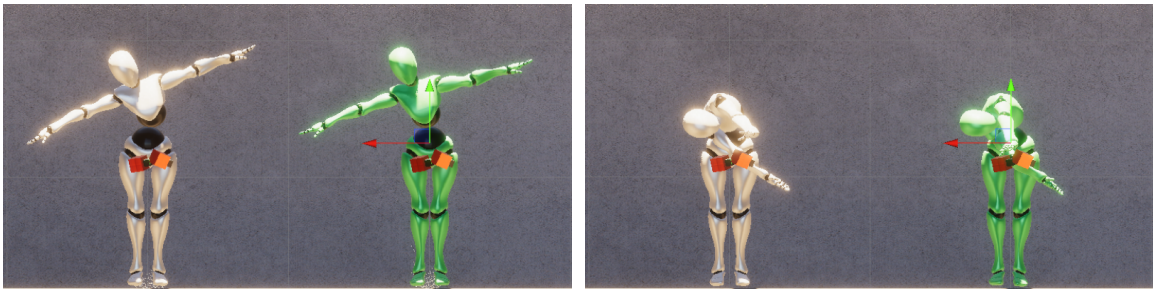
The plot in 5.1 displays the generation performance comparison of MoE and LSTM for the FE-OMG models. The confidence interval at 95% confidence level is also plotted in the

figures. It illustrates that AE+MoE performs significantly better than AE+LSTM on three of the four metrics.

Figure 5.1: The generation performance of FE-OMG models and the respective confidence interval (95%) are displayed. The values are averaged over the domains. The results show that the MoE module achieves better performance than the LSTM module.

Figure 5.2 shows two frames of the clip that is generated by AE+MoE and AE+LSTM. The left white character shows the target pose and the right green character shows the generated pose. Both characters should reach their hands towards the targets which are the red cubes in the scene. AE+MoE is able to generate accurate and smooth animations whereas AE+LSTM fails to produce animations of the same quality, and it introduces a visible amount of jittering. The jittering is also indicated by the higher sum of angular updates in Figure 5.1.

In Figure 5.3, the generation performance comparison of the relevant FE-OMG and FC-OMG models are displayed. The LSTM variants are not included in the graphs, but the exact performance data can be found in Table D.1. By comparing the results, it is clear that AE+MoE, RBF-CAT+MoE and VAE-CAT+MoE are the best performing models. FC-CAT-OMG variants perform better than the FC-IN-OMG variants. For clustering models, RBF and VAE achieve similar performance whereas DEC is slightly worse. However, the VAE variant also introduces a small amount of jittering, as indicated by the higher sum of angular updates. The screenshots of the generated clips by the models are presented in Appendix B. The animation quality matches the findings on the numerical results, that FC-CAT+OMG generates accurate and smooth animations than FC-IN-OMG.



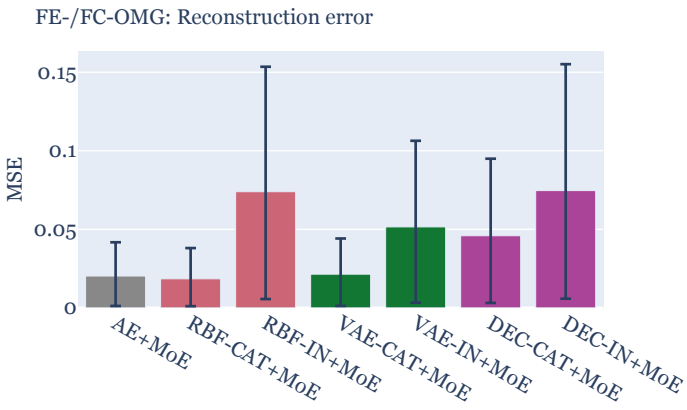
(a) AE+MoE at frame 22

(b) AE+MoE at frame 35

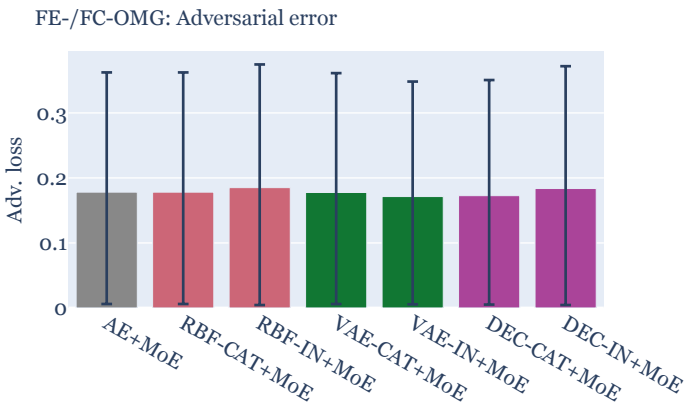
(c) AE+LSTM at frame 22

(d) AE+LSTM at frame 35

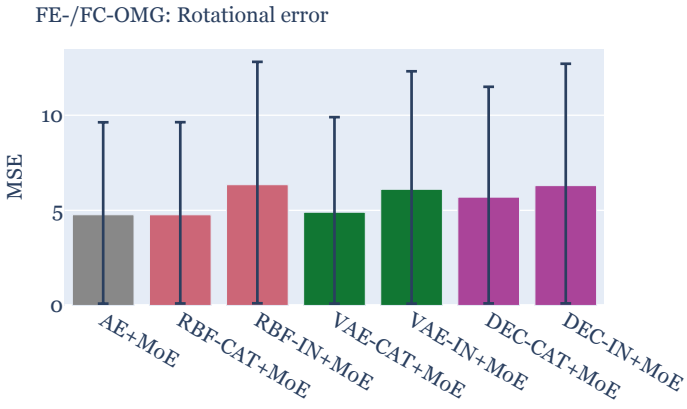
Figure 5.2: The screenshots of the generated clip (the green character) by AE+MoE and AE+LSTM, compared against the target clip (the white characeter). AE+MoE achieves better generation performance than AE+LSTM in terms of accuracy and smoothness.



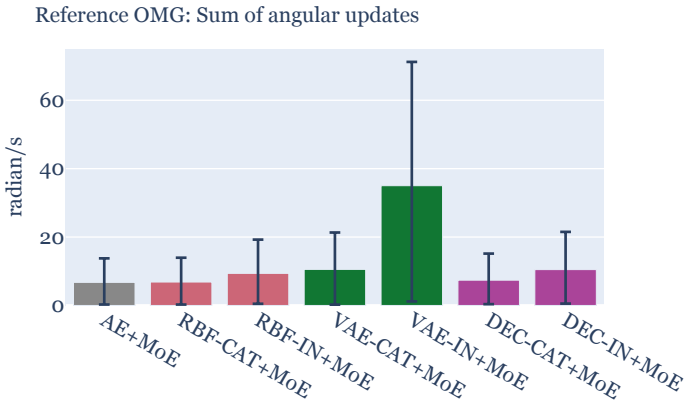
(a) Reconstruction error



(b) Adversarial error



(c) Rotation error



(d) Sum of angular updates

Figure 5.3: The reference performance of FE-OMG and FC-OMG models. The values are averaged over the domains. It shows the FE-OMG and FC-CAT-OMG are the better performing models.

Figure 5.4 displays the reconstruction error of FS-OMG and FS-CAT-OME models. All models achieve good performance, even when compared to the other FE-OMG and FC-OMG models that operate on the full-resolution pose data. Among the four models, RBF-CAT+MoE is the best performing model and DEC-CAT+MoE is the worst of them.

Figure 5.5 shows the reconstruction error of the FE-OMG, FC-CAT-OMG and FS-OMG models for each of the rigs. Even when operating at the reduced feature set, the models manage to achieve comparable generation performance with slight visual artefacts, as visualised in the Figure 5.6. For all RAE approaches, AE+MoE, RBF-CAT+MoE and VAE-CAT+MoE achieve consistent performance on all the rigs, while DEC-CAT+MoE performs worse on all the rigs.

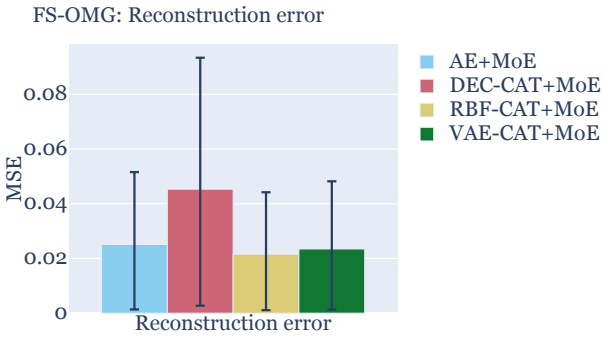
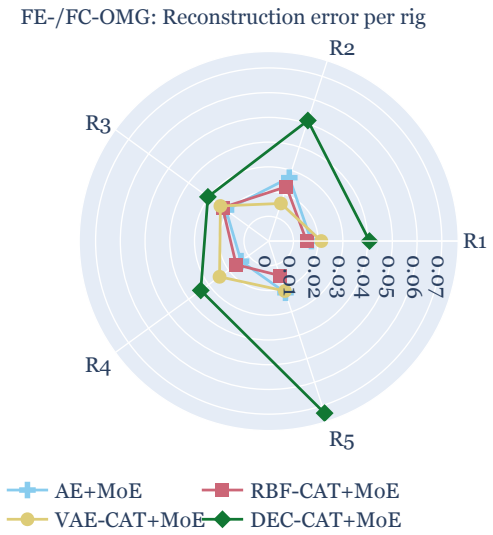
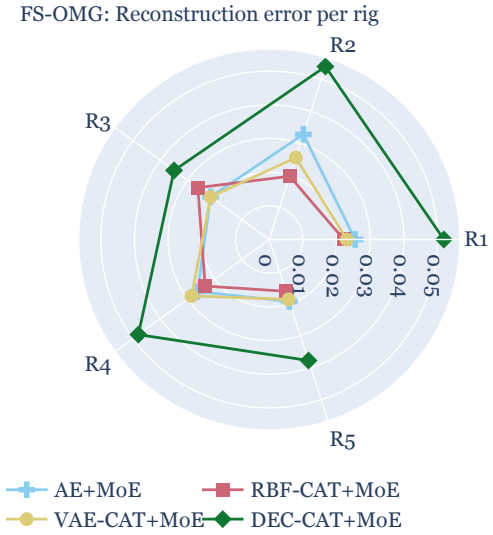


Figure 5.4: The generation performance of FS-OMG models, and the respective confidence interval (95%) are displayed. The values are averaged over the domains. All models have achieved good performance even when compared to the FE-OMG and FC-OMG models



(a) FE-OMG and FC-CAT-OMG

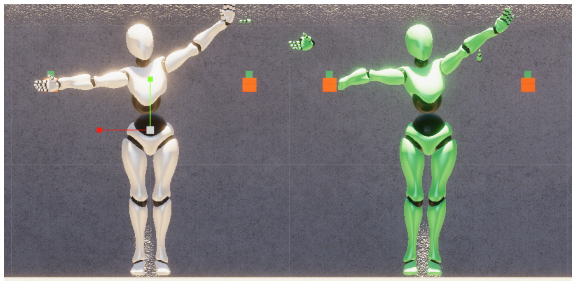


(b) FS-OMG and FS-CAT-OMG

Figure 5.5: The reconstruction error of FE-OMG, FC-CAT-OMG and FS-OMG models on each of the rigs. It shows that the performance of the models are consistent over the different domains, and DEC-CAT-OMG is the worse-performing model.

(a) AE+MoE at frame 22

(b) AE+MoE at frame 35



(c) Visual artefacts of Ae+MoE at frame 96

Figure 5.6: The screenshots of the generated clip (the green character) by AE+MoE on reduced feature set, compared against the target clip (the white character). The input pose data only contains information about 6 joints, and the model internally upsamples to the full resolution pose at the output. The generated clips show comparable quality as observed in Figure 5.2 with slight visual artefacts.

5.3 Transferring quality

In this section, the transferring quality of
the relevant models is presented.

The quality is quantified by the improvements over the cold-performance and the deterioration over the reference-performance for the respective model. The plots illustrate how the warm-performance data relate to the cold-performance data or the reference-performance data. For instance, a positive value in the reconstruction error figure implies a reduction over the previous error and a negative value implies an increase.

5.3.1 FE-OMG

Figure 5.7 displays the performance gains of AE+MoE when warm-started with the parameters of the fully trained AE+MoE model on R1. The performance values are averaged over R2, R3, R4 and R5. Frozen-AE+MoE denotes the ”frozen” training strategy where only the autoencoder is being trained, and Trained-AE+MoE denotes the ”trained”

training strategy where all the modules are being trained. It shows that warm-starting the model yields approximately a 40% reduction in reconstruction error over the cold-performance. Three of the four metrics show an improvement. The sum of angular updates metric shows a decline in performance, which can be explained by the model starts to generate more varied poses.

Figure 5.8 displays the performance of AE+MoE when compared against the reference-performance. It shows that reconstruction error is roughly doubled when compared to the reference values. An increase of 20%-30% is observed in both the adversarial error and the sum of angular update metric, whereas the rotational error is about the same as the reference values.

In both cases, it shows that by adopting the ”trained” strategy, the performance is slightly improved.

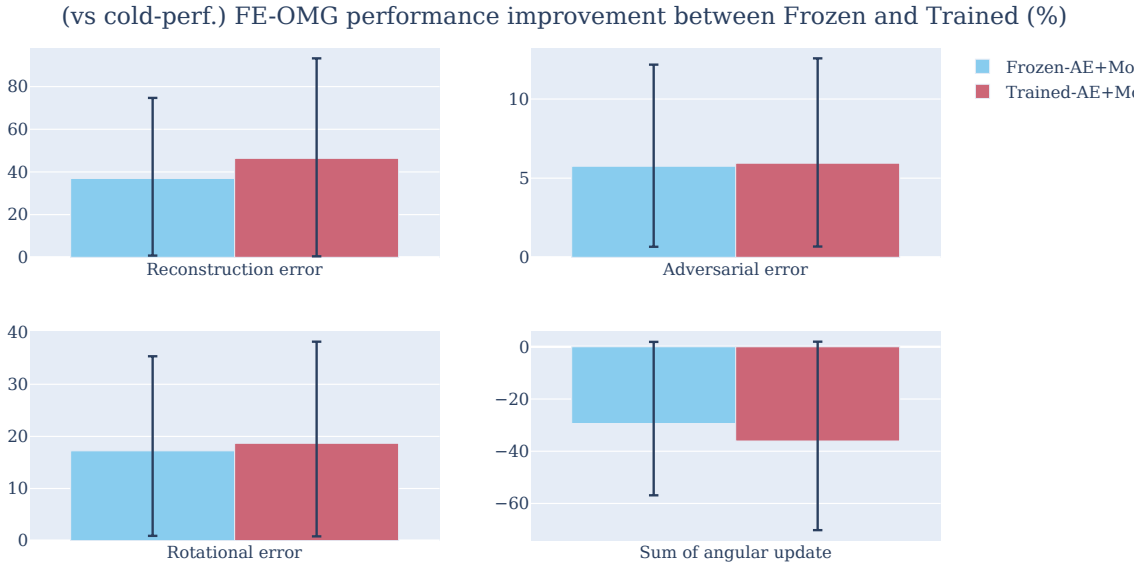


Figure 5.7: The transferring quality of AE+MoE in relation to the cold-performance. It illustrates that by warm-starting the model yields approximately 40% reduction in the reconstruction error when compared to the cold-performance.

5.3.2 FC-OMG

Figure 5.9 and 5.10 present the reconstruction error of the transferred FC-models in relation to the cold-performance and the reference-performance. The confidence interval at 95% confidence level is also plotted. Similar to the reference performance, FC-CAT-OMG models are better than FC-IN-OMG models in terms of improvements over the cold-performance, except for DEC-CAT+MoE. Despite an approximate 40% reduction

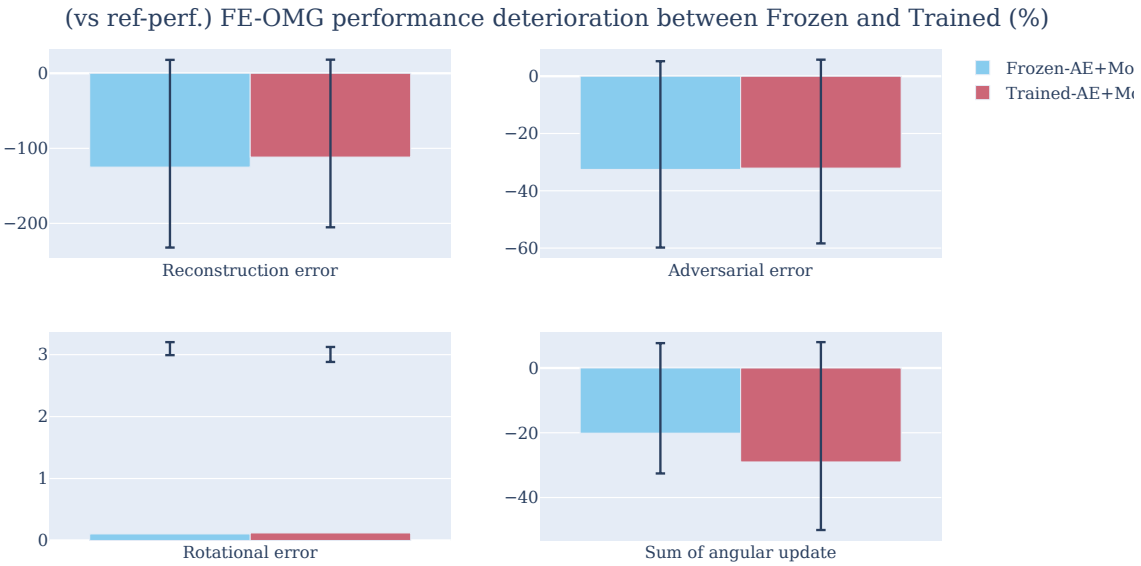
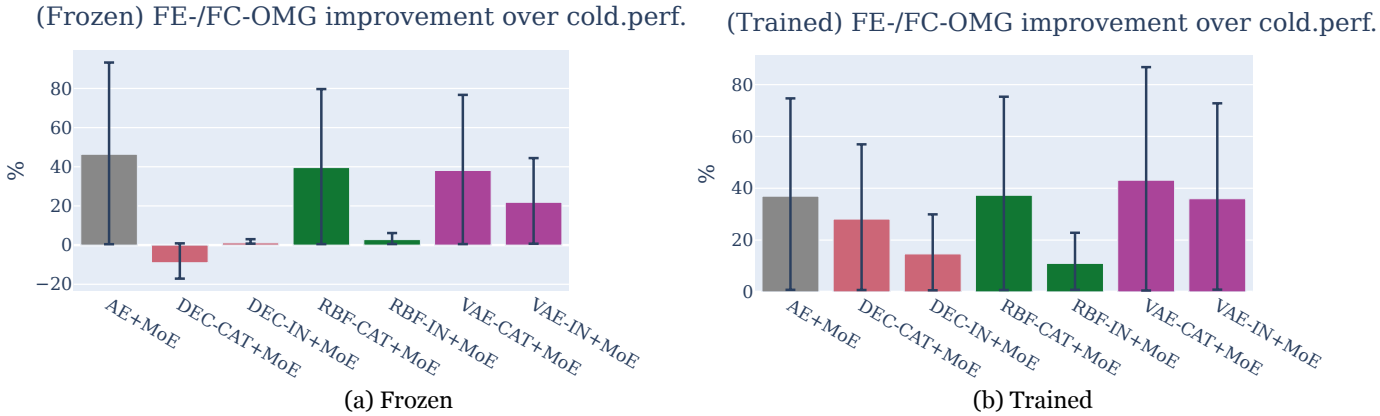


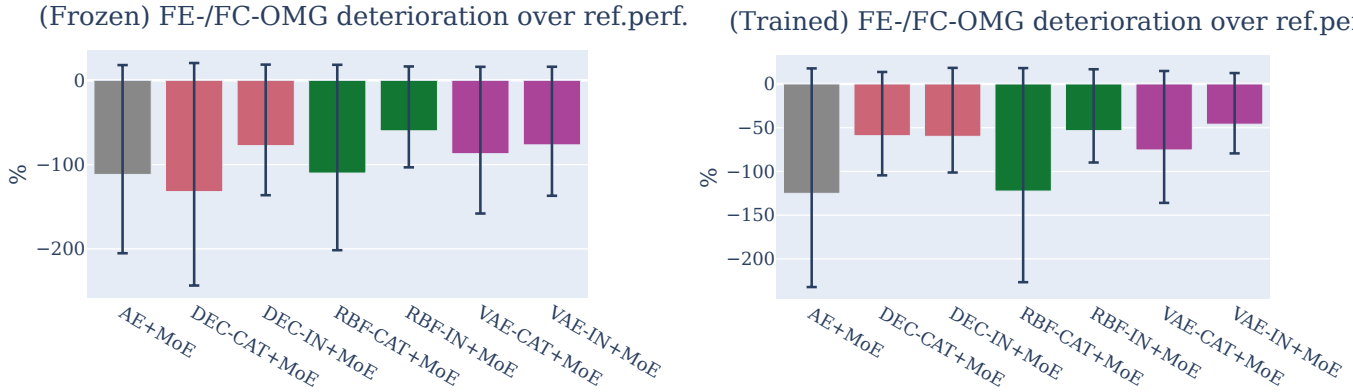
Figure 5.8: The transferring quality of AE+MoE in relation to the reference-performance. The reconstruction error of the model is approximately 2 times larger the reference-performance. Allowing the entire model to be trained does not significantly improve the performance.

in the reconstruction errors, they are still about 2 times larger than the reference-performance for RBF-CAT-MoE and VAE-CAT-MoE. It is the same behaviour as observed with AE+MoE. The plots of the other metrics can be found in Appendix C. In the both figures, by adopting the ”trained” strategy, the performance of the DEC-MoE models is greatly improved, but only marginal improvements for the other models.



(a) Frozen (b) Trained

Figure 5.9: The reconstruction error improvement over cold-performance of the FC-OMG models. The models are warm-started with the parameters from the pre-trained instances on R1 and trained with 20% of the training data with the ”frozen” strategy. The values are averaged over the rigs R2-R5. The confidence interval at the 95% significance level is plotted.



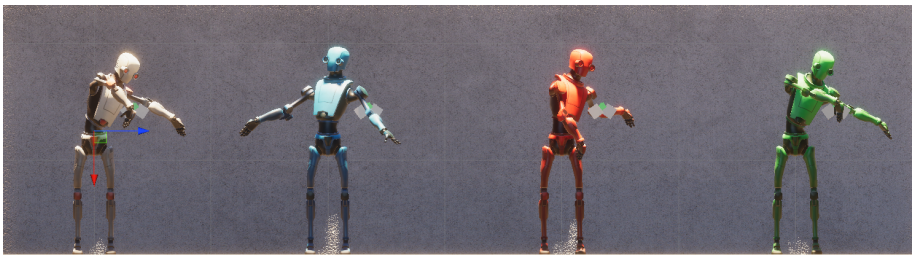
(a) Frozen (b) Trained

Figure 5.10: The reconstruction error deterioration over the reference-performance of the FC-OMG models.

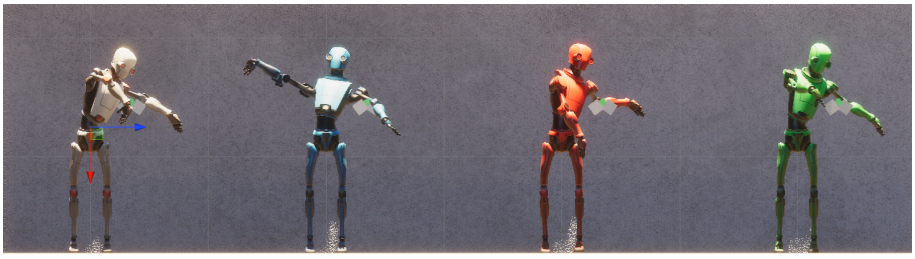
The sample frames from the generated animation clip using FE-OMG and FC-CAT-OMG models on R3 are visualised in Figure 5.11. The white character shows the target pose; the blue variant shows the pose from the cold-started model; the red one shows the pose from the warm-started model with the ”frozen” strategy, and the green one shows the pose from the transferred model with the ”trained” strategy.

From the observation of the generated clips, all the models, when cold-started, fail to produce meaningful animations. Despite that, the generated poses are semantically correct (not falling apart), the models can only produce generic responses. Warm-starting the model does improve the overall accuracy of the generated motions, but not as precise and smooth as the reference models. Allowing all the modules to be trained, improves the overall precision slightly. AE, RBF and VAE variants manage to generate roughly accurate motions, where the character actually reaches the hands towards the targets, though not completely reaching them. Whereas the DEC variant failed to produce accurate animation. Similar to the behaviour of its reference counterpart, the VAE variant produces visible stuttering animations when compared to the RBF and the DEC variants. AE+MoE and RBF-CAT+MoE are the best performing models that are able to generate somewhat correct and smooth animations.

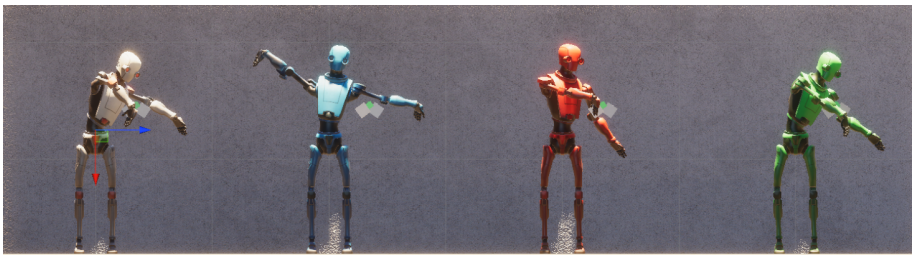
Figure 5.12 shows the transferring quality of all models in terms of reconstruction error, including the LSTM variants. The two outer plot lines show the improvement (%) over the cold-performance for each of the models and each of the training strategy. The two inner plot-lines show the performance in relation to the reference-performance. It is particularly interesting because AE+MoE and the FC-CAT-OMG models that are the best performing models have the most performance gains over the cold-performance,



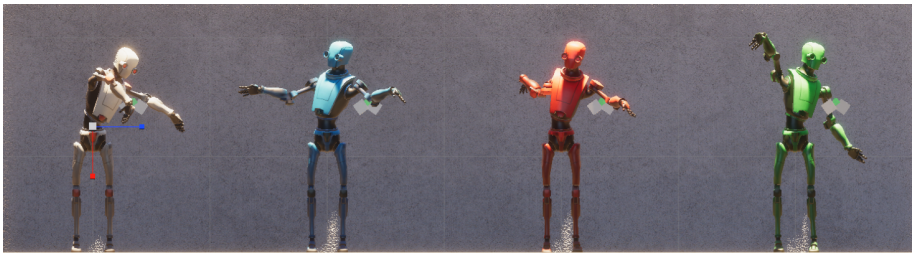
(a) AE+MoE at frame 35



(b) RBF-CAT+MoE at frame 35



(c) VAE-CAT+MoE at frame 35



(d) DEC-CAT+MoE at frame 35

Figure 5.11: The generated poses from RBF-CAT+MoE, VAE-CAT+MoE and DEC-CAT+MoE, compared against the target pose (white). The three variants are the cold-started model (blue), the warm-started model with the ”frozen” strategy (red) and the warm-started model with the ”trained” strategy (green). It shows that both the RBF and the VAE variants manage to generate more accurate animations than the DEC variant.

which is also true for LSTM variants. However, they are not reaching anywhere near their reference-performance as much as the other less performing models such as the FC-IN-OMG models. Those models, especially the LSTM variants demonstrate decent performance gains over their cold-performance, and the performance is also comparable to their reference performance. Allowing all the modules of models to be trained does not yield any drastic improvements for all of the models.

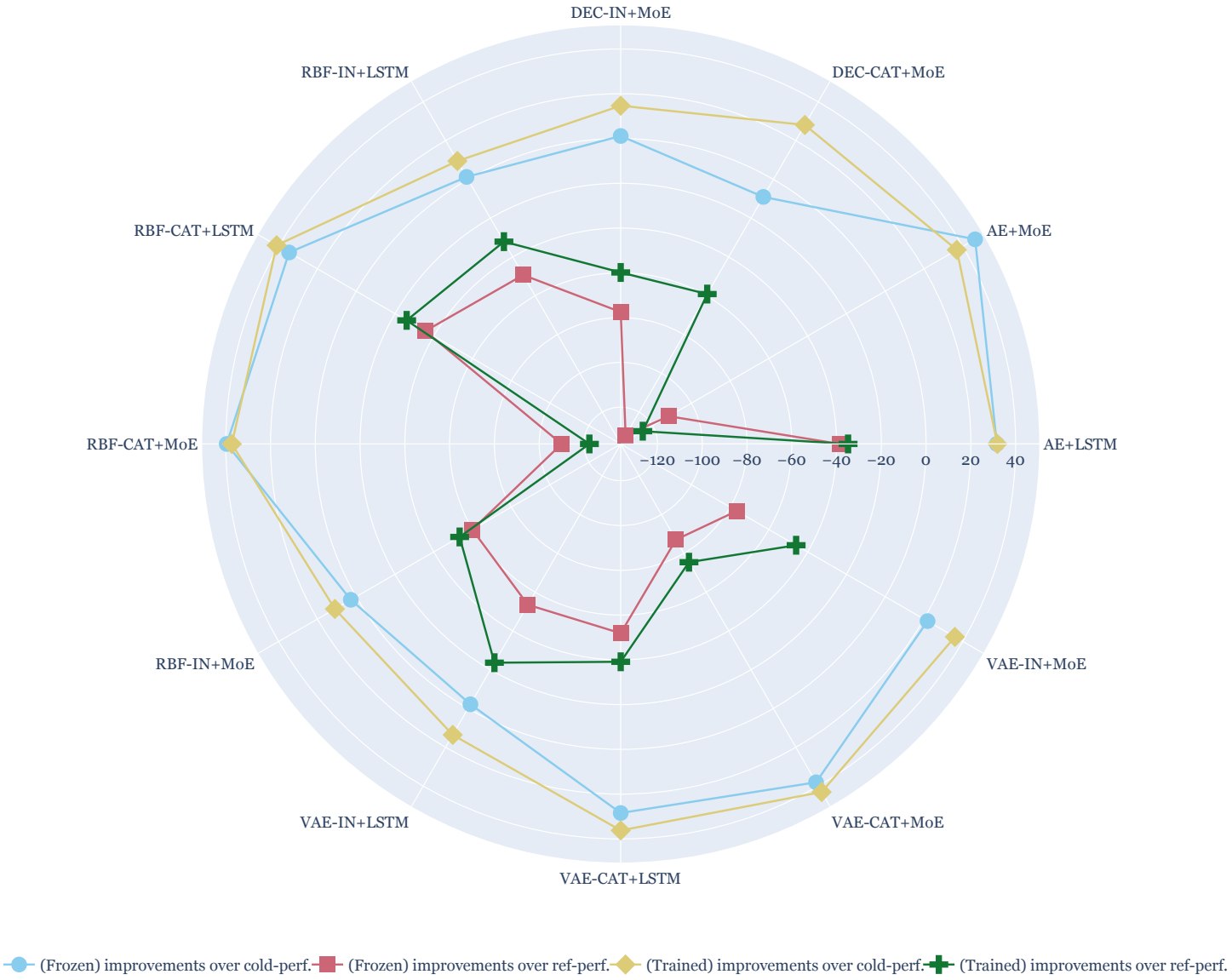


Figure 5.12: The reconstruction error in relation to the cold- and the reference-performance of FE-OMG and FC-OMG models. The values are averaged over the domains. It illustrates that the best performing models have the most performance gains but do not reach their reference-performance as much as the other models do.

5.3.3 FS-OMG

This section presents the transferring quality of using the FS-OMG models on the reduced feature set of R2, R3, R4 and R5. The Figures 5.13 and 5.14 illustrates the transferring quality in relation to the reference-performance using FS-OMG and FS-CAT-OMG models. It shows that AE+MoE does not gain much from allowing the MoGen module to be trained, whereas, for the FS-CAT-OMG models, the reconstruction error

is almost halved from ”frozen” to ”trained”. The gain is also visible and drastic for other metrics.

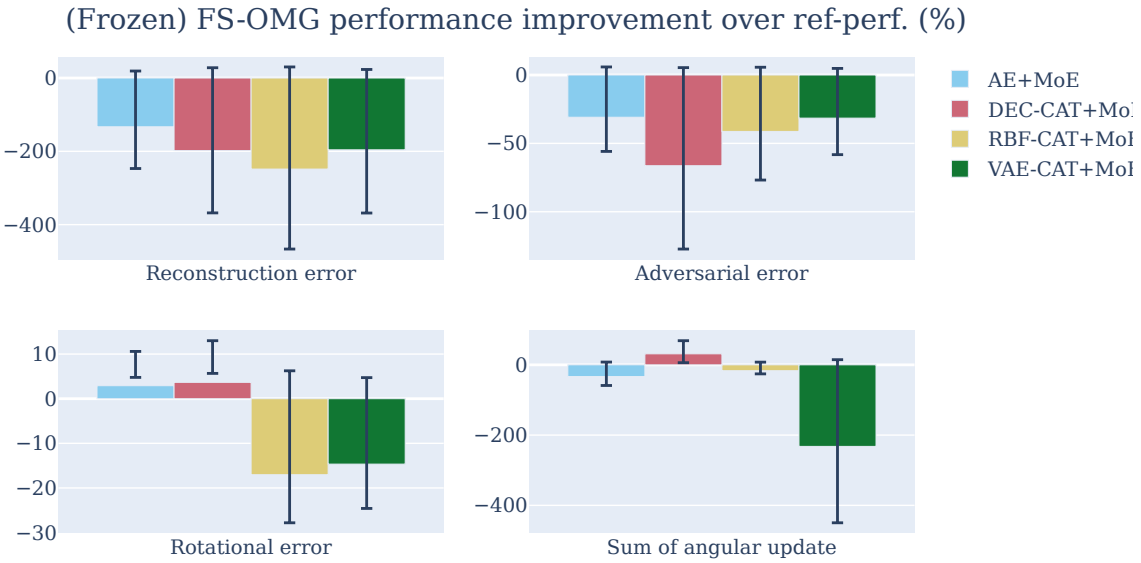


Figure 5.13: The transferring quality of FS-OMG and FS-CAT-OMG models with the ”frozen” training strategy, in relation to the reference performance.

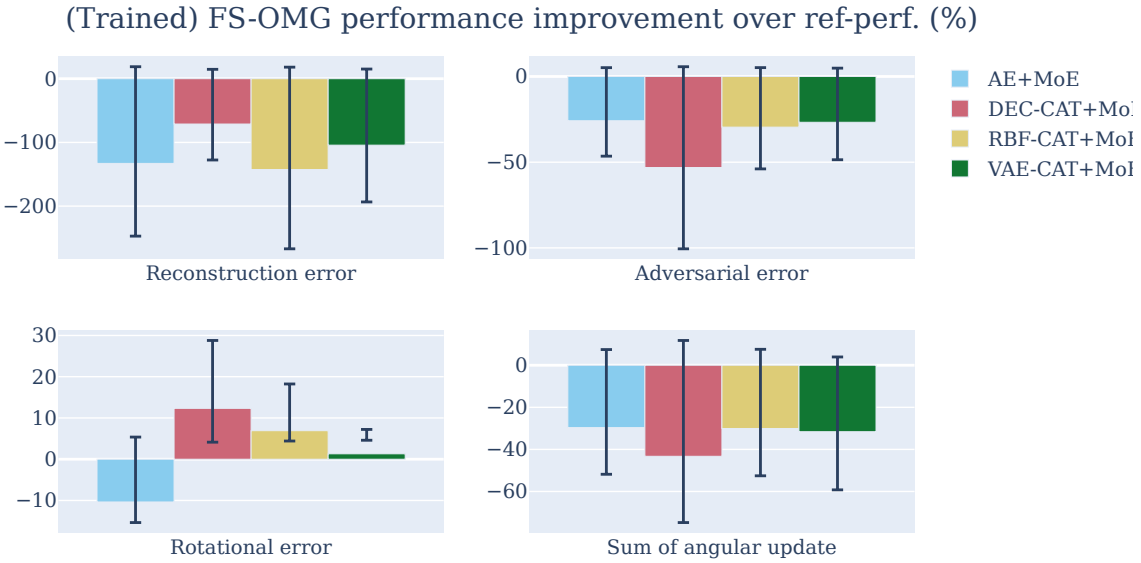


Figure 5.14: The transferring quality of FS-OMG and FS-CAT-OMG models with the ”frozen” training strategy, in relation to the reference performance. Comparing to 5.13, it shows that allowing the entire model to be trained does improve the performance of all models significantly.

Figure 5.15 visualises the sample frame from the generated clip using FS-OMG and FS-CAT-OMG models. The input pose vectors to the model only contain information about 6 joints (feet, hands, head and spine) and the model outputs the upsampled full-resolution pose vectors. The poses appear semantically correct but only RBF-CAT-MoE is able to

produce more accurate poses than the other three variants. The overall quality of the generated animations is not at the same level as the FC-CAT-OMG models that operate on the full resolution pose data, but it is better than expected as the models are also forced to learn the upsampling.

5.3.4 Statistical significance of the improvements

This section presents the p-values computed from T-test and Mann-Whitney U test, comparing the reconstruction error populations on the test set, from the cold-started model and the warm-started model with the ”frozen” strategy. Table 5.2 lists the rounded p-values of the relevant models. The null hypothesis H_0 [3.6.1] is rejected if and only if the p-values from both tests are less than 0.05, indicating a significance level at 95%.

Name	T-test	Mann-Whitney U test	Reject
AE+MoE	0.00	0.00	True
AE+LSTM	0.00	0.00	True
DEC-CAT+MoE	0.00	0.00	True
DEC-IN+MoE	0.11	0.02	**False**
RBF-CAT+MoE	0.00	0.00	True
RBF-IN+MoE	0.36	0.14	**False**
VAE-CAT+MoE	0.00	0.00	True
VAE-IN+MoE	0.00	0.00	True

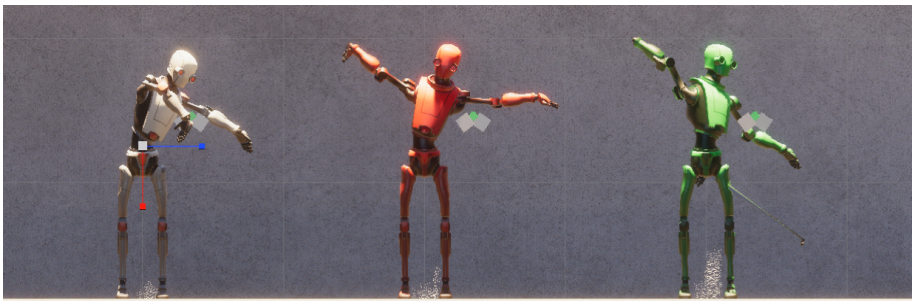
Table 5.2: The table displays the p-values computed using T-test and Mann-Whitney U test on two independent test errors populations (cold- and warm-performance) of the FE-/FC-OMG models. It shows that the null hypothesis is considered to be rejected all the models except DEC-IN-MoE and RBF-IN+MoE.

5.4 Summary

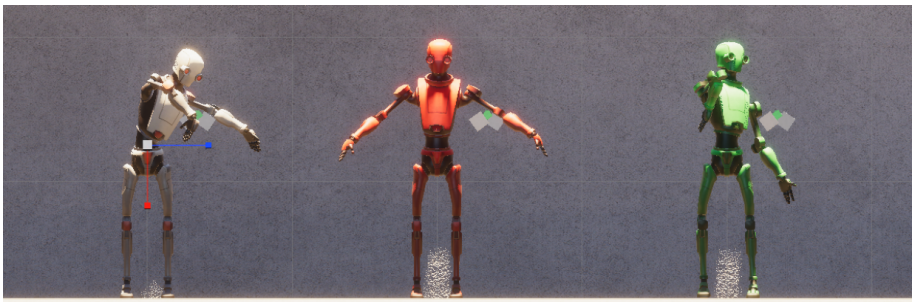
This section presents the final comparison of the best performing models for the three RAE approaches. Figure 5.16 shows the comparison of the generation performance of AE+MoE, RBF-CAT+MoE and VAE-CAT+MoE in the three scenarios: reference performance where the models are fully trained with all the training data for 50 epochs; FE/FC is the scenario where the models are warm-started with the trained parameters from the reference instances on R1, and the models are trained with only 20% training data for 30 epochs; FS is the performance of the warm-started models that operate on the reduced feature set. It shows that all the models achieve comparable performance in all three scenarios. where the transferred reconstruction error is about 2 times larger than the reference values, for all models. The values are averaged over R2, R3, R4 and R5. The confidence interval at 95% confidence level is also displayed.

(a) AE+MoE at frame 35

(b) RBF-CAT-MoE at frame 35



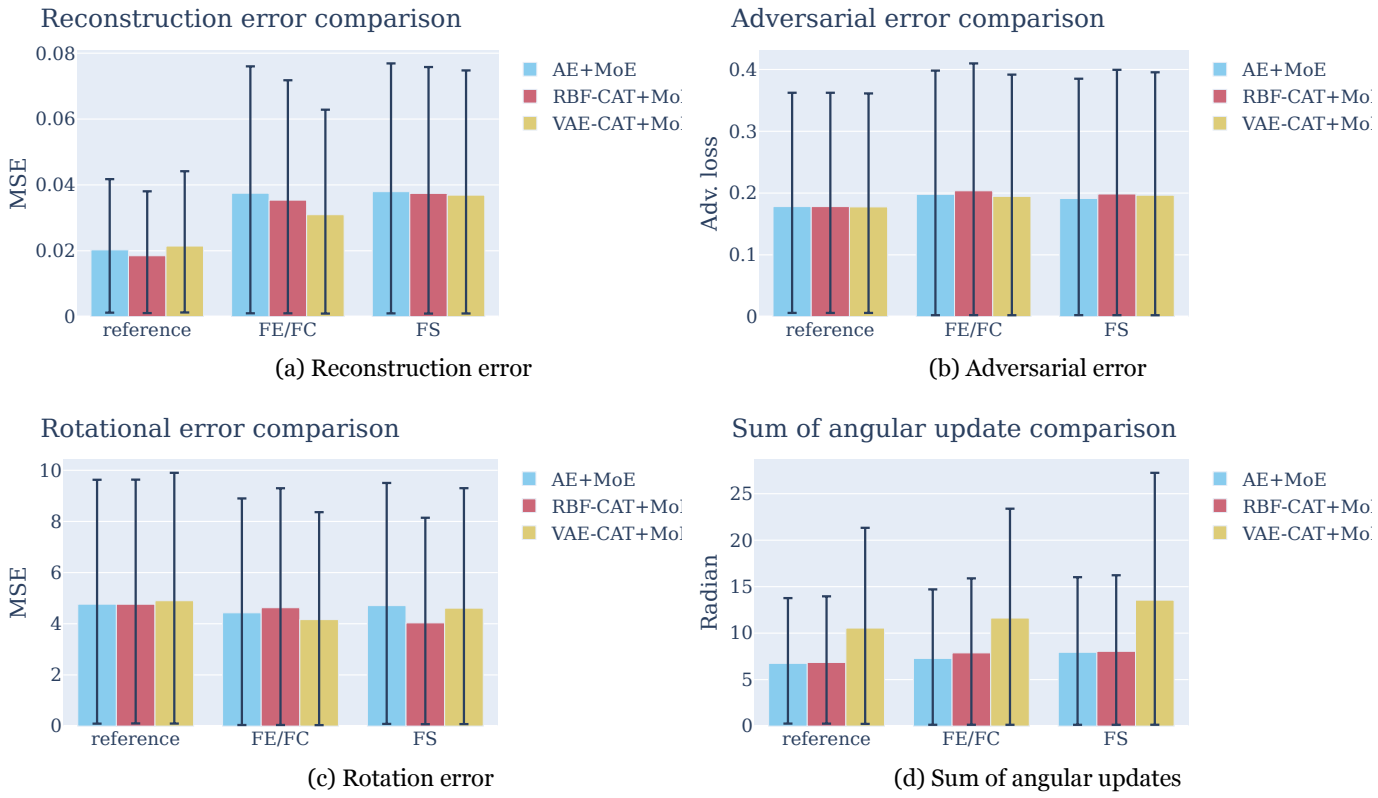
(c) VAE-CAT-MoE at frame 35



(d) DEC-CAT-MoE at frame 35

Figure 5.15: The sample frame from the generated clip using AE+MoE, FS-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 is able to produce more accurate poses.

Figure 5.17 displays the reconstruction error per rig of the warm-started models using the three RAE approaches. The performance is consistent on all the rigs for each of the RAE approach.



(a) Reconstruction error

(b) Adversarial error

(c) Rotation error

(d) Sum of angular updates

Figure 5.16: The performance comparison of the three best performing models in the three scenarios: reference, warm-start with limited training and data, warm-start with reduced feature set

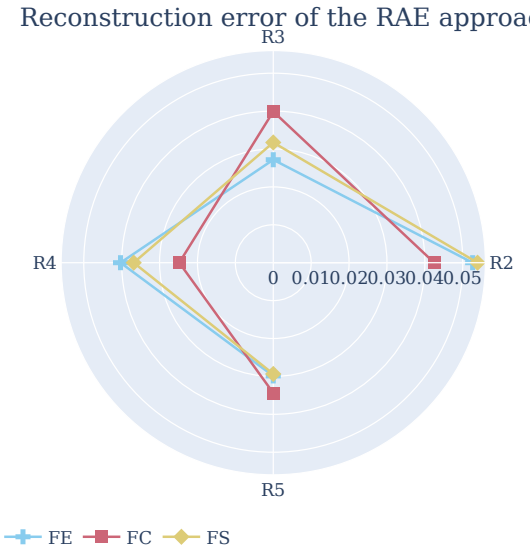


Figure 5.17: The reconstruction error of the respective best performing model for each RAE method per domain. The models are (FE) AE+MoE, (FC) RBF-CAT+MoE, (FS) AE+MoE, with the ”trained” strategy.

6 Discussion

This section discusses the transferability and the limitations of the obtained results. In Sections 6.1 and 6.2 discuss the results in relation to the research questions defined in 1.1.

6.1 Generation performance of OMG

All the implemented OMG models are capable of learning from the datasets and generate correct animations with good quality, as presented in Section 5.2. FE-OMG and FC-CAT-OMG models perform better than FC-IN-OMG models, where MoE is the better Motion Generation network (MoGen) than LSTM. AE+MoE and RBF-CAT+MoE perform best and can produce near-identical animations when compared to the target animations. Although VAE-CAT-MoE achieves similar numerical results, the generated animations contain visible jittering, which might be caused by the random sampling mechanism in the VAE layer. For the other models, the generated animations are less accurate and smooth.

Even when operating on the reduced feature set (only 6 joints), AE+MoE and RBF-CAT+MoE manage to produce good quality animations with only slight visual artefacts that can be eliminated with post-processing. This is particularly interesting because not only does it provide a simpler way to achieve rig-agnostic encoding (RAE), but also indicates the possibilities of an MoE-based motion generation model. Moreover, it also reduces the memory bandwidth required for running the network, because the size of the input pose data is reduced.

6.1.1 (RQ.1) Which of MoE and LSTM is better suited for OMG?

Regarding the research question RQ.1, it is clear that MoE is better suited for OMG than LSTM in terms of the generation performance and the model complexity, as LSTM required about three times more memory than the MoE counterpart, indicated in Table 5.1. Moreover, the LSTM model fails to produce accurate and smooth animations.

6.2 Transferring quality of the RAE approaches

The results in Section 5.3 demonstrate the transferring quality of the three RAE approaches, with both training strategies. The transferring quality is quantified by the performance improvement over the performance from the cold-started models and the performance deterioration over the reference-performance.

The results show that all the models, when cold-started, fail to generalise over the test set and are only able to produce generic poses. Warm-starting the models with the parameters of the pre-trained instances for another rig, does immediately improve the overall performance of all the models, as demonstrated in Figure 5.7 with the ”frozen” strategy, where only the autoencoder is trained with the limited motion data from the respective domain. AE+MoE and RBF-CAT+MoE are the best performing models in terms of all four metrics, and they are capable of generating accurate animations with limited training and data, albeit not completely correct and not as smooth as their reference counterparts.

Moreover, the models that have worse reference-performance, when transferred to another domain, can still achieve comparable performance. Whereas the models that have better reference-performance cannot get equally close to the same level, as presented in Section 5.3.2. However, these models gain significantly more improvement over their cold-performance.

Finally, allowing all the modules to be trained (the ”trained” strategy) does improve the overall performance for all models, but only slightly. This might be caused by the limited data is not enough or the catastrophic forgetting phenomenon [MC89], where the models get overfitted on the training data and forget the previously learnt knowledge, which reduces the generalisation performance.

6.2.1 (RQ.2) Which of RBF, VAE and DEC is better suited for OMG?

Regarding the research question RQ.2, both the generated animation clips and the numerical results indicate that FC-CAT-OMG models achieve better performance than FC-IN-OMG models. It is not surprising, because the MoGen module in the former models operates on a larger input pose space (256+128) than the latter models (128). The motivation for FC-IN-OMG is that it ensures that the input vectors to the MoGen module are always from the same subspace, feature-wise for all the rigs.

The hypothesis is that it would facilitate the transferring to other domains, which is confirmed in Figure 5.12, where FC-IN-OMG models demonstrate less deterioration over their reference-performance when transferred, compared to the FC-CAT-OMG counterparts. Comparing RBF, VAE and DEC in Section 5.3.2, it is clear that RBF and VAE are better than DEC in terms of all metrics and the quality of generated animations. The reason could be that DEC outputs a vector of probability, which sums to 1, which means

for an output vector of dimension 128, the values in the vectors become fairly small and might not be uniformly distributed.

VAE-CAT-MoE performs slightly worse than the RBF variant in both the quality of the generated animations and the numerical results. The randomly sampling mechanism in the VAE layer also introduces a visible amount of stuttering and it is also indicated by the higher sum of angular updates in Figure 5.16d, which the RBF variant does not have. Furthermore, the VAE variant also has slightly more parameters than the RBF variant. Therefore, RBF is the better alternative for FC-OMG in terms of generation performance, transferring quality and model complexity.

The hypothesis H_1 is accepted and confirmed by the statistical evaluation presented in Section 5.3.4, as the null hypothesis is rejected for the most relevant models.

6.2.2 Which transfer learning technique achieves better transferring quality for OMG?

Comparing AE+MoE, RBF-CAT+MoE and VAE-CAT+MoE in Figure 5.16, it is difficult to tell which model is better than the other two, as they all perform similarly in terms of the four metrics. VAE-CAT+MoE achieves a slightly lower reconstruction error than the other two models, but it is less stable and smooth. RBF-CAT+MoE is performing slightly better than AE+MoE but it has about 24% more parameters than the latter. Although that the three RAE approaches FE/FC and FS produce similar results in terms of the four metrics, the quality of the generated animations are not similar at all. Only AE+MoE and RBF-CAT+MoE are capable of generating roughly accurate animations, with limited training and data on new domains. It implies that the numerical metrics do not capture the true performance of the models.

However, it is safe to state that feature encoding (AE+MoE) is the better rig-agnostic encoding approach, as it provides excellent generation performance when fully trained, and good transferring quality. It is also less complex than the feature clustering (RBF-CAT+MoE) approach.

6.3 Transferability

In theory, the results of this research do apply to the other motion synthesis frameworks as MANN, NSM, LMP-MoE, MVAE and TR-LSTM [Zha+18; Sta+19; Sta+20; Lin+20; Har+20]. Model-wise, the OMG shares the same internal generation process as the MoE-based frameworks (MANN, NSM, LMP-MoE, MVAE), where the parameters of the main

network are generated by a gating network that is conditioned on some covariate. The main network is conditioned on the pose data, some condition features and some control signals, and outputs the pose and the updates to the features for the next frame. The gating network in OMG is conditioned on a local motion phase variable that is a simplified variant of the one described in LMP-MoE [Sta+20]. It should not affect the transferring quality of the RAE approaches, as they only concern the pose representation to the network.

Data-wise, the pose data is formatted in the same way as in LMP-MoE, the only fundamental differences between the pose data in this research to the pose data used in LMP-MoE are the rigs and the visual quality of the motion data. Since their data is motion-captured animations, and the motion data in this research is procedurally generated using an IK-solver. Furthermore, from the perspective of the model, the pose is only a collection of individual data points (joints) and the models do not consider the kinematic linkages between the joints.

6.4 Limitations

The four evaluation metrics used to quantify the generation performance are not good enough, as they fail to capture the true performance of the models. Since a lower reconstruction error does not necessarily correspond to better animation quality. The adversarial discriminator is only trained with the autoencoder, but not with the OMG, thus the adversarial loss might not reflect on the true quality of generated poses. The rotational error is computed as the mean squared error between the two orientations, which might not be good as it only performs a value-wise comparison and the orientation vectors, that are not guaranteed to be orthonormal. It would have been better if the angular difference is used instead. Furthermore, the reconstruction errors are not normalised, meaning they cannot be directly compared between the rigs, because the rigs have different sizes.

Normalised-power-spectrum similarity (NPSS) might be a good metric candidate, as it compares the animation sequences in their power spectrum that is time-invariant [Gop+18].

Due to the time constraint, the training is heavily limited when performing the experiments. Each model is trained for only about $\frac{1}{3}$ number of epochs of what was initially planned. For the same reason, the models are not tuned for the feature selection approach. Despite having only 6 joints in the pose representation, the same configurations of the models as the other RAE approaches are used,

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 research, 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 use multiple frames as input in the framework, it would improve the overall performance of the models.

6.5 Future Work

Although the results should apply to the more advanced motion synthesis frameworks, as discussed in Section 6.3, it is definitely interesting to test the RAE approaches on those frameworks with motion-captured animation data. Moreover, further tuning and experimenting of the feature clustering and the feature selection approaches are attractive, because they can achieve feature-wise rig-agnostic encoding. For feature clustering, it might be helpful to initialise the cluster centres with the key poses that are chosen with the help of expert knowledge. For feature selection, further research in the upsampling process is desired, as this approach reduces the required bandwidth of running the framework which is especially beneficial for real-time applications such as video games.

For the broader application and the generalisation, it is necessary to experiment with the transferring across different motion types (tasks). In this research, all the motion data for all the rigs are procedurally generated in the same way, with small motion-wise differences, which might be easy for the transferred models to reconstruct. Also, the differences between the tested rigs are not particularly large, hence it would be interesting to experiment with rigs that are much more complex, more varied in sizes and different types.

6.6 Conclusion

In this research, three rig-agnostic encoding (RAE) approaches are presented: feature encoding (FE-OMG), feature clustering (FC-OMG) and feature selection (FS-OMG), for knowledge transferring of the proposed Objective-driven motion generation model (OMG). OMG 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 much the RAE approaches improve the learning of OMG on new domains (character rigs) with limited training and data, by warm-starting the model with the parameters of the pre-trained instance on a previous domain. The

results show that both FE-OMG and FC-OMG perform better than FS-OMG, in terms of the generation performance and the transferring quality, with FC-OMG pulling slightly ahead. All the transferred models demonstrate huge improvements over the vanilla instances.

In conclusion, all three RAE approaches can be used for transfer learning of motion synthesis frameworks, where the FE-OMG model is the better performing variant with respect to the model complexity.

References

[Bis09] Bishop, Christopher M. *Pattern recognition and Machine learning*. Springer Science, 2209.

[BKG20] Bank, Dor, Koenigstein, Noam, and Giryes, Raja. "Autoencoders". In: *arXiv* (2020). eprint: 2003.05991.

[Cha16] 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.

[Che] Chen, Nuo. *Deep-learning-based-rig-agnostic-encoding*. URL: https://github.com/Neroro64/Deep-learning-based-rig-agnostic-encoding. (accessed: 2021-06-09).

[Dai+08] Dai, Wenyuan et al. "Self-taught clustering". In: *Proceedings of the 25th international conference on Machine learning - ICML '08* (2008), pp. 200–207. DOI: 10.1145/1390156.1390182.

[Fra+15] Fragkiadaki, Katerina et al. "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.

[Gop+18] Gopalakrishnan, Anand et al. "A Neural Temporal Model for Human Motion Prediction". In: *arXiv* (2018). eprint: 1809.03036.

[HA] Hyndman, Rob J and Athanasopoulos, George. *Forecasting: Principles and Practice*. URL: https://otexts.com/fpp2/AR.html. (accessed: 2021-06-09).

[Har+20] Harvey, Félix G. et al. "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.

[HKS17] 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.

[Hod+95] Hodgins, Jessica K. et al. "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. URL: https://doi.org/10.1145/218380.218414.

[Hol+20] Holden, Daniel et al. “Learned motion matching”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), 53:1–53:12. ISSN: 0730-0301. DOI: 10.1145/3386569.3392440.

[HS97] 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.

[KSH17] 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.

[Lin+20] Ling, Hung Yu et al. “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.

[Mao+16] Mao, Xudong et al. “Least Squares Generative Adversarial Networks”. In: *arXiv* (2016). eprint: 1611.04076.

[MC12] Min, Jianyuan and Chai, Jinxiang. “Motion graphs++”. In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), pp. 1–12. ISSN: 0730-0301. DOI: 10.1145/2366145.2366172.

[MC89] 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.

[Par12a] Parent, Rick. “Computer Animation (Third Edition)”. In: (2012), pp. 111–160. DOI: 10.1016/b978-0-12-415842-9.00004-6.

[Par12b] Parent, Rick. “Computer Animation (Third Edition)”. In: (2012), pp. 161–185. DOI: 10.1016/b978-0-12-415842-9.00005-8.

[Pen+18] Peng, Xue Bin et al. “DeepMimic”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3197517.3201311. eprint: 1804.02717.

[SHZ19] 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.

[Sta+19] Starke, Sebastian et al. “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.

[Sta+20] Starke, Sebastian et al. “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.

[THR07] 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. URL: https://proceedings.neurips.cc/paper/2006/file/1091660f3dff84fd648efe31391c5524-Paper.pdf.

[Vil+18] Villegas, Ruben et al. “Neural Kinematic Networks for Unsupervised Motion Retargetting”. In: *arXiv* (2018). eprint: 1804.05653.

[XGF15] Xie, Junyuan, Girshick, Ross, and Farhadi, Ali. “Unsupervised Deep Embedding for Clustering Analysis”. In: *arXiv* (2015). eprint: 1511.06335.

[Zha+18] Zhang, He et al. “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.

[Zhu+19] Zhuang, Fuzhen et al. “A Comprehensive Survey on Transfer Learning”. In: *arXiv* (2019). eprint: 1911.02685.

Appendix - Contents

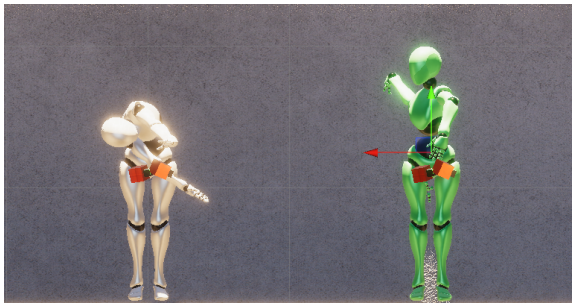
A The complete list of model properties

Name	Parameters (M)	Memory (MB)	Inference time (ms)
AE	0.5	1.82	10.8
AE+MoE	2.9	11.78	232.84
AE+LSTM	8.8	35.22	194.33
RBF-CAT+LSTM	9.1	36.4	204.53
RBF-IN+LSTM	8.6	34.3	194.92
RBF-CAT+MoE	3.7	14.6	287.36
RBF-IN+MoE	2.7	10.87	209.69
VAE-CAT+LSTM	9.1	36.53	194.24
VAE-IN+LSTM	8.6	34.43	185.09
VAE-CAT+MoE	3.7	14.73	280.83
VAE-IN+MoE	2.7	11	202.8
DEC-CAT+MoE	3.7	14.6	288.8
DEC-IN+MoE	2.7	10.87	213.85

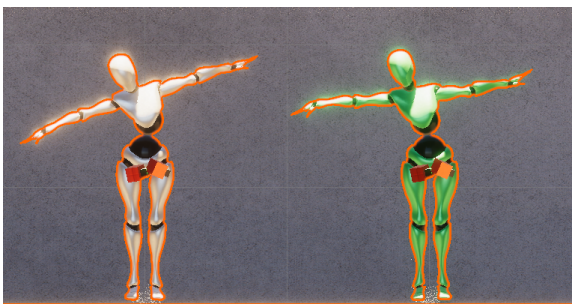
Table A.1: Properties of the implemented models for R1.

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

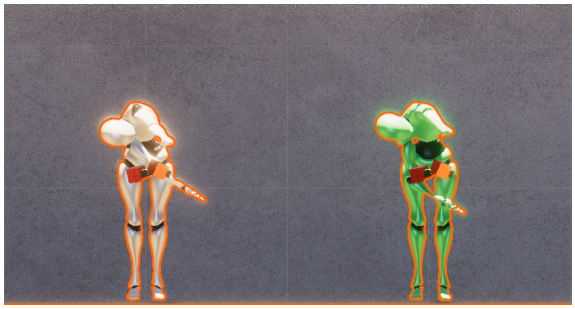
(a) RBF-IN+MoE at frame 22



(b) RBF-IN+MoE at frame 35

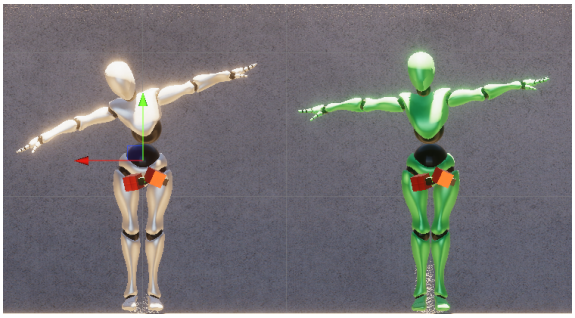


(c) RBF-CAT+MoE at frame 22



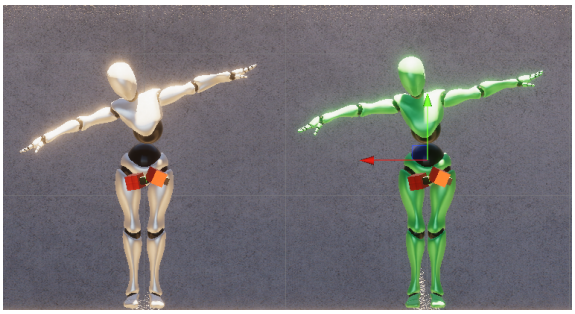
(d) RBF-CAT+MoE at frame 35

Figure B.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).

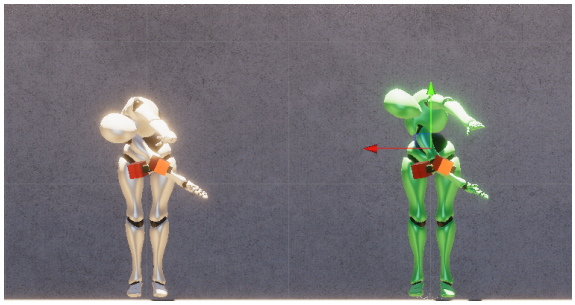


(a) VAE-IN+MoE at frame 22

(b) VAE-IN+MoE at frame 35

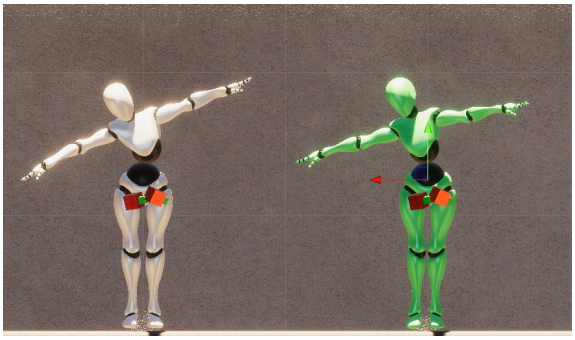


(c) VAE-CAT+MoE at frame 22

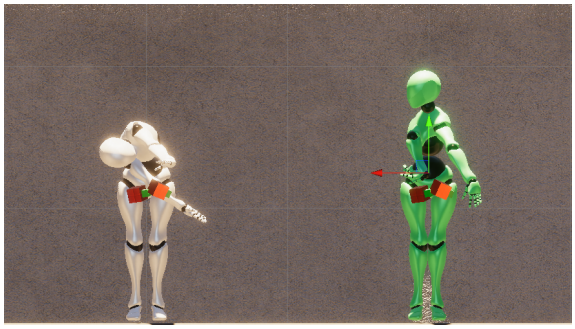


(d) VAE-CAT+MoE at frame 35

Figure B.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).

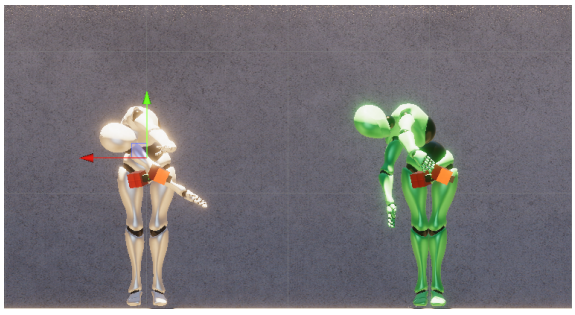


(a) DEC-IN+MoE at frame 22



(b) DEC-IN+MoE at frame 35

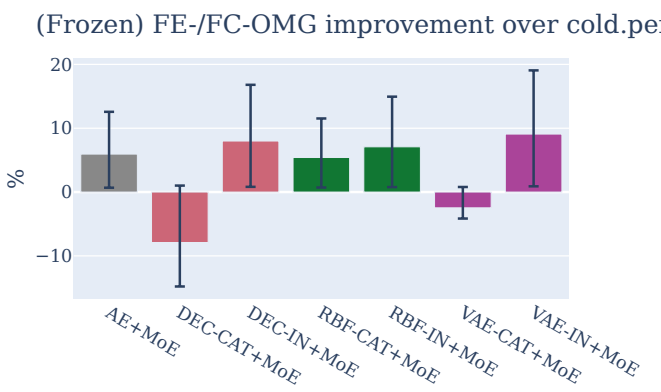
(c) DEC-CAT+MoE at frame 22



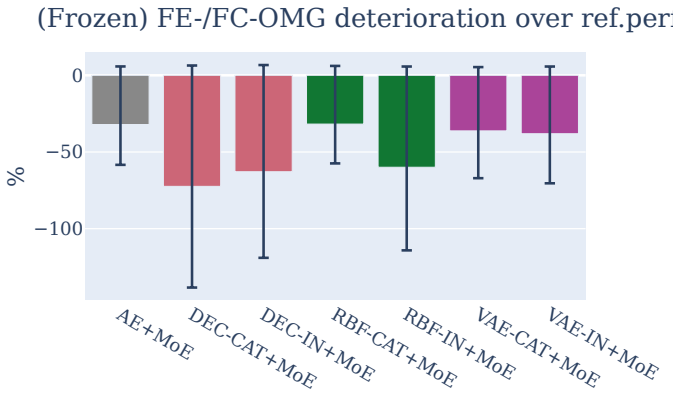
(d) DEC-CAT+MoE at frame 35

Figure B.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).

C More plots of the transferring quality of the reference FC-OMG models

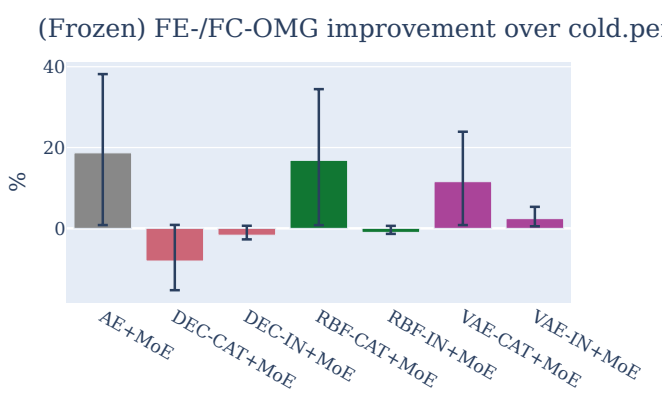


(a) Improvement over cold-performance

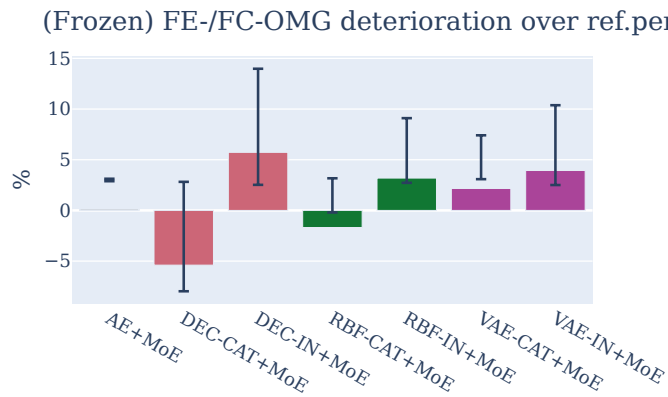


(b) Deterioration over reference-performance

Figure C.1: The adversarial error improvement over cold-start performance and the deterioration over the reference performance of the respective FC-OMG model (”frozen”).

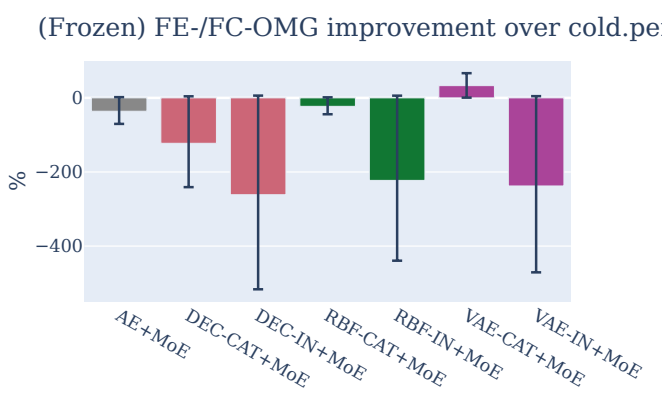


(a) Improvement over cold-performance

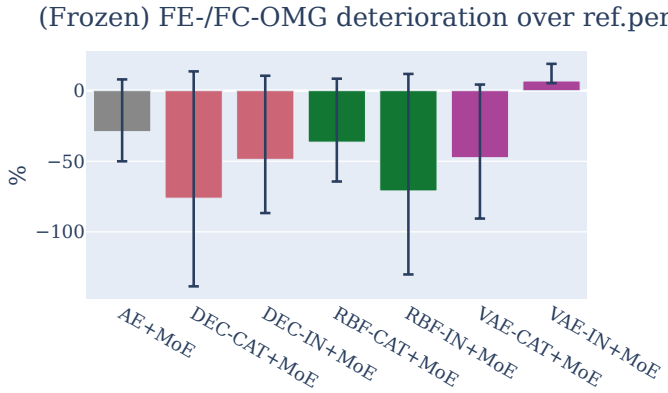


(b) Deterioration over reference-performance

Figure C.2: The rotational error improvement over cold-start performance and the deterioration over the reference performance of the respective FC-OMG model (”frozen”).

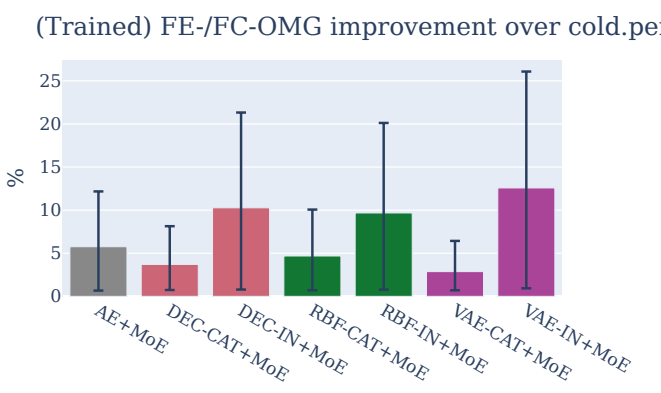


(a) Improvement over cold-performance

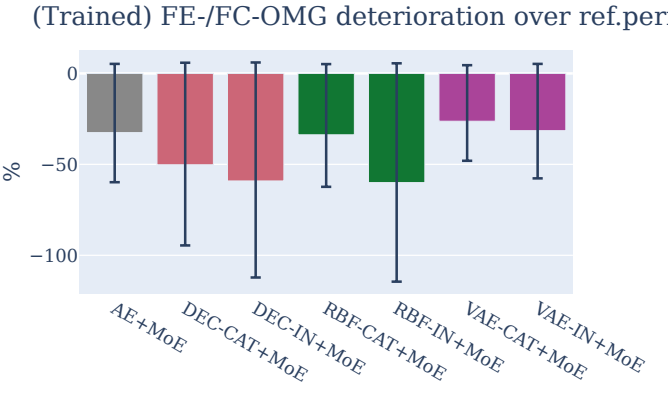


(b) Deterioration over reference-performance

Figure C.3: The sum of angular update improvement over cold-start performance and the deterioration over the reference performance of the respective FC-OMG model (”frozen”).

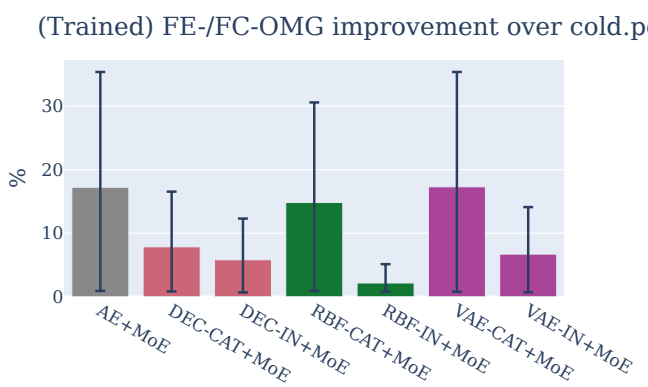


(a) Improvement over cold-performance

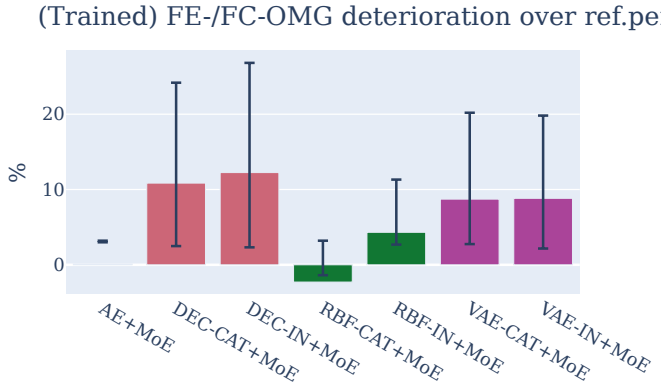


(b) Deterioration over reference-performance

Figure C.4: The adversarial error improvement over cold-start performance and the deterioration over the reference performance of the respective FC-OMG model (”trained”).

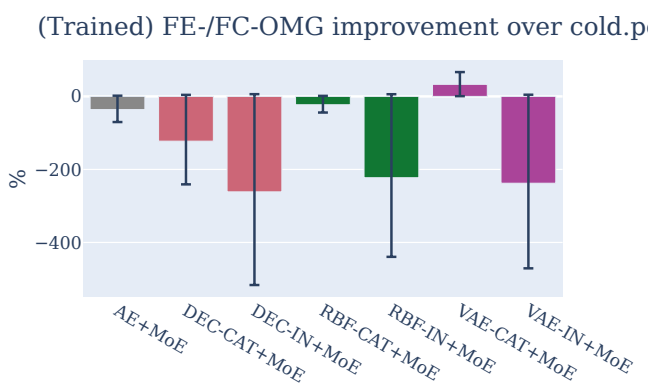


(a) Improvement over cold-performance

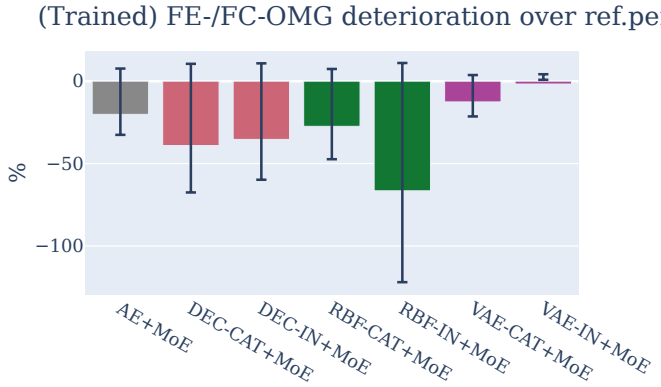


(b) Deterioration over reference-performance

Figure C.5: The rotational error improvement over cold-start performance and the deterioration over the reference performance of the respective FC-OMG model ("trained").



(a) Improvement over cold-performance



(b) Deterioration over reference-performance

Figure C.6: The sum of angular update improvement over cold-start performance and the deterioration over the reference performance of the respective FC-OMG model ("trained").

D

The complete list of model performance data

D.1 Reference performance results

Name	Reconstruction error (MSE)	Adversarial error (Adv. Loss)	Rotational error (MSE)	Sum of angular update (radian / s)
AE_R2	0.229	22.059	6.595	6.012
AE_R3	0.36	42.061	6.311	5.029
AE_R4	0.312	23.256	6.628	5.251
AE_R5	0.301	21.394	5.48	4.911
AE+LSTM_R1	0.046	0.028	11.139	13.12
AE+LSTM_R2	0.049	0.158	4.331	19.588

AE+LSTM_R3	0.029	0.404	8.811	21.192
AE+LSTM_R4	0.062	0.153	3.198	13.745
AE+LSTM_R5	0.031	0.17	2.671	10.557
AE+MoE_R1	0.011	0.068	10.534	5.541
AE+MoE_R2	0.024	0.148	5.381	8.446
AE+MoE_R3	0.01	0.571	5.826	9.161
AE+MoE_R4	0.012	0.137	2.202	4.972
AE+MoE_R5	0.012	0.164	2.049	5.634
DEC-CAT+MoE_R1	0.029	0.054	10.16	5.945
DEC-CAT+MoE_R2	0.038	0.155	4.67	9.513
DEC-CAT+MoE_R3	0.014	0.501	6.453	10.329
DEC-CAT+MoE_R4	0.024	0.141	3.294	5.864
DEC-CAT+MoE_R5	0.032	0.164	2.294	5.303
DEC-IN+MoE_R1	0.052	0.04	9.457	6.265
DEC-IN+MoE_R2	0.06	0.159	5.905	14.889
DEC-IN+MoE_R3	0.029	0.32	8.217	16.434
DEC-IN+MoE_R4	0.084	0.151	4.715	7.266
DEC-IN+MoE_R5	0.025	0.166	2.084	7.57
RBF-CAT+LSTM_R1	0.047	0.031	9.091	12.172
RBF-CAT+LSTM_R2	0.049	0.157	4.477	16.304
RBF-CAT+LSTM_R3	0.025	0.399	7.344	23.106
RBF-CAT+LSTM_R4	0.06	0.14	2.737	14.767
RBF-CAT+LSTM_R5	0.022	0.168	2.346	13.622
RBF-CAT+MoE_R1	0.011	0.056	9.005	5.472
RBF-CAT+MoE_R2	0.018	0.155	5.033	8.421
RBF-CAT+MoE_R3	0.016	0.528	6.929	9.598
RBF-CAT+MoE_R4	0.013	0.136	1.86	5.401
RBF-CAT+MoE_R5	0.009	0.163	1.517	5.351
RBF-IN+LSTM_R1	0.066	0.031	10.113	10.56
RBF-IN+LSTM_R2	0.08	0.164	5.094	18.396
RBF-IN+LSTM_R3	0.033	0.386	7.847	27.836
RBF-IN+LSTM_R4	0.093	0.146	3.846	11.789
RBF-IN+LSTM_R5	0.033	0.18	2.796	13.03
RBF-IN+MoE_R1	0.053	0.051	11.364	5.743
RBF-IN+MoE_R2	0.047	0.156	4.576	11.672
RBF-IN+MoE_R3	0.031	0.395	7.426	16.257
RBF-IN+MoE_R4	0.073	0.146	3.681	6.019
RBF-IN+MoE_R5	0.024	0.166	2.459	7.206
VAE-CAT+LSTM_R1	0.048	0.028	9.813	15.271
VAE-CAT+LSTM_R2	0.043	0.162	4.471	18.785
VAE-CAT+LSTM_R3	0.025	0.458	8.249	23.388
VAE-CAT+LSTM_R4	0.052	0.139	3.141	14.535
VAE-CAT+LSTM_R5	0.024	0.166	2.342	14.668
VAE-CAT+MoE_R1	0.016	0.072	8.82	9.608
VAE-CAT+MoE_R2	0.012	0.154	5.123	9.996
VAE-CAT+MoE_R3	0.015	0.553	6.445	13.058
VAE-CAT+MoE_R4	0.024	0.141	2.311	10.71
VAE-CAT+MoE_R5	0.012	0.16	2.124	9.366
VAE-IN+LSTM_R1	0.062	0.042	11.249	19.178
VAE-IN+LSTM_R2	0.074	0.162	4.506	26.553
VAE-IN+LSTM_R3	0.032	0.406	8.21	31.673
VAE-IN+LSTM_R4	0.1	0.149	4.148	12.719
VAE-IN+LSTM_R5	0.034	0.179	2.439	15.884
VAE-IN+MoE_R1	0.031	0.059	10.818	30.045
VAE-IN+MoE_R2	0.036	0.154	5.373	32.615
VAE-IN+MoE_R3	0.036	0.342	8.61	44.301
VAE-IN+MoE_R4	0.04	0.149	3.673	35.877
VAE-IN+MoE_R5	0.023	0.159	2.339	32.24

D.2 FE-/FC-OMG performance results

Name	Reconstruction error (MSE)	Adversarial error (Adv. Loss)	Rotational error (MSE)	Sum of angular update (radian / s)
AE+LSTM_F_R3	0.071	0.372	7.502	28.415
AE+LSTM_F_R4	0.033	0.139	3.934	18.043
AE+LSTM_F_R5	0.033	0.184	3.129	14.186
AE+LSTM_RAW_R2	0.115	0.196	5.833	22.107
AE+LSTM_RAW_R3	0.099	0.375	7.649	25.623

AE+LSTM_RAW_R4	0.044	0.154	4.542	10.369
AE+LSTM_RAW_R5	0.051	0.19	3.375	3.798
AE+LSTM_T_R2	0.069	0.156	5.217	14.593
AE+LSTM_T_R3	0.072	0.304	7.917	25.004
AE+LSTM_T_R4	0.036	0.148	3.813	9.946
AE+LSTM_T_R5	0.024	0.182	2.397	16.637
AE+MoE_F_R2	0.035	0.157	4.436	10.382
AE+MoE_F_R3	0.025	0.293	5.287	9.246
AE+MoE_F_R4	0.022	0.142	2.956	5.482
AE+MoE_F_R5	0.014	0.19	2.772	5.397
AE+MoE_RAW_R2	0.064	0.162	5.728	7.798
AE+MoE_RAW_R3	0.067	0.368	8.062	10.574
AE+MoE_RAW_R4	0.029	0.139	3.218	4.045
AE+MoE_RAW_R5	0.033	0.192	2.545	3.62
AE+MoE_T_R2	0.044	0.163	4.944	8.957
AE+MoE_T_R3	0.026	0.324	5.907	9.752
AE+MoE_T_R4	0.028	0.144	3.676	5.174
AE+MoE_T_R5	0.013	0.184	2.269	5.25
DEC-CAT+MoE_F_R2	0.082	0.177	5.979	19.317
DEC-CAT+MoE_F_R3	0.078	0.434	8.013	14.006
DEC-CAT+MoE_F_R4	0.039	0.163	3.972	5.628
DEC-CAT+MoE_F_R5	0.036	0.166	3.107	5.629
DEC-CAT+MoE_RAW_R2	0.073	0.165	6.165	8.138
DEC-CAT+MoE_RAW_R3	0.067	0.371	7.903	8.571
DEC-CAT+MoE_RAW_R4	0.028	0.136	2.764	4.381
DEC-CAT+MoE_RAW_R5	0.038	0.188	2.931	1.974
DEC-CAT+MoE_T_R2	0.052	0.16	5.619	14.846
DEC-CAT+MoE_T_R3	0.053	0.368	7.299	10.745
DEC-CAT+MoE_T_R4	0.024	0.146	3.613	5.837
DEC-CAT+MoE_T_R5	0.021	0.19	2.182	4.994
DEC-IN+MoE_F_R2	0.127	0.179	6.299	13.023
DEC-IN+MoE_F_R3	0.092	0.335	8.102	24.612
DEC-IN+MoE_F_R4	0.043	0.159	3.832	4.675
DEC-IN+MoE_F_R5	0.038	0.17	3.053	8.279
DEC-IN+MoE_RAW_R2	0.123	0.186	5.762	4.81
DEC-IN+MoE_RAW_R3	0.1	0.384	8.022	11.689
DEC-IN+MoE_RAW_R4	0.043	0.158	4.346	2.687
DEC-IN+MoE_RAW_R5	0.042	0.18	3.469	1.195
DEC-IN+MoE_T_R2	0.105	0.172	5.723	11.883
DEC-IN+MoE_T_R3	0.088	0.36	7.552	20.172
DEC-IN+MoE_T_R4	0.04	0.15	4.454	6.015
DEC-IN+MoE_T_R5	0.031	0.167	2.643	8.177
RBF-CAT+LSTM_F_R2	0.079	0.159	5.576	18.799
RBF-CAT+LSTM_F_R3	0.066	0.367	7.371	27.677
RBF-CAT+LSTM_F_R4	0.032	0.142	3.719	20.49
RBF-CAT+LSTM_F_R5	0.036	0.178	3.205	14.287
RBF-CAT+LSTM_RAW_R2	0.124	0.223	5.662	15.25
RBF-CAT+LSTM_RAW_R3	0.104	0.353	7.239	35.145
RBF-CAT+LSTM_RAW_R4	0.042	0.151	4.526	12.289
RBF-CAT+LSTM_RAW_R5	0.047	0.19	3.101	4.197
RBF-CAT+LSTM_T_R2	0.067	0.165	5.244	15.766
RBF-CAT+LSTM_T_R3	0.066	0.336	7.441	29.057
RBF-CAT+LSTM_T_R4	0.03	0.139	3.555	16.273
RBF-CAT+LSTM_T_R5	0.02	0.188	2.894	15.332
RBF-CAT+MoE_F_R2	0.03	0.157	5.023	10.825
RBF-CAT+MoE_F_R3	0.029	0.365	5.289	11.396
RBF-CAT+MoE_F_R4	0.021	0.14	2.355	5.113
RBF-CAT+MoE_F_R5	0.012	0.171	1.879	5.812
RBF-CAT+MoE_RAW_R2	0.05	0.156	5.956	7.012
RBF-CAT+MoE_RAW_R3	0.062	0.361	8.36	9.818
RBF-CAT+MoE_RAW_R4	0.032	0.143	3.459	5.721
RBF-CAT+MoE_RAW_R5	0.019	0.197	1.884	5.179
RBF-CAT+MoE_T_R2	0.045	0.154	5.327	9.397
RBF-CAT+MoE_T_R3	0.04	0.346	6.114	10.144
RBF-CAT+MoE_T_R4	0.015	0.133	2.825	5.825
RBF-CAT+MoE_T_R5	0.019	0.199	2.623	6.133
RBF-IN+LSTM_F_R2	0.12	0.173	5.509	13.827
RBF-IN+LSTM_F_R3	0.102	0.369	8.5	35.882

RBF-IN+LSTM_F_R4	0.046	0.155	4.262	15.929
RBF-IN+LSTM_F_R5	0.045	0.183	3.614	14.897
RBF-IN+LSTM_RAW_R2	0.121	0.191	5.435	16.746
RBF-IN+LSTM_RAW_R3	0.105	0.37	8.797	33.945
RBF-IN+LSTM_RAW_R4	0.042	0.155	4.213	11.019
RBF-IN+LSTM_RAW_R5	0.044	0.187	3.356	15.502
RBF-IN+LSTM_T_R2	0.11	0.17	5.324	12.878
RBF-IN+LSTM_T_R3	0.095	0.32	8.444	36.747
RBF-IN+LSTM_T_R4	0.039	0.149	3.912	12.159
RBF-IN+LSTM_T_R5	0.043	0.195	3.111	19.856
RBF-IN+MoE_F_R2	0.122	0.175	6.035	12.201
RBF-IN+MoE_F_R3	0.096	0.274	8.27	28.04
RBF-IN+MoE_F_R4	0.042	0.151	3.701	5.243
RBF-IN+MoE_F_R5	0.038	0.165	3.024	8.207
RBF-IN+MoE_RAW_R2	0.121	0.178	5.757	5.69
RBF-IN+MoE_RAW_R3	0.1	0.359	7.796	9.568
RBF-IN+MoE_RAW_R4	0.04	0.153	3.786	3.503
RBF-IN+MoE_RAW_R5	0.039	0.186	3.361	1.523
RBF-IN+MoE_T_R2	0.105	0.164	5.305	13.744
RBF-IN+MoE_T_R3	0.088	0.237	8.234	20.673
RBF-IN+MoE_T_R4	0.045	0.142	4.281	6.775
RBF-IN+MoE_T_R5	0.03	0.171	3.019	7.926
VAE-CAT+LSTM_F_R2	0.088	0.171	5.815	25.186
VAE-CAT+LSTM_F_R3	0.071	0.367	7.58	32.116
VAE-CAT+LSTM_F_R4	0.036	0.142	4.063	20.186
VAE-CAT+LSTM_F_R5	0.034	0.187	2.963	17.843
VAE-CAT+LSTM_RAW_R2	0.118	0.178	5.584	16.888
VAE-CAT+LSTM_RAW_R3	0.103	0.406	7.615	26.356
VAE-CAT+LSTM_RAW_R4	0.043	0.153	4.182	4.069
VAE-CAT+LSTM_RAW_R5	0.043	0.187	3.17	16.228
VAE-CAT+LSTM_T_R2	0.075	0.173	5.322	20.191
VAE-CAT+LSTM_T_R3	0.065	0.306	8.451	29.023
VAE-CAT+LSTM_T_R4	0.034	0.145	4.249	17.361
VAE-CAT+LSTM_T_R5	0.025	0.178	2.37	14.048
VAE-CAT+MoE_F_R2	0.033	0.156	5.103	15.67
VAE-CAT+MoE_F_R3	0.03	0.381	5.505	19.524
VAE-CAT+MoE_F_R4	0.021	0.145	3.327	13.346
VAE-CAT+MoE_F_R5	0.013	0.181	2.021	12.32
VAE-CAT+MoE_RAW_R2	0.061	0.153	5.79	37.513
VAE-CAT+MoE_RAW_R3	0.057	0.41	6.921	27.231
VAE-CAT+MoE_RAW_R4	0.027	0.136	3.685	16.259
VAE-CAT+MoE_RAW_R5	0.019	0.192	2.066	18.993
VAE-CAT+MoE_T_R2	0.031	0.15	4.985	14.142
VAE-CAT+MoE_T_R3	0.023	0.264	5.221	13.609
VAE-CAT+MoE_T_R4	0.015	0.137	3.15	8.996
VAE-CAT+MoE_T_R5	0.017	0.183	2.586	9.752
VAE-IN+LSTM_F_R2	0.135	0.21	5.599	17.544
VAE-IN+LSTM_F_R3	0.105	0.528	8.708	38.294
VAE-IN+LSTM_F_R4	0.052	0.167	4.002	12.064
VAE-IN+LSTM_F_R5	0.045	0.183	3.529	24.149
VAE-IN+LSTM_RAW_R2	0.124	0.178	5.392	17.502
VAE-IN+LSTM_RAW_R3	0.108	0.362	8.548	45.222
VAE-IN+LSTM_RAW_R4	0.044	0.159	4.192	10.974
VAE-IN+LSTM_RAW_R5	0.045	0.187	3.424	11.803
VAE-IN+LSTM_T_R2	0.113	0.168	5.32	22.51
VAE-IN+LSTM_T_R3	0.094	0.33	7.92	39.346
VAE-IN+LSTM_T_R4	0.046	0.151	3.922	19.672
VAE-IN+LSTM_T_R5	0.04	0.177	2.959	24.382
VAE-IN+MoE_F_R2	0.071	0.165	6.007	23.89
VAE-IN+MoE_F_R3	0.084	0.268	7.721	44.402
VAE-IN+MoE_F_R4	0.039	0.166	4.117	28.464
VAE-IN+MoE_F_R5	0.037	0.168	3.423	24.016
VAE-IN+MoE_RAW_R2	0.112	0.152	6.257	17.369
VAE-IN+MoE_RAW_R3	0.105	0.377	8.837	16.712
VAE-IN+MoE_RAW_R4	0.04	0.146	4.086	8.878
VAE-IN+MoE_RAW_R5	0.045	0.185	3.41	4.027
VAE-IN+MoE_T_R2	0.058	0.167	5.73	32.947
VAE-IN+MoE_T_R3	0.06	0.417	7.63	32.471

VAE-IN+MoE_T_R4	0.032	0.14	4.187	43.394
VAE-IN+MoE_T_R5	0.039	0.172	2.949	27.929

D.3 FS-OMG performance results

Name	Reconstruction error (MSE)	Adversarial error (Adv. Loss)	Rotational error (MSE)	Sum of angular update (radian / s)
AE+MoE_F_R2	0.038	0.312	2.81	10.97
AE+MoE_F_R3	0.024	0.142	3.891	5.467
AE+MoE_F_R4	0.014	0.176	5.433	5.248
AE+MoE_R1	0.017	0.066	4.878	5.443
AE+MoE_R2	0.031	0.157	5.796	7.979
AE+MoE_R3	0.013	0.5	1.51	8.527
AE+MoE_R4	0.025	0.136	4.101	5.552
AE+MoE_R5	0.012	0.154	6.016	5.236
AE+MoE_T_R1	0.047	0.157	6.454	9.552
AE+MoE_T_R2	0.04	0.286	2.656	10.34
AE+MoE_T_R3	0.021	0.135	4.656	6.066
AE+MoE_T_R4	0.017	0.178	5.065	5.806
DEC-CAT+MoE_F_R1	0.093	0.165	7.217	7.452
DEC-CAT+MoE_F_R2	0.096	0.392	2.59	6.57
DEC-CAT+MoE_F_R3	0.046	0.152	4.413	2.606
DEC-CAT+MoE_F_R4	0.042	0.181	6.151	1.223
DEC-CAT+MoE_R1	0.036	0.054	5.052	6.776
DEC-CAT+MoE_R2	0.04	0.158	4.597	9.287
DEC-CAT+MoE_R3	0.017	0.486	4.089	10.627
DEC-CAT+MoE_R4	0.038	0.142	4.177	6.17
DEC-CAT+MoE_R5	0.017	0.159	6.536	5.613
DEC-CAT+MoE_T_R1	0.058	0.164	7.398	12.853
DEC-CAT+MoE_T_R2	0.057	0.378	2.941	13.047
DEC-CAT+MoE_T_R3	0.034	0.146	3.612	5.97
DEC-CAT+MoE_T_R4	0.022	0.174	5.876	5.976
RBF-CAT+MoE_F_R1	0.049	0.16	7.761	8.657
RBF-CAT+MoE_F_R2	0.059	0.334	3.423	9.144
RBF-CAT+MoE_F_R3	0.03	0.141	2.823	5.655
RBF-CAT+MoE_F_R4	0.025	0.19	3.139	5.282
RBF-CAT+MoE_R1	0.017	0.074	7.202	5.481
RBF-CAT+MoE_R2	0.017	0.157	2.521	7.716
RBF-CAT+MoE_R3	0.013	0.531	2.585	9.463
RBF-CAT+MoE_R4	0.022	0.144	4.132	5.302
RBF-CAT+MoE_R5	0.009	0.17	6.041	5.309
RBF-CAT+MoE_T_R1	0.052	0.171	5.356	10.291
RBF-CAT+MoE_T_R2	0.042	0.332	2.765	10.707
RBF-CAT+MoE_T_R3	0.019	0.139	2.785	5.593
RBF-CAT+MoE_T_R4	0.012	0.181	4.65	5.588
VAE-CAT+MoE_F_R1	0.049	0.162	6.997	42.476
VAE-CAT+MoE_F_R2	0.056	0.401	4.364	47.101
VAE-CAT+MoE_F_R3	0.034	0.136	5.654	14.592
VAE-CAT+MoE_F_R4	0.026	0.168	5.82	29.245
VAE-CAT+MoE_R1	0.016	0.078	5.342	10.356
VAE-CAT+MoE_R2	0.02	0.15	2.328	11.252
VAE-CAT+MoE_R3	0.014	0.454	3.681	11.458
VAE-CAT+MoE_R4	0.032	0.149	4.053	10.54
VAE-CAT+MoE_R5	0.011	0.167	5.961	9.272
VAE-CAT+MoE_T_R1	0.039	0.162	7.18	14.113
VAE-CAT+MoE_T_R2	0.04	0.323	2.338	16.989
VAE-CAT+MoE_T_R3	0.021	0.136	3.891	10.53
VAE-CAT+MoE_T_R4	0.018	0.179	4.964	12.572

Table D.3: Reduced feature set