University of Reading

Department of Computer Science

Evaluation of machine learning techniques on reducing computational expense for numerical weather prediction

Jack Lau

Supervisor: Dr. Varun Ojha

A report submitted in partial fulfilment of the requirements of
the University of Reading for the degree of
Master of Science in *Data Science and Advanced Computing*

September 17, 2021

Declaration

I, Jack Lau, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Jack Lau
September 17, 2021

Abstract

Modern numerical weather forecasting systems use many of the world's most powerful computers, due to the requirements of processing a gigantic amount of data and executing numerous complicated calculations to derive the final forecasting outputs. There is always a desperate need to mitigate such burden, so that more saved computation power can be reinvested to further improve the accuracy and resolution of the weather predictions. In this project, we specifically investigate if supervised machine learning models can effectively predict the local Lyapunov exponents (LLE) in two low-dimensional dynamical systems, with less computation expense than calculating them using traditional numerical methods. Our results indicate that all the selected supervised learning algorithms can achieve sufficiently good predictability. In additions, the regression tree model is able to deliver similar predicting performance as the three deep neural network models, but with a speed of 2 orders of magnitude faster. Regression tree is also 2 to 10 times faster than using the traditional numerical computation method. We believe the goal of using supervised machine learning for LLE prediction is achieved, and we also proved that it is possible to reduce the computation cost by applying machine learning. Such positive findings suggest it is worthwhile to proceed to the exploration of the applicability of ML techniques in more complex dynamical systems.

Keywords: supervised machine learning, numerical weather prediction, Lyapunov exponents

Report's total word count: 13,443

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Varun Ojha, for providing invaluable guidance throughout the project.

I would also like to thank for Professor Alberto Carrassi and Dr. Javier Amezcua for their precious advice and brilliant ideas on the research.

It is my pleasure to collaborate with Daniel Ayers in this project, who has made tremendous contributions in all aspects of our work.

I am extremely grateful to my wife and my three daughters for their love, prayers, understanding and continuing support.

Last but not least, sing praises and give thanks to God, the origin of wisdoms, for His showers of blessings and abundant provision that make the completion of this report possible.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
DNN	Deep Neural Network
GPU	Graphics Processing Unit
LE	Lyapunov Exponent
LLE	Local Lyapunov Exponent
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
NWP	Numerical Weather Prediction
RNN	Recurrent Neural Network
RT	Regression Tree
TPU	Tensor Processing Unit

Chapter 1. Introduction

1.1 Background

Numerical weather prediction (NWP) uses mathematical models of the atmosphere and oceans to predict the weather based on current weather conditions. The weather forecast is obtained by propagating the models forward from one time to another. This process involves manipulating a vast number of datasets and performing complex calculations, which requires some of the most powerful supercomputers in the world. However, even using facilities with such ultra-high computation capability, it is still desirable to save computational expense when propagating the model forward and invest it in higher resolution, better parameterisation and more sophisticated data assimilation.

The weather is a chaotic dynamical system. Certain weather events are less predictable (i.e., more dynamically unstable) than others, which means that when modelling such events, errors grow more rapidly. It is possible to mitigate rapid error growth with more accurate methods that require greater computation power. Therefore, it is preferable to adopt such expensive methods only when necessary. This leads to the idea of introducing an adaptive computational scheme to the NWP model.

In this project, we are interested in the prediction of the Lyapunov exponents (LEs) of the dynamical systems. The spectrum of LEs is characteristic for a dynamical system given certain properties are satisfied. LEs and their corresponding Lyapunov vectors are useful in NWP, for instance, data assimilation (Carrassi *et al.*, 2020; Palatella *et al.*, 2013), ensemble prediction systems (Toth and Kalnay, 1997). The LEs of a system are calculated as an average of finite-time Lyapunov exponents. Here these components are referred as the local Lyapunov exponents (LLEs), where the LLEs are calculated from a sufficiently long system trajectory, together with a series of calculation. This requires a substantial use of computational resources, particularly for high-dimensional systems.

On the other hand, in the field of machine learning, it is known that supervised learning methods generate models that can provide useful information about time series and dynamical systems, including future time-steps prediction (Lim and Zohren, 2021), and providing analysis by classification and regression (Goodfellow *et al.*, 2016; LeCun *et al.*, 2015). Using machine learning can be beneficial in weather forecasting for it is less sensitive to the perturbations than the physical models of the weather, and for building the ML models, it does not require in-depth knowledge of the underlying physical models (Holmstrom *et al.*, 2016).

1.2 Problem statement

We investigate how to use machine learning (ML) techniques to predict the local Lyapunov exponents (LLEs) of low dimension dynamical systems. The two dynamical systems concerned are Rössler (Rössler, 1976) and Lorenz-63 (Lorenz, 1963), and they are defined by the below equations:

Rössler

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + ay \\ \dot{z} &= b - z(x - c) \end{aligned} \tag{1}$$

Lorenz-63

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{2}$$

where a, b, c and σ, ρ, β are the constant parameters (properties) of the two systems respectively.

It is a supervised regression problem since the ML models will be built to predict the exact values (real numbers) of the corresponding LLEs of the dynamical systems directly, given the target values of the training samples are available. The supervised ML algorithms to be evaluated include both neural networks and tree-based methods. The performances of the algorithms are evaluated by using both pointwise accuracy and statistical accuracy. The computation efficiency is also assessed by measuring and comparing the wall clock execution times of the predictions.

1.3 Aims and objectives

The goal is to use machine learning to forecast the LLE spectrum by predicting the LLE values at any given time step with just inputting a short historical trajectory from the current time point. It is envisioned that the trained machine learning algorithm could then be served as a "light-weight" alternative to provide information for interpreting the local dynamical instability of the systems. Here the "light-weight" refers to the relieving computational burden for deriving the LLEs. Further, it is hoping to pave a path for extending such study to higher dimensional systems, which are more comparable to those used in modern weather prediction domain.

To achieve the aim, the project sets the following objectives:

1. To investigate and critique the computation demands of state-of-the-art NWP and the current applications of machine learning in NWP;
2. To propose the ML models for learning and predicting LLEs of two simple ODE (ordinary differential equations) dynamical systems (i.e., Rössler and Lorenz-63 systems);
3. To implement the proposed ML models using state-of-the-art tools and libraries;
4. To design and carry out the experiments for comparing the performance of the models;
5. To evaluate the result of the experiments and identify the limitations and propose improvement;
6. To apply and evaluate the ML models to a more complicated and realistic system (i.e., a PDE (partial differential equation) system with spatial dimension);
7. To reflect personal experience about self-knowledge gained and skills developed.

1.4 Solution approach

This project will adopt both qualitative and quantitative approaches, and will undertake the following tasks:
Literature review: selecting and reviewing relevant literatures of NWP and machine learning.
Data generation: develop programs to implement the selected numerical models for generating sample data for training and testing.
High-level model design: outlining the high-level design of the machine learning models for the defined problems.
Detailed model design: defining the detailed architecture and configuration of the machine learning models.
Model implementation: developing and testing the designed models with using selected tools.
Model training, tuning and testing: prepare data for training and testing the models. Automated tuning with a Bayesian optimisation in parameter space will also be employed for fine tuning the hyperparameters to ensure the fairness and objectivity of the results for the comparative experiments.
Experiments: designing and performing experiments to measure and compare the prediction performance of different models.
Evaluation: appreciation of the results obtained from the experiments, identify the limitations of the models and suggest potential improvements.

1.5 Summary of contributions and achievements

In the project, we have demonstrated the capabilities of popular supervised ML models on predicting the LLEs on Rössler and Lorenz-63 systems. The results show those ML models can achieve satisfactory performance in terms of both pointwise and statistical accuracy with only using recent time-steps as input. This is a green sign for proceeding the exploration of the applicability of ML techniques in more complex dynamical systems.
An end-to-end machine learning application has been built, with incorporating a comprehensive workflow of the machine learning covering data generation, data scaling, data shuffling, train-test data splitting and *k*-fold cross-validation. Pipeline is used to define a series of sequential steps in machine learning (e.g., data preparation, modelling operation, prediction transformation) to be applied correctly and consistently. The best practices of applying ML have been followed and are proved to be practicable.
A set of Python programs are developed for training pipeline, hyperparameter tuning, experiment execution and results plotting. The source codes, data files, results and plots are all shared through GitHub. This makes all the works in the project reproducible and traceable. We believe this can promote the exchange of experience and knowledge and foster further research.

1.6 Organization of the report

This report is organised into 7 chapters. Chapter 2 details the literature review of this project. In Chapter 3, the methodology used is described. The results and analysis based on the methodology applied are presented in Chapter 4. The findings from our results and analysis section are consolidated in Chapter 5, and the interpretation of the obtained results is also discussed in this chapter. We summarise our project in Chapter 6 as the conclusions, as well as proposing the relevant future work. Finally, we present the experience learned as the reflection to wrap up this report.

Chapter 2. Literature Review

2.1 Computationally intensive nature of NWP

Modern NWP is extremely demanding in computation and I/O capacity. For instance, to simulate the global climate system at the resolution to the single kilometre scale, one would need a high-performance computing system equipped with nearly 5,000 GPUs (Fuhrer *et al.*, 2018). On such system, a 10-day long moist simulation at 930 m grid spacing still requires over 15 hours execution time. In the article of Wedi *et al.* (2020), we can catch another glimpse of this aspect: the paper presented in the latest developed simulation model of the Earth's atmosphere, in order to complete 1 simulated week, the model took 1 day to run, with using 960 computing nodes from *Summit*, the current world's second fastest supercomputer as of June 2021, where each node housing two 22-core IBM Power9 CPUs and six NVIDIA Tesla V100 GPUs (TOP500, 2021). The model also generated 450TB of output data.

Specifically, when looking into the computation of LLEs of dynamic systems, numerical methods are used. Recalling those methods typically involve generating a long system trajectory, then combine with computing eigenvectors and eigenvalues via QR decomposition, and integrate the matrix differential equation (Pikovsky and Politi, 2016). Such algorithms do not scale well and require significant amount of computational cost. Consequently, computing the full LE spectrum for a modern weather prediction system is too expensive to be done during the forecast cycle.

2.2 Machine learning in NWP

In the area of weather and climate forecasting, both supervised and unsupervised ML have been used in NWP. They were found to be successful in many applications in different parts of the NWP, namely observations, data assimilation, numerical weather forecasts and post-processing (Hewson and Pillosu, 2020) and dissemination (Düben *et al.*, 2021). Chantry *et al.* (2021) pointed out that machine learning can take up or assist weather prediction at different timescales, particularly on now-casting and seasonal forecasting, but it requires sufficient high-quality data for training.

Particularly, supervised ML algorithms are applied for various purposes in NWP. Two of the typical areas include (i) emulating the dynamics of a system (Brajard *et al.*, 2020; Pathak *et al.*, 2017, 2018; Schultz *et al.*, 2021), and (ii) improving a physics-based model with data-driven correction and parameterisation (Bonavita and Laloyaux, 2020; Brajard *et al.*, 2021; Gottwald and Reich, 2021; Nguyen *et al.*, 2020; O'Gorman and Dwyer, 2018). Note that both approaches aim for building surrogate models for the evolution function of the dynamical systems. By evolution function, it means the function that takes the state at time t as input and outputs the state at $t + \Delta t$. Therefore, if calculation of LLEs is required in such cases, we need to use the evolution function to calculate a long trajectory, and then going through the lengthy numerical computation mentioned in Section 2.1 above.

2.3 Supervised ML algorithms

Supervised ML is the machine learning task of training or learning a function that can map an input to an output using labelled training data. Some of the most widely used supervised ML

algorithms include decision trees, support vector machines, K-nearest neighbour algorithm and neural networks. In the time-series problem domain, we see numerous usages of deep neural network architectures (Chantry *et al.*, 2021; Düben *et al.*, 2021; Lim and Zohren, 2021), such as CNN and RNN. We also see some successful deployments of classical ML model like decision trees (Hewson and Pillosu, 2020; Murugan Bhagavathi *et al.*, 2021).

Supervised ML is thought to be an appealing alternative to traditional methods used in NWP because the its objective is to approximate a target model with a reduced surrogate using only a subset of variables, which is assuredly more computationally manageable (Gottwald and Reich, 2021). This is especially evident for high-dimensional systems and high-volume data, in which the traditional algorithms in NWP are not scalable to cope with (Schultz *et al.*, 2021).

2.4 Critique of the review

It is learned that NWP systems nowadays use tremendous amount of computer resources. Even under the situation of modern compute is cheap and abundant, it is still worthwhile to reduce its usage whenever possible and save for the most critical processes in weather forecasting. It is evident that in the field of NWP, machine learning has been playing a key role, and its influence is only getting stronger in future, partly due to its potential to cut down the computation resource usage.

While the majority of the research on applying supervised ML to dynamic systems focused on reconstructing the evolution functions, we argue that predicting the LLEs directly using machine learning models is desirable and could possibly reduce the corresponding computational expenses.

2.5 Summary

In this chapter, we have studied that modern NWP system consumes a massive amount of CPU and GPU resources for weather forecasting. We understand that calculating Lyapunov exponents of chaotic systems is also computationally intensive. Conversely, the adoption rate of ML technologies in NWP is high and will continue to thrive, due to their ability in making the computation usage tractable. Finally, the fact that the lack of study on predicting LLEs directly using MLs motivates us to focus on this specific direction in the project.

Chapter 3. Methodology

3.1 Machine learning for regression problem

In the problem of this study, the input for the ML models is the system state at the current and several previous timesteps. The target is the LLEs calculated using the method of integrating perturbations from the current time t to $t + \tau$ (Ayers *et al.*, 2021). Generally, we have:

$$\text{(input, target)} = \left(\left[\mathbf{x}_{k_n}, \mathbf{x}_{k_{n-1}}, \dots, \mathbf{x}_{k_1}, \mathbf{x}_{k_0}\right], \quad \left(\lambda_k^{r_1}, \dots, \lambda_k^{r_{d-1}}, \lambda_k^{r_d}\right) \right) \tag{3}$$

where

- $\mathbf{x}_k$ denotes the system state at $k\Delta t$
- $k_i > k_j$ for any $i < j$
- $k_0 = k$
- $\{\lambda_k^{r_i} : i = 1, \dots, d\}$ is a set of LLEs computed from the interval $[k\Delta t, k\Delta t + \tau]$

The two chaotic systems concerned, i.e., Rössler and Lorenz-63, are three-dimensional systems, and three LLEs will be predicted in our problem. Note that the size of one time-step is set to 0.01, and LLEs are calculated over a four time-steps window, so $\tau = 0.04$.

3.1.1 Model selection

With referencing to the findings in literature review, decision tree (DT) and deep neural networks (DNNs) are chosen for the supervised ML model evaluation. On one hand, it is observed that these models are widely used within the NWP sector; on the other hand, they are paradigmatic algorithms with high acceptance in the ML community (LeCun *et al.*, 2015; Wu *et al.*, 2008). For DNNs, three different architectures are selected for assessment: multi-layer perceptron (MLP), convolutional neural network (CNN) and recurrent neural network (RNN).

3.1.1.1 Decision trees

Decision trees (DTs) (Breiman *et al.*, 2017) is a supervised learning method that is applicable to both classification and regression problems. It works as creating a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. DT is a non-parametric model, meaning that it makes no assumptions on the training data (e.g., distribution, independency). If a DT is built with the target variable as discrete values (e.g., categorical values), it is also called a classification tree. For predicting the target variable as continuous values (typically real numbers), it is a regression tree (RT). Decision trees have long been one of the most popular machine learning algorithms as they are simple to understand, interpret and visualise (Wu *et al.*, 2008). The down sides of using DTs are they are prone to overfitting, and they could be unstable due to their sensitivity to the training data.

3.1.1.2 Multi-layer perceptron

The Multi-layer perceptron (MLP) is a simple feed-forward artificial neural network, where the units (neurons) in the network are arranged into a graph without any cycles, so that all the computation can be done sequentially. The MLP contains the input layer as the first layer, and

its units take the values of the input features. The last layer is the output layer, and all the layers in between are known as hidden layers. Here we describe the MLPs are fully connected, meaning that for every neuron in any layer of the network, it is connected to all the neurons in the previous layer. MLP is a classical type of neural network. It is very flexible and can be used for approximating a mapping from inputs to outputs.

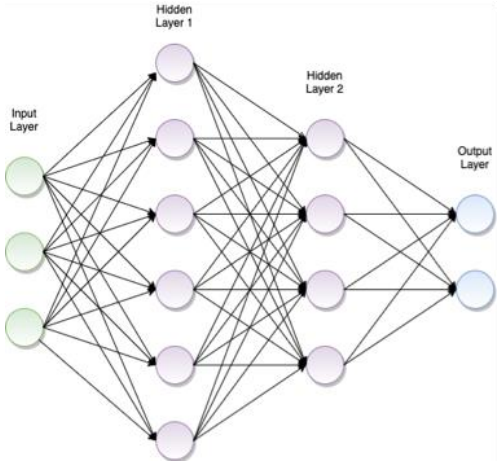

Figure 1: A typical MLP with 2 hidden layers

3.1.1.3 Convolutional neural network

Convolutional neural networks (CNN) share similar characteristics with MLP, they are both made up of neurons that have learnable weights and biases. However, CNNs assume the inputs are images and their architecture are designed for processing images. Particularly, the layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. The neurons of a CNN layer will only be connected to a small region of its previous layer, instead of fully connected. A CNN is constructed by three main types of hidden layers:

- Convolutional layer: compute the output of neurons that are connected to local regions of the input
- Pooling layer: perform down-sampling operation along the spatial dimensions
- Fully connected layer: compute the final output for the network

Although CNNs were originally designed for image processing (LeCun *et al.*, 1999), their usage has already been further extended to handling other problems, such as time-series classification (Zhao *et al.*, 2017).

3.1.1.4 Recurrent neural network

Recurrent neural network (RNN) is another class of artificial neural networks. Instead of having a simple feed-forward architecture like MLP, RNNs incorporate directed graph formed by the connections between nodes along a temporal sequence. Such characteristic allows it to exhibit temporal dynamic behaviour.

Long-short term memory, or LSTM (Hochreiter and Schmidhuber, 1997), is a typical RNN architecture and has recently gained its reputation on effectively modelling a number of ML problems related to sequential data, such as handwriting recognition, audio and video analysis (Greff *et al.*, 2016). A LSTM unit has a cell, an input gate, and output gate and a forget gate. The cell is responsible for remembering values over arbitrary time, and the gates regulate the

flow of the information in and out of the cell. This specific architecture makes it effective to capture long-term temporal dependencies (Hochreiter and Schmidhuber, 1997). The LSTM model accepts time-series data consisting of multiple time steps.

3.1.2 Input data generation

The dynamical systems chosen, namely Rössler and Lorenz-63 systems, are autonomous deterministic models. That means they do not explicitly depend on the independent variable (e.g., time) and if the initial state is given, the future states of such systems can be predicted theoretically. Due to this nature, the input samples for training and testing of the supervised ML experiments can be generated by calculation through numerical computation methods. The trajectory data of the two systems are calculated using the fourth order Runge-Kutta scheme (RK4) (Epperson, 2021, p.413-419) with timestep $\Delta t = 0.01$.

With consideration of restricting the input data size, such that to limit the computational requirements of the ML models, it is desirable to feed in only a small number of recent time-steps as input, while still be able to provide sequential information to the ML models. For each sample, the most recent six time-steps in the trajectory are chosen as the input features, such that the input will be:

$$[\mathbf{x}_{k-5}, \mathbf{x}_{k-4}\mathbf{x}_{k-3}, \mathbf{x}_{k-2}, \mathbf{x}_{k-1}, \mathbf{x}_k]$$

where $\mathbf{x}_k$ denotes current time-step, $\mathbf{x}_{k-n}$ as the n^{th} previous time-step from current one. Each of the time-steps consists of 3 components (x, y, z), representing the three-dimensional state of the dynamical systems. Therefore, the size of a single set of input feature is 18 (i.e., 6 time-steps $\times$ 3 state components).

Note that an initial transient period of the first 500 time-steps in the generated trajectory is discarded to ensure the trajectory is in the attractor, which refers to a set of states toward which is invariant under the dynamics.

The expected outputs of the models are the first three LLE values corresponding to each time-step $\mathbf{x}_k$, i.e.:

$$\left(\lambda_k^{r_1}, \lambda_k^{r_2}, \lambda_k^{r_3}\right)$$

These 3 LLEs are denoted as LLE_1, LLE_2 and LLE_3 in the subsequent content of this report.

Therefore, equation (3) becomes:

$$(\text{input, target}) = \left([\mathbf{x}_{k-5}, \mathbf{x}_{k-4}\mathbf{x}_{k-3}, \mathbf{x}_{k-2}, \mathbf{x}_{k-1}, \mathbf{x}_k], \quad (LLE_1, LLE_2, LLE_3) \right) \qquad (4)$$

We use the parameters (a, b, c) = (0.37, 0.2, 5.7) and (σ, ρ, β) = (10, 28, 8/3). The LLEs at time t are calculated by integrating perturbations from time t to $t + \tau$, where $\tau = 0.04$.

3.1.3 Exploratory data analysis

The process of exploratory data analysis is useful for gaining a high-level understanding of the dataset, such as data size and data distributions, which is crucial for subsequent processes

including data pre-processing, model design etc. Specifically in our project, this analysis can also help verifying the correctness of the generated data with comparing with those from other literatures numerically and visually.

Following the method described in Ayers *et al.* (2021), the LLEs are calculated using a trajectory of 720,000 computed time-steps with $\tau = 0.04$, and a transient period of 1,200 iterations. The LEs are calculated as the cumulative mean of the corresponding LLE values and are shown in Table 1. It can be observed that for both systems, the computed values of LE_1 are positive, LE_2 values are very close to zero, while the values are negative for LE_3.

Taking the reference values from Sprott (2003), the calculated LEs of Rössler are $(0.0714, 0, -5.3943)$, which are quite close to our computed values, despite our value of the parameter a is 0.37 (0.2 in Sprott, 2003). For Lorenz-63, we share the same parameter values with the cited source, and their LE values $(0.9056, 0, -14.5723)$ nearly coincide with ours. Thus, it is confident that our calculated values are consistent with those reference figures.

Table 1: The LEs of the Rössler and Lorenz-63 systems as calculated using 720,000 iterations with τ = 0.04, and a transient period of 1200 iterations. The LE is shown with the variation of the last 2000 iterations (Ayers et al., 2021).

System	**LE_1**	**LE_2**	**LE_3**
Rössler	0.19597 ± 0.00011	0.0000075 ± 0.0001275	−4.30097 ± 0.00068
Lorenz-63	0.90495 ± 0.000145	0.001975 ± 0.000055	−14.571345 ± 0.000105

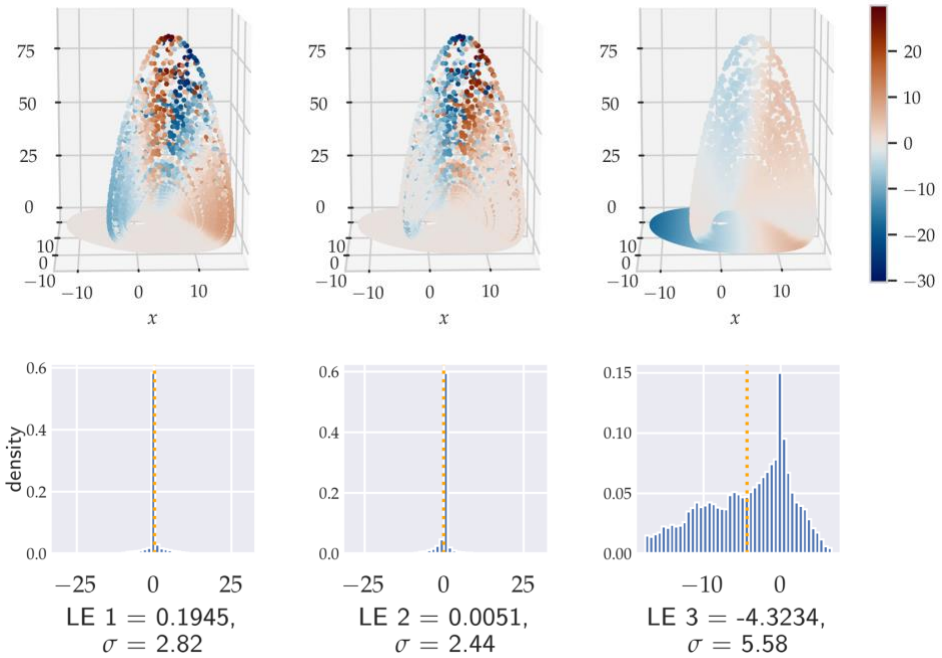

Figure 2: A trajectory of the Rössler system coloured by LLE values (top row) and the corresponding statistical distribution of the LLEs via histograms (bottom row); the mean of the LLE values is marked as orange dash line in each column. The mean and standard deviation of the LLE values are shown underneath (Ayers et al., 2021).

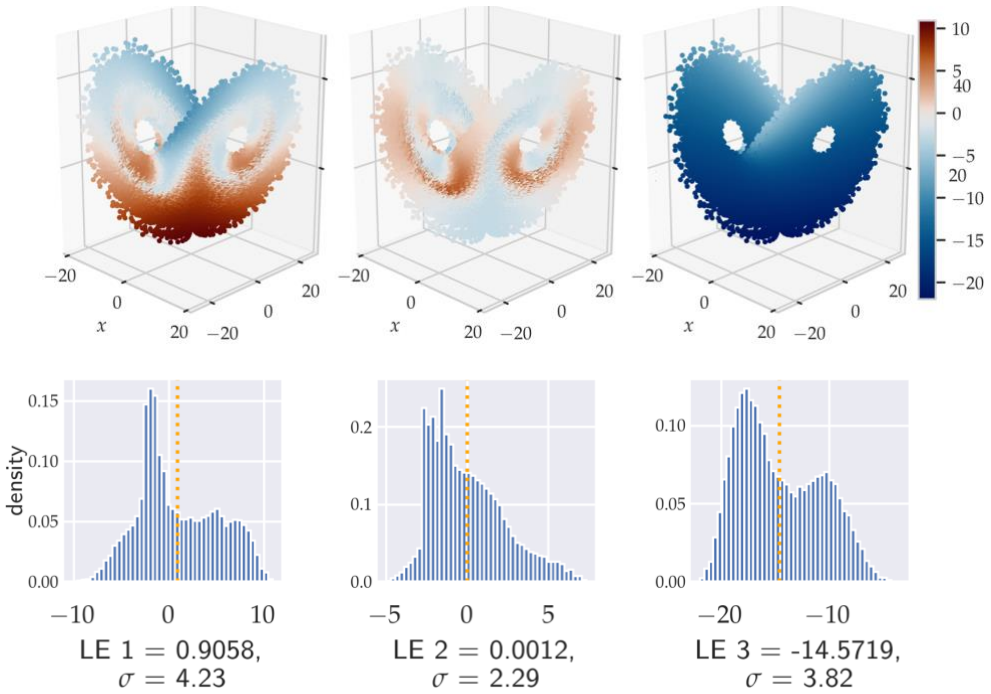

Figure 3: A trajectory of the Lorenz-63 system coloured by LLE values (top row) and the corresponding statistical distribution of the LLEs via histograms (bottom row); the mean of the LLE values is marked as orange dash line in each column. The mean and standard deviation of the LLE values are shown underneath (Ayers et al., 2021).

Figure 2 and Figure 3 show the points of the trajectory plotted in phase space for Rössler and Lorenz-63 respectively. The points in the phase space are coloured based on the LLE values. The bottom row shows the histograms of the three LLEs, illustrating their statistical distributions.

The phase space plot allows us to examine the local heterogeneity of the LLE values in the phase space. Local heterogeneity refers to the degree of variation of an attribute's value of the neighbours at a local region. The higher the degree of variation, the greater the local heterogeneity. In Rössler system, the LLE_3 has the lowest local heterogeneity as we see the colour transition is mostly smooth. The LLE_1 and LLE_2 exhibit quite high level of local heterogeneity relatively. For Lorenz-63 system, since the colour transitions are smoother, all LLEs are generally showing less mixing when comparing with those of the Rössler.

The histograms of Rössler's LLE_1 and LLE_2 show that their statistical distributions are quite symmetric, but the distributions are highly concentrated near their mean values. The histogram of LLE_3 is skewed left, and the range of the LLE_3 values is smaller than that of LLE_1 and LLE_2. The LLE distributions of Lorenz-63 are all skewed to right. The values of LLE_1 have largest range, followed by those of LLE_3 and then LLE_2.

3.1.4 Input features selection

As mentioned in Section 3.1.2, we select 6 most recent time-steps from the trajectory for each input sample, with a full system state (i.e., (x, y, z)) included in each time-step, giving a total of 18 input features.

Additionally, we would like to have a study on the impact of availability of historic trajectory to the prediction. Therefore, all the experiments will be repeated with using only current time-step ($\mathbf{x}_k$) in input samples. There will then have two set of samples:

Set 1: All time-steps (ATS)
(input, target) $= ([\mathbf{x}_{k-5}, \mathbf{x}_{k-4}\mathbf{x}_{k-3}, \mathbf{x}_{k-2}, \mathbf{x}_{k-1}, \mathbf{x}_k], \quad (LLE_1, LLE_2, LLE_3))$

Set 2: One time-step (OTS)
(input, target) $= ([\mathbf{x}_k], \quad (LLE_1, LLE_2, LLE_3))$

The latter case will have only 3 input features, i.e., the complete spatial status (x, y, z) for each $\mathbf{x}_k$. Note that separate ML models must be built for different input time-step cases.

3.1.5 Data pre-processing

3.1.5.1 Train-validation-test split

The goal of ML is to discover a model that learns the relationship between the input and output variables using the training dataset. The desired outcome is to establish a trained model that works equally well on any input dataset. To achieve this, one of the strategies is to avoid the models from learning the test dataset, that causes the model overfitting the test samples (i.e., performing too well on that specific test samples), and eventually loses the generalisation to predict other unseen data. This is one kind of the **data leakage** problems in machine learning.

The best practice is to split the available samples into separate sets, each for different purpose. Typically, we reserve the majority of the data as the training set, and one set for validation during the training of neural network models, and lastly, a test set for evaluating the trained models.

3.1.5.2 Data Scaling

Data scaling is a recommended pre-processing before feeding data into machine learning models like neural networks. The data scaling process aims to reduce the potential impact to the learning of the models due to the differences in the scales across input features. Standardisation is one of the commonly used data scaling techniques. Standardised data results in improving the back-propagation algorithm's performance in neural networks (Haykin, 2010). In the project, the input features are scaled using `StandardScaler` from scikit-learn (Pedregosa *et al.*, 2011), while the target values are standardised using scikit-learn's `TransformedTargetRegressor`. By applying standardisation, the input features and the target values are transformed by removing the mean (i.e., mean = 0) and scaling to unit variance (i.e., variance = 1). The standardised value x' of a sample x is calculated as:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{5}$$

where $\bar{x}$ is the mean of the sample vector, and σ is its standard deviation.

Note that the data scaling is NOT required to apply to RT models since it has no impact to the models' performance. In addition, to avoid any information leaks from test or validation datasets to the models during training, only the training dataset will be used to build standardisation scaler, and then the same scaling will be applied to validation and test dataset.

3.1.6 Pipelining

Pipelining is a technique in ML, that is used to define a series of sequential steps in machine learning (e.g., data preparation, modelling operation, prediction transformation) to be applied correctly and consistently. In our project, the samples must be pre-processed for standardisation, and the target values also need to be scaled before training, regardless of what ML algorithm is used. Pipelines help to encapsulate all these data transformation processes into a single entity, such that the model training and prediction can be done in a single call, and all these transformation steps will be executed in correct order.

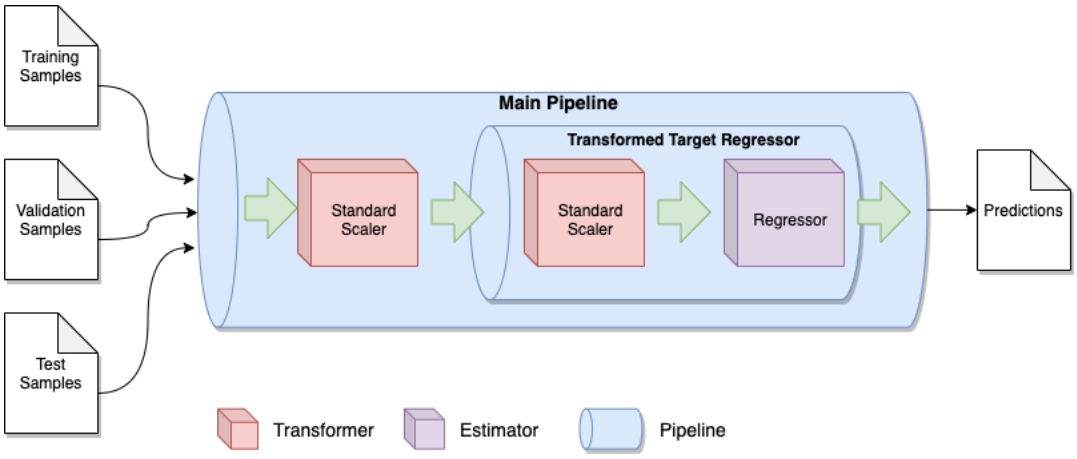

Figure 4: Use of pipeline in ML modelling

Figure 4 depicts the application of pipelines in our project. The pipeline is implemented using scikit-learn library. A main pipeline object is created, it consists of a **StandardScaler** as the data transformer of the input features, and the **TransformedTargetRegressor** is added as a nested pipeline object for standardising the target values (i.e., actual LLE values). The actual ML model, or the **Regressor**, is embedded as the estimator object in it. When training a model, the **`fit()`** method of the main pipeline object is called, and fed with the testing samples (and validation samples for DNN models). The **`predict()`** method is called for prediction after the pipeline object is trained, with passing the test samples as input instead.

Besides bringing the benefit of encapsulation, pipelining helps avoid data leakage by preventing data transformation process (such as scaling) being applied to the entire dataset. When training and predicting using pipeline, it ensures that the data transformation is prepared based on the training dataset only, then is applied to all other datasets.

3.1.7 Techniques to improve deep neural networks

The highly configurable architecture of deep neural networks is a double-side sword: while it offers immense power to learn internal representation from raw input data; it also has higher

likelihood to suffer from overfitting. In this section, we focus on utilising several techniques on combating the overfitting during the learning process of DNN models.

3.1.7.1 Regulariser

The purpose of adding regulariser is to introduce penalty to the loss function of the DNN model, so as to penalise the magnitude of the model parameters. Empirical results show incorporating regulariser into DNN models may reduce their generalisation errors but not the training error (Goodfellow *et al.*, 2016). There are 2 types of regulariser available:

- **L1 regulariser**: adding a penalty proportional to the absolute magnitude of the network's weights into the training loss function
- **L2 regulariser**: adding a penalty proportional to the squared values of the network's weights into the training loss function

Note that there is no simple answer to decide whether adding regulariser or not, nor choosing which type of regulariser will help. It can differ from problem to problem.

3.1.7.2 Early stopping

When training complex DNN models, it is often observed that the training error continues to reduce as training epochs pass, but at certain point the validation error starts to grow, which indicates the model begins to overfit. Early stopping provides a straightforward solution that it stops the training when there is no improvement on validation loss. Instead of returning the final trained parameters, we can return the optimal set that can produce the minimum loss throughout the training.

3.1.7.3 Dropout

Dropout works by simply excluding input or hidden units at random in a single batch of stochastic gradient descent during training (Srivastava *et al.*, 2014). Different probability values are assigned for dropping input units and hidden units respectively. Previous experiments showed such approach has surprisingly positive impact on inhibiting the overfitting issue. However, our experience on using such feature in our problem does not have significant performance gain observed. As a result, dropout is not adopted.

3.1.8 Hyperparameter tuning

The hyperparameters of a model refer to a set of configurations that is external to the model, and their values cannot be learned by the model from the data. Take decision tree as example, its hyperparameters include the maximum depth of the tree, minimum number of samples of a leaf node etc.; while the number of neurons, number of layers and learning rate of the optimiser are some of the hyperparameters of neural networks. Hyperparameters are different from parameters of models, as the latter ones are internal to the models, or literally part of the models, which are learned from historical training data (for instance, the weights and biases are the parameters of a neural network model).

Hyperparameters are difficult to learn from training data because doing so will often push the capacity of the model to its maximum capacity, causing overfitting. In other words, it results in a less generalised model (Goodfellow *et al.*, 2016).

However, there is a need to determine the best possible hyperparameter values for each model in order to unleash its maximum prediction capability while maintaining a reasonably good generalisation. This process is called the hyperparameter tuning, and it is usually done during the later stage of the model development. There are several strategies for performing hyperparameter tuning: manual tuning, grid search, random search and Bayesian optimisation.

Manual tuning is a trial-and-error method by specifying a set of hyperparameter values to a model, then run the training and testing on the model and capture its performance scores. Then a different set of hyperparameter values is applied to the same model, rerun the training and testing, and record the scores again. The set of hyperparameters is chosen with the corresponding model achieving highest score among all trials.

Grid Search simply uses brute-force search on the hyperparameter space, it tries every possible configuration in the given search space and finds out the best one that gets the best score. This strategy allows the search to be run in parallel as it does not depend on the search history.

Random Search (Bergstra and Bengio, 2012) is similar to the grid search, but it picks the point randomly from the configuration space. This approach has the advantage on exploring the hyperparameters space more widely than grid search, since the values of the configuration are selected randomly, it is very improbable to pick the same value more than once.

Bayesian Optimisation (Mockus, 2012) uses surrogate function to approximate the objective function. The surrogate function is formed based on the sampled points, and it is much cheaper to evaluate than the original objective function. The Gaussian Process is usually used to return several surrogate functions, with corresponding probability attached to each of them. An acquisition function is used to drive the proposition of new points in the search space to test. The more models we train, the more confident the surrogate will become about the next promising points to sample.

In comparing different optimisation approaches mentioned above, the manual tuning is very straightforward and easy to control, but it is not scalable and requires extra manual effort on keeping track of the trial results for running it systematically. The grid search does all possible searches exhaustively and automatically on a pre-defined configuration space. It is suitable for the case with a few hyperparameters to tune and is possible to run multiple searches concurrently since they have no inter-dependency. However, it does not work well when encountering the problem of *curse of dimensionality*. It means when the dimension of the hyperparameters increases, the search cases will grow exponentially, making the grid search prohibitively expensive. Random search is a better alternative of grid search for its capability on exploring more search space and works better in higher dimension case, but it shares a common shortcoming with grid search, that they do not utilise the previous search history for improvement. Finally, the Bayesian optimisation method provides a more refined way for the tuning with (i) using surrogate function to reduce the search cost, and (ii) accumulating the search history to continuously improve the optimisation result.

As a result, the Bayesian optimisation algorithm is believed to be the most appropriate way for obtaining the optimal set of hyperparameters of different ML models. These sets of “best” hyperparameters will then be used to build the final ML models for running the evaluation experiments.

The scikit-optimize library (Head *et al.*, 2020) is chosen for implementing the Bayesian optimisation process in our project. It works natively with models built by scikit-learn and supports tuning Keras / Tensorflow (Chollet, 2015; Abadi *et al.*, 2016) NN models with using simple wrapper class. The scikit-optimize library shares very similar features with the Grid Search feature from scikit-learn, and we can utilise the features like *k*-fold cross-validation and parallel execution when running the Bayesian optimisation.

3.1.9 Tools & libraries

In developing the ML models and carrying out the experiments in this project, Python is used as the programming language. Various supporting tools and libraries are also used as listed in Table 2.

Table 2: List of tools and libraries used in the project

Tool / Library	**Purpose**
scikit-learn (Pedregosa *et al.*, 2011)	• Implementation of decision tree models • Implementation of model pipelines • Grid search for hyperparameter tuning
Keras / Tensorflow (Chollet, 2015; Abadi *et al.*, 2016)	• Implementation of neural network models: MLP, CNN and LSTM
scikit-optimize (Head *et al.*, 2020)	• Bayesian optimisation for hyperparameter tuning
PyCharm	• Python Integrated Development Environment (IDE)
Google Colab	• Online machine learning platform for Python with GPU and TPU (Tensor processing unit) resources available • Running compute intensive process, such as Bayesian optimisation

3.2 Performance metrics

3.2.1 Pointwise accuracy

To evaluate pointwise accuracy, the R^2 score is used, it is also known as the coefficient of determination, given by:

$$R^2 = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2} \in (-\infty, 1] \qquad (6)$$

Where y is the target output, $\hat{y}$ is the model's prediction, and $\bar{y}$ is the mean of the target outputs. In traditional linear regression using original least squares (OLS), we have that $0 \leq R^2 \leq 1$, and the fraction $\sum(y - \hat{y})^2 / \sum(y - \bar{y})^2$ represents the proportion of the variance of the data remaining to be explained by a model. For non-linear regression, as is the case in machine learning, one can have negative values of R^2, and in fact there is no lower bound for this metric. The interpretation is not as direct as in OLS, but one can say that an R^2 score of 1 is optimal, and an R^2 score of 0 is as good as guessing the mean of the target values every time.

3.2.2 Statistical accuracy

We evaluate the statistical accuracy of the machine learning model with two methods: Bhattacharyya distance and quantile-quantile (Q-Q) plots. The Bhattacharyya distance measures the amount of difference between the statistical distributions of two samples. For probability distributions p and q over the same finite domain X, the Bhattacharyya distance is defined as:

$$D_B(p,q) = -\ln\left(\sum_{x\in X}\sqrt{p(x)q(x)}\right) \in [0,\infty) \qquad (7)$$

If p and q are identical (the optimal case) then $D_B(p,q) = 0$. The LLEs take real values, so we quantize the target and predicted values into 50 equally sized bins.

Q-Q plots provide a non-parametric comparison of two probability distributions by plotting their quantiles against each other, and are useful to show which parts of the target distribution are well represented by the predictions.

3.3 Finalised models

Upon completion of the hyperparameter tuning for model optimisation, the models are built using the optimal hyperparameters obtained from Bayesian optimisation, trained with full set of training dataset (also with validation dataset for DNNs models) and evaluated using the test dataset.

3.4 Experiments design and setup

3.4.1 Main experiment

The experiment involves evaluating the four supervised ML models: RT, MLP, CNN and LSTM. Three regression trees are built for predicting the three LLEs separately, while a single model of each DNN type is used to predict the three LLEs altogether. The regression tree is implemented using scikit-learn, and Keras/Tensorflow is used to construct the three DNN models. The four models will be tested on the two dynamical systems, and both the 6 time-steps and 1 time-step input will be tested separately. Therefore, there will have 16 combinations of configuration on [Dynamical system, ML model, Input time-steps] in total.

As mentioned in Section 3.1.6, the data transformer and the ML predictor are linked together in a pipeline. The formed pipeline is then used in both model training and testing during the experiment.

The models are first undergone hyperparameter tuning using Bayesian optimisation to determine the best hyperparameters to be used for the experiment. The hyperparameter search space for each ML model is defined and is passed to the **`BayesSearchCV()`** method. We specify the optimisation routine to run 100 iterations with a 5-fold cross validation for each iteration, then the mean test score is calculated from the 5-fold results for a single iteration. The optimisation method will finally return the set of hyperparameter values that achieves the maximum mean test score. Note that Bayesian optimisation will be run for each of the 16

configurations mentioned above, such that each configuration will have its own optimal hyperparameters.

In the main experiment, each combination of the configuration will be trialled 30 times, to ensure the results are statistically significant. For each trial, a separate ML model (i.e., the pipeline object) will be created, trained and tested. This approach avoids the model from accumulating the knowledge from previous trials by forcing it to be trained from scratch in each trial. So, the performance metrics can be fairly measured for all trials. The process flow of a single trial of the experiment is illustrated in Figure 5. The detailed algorithms on how the experiment iterates, as well as the computation of performance metrics can be referred to Algorithm 1 and 2 respectively.

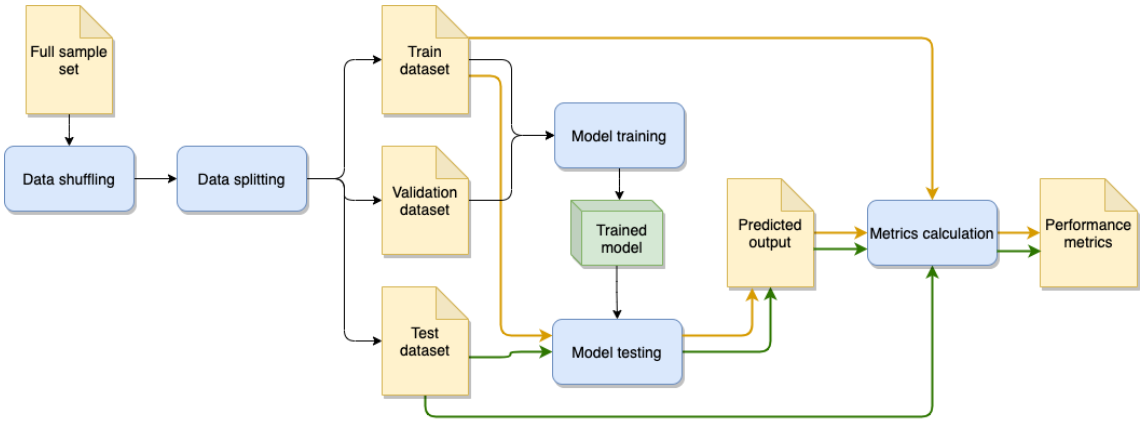

Figure 5: Process flow of a single trial of the experiment. The orange lines indicate the train dataset is required to be used for model testing (prediction) and metrics calculation for obtaining the performance metrics of train dataset. Green lines are used to distinguish the use of test dataset for similar purpose.

Algorithm 1: Main experiment

```
procedure run_experiment()
    Load full_dataset from file;
    for each dynamical_system in [Rossler, Lorenz-63]:
        for each input_feature_set in [all_time_steps, one_time_step]:
            for each instance_id in [1 to 30]:
                Shuffle full_dataset;
                Split full_dataset into train_set, validation_set, test_set;
                for each ml_model in [RT, MLP, CNN, LSTM]:
                    Create ML model pipeline object;
                    Train ML model pipeline with train_set and validation_set;
                    Perform model prediction using train_set;
                    Perform model prediction using test_set;
                    Save actual and prediction results to file;
                end for
            end for
        end for
    end for
end procedure
```

Algorithm 2: Computing the performance metrics

```
procedure compute_metrics()
    Load actual and prediction results from file;
    Groups = Get all combinations of [dynamical_system, input_feature_set, instance_id,
                Ml_model];
    for each group in Groups:
        Calculate R² scores;
        Calculate Bhattacharyya distances;
        Save metrics to file;
    end for
end procedure
```

100,000 generated samples will be used throughout the experiment. In each trial, these samples are shuffled and split into training, validation & test sets. Typically, only the test dataset is used to calculate the metrics for the ML model evaluation. However, we also compute the R^2 scores and Bhattacharyya distances of the training dataset, for they are useful for inspecting the degree of overfitting of a trained model by comparing its performance on training dataset against test dataset.

Table 3: Data splitting for the experiment

Full set sample size	100,000
Train, validation & test splitting ratio	0.6 : 0.2 : 0.2
Train dataset size	60,000
Validation dataset size (Note: used by DNN models only)	20,000
Test dataset size	20,000

Spare samples and samples for hyperparameter tuning

For the case of plotting the predicted results sequentially is required (e.g., plots for moving mean and cumulative mean of LLEs, refer to Section 4.4), a separate spare test dataset of the size of 20,000 is reserved for such purpose. For hyperparameter tuning, we will still use the same selected full set (100,000 samples), shuffle it, and feed it into the `BayesSearchCV()` routine.

3.4.2 Experiment for computation cost comparison

To compare the computation cost of deriving the LLE values using the ML models and the numerical computation method, we construct the four ML models with using all 6 time-steps as input features, execute 4,000 predictions with predicting 1 sample at a time. We measure the execution time for each of the predictions. For numerical method, acquiring the three LLE values at time t involves the following steps:

1. Generate the system states of 4 consecutive time-steps from t to $t + 3\Delta t$;
2. Use the 4 time-steps generated to compute LLE_1, LLE_2 and LLE_3 at time t.

Therefore, we use the total duration of these two steps to compare a single prediction time of the ML models. Again, 4,000 measurements will be taken for numerical method.

3.5 Summary

In this chapter, we have discussed in detailed the methodology being applied in the project. We start with formally defining the regression problem to be solved by ML, stating what ML models are selected, and the preparation, exploration and selection of input data. We adopt various techniques covering hyperparameter tuning, data pre-processing, pipelining and regularisation on neural networks, to aim for bringing the best practices on improving the quality of the experiment results. We also present the performance metrics used for evaluation and explain how they should be interpreted. Finally, the design and setup of the experiments are elaborated, such that reader can have a deeper understanding on the approach used by the experiments, as well as the rationale behind.

Chapter 4. Results & Analysis

4.1 Hyperparameter tuning

Our initial trials of using Bayesian optimisation for hyperparameter tuning took too long to complete a single case of the search of optimal hyperparameters. Eventually, the configuration of the optimisation process was adjusted (trimmed down) to the followings:

Number of iterations	: **50**
k-fold cross validation	: **5**
Number of jobs to run in parallel	: **-1** (i.e., as many as possible)
Hyperparameter search space	: Refer to Table 8

The process was also ported to and executed on Google Colab Pro+ platform (a paid version of Colab), for it provides abundant GPU and TPU (Tensor Processing Unit) resources for optimising DNN models. The durations of running the hyperparameter tuning are detailed in Table 9. The total execution time spent was over 102 hours.

The optimal hyperparameters obtained for all combinations on [ML model, System, No. of time-steps] can be found in Section B.3.

The Bayesian optimisation process can be shown being effective in finding optimal hyperparameter values by referring to the plots in Section A.1. Those plots illustrate the convergence of the maximum test score of the ML model in all cases as iterations move along. Examples extracted as in Figure 6 show the described trend of the maximum test scores appears in all ML models for Rössler system with all time-steps input. Particularly, CNN quickly converges to the near-optimal test score only after 4 iterations.

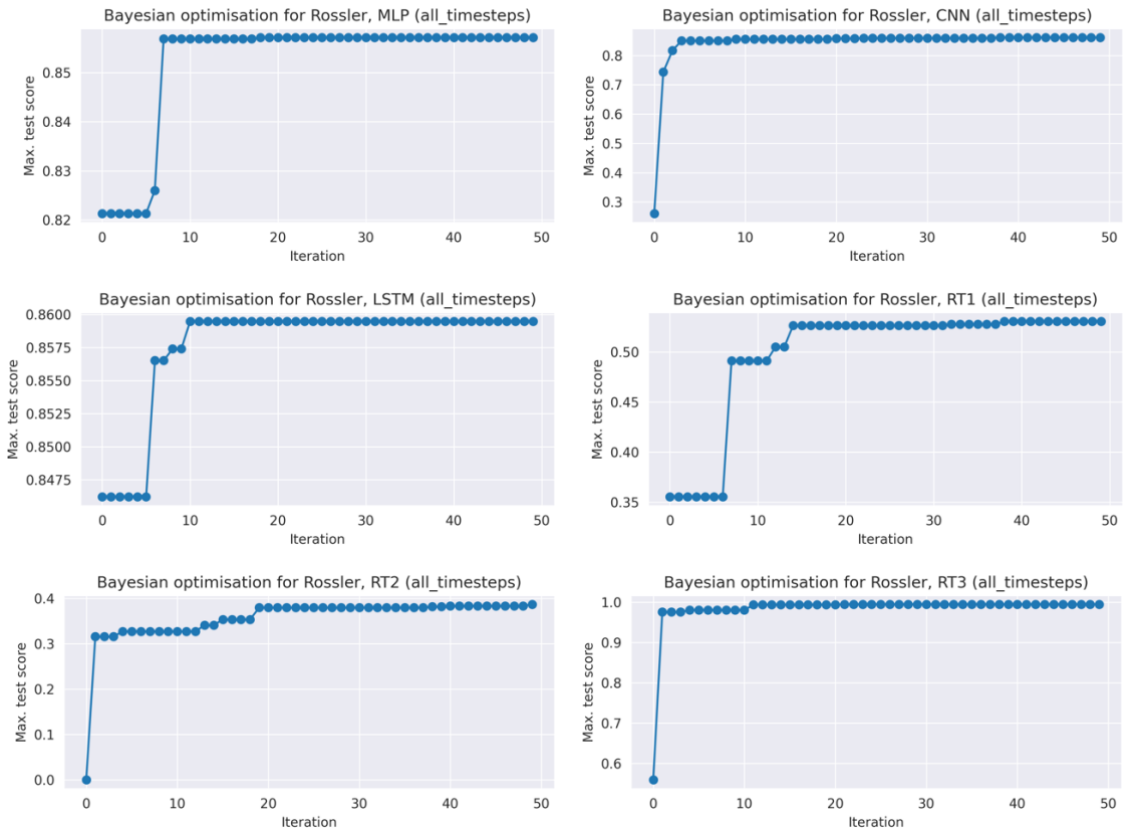

Figure 6: Maximum test score vs. iterations of Bayesian optimisation for different ML models (Rössler, All time-steps)

4.2 Performance metrics of experiment results

When reviewing R^2 scores of predicting the LLEs calculated from the 30-trial experiments for the two dynamical systems, two input feature sets and on four ML models in Table 4 and Table 6, we see that:

LLE$_1$: reasonably good prediction with mean R^2 scores of 0.47 to 0.54 for Rössler system, well predicted for Lorenz-63 with at least 0.84 of mean R^2 scores (except LSTM).

LLE$_2$: only with moderately good prediction with scores from 0.29 to 0.39 for Rössler system, reasonably well predicted for Lorenz-63 with a range of 0.44 to 0.76 of mean R^2 scores but the LSTM case.

LLE$_3$: very well predicted with all scores being over 0.93.

The Bhattacharyya distances, in contrast, the smaller the better. Referring to Table 5 and Table 7, it is noticed that all ML models achieve good statistical accuracy generally. As expected, the Bhattacharyya distances follow the behaviour of R^2 scores: the best are of predicting LLE$_3$, followed by LLE$_1$, then LLE$_2$. However, unlike R^2 scores, there is no apparent better statistical accuracies of the Lorenz-63 system over those of the Rössler; in the case of CNN, Lorenz-63 outperforms obviously, but in another case like LSTM, the reverse holds.

Table 4: R^2 scores of the experiment results for Rössler system. Each entry shows the mean of the 30 trials, with the corresponding standard deviation in parentheses.

Rössler R^2 scores

	All time-steps			One time-step		
	LLE_1	LLE_2	LLE_3	LLE_1	LLE_2	LLE_3
CNN	0.5054 (0.0349)	0.3530 (0.0419)	0.9956 (0.0044)	0.5279 (0.0333)	0.3711 (0.0518)	0.9960 (0.0040)
LSTM	0.5319 (0.0462)	0.3788 (0.0571)	0.9955 (0.0023)	0.4657 (0.0633)	0.2921 (0.0769)	0.9869 (0.0074)
MLP	0.5363 (0.0243)	0.3897 (0.0274)	0.9975 (0.0006)	0.5323 (0.0211)	0.3837 (0.0249)	0.9978 (0.0005)
RT	0.5161 (0.0268)	0.3681 (0.0278)	0.9946 (0.0002)	0.5155 (0.0248)	0.3506 (0.0299)	0.9944 (0.0002)

Table 5: Bhattacharyya distances of the experiment results for Rössler system. Each entry shows the mean of the 30 trials, with the corresponding standard deviation in parentheses.

Rössler Bhattacharyya distances

	All time-steps			One time-step		
	LLE_1	LLE_2	LLE_3	LLE_1	LLE_2	LLE_3
CNN	0.0180 (0.0140)	0.0333 (0.0164)	0.0029 (0.0034)	0.0146 (0.0129)	0.0275 (0.0188)	0.0022 (0.0022)
LSTM	0.0141 (0.0122)	0.0194 (0.0080)	0.0022 (0.0019)	0.0179 (0.0120)	0.0360 (0.0168)	0.0061 (0.0045)
MLP	0.0151 (0.0126)	0.0227 (0.0118)	0.0013 (0.0006)	0.0141 (0.0098)	0.0235 (0.0108)	0.0010 (0.0005)
RT	0.0231 (0.0292)	0.0333 (0.0241)	0.0875 (0.0099)	0.0209 (0.0223)	0.0303 (0.0205)	0.1097 (0.0130)

Table 6: R^2 scores of the experiment results for Lorenz-63 system. Each entry shows the mean of the 30 trials, with the corresponding standard deviation in parentheses.

Lorenz-63 R^2 scores

	All time-steps			One time-step		
	LLE_1	LLE_2	LLE_3	LLE_1	LLE_2	LLE_3
CNN	0.9304 (0.0047)	0.7613 (0.0159)	0.9993 (0.0002)	0.9169 (0.0081)	0.7153 (0.0261)	0.9992 (0.0005)
LSTM	0.7594 (0.0302)	0.2933 (0.0803)	0.9838 (0.0028)	0.7404 (0.0037)	0.2545 (0.0088)	0.9766 (0.0008)
MLP	0.8350 (0.0438)	0.4449 (0.1446)	0.9925 (0.0033)	0.9217 (0.0038)	0.7325 (0.0123)	0.9993 (0.0003)
RT	0.8801 (0.0027)	0.6540 (0.0053)	0.9936 (0.0003)	0.8672 (0.0025)	0.6166 (0.0046)	0.9324 (0.0305)

Table 7: Bhattacharyya distances of the experiment results for Lorenz-63 system. Each entry shows the mean of the 30 trials, with the corresponding standard deviation in parentheses.

Lorenz-63 Bhattacharyya distances

	All time-steps			One time-step		
	LLE_1	LLE_2	LLE_3	LLE_1	LLE_2	LLE_3
CNN	0.0050 (0.0016)	0.0333 (0.0055)	0.0005 (0.0002)	0.0080 (0.0028)	0.0399 (0.0079)	0.0005 (0.0002)
LSTM	0.0476 (0.0082)	0.1527 (0.0245)	0.0061 (0.0013)	0.0647 (0.0029)	0.2498 (0.0129)	0.0127 (0.0018)
MLP	0.0243 (0.0095)	0.1526 (0.0541)	0.0031 (0.0015)	0.0069 (0.0019)	0.0380 (0.0064)	0.0006 (0.0004)
RT	0.1200 (0.0163)	0.1540 (0.0178)	0.0451 (0.0152)	0.1446 (0.0171)	0.1735 (0.0189)	0.1591 (0.0391)

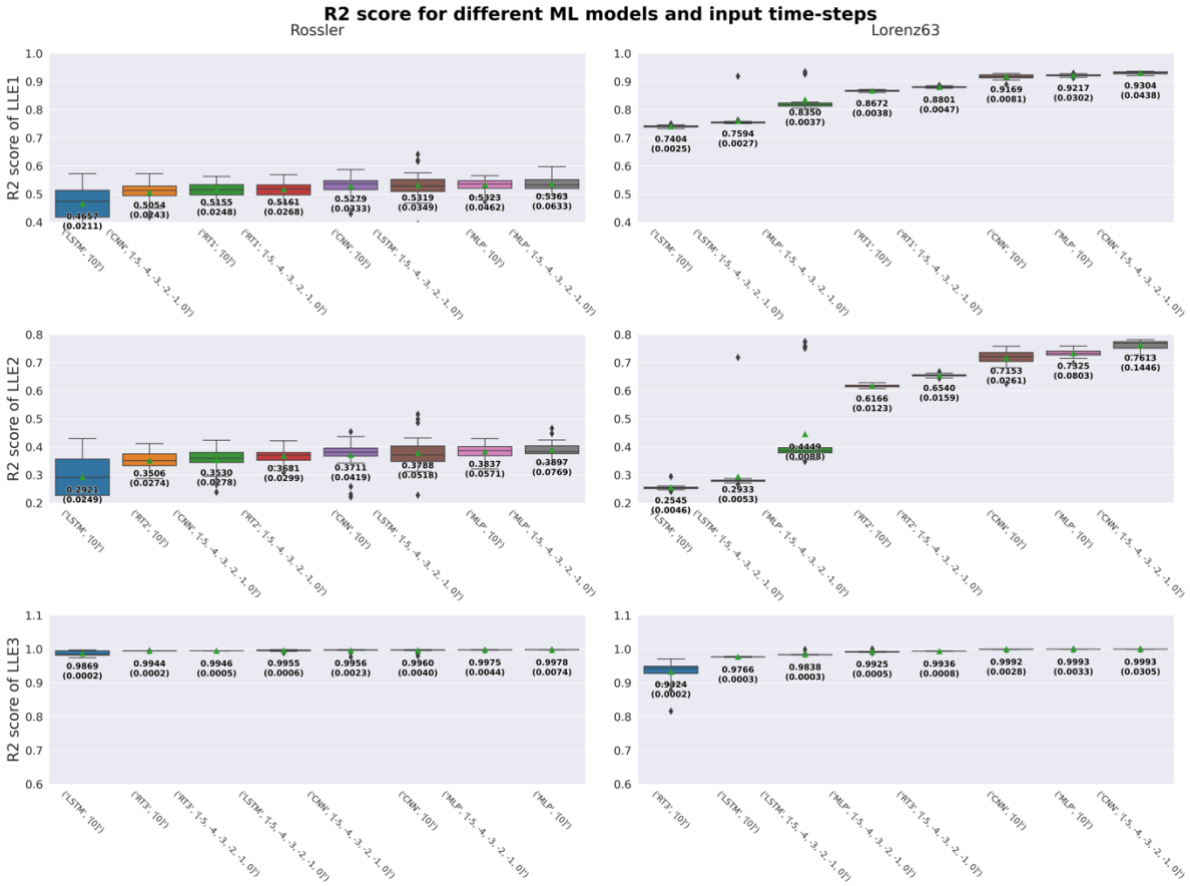

Figure 7: Boxplot of R^2 scores for different combinations of ML models with input feature sets. Each row shows results of different LLEs and the two columns show results of two dynamical systems. The green triangles show the mean scores of the test datasets of the 30 trials. Each box is labelled with the mean and standard deviation (in parentheses) of R^2 scores in bold font.

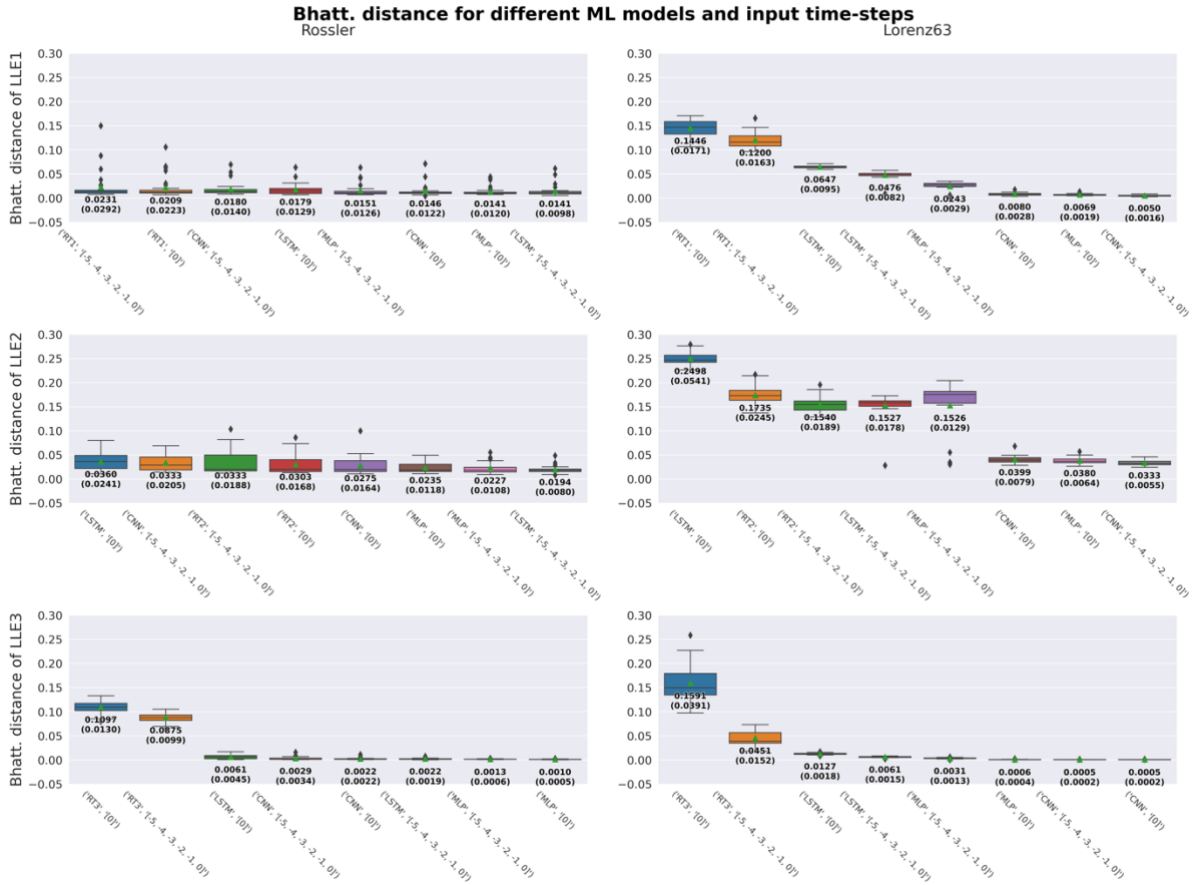

Figure 8: Boxplot of Bhattacharyya distances for different combinations of ML models with input feature sets. Each row shows results of different LLEs and the two columns show results of two dynamical systems. The green triangles show the mean scores of the test datasets of the 30 trials. Each box is labelled with the mean and standard deviation (in parentheses) of Bhattacharyya distances in bold font.

4.3 Q-Q plots

The Q-Q plots help compare the statistical distribution of predicted against actual LLE values visually. The Q-Q plots of the 3 LLEs of both training and test datasets, for all combinations of chaotic systems, ML models and input feature sets, can be found in Section A.2. For LLE_1 and LLE_2 of the Rössler system, the predictions do well when the target value is close to the mean, but they fail to represent the more extreme values. Those extreme values have tendency of having smaller predicted values than their actual ones, which results in a S-shape shown in the Q-Q plots. LLE_3 has a very well predicted outcome, however. The blue curves in their plots, showing the relationship between predicted and actual LLE values, overlap with the ideal scenarios (red dash lines). In Lorenz-63 system, the overall predictions can be seen better for LLE_1 and LLE_2, except LSTM struggles to represent the positive extremes of LLE_2.

When comparing the Q-Q plots on test dataset with their corresponding plots on training dataset, we find their statistical accuracies are very similar. This tells us that the ML models perform more or less equally well on training data and test data, meaning the models do not exhibit overfitting on training data. In other words, their generalisation levels are high.

The Q-Q plots for RT looks like a stepping function (refer to Figure 9), it is due to the node splitting nature of RT during training, multiple predicted outputs are allocated to the same leaf node, meaning that they share the same predicted values. While the neural networks can reproduce a smoother output curve because each output is independent to each other. Also, the networks are more flexible to reconstruct the regression function by using the weights, biases and activation functions on a large number of neurons.

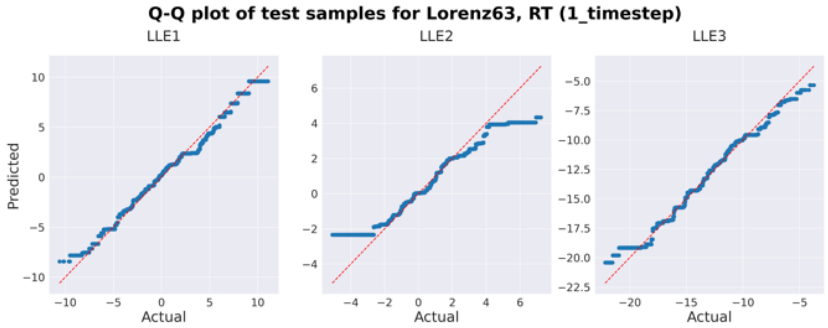

Figure 9: Stepping function appearance of Q-Q plot for regression tree

4.4 Cumulative mean & moving mean plots

We also present the plot of the cumulative mean of LLEs against 1,000 sequential time-steps of the best performing [ML model, Input feature] case for Rössler and Lorenz-63, that is [MLP, all time-steps] (Figure 10) and [CNN, all time-steps] (Figure 11) respectively. The blue lines in the plots represent the actual LLE values and the orange lines the predicted ones. The two thick lines tell us that the cumulative average of the predicted and actual LLE values are very close to each other throughout the sequence. But those thin lines representing the LLE values, show the predictions fail to represent the extreme values of LLE_1 and LLE_2. Such pattern can be seen in both dynamical systems. This observation aligns with what the Q-Q plots have revealed.

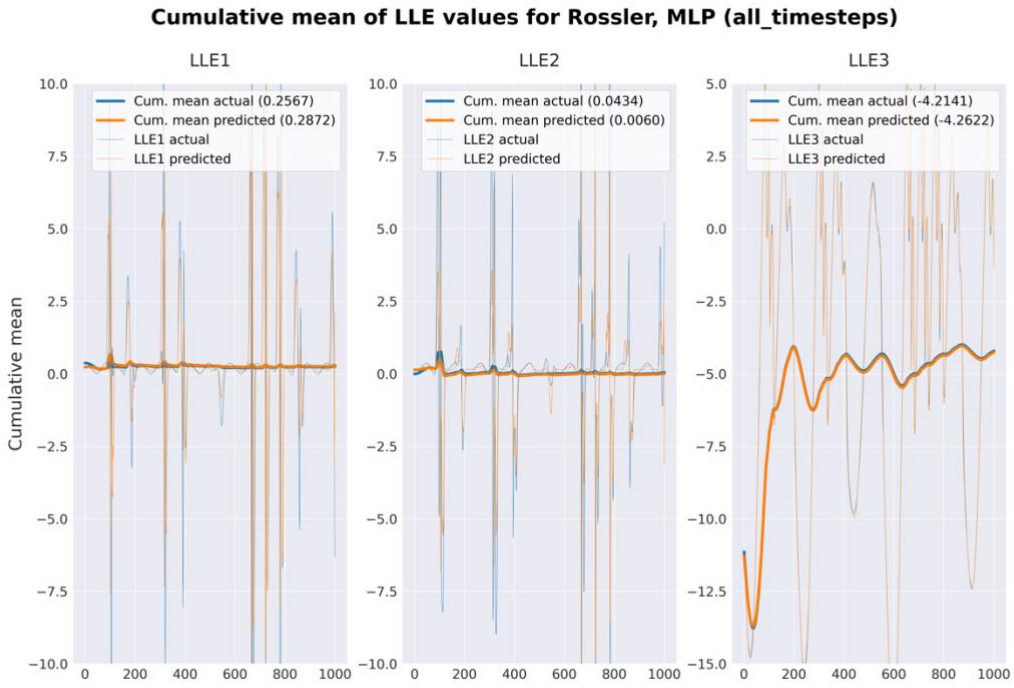

Figure 10: Cumulative mean plot of LLE values for the best performing case in Rössler (MLP, all time-steps)

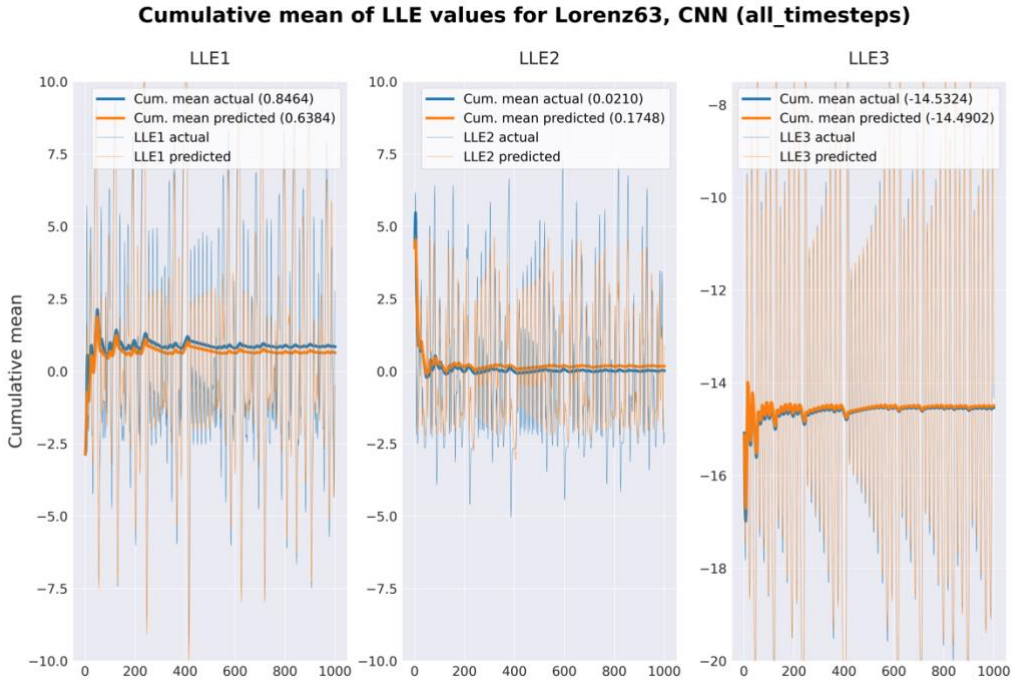

Figure 11: Cumulative mean plot of LLE values for the best performing case in Lorenz-63 (CNN, all time-steps)

Figure 12 and Figure 13 are the moving mean plots of LLE values of the same two best performing cases used for LLE cumulative mean plotting. The moving mean of the LLEs at a specific time-step, presented by the corresponding thick line, is calculated by taking the mean

value of LLE values from the previous 50 time-steps to the subsequent 50 time-steps. These plots give another perspective in visualising that the predictions resemble the actual values well in a local region. Moreover, it is clearer that the predicted values cannot represent the actual extremes by examining the thick lines in LLE_1 and LLE_2.

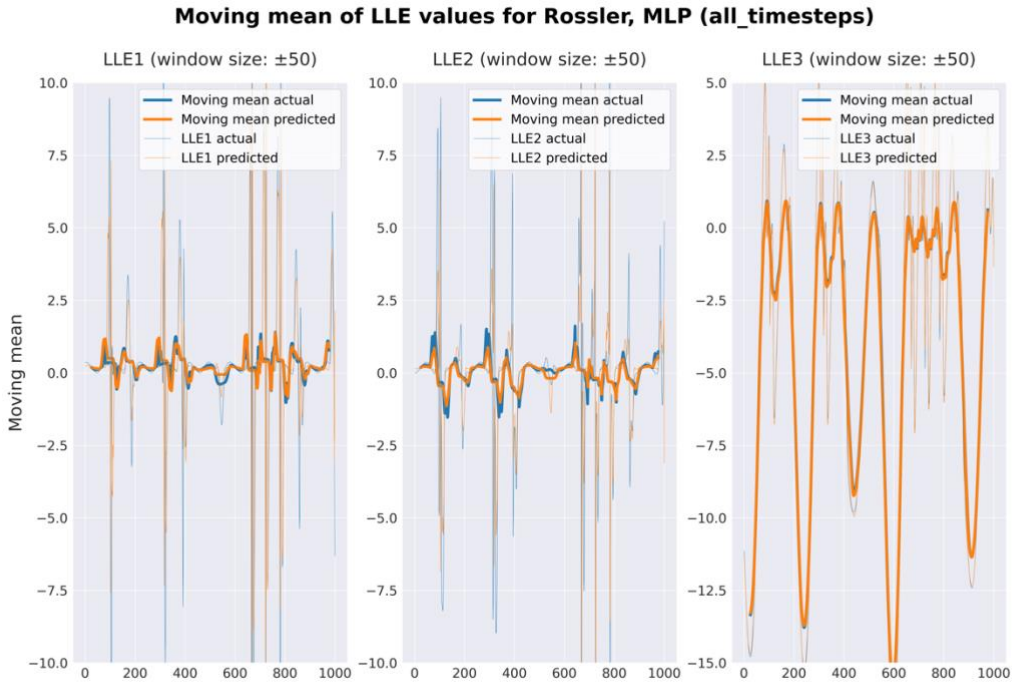

Figure 12: Moving mean plot of LLE values for the best performing case in Rössler (MLP, all time-steps)

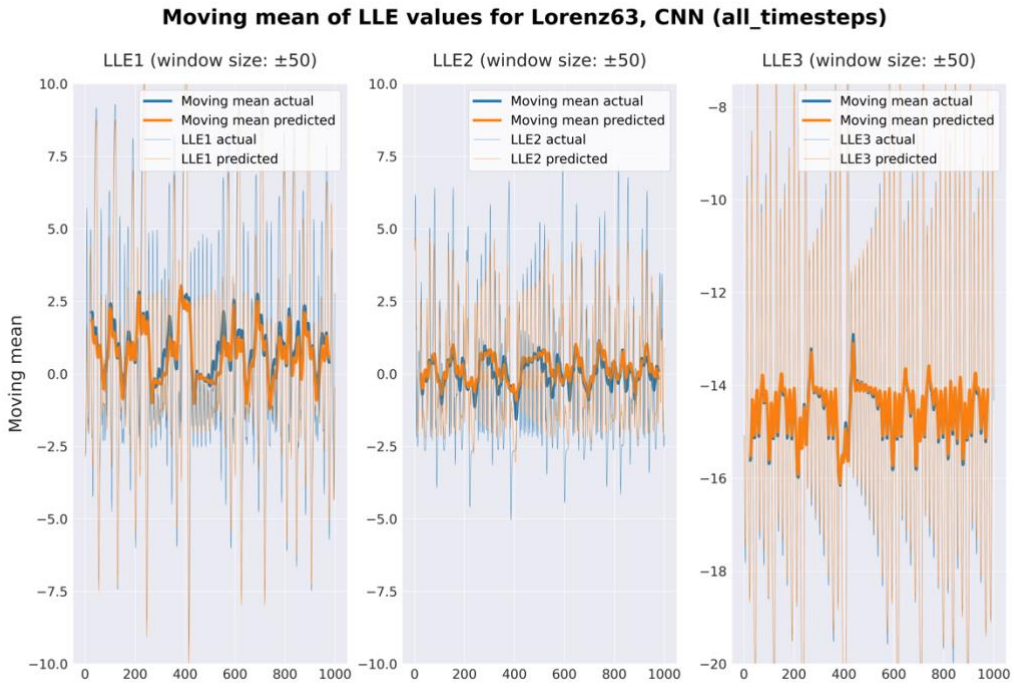

Figure 13: Moving mean plot of LLE values for the best performing case in Lorenz-63 (CNN, all time-steps)

4.5 Computation time plots for different ML algorithms

The experiment for measuring the execution time of LLE acquisition (by either prediction or computation), using ML models and numerical computation algorithm, was performed in an Apple MacBook Pro laptop with M1 CPU. The wall clock durations were measured for each execution.

As shown in Figure 14, the average execution time for predicting 3 LLE values is about 0.23 second for CNN, MLP and LSTM, while that for RT is around 0.0001 second. With using numerical computation method to calculate the system states of 4 time-steps, and then the values of LLE_1, LLE_2 and LLE_3, it takes 0.0002 second for Rössler and 0.001 second for Lorenz-63. It can be concluded that RT is the fastest algorithm, it is 200 times faster than all neural network models. When comparing with the numerical computation method, RT doubles the speed in Rössler and is an order of magnitude faster in Lorenz-63. DNNs are the slowest algorithms among all.

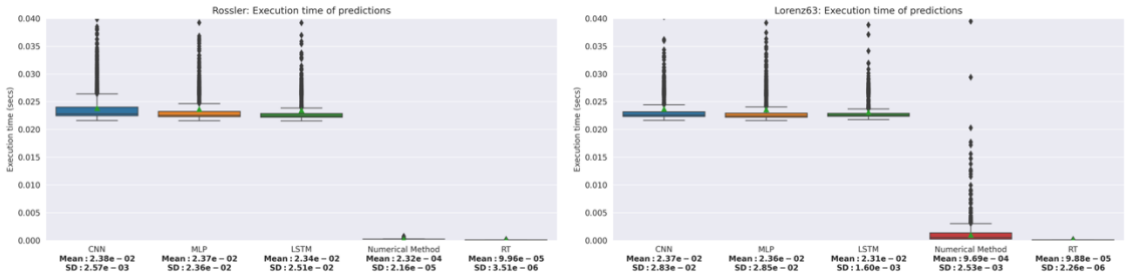

Figure 14: Boxplots of prediction execution times of ML algorithms and numerical computation method to obtain LLEs for Rössler (left) and Lorenz-63 (right)

4.6 Summary

In this chapter, the results of hyperparameter tuning are reviewed. The convergence nature of the maximum test scores is discovered, showing the effectiveness of the Bayesian optimisation algorithm. The mean R^2 scores and Bhattacharyya distances of different ML models on the two chaotic systems and two sets of input time-steps are compared, leading to the finding of LLE_3 being predicted superbly, while LLE_1 being reasonably well and LLE_2 only predicted fairly. We also present the Q-Q plots that show how plausible the predicted targets can represent the actual values visually. Moreover, the plots of cumulative mean and moving mean of LLEs provide another perspective on the predictability of the ML models, which is found to be consistent with the Q-Q plots. Finally, the computation time of the ML models against the traditional numerical computation method on deducing the LLE values are compared, which helps us realise the fact that RT is the fastest ML algorithm in our problem, that can even beat the numerical method.

Chapter 5. Discussion

5.1 Discussion

Firstly, from analysing the optimal hyperparameters of all cases, it is observed that for DNNs, they often perform better with slower learning rate, more hidden layers, more neurons or units per layer, and with the use of regularisers. For regression trees, growing deeper trees, using "best split" strategy on node splitting, and having large leaf nodes are generally good for model performance. The results match with the conventional understanding of ML models that larger trees, or bigger and deeper neural networks usually have greater learning power.

We also see from the experiments' results that all supervised ML algorithms achieve similar prediction performance. They get very high R^2 scores for predicting LLE_3, to reasonably good for LLE_1, and then moderately well for LLE_2; In statistical point of view, their performances are also quite consistent with the pointwise accuracies. However, regression trees are obviously distinguished from others by their speed of prediction.

Several patterns come into view:

- The pointwise accuracy of LLE_3 predictions is higher than that of LLE_1, followed by LLE_2
- The pointwise accuracy of LLE prediction the Lorenz-63 system is higher than that of the Rössler system
- Using 6 input time-steps does not contribute significant improvement to the ML models when compared with 1 input time-step
- All models exhibit high level of generalisation on prediction

The first two patterns might be explained by the local heterogeneity of LLEs in phase space of the dynamical systems. The local mixing of LLE_3 is low, and it is higher in the other two LLEs. Likewise, Lorenz-63 exhibits greater degree of homogeneity locally than Rössler. Consequently, in the regions with very low local heterogeneity, the predictions achieve good accuracy.

On the other hand, we expect providing more spatial-temporal data to the ML predictors will boost their capabilities, particularly with a few near historic time-steps. However, the experiment results do not follow our anticipation. We suspect that the relationship between the sequential trajectory steps and the LLE values of the chaotic systems is not strong. This argument could be complemented by the fact that the LSTM model, which is reputably good at modelling sequential data, does not produce results standing out from other ML algorithms used.

Having discovered that all trained models can achieve good generalisation, we are confident that the measures we applied to combat overfitting (e.g., data splitting, using regulariser and early stopping in DNNs) are effective.

5.2 Significance of the findings

We have shown that supervised learning methods can predict LLEs upon input of the recent system trajectory. We suggest that the ML performance is determined by the degree of heterogeneity of the LLE values. Specific ML model is proven to be a computational cost saving alternative of the traditional numerical algorithms. Our findings are Interesting for

weather prediction where previously the computational cost of calculating LLEs prevented them being used in NWP.

5.3 Limitations

Our project cannot give a comprehensive picture of how well the supervised ML can address our problem, due to our scope is unable to cover the entire spectrum of supervised ML algorithms. Similarly, we cannot explore every possible configuration in our ML modelling.
In addition, the learning for the approximation is dependent on the temporal resolution and the numerical accuracy of input data.
The hyperparameter tuning for neural networks takes very long time to complete, even using the most efficient Bayesian optimisation method. The final hyperparameter search space on the tuning is a trimmed down version of our original one, in order to allow the process to complete in a reasonable time. Therefore, the obtained "best" hyperparameters may not be the global optima.

5.4 Summary

We come to a more in-depth discussion in this chapter, following the review and analysis of the experiment results in the previous section. For the optimal hyperparameters, several patterns on the optimal values are observed, and they align with our general understanding of the ML models. We also try to explain the behaviour on the performance of the ML models. The importance of our findings is shared, and it is believed that our results will have positive impact to the research community. Lastly, some identified limitations are also highlighted to wrap up this chapter.

Chapter 6. Conclusions and Future Work

6.1 Conclusions

In the project, we have studied the characteristics of the computation demanding nature of the numerical weather prediction systems. It is also learned that machine learning techniques are adopted in a broad range of applications related to weather forecasting. Four supervised machine learning algorithms are proposed to solve our problem of predicting the local Lyapunov exponents of the two chaotic systems, namely Rössler and Lorenz-63. The experiments for evaluating the performance of the machine learning models are designed and carried out. The prediction performance of the ML models is measured quantitatively by using R^2 scores and Bhattacharyya distances from the perspective of pointwise accuracy and statistical validity respectively. Different analysis techniques such as Q-Q plots, boxplots, cumulative means and moving means plots are used to appraise the models qualitatively. The impact on the computation cost on using ML algorithms is assessed by comparing the average execution time used for ML predictions and the calculation time of LLE values using numerical computation method.

The results demonstrated the selected supervised ML models have satisfactory performance in both pointwise and statistical accuracies on two low-dimensional systems. In terms of the saving of computation expenses, we showed that the regression tree is particularly suitable for fulfilling this objective, which is 200 times faster than the neural network counterparts, and is also 2 to 10 times faster than using numerical computation algorithm. With further analysing the results, we also discovered that the prediction accuracy of the ML model is influenced by the local heterogeneity of the LLE values.

During the life cycle of ML models development, a number of ML best practices and techniques are successfully applied, including hyperparameter tuning, data splitting, data scaling, pipelining, regularisation for deep neural networks and etc.

Consequently, the goal of using supervised machine learning for LLE prediction is achieved. It is equally significant that the possibility of reducing computation cost by applying ML to the targeted problem is verified. Such accomplishments encourage us to believe that the way for carrying on further relevant research is paved.

6.2 Future work

There are a few ideas that could be considered as the ongoing research:

1. Climate systems are much more complex and have way larger dimensions than what we covered in this project. Very often in such systems, not all variables can be "observed" and are available as input. It will be interesting if we explore the impact to the performance of the ML models on providing only partial observations as the input features. Typical approach is trying to include merely a subset of the system state components, for instance, using (x, y) instead of (x, y, z) in each input time-step.

2. Logically, we should not just focus on low dimensional chaotic systems. To move one step forward for catering the realistic weather forecasting problem, we suggest

evaluating spatially extended systems, e.g., partial differential equation (PDE) systems. In particular, studying the Lorenz 96 system (Lorenz, 1996) and the Kuramoto-Sivashinski system (Kuramoto, 1978; Sivashinsky, 1977) are suggested. The recommended systems to be studies remain relatively simple, for they are either an ODE system with configurable number of variables, or just a fourth order PDE system. Nevertheless, we will enter into another stage of modelling chaotic systems with studying them, and will steer our work in the right direction towards very high dimensional systems.

3. In reality, annotated inputs may not be obtained easily, or it is too expensive for the process of data labelling in terms of time and monetary cost. It is therefore sensible to also explore the unsupervised ML approach. In this case, unstable weather events are treated as "abnormal" behaviours of the model, and unsupervised ML algorithm can be used to identify these anomalies automatically, without the need of labelled input data.

Chapter 7. Reflection

I found undertaking this project is a challenging experience. Firstly, applying ML into weather forecasting is a very appealing topic for me. However, it involves a lot of new concepts, especially the idea of dynamical systems. This area requires substantial knowledge on the mathematical models, as well as the understanding of the numerical computation algorithms. It is a bit overwhelming to me. Fortunately, my project partner Daniel, who is a PhD student in Meteorology, did contribute a lot on the research and study in this area. He also helped me understand most of the fundamental concepts, which are crucial and relevant in designing and implementing the experiments, as well as analysing and interpreting the results.

We have spent a tremendous amount of time on refining and repeating the experiments to evaluate the ML models' performance on the two dynamical systems. The complexity of the execution and the numerous technical issues encountered were out of our expectation. Therefore, in the late stage of the project, we decided to trim down our original scope, such that the evaluation of more complicated systems, and the exploration of the unsupervised learning approach are put into the list of suggested future works. However, the effort we spent to focusing on a smaller scope pays off: we now have gained a pretty solid experience on how a whole ML application should be implemented. We believe things will be much smoother if we were to do the project implementation again, even involving more complicated systems.

Furthermore, we found the process of hyperparameter tuning is particularly daunting. It will take impracticably long to complete if we specify a large search space albeit using the most efficient Bayesian optimisation. We finally need to compromise to restrict the hyperparameter space to a smaller one, especially for LSTM model. It remains a doubt of how optimised the models we have achieved comparing with the global supreme. We can foresee this is one of the hurdles to be overcome in real-life NWP use cases where the scale of the problem drastically increases.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. and Ghemawat, S., 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Ayers, D., Lau, J., Amezcua, J., Carrassi, A., Ojha, V., 2021. Supervised learning to estimate local dynamical instabilities in chaotic systems: computation of local lyapunov exponents. *Quarterly Journal of the Royal Meteorological Society*, 2021. In preparation.

Bergstra, J. and Bengio, Y., 2012. Random search for hyperparameter optimization. *Journal of machine learning research*, *13*(2).

Bonavita, M. and Laloyaux, P., 2020. Machine learning for model error inference and correction. *Journal of Advances in Modeling Earth Systems*, *12*(12), p.e2020MS002232.

Brajard, J., Carrassi, A., Bocquet, M. and Bertino, L., 2020. Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model. *Journal of Computational Science*, *44*, p.101171.

Brajard, J., Carrassi, A., Bocquet, M. and Bertino, L., 2021. Combining data assimilation and machine learning to infer unresolved scale parametrization. *Philosophical Transactions of the Royal Society A*, *379*(2194), p.20200086.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J., 2017. *Classification and regression trees*. Routledge.

Carrassi, A., Bocquet, M., Demaeyer, J., Grudzien, C., Raanes, P. and Vannitsem, S., 2020. Data assimilation for chaotic dynamics. *arXiv preprint arXiv:2010.07063*.

Chantry, M., Christensen, H., Dueben, P. and Palmer, T., 2021. Opportunities and challenges for machine learning in weather and climate modelling: hard, medium and soft AI.

Chollet F. *et al.* (2015) *Keras*. Available at https://keras.io (Accessed: 30 August 2021)

Düben, P., Modigliani, U., Geer, A., Siemen, S., Pappenberger, F., Bauer, P., Brown, A., Palkovic, M., Raoult, B., Wedi, N. and Baousis, V., 2021. Machine learning at ECMWF: A roadmap for the next 10 years. *ECMWF Technical Memoranda*, (878).

Epperson, J.F., 2021. *An introduction to numerical methods and analysis*. John Wiley & Sons, pp.413-419

Fuhrer, O., Chadha, T., Hoefler, T., Kwasniewski, G., Lapillonne, X., Leutwyler, D., Lüthi, D., Osuna, C., Schär, C., Schulthess, T.C. and Vogt, H., 2018. Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0. *Geoscientific Model Development*, *11*(4), pp.1665-1681.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT press.

Gottwald, G.A. and Reich, S., 2021. Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation. *Physica D: Nonlinear Phenomena*, *423*, p.132911.

Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R. and Schmidhuber, J. (2016) LSTM: A search space odyssey.*IEEE transactions on neural networks and learning systems*, *28*(10), pp.2222-2232.

Haykin, Simon. *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

Head, Tim, Kumar, Manoj, Nahrstaedt, Holger, Louppe, Gilles, & Shcherbatyi, Iaroslav. (2020). scikit-optimize/scikit-optimize (v0.8.1). Zenodo. https://doi.org/10.5281/zenodo.4014775

Hewson, T.D. and Pillosu, F.M., 2020. A new low-cost technique improves weather forecasts across the world. *arXiv preprint arXiv:2003.14397*.

Hochreiter, S. and Schmidhuber, J. (1997) Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780.

Holmstrom, M., Liu, D. and Vo, C., 2016. Machine learning applied to weather forecasting. *Meteorol. Appl*, pp.1-5.

Kuramoto, Y., 1978. Diffusion-induced chaos in reaction systems. *Progress of Theoretical Physics Supplement*, *64*, pp.346-367.

LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *nature*, *521*(7553), pp.436-444.

LeCun, Y., Haffner, P., Bottou, L. and Bengio, Y., 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision* (pp. 319-345). Springer, Berlin, Heidelberg.

Lim, B. and Zohren, S., 2021. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, *379*(2194), p.20200209.

Lorenz, E.N., 1963. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, *20*(2), pp.130-141.

Lorenz, E.N., 1996, September. Predictability: A problem partly solved. In *Proc. Seminar on predictability* (Vol. 1, No. 1).

Mockus, J., 2012. Bayesian approach to global optimization: theory and applications (Vol. 37). Springer Science & Business Media.

Murugan Bhagavathi, S., Thavasimuthu, A., Murugesan, A., George Rajendran, C.P.L., Raja, L. and Thavasimuthu, R., 2021. Weather forecasting and prediction using hybrid C5. 0 machine learning algorithm. *International Journal of Communication Systems*, *34*(10), p.e4805.

Nguyen, D., Ouala, S., Drumetz, L. and Fablet, R., 2020. Variational Deep Learning for the Identification and Reconstruction of Chaotic and Stochastic Dynamical Systems from Noisy and Partial Observations. *arXiv preprint arXiv:2009.02296*.

O'Gorman, P.A. and Dwyer, J.G., 2018. Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *Journal of Advances in Modeling Earth Systems*, *10*(10), pp.2548-2563.

Palatella, L., Carrassi, A. and Trevisan, A., 2013. Lyapunov vectors and assimilation in the unstable subspace: theory and applications. *Journal of Physics A: Mathematical and Theoretical*, *46*(25), p.254020.

Pathak, J., Lu, Z., Hunt, B.R., Girvan, M. and Ott, E., 2017. Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *27*(12), p.121102.

Pathak, J., Hunt, B., Girvan, M., Lu, Z. and Ott, E., 2018. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical review letters*, *120*(2), p.024102.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, pp.2825-2830.

Pikovsky, A. and Politi, A., 2016. *Lyapunov exponents: a tool to explore complex dynamics*. Cambridge University Press.

Rössler, O.E., 1976. An equation for continuous chaos. *Physics Letters A*, *57*(5), pp.397-398.

Schultz, M.G., Betancourt, C., Gong, B., Kleinert, F., Langguth, M., Leufen, L.H., Mozaffari, A. and Stadtler, S., 2021. Can deep learning beat numerical weather prediction?. *Philosophical Transactions of the Royal Society A*, *379*(2194), p.20200097.

Sivashinsky, G.I., 1977. Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations. *Acta astronautica*, *4*(11), pp.1177-1206.

Sprott, J.C., 2003. *Chaos and time-series analysis* (Vol. 69). Oxford: Oxford university press.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), pp.1929-1958.

TOP500 (2021) *June 2021 | TOP500*. Available at https://www.top500.org/lists/top500/2021/06/ [Accessed 4 September 2021]

Toth, Z. and Kalnay, E., 1997. Ensemble forecasting at NCEP and the breeding method. *Monthly Weather Review*, *125*(12), pp.3297-3319.

Wedi, N.P., Polichtchouk, I., Dueben, P., Anantharaj, V.G., Bauer, P., Boussetta, S., Browne, P., Deconinck, W., Gaudin, W., Hadade, I. and Hatfield, S. (2020) A baseline for global weather and climate simulations at 1 km resolution. *Journal of Advances in Modeling Earth Systems*, *12*(11), p.e2020MS002192.

Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Philip, S.Y. and Zhou, Z.H., 2008. Top 10 algorithms in data mining. *Knowledge and information systems*, *14*(1), pp.1-37.

Zhao, B., Lu, H., Chen, S., Liu, J. and Wu, D., 2017. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, *28*(1), pp.162-169.

Appendix A. Plots

A.1 Bayesian optimisation best test score plots

A.1.1 Rössler

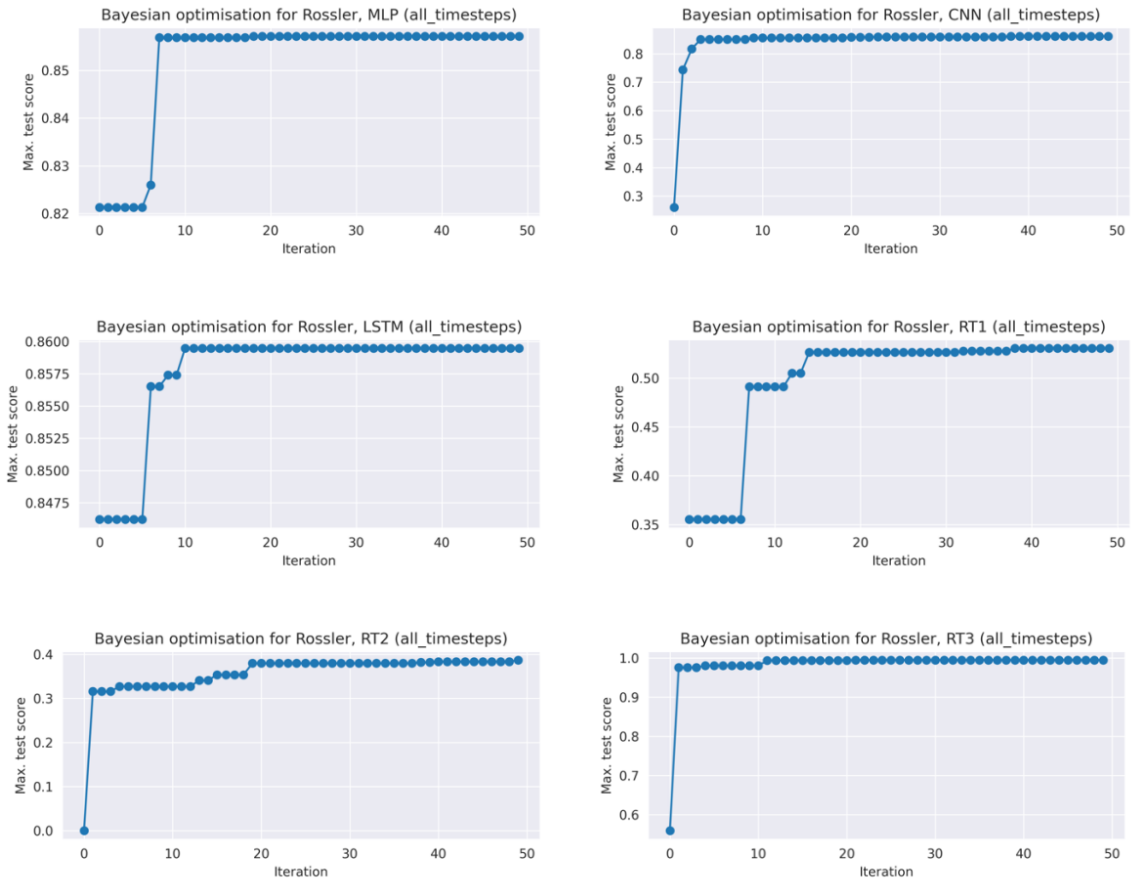

Figure 15: Maximum test score vs. Bayesian optimisation iterations for different ML models (Rössler, All time-steps)

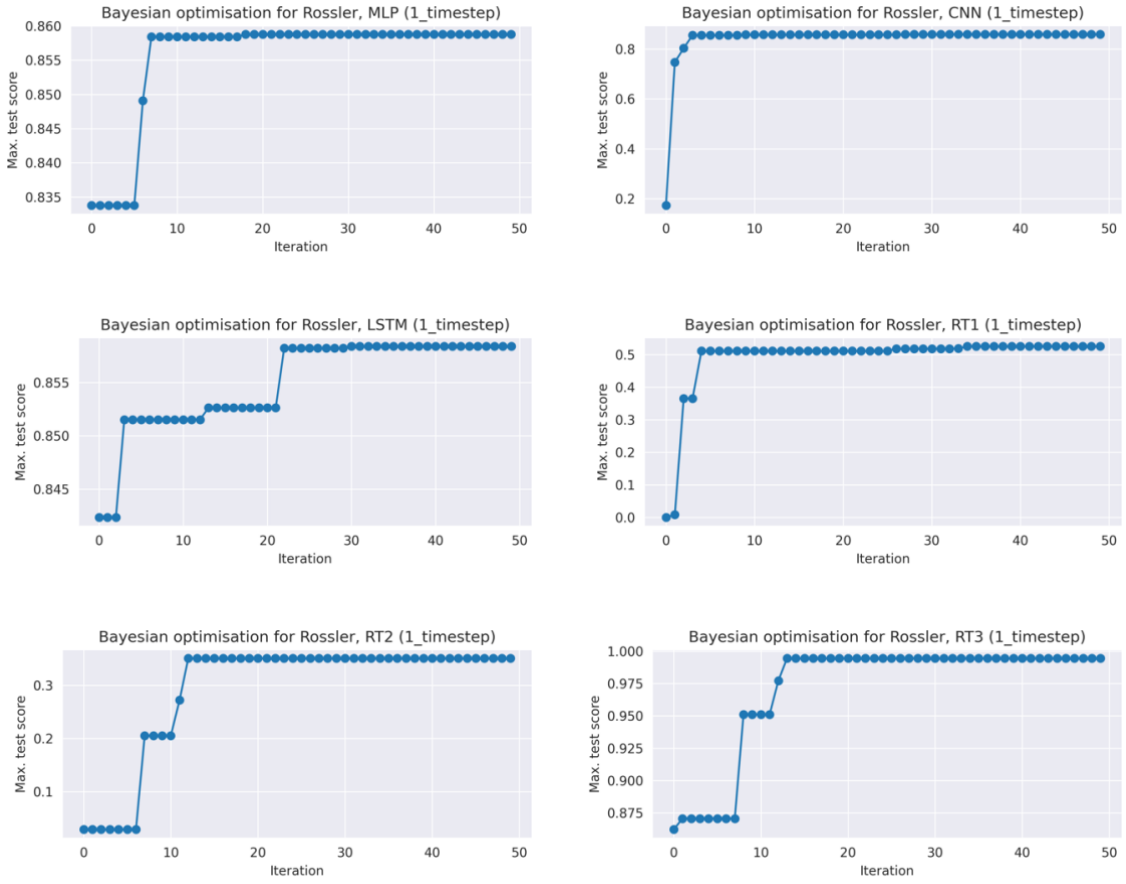

Figure 16: Maximum test score vs. Bayesian optimisation iterations for different ML models (Rössler, One time-step)

A.1.2 Lorenz-63

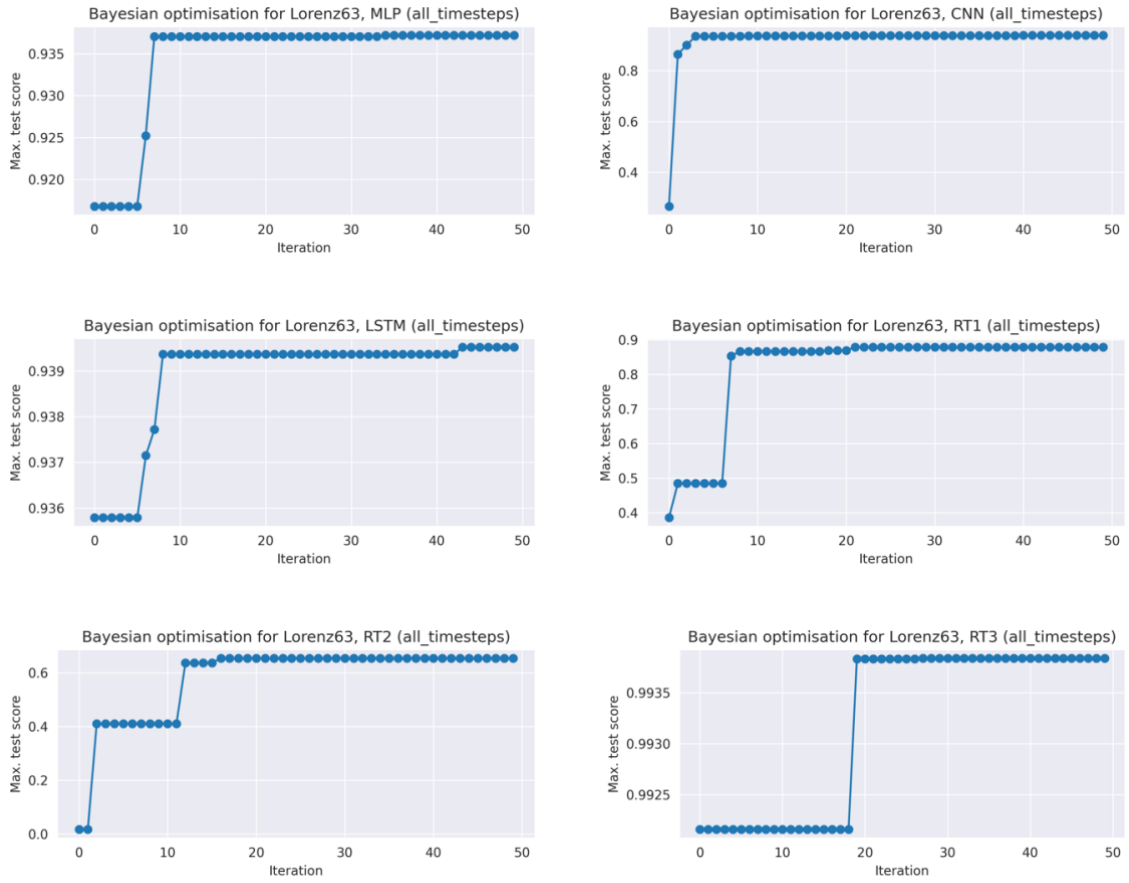

Figure 17: Maximum test score vs. Bayesian optimisation iterations for different ML models (Lorenz-63, All time-steps)

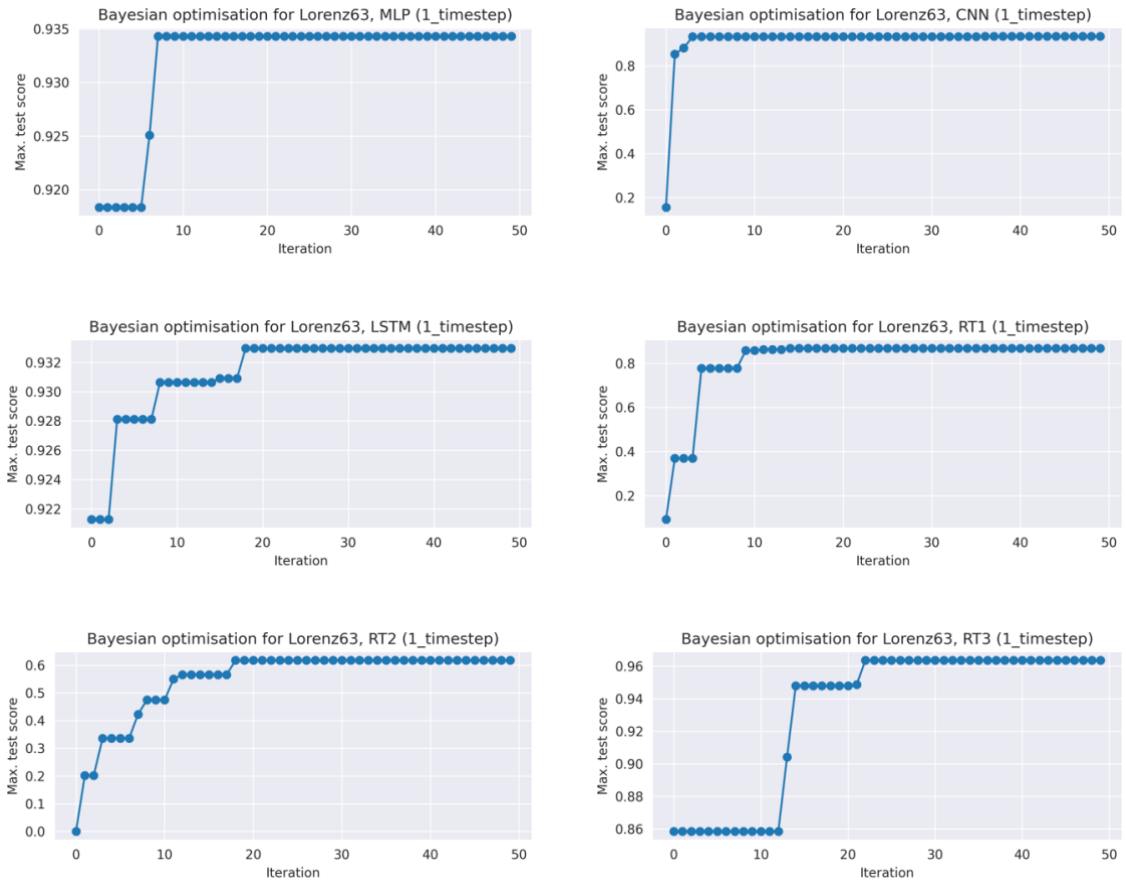

Figure 18: Maximum test score vs. Bayesian optimisation iterations for different ML models (Lorenz-63, One time-step)

A.2 Q-Q plots

A.2.1 Rössler

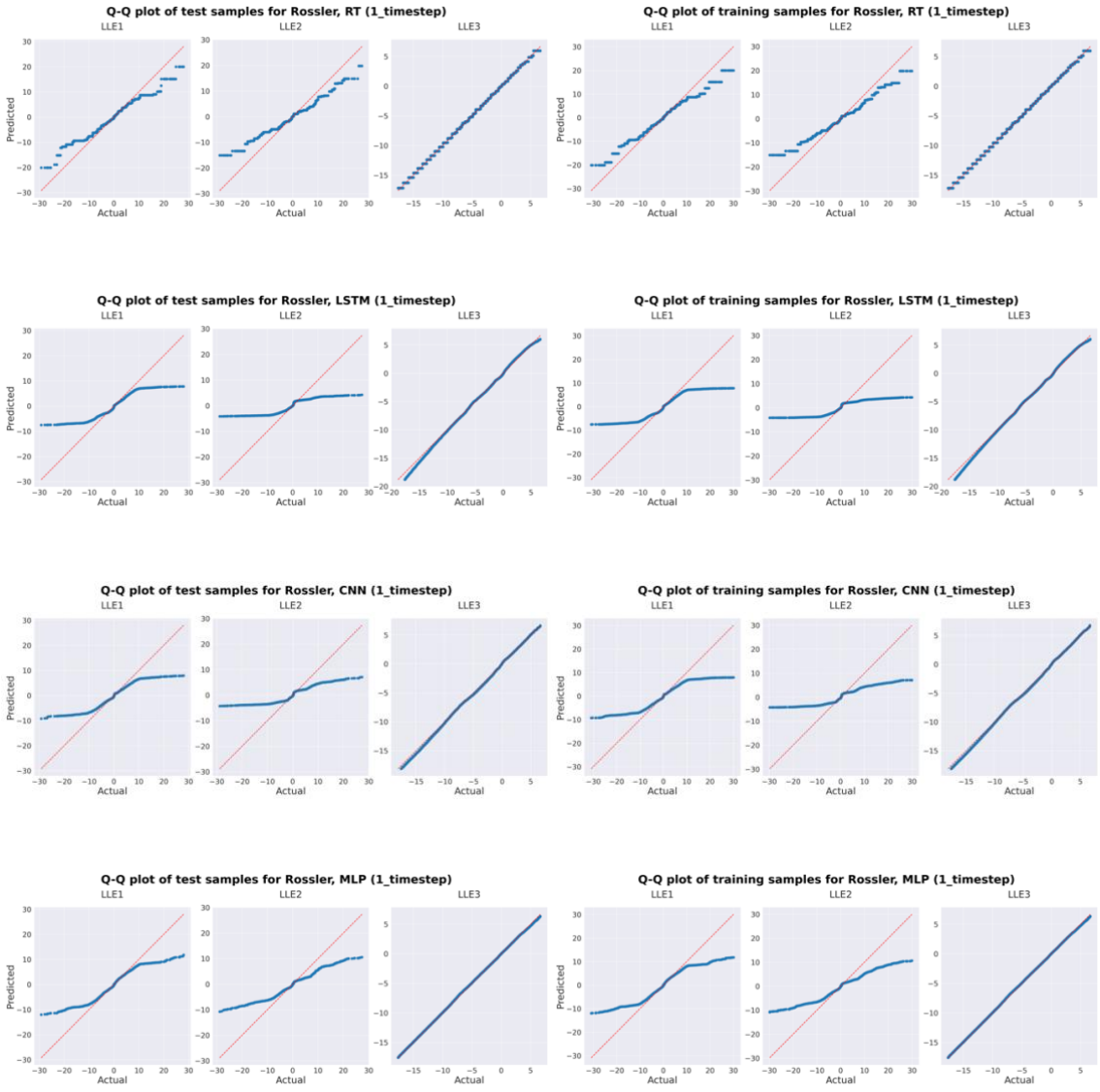

Figure 19: Q-Q plots of Rössler system, one time-step cases. Each row represents one ML model, with columns representing test (left) and training (right) samples

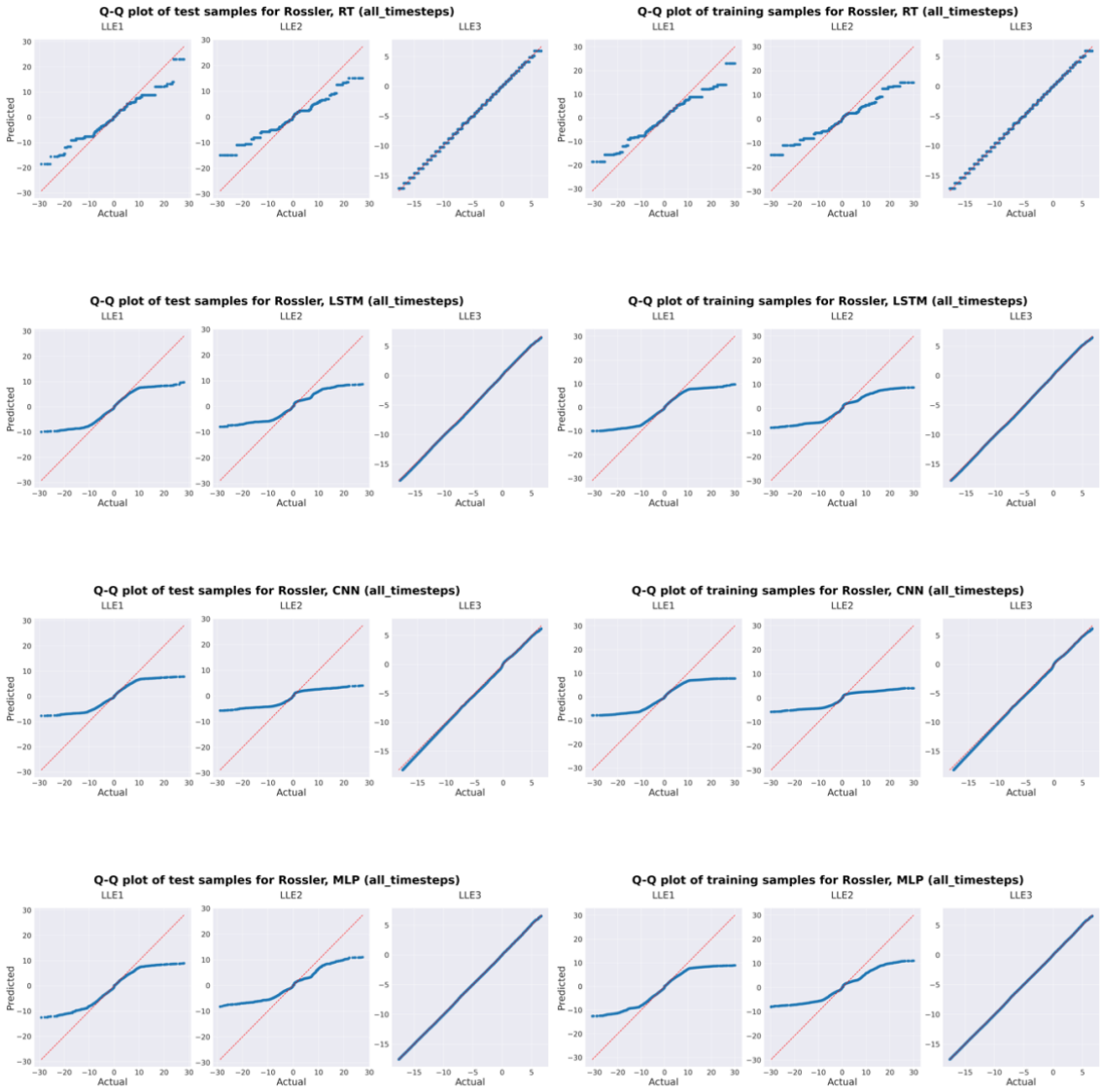

Figure 20: Q-Q plots of Rössler system, all time-steps cases. Each row represents one ML model, with columns representing test (left) and training (right) samples

A.2.2 Lorenz-63

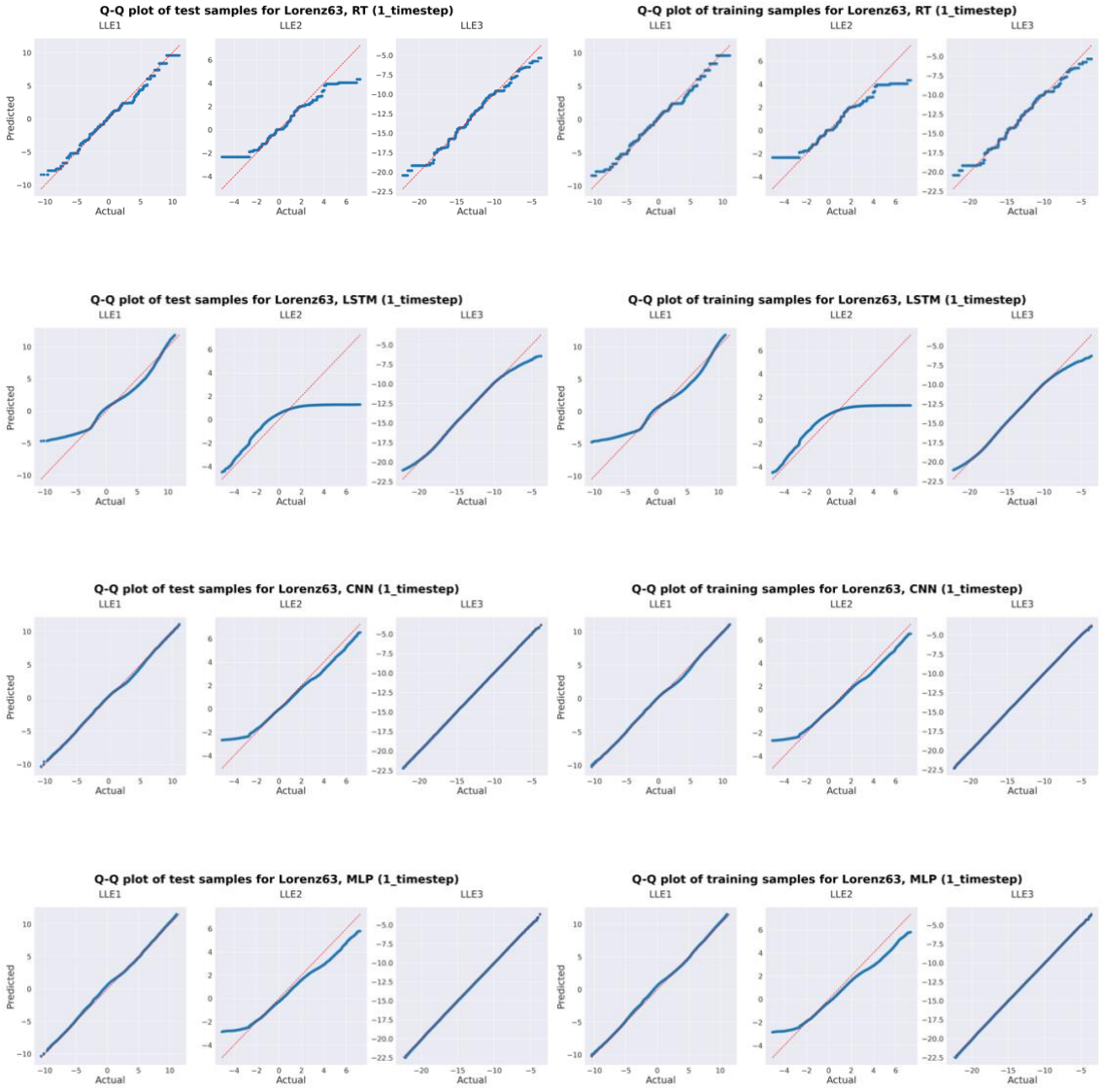

Figure 21: Q-Q plots of Lorenz-63 system, one time-step cases. Each row represents one ML model, with columns representing test (left) and training (right) samples

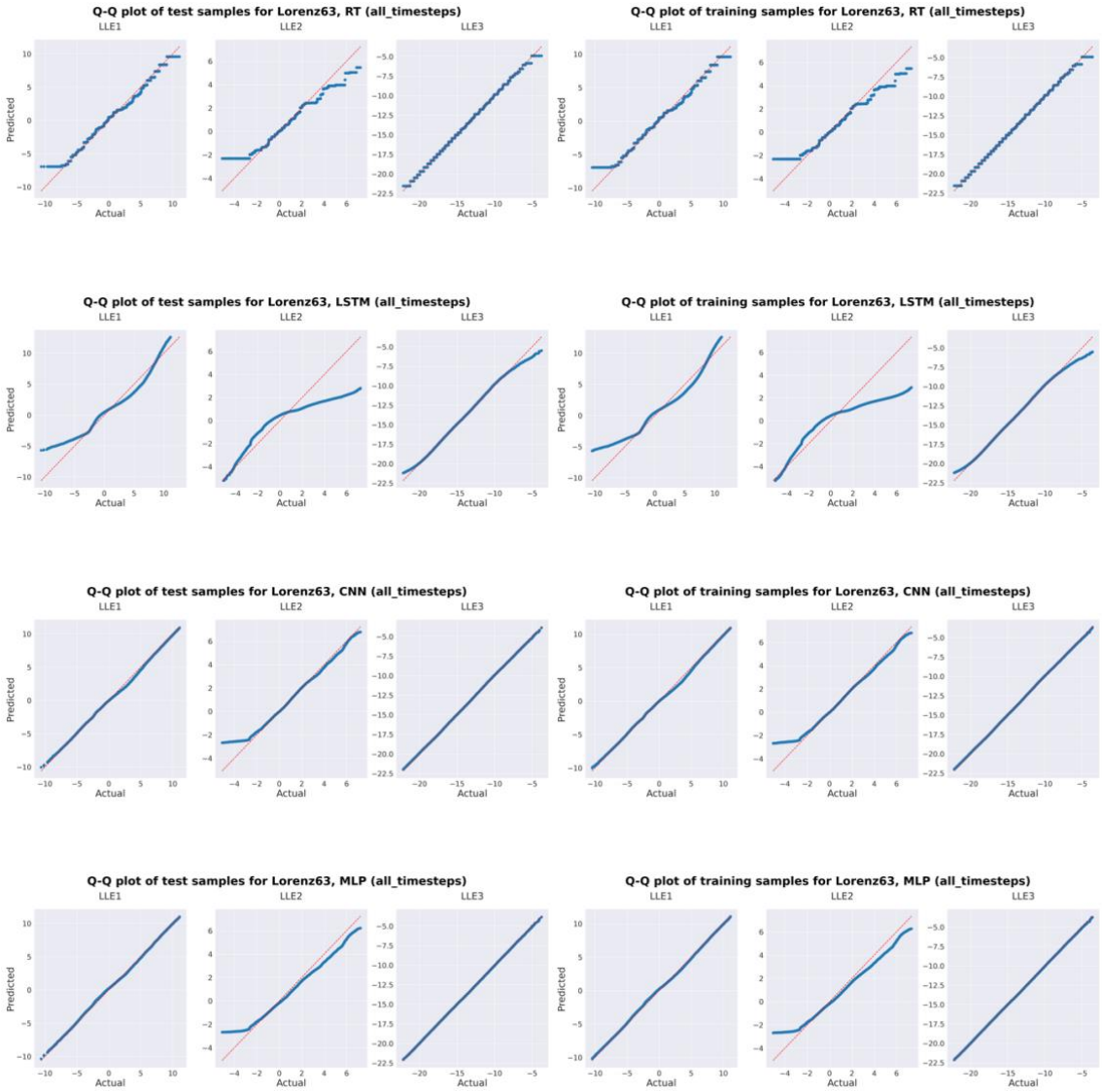

Figure 22: Q-Q plots of Lorenz-63 system, all time-step cases. Each row represents one ML model, with columns representing test (left) and training (right) samples

A.3 Cumulative means plots of LLEs

A.3.1 Rössler

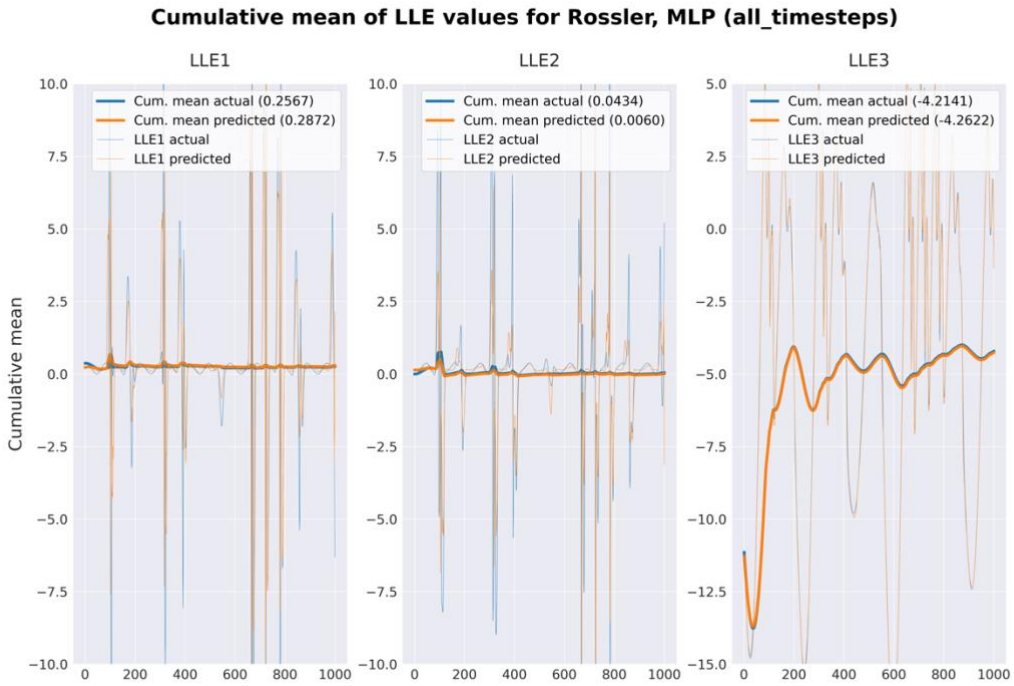

Figure 23: Plot of cumulative mean of LLE values of MLP for Rössler system, all time-step case. The 3 columns represent the three LLEs

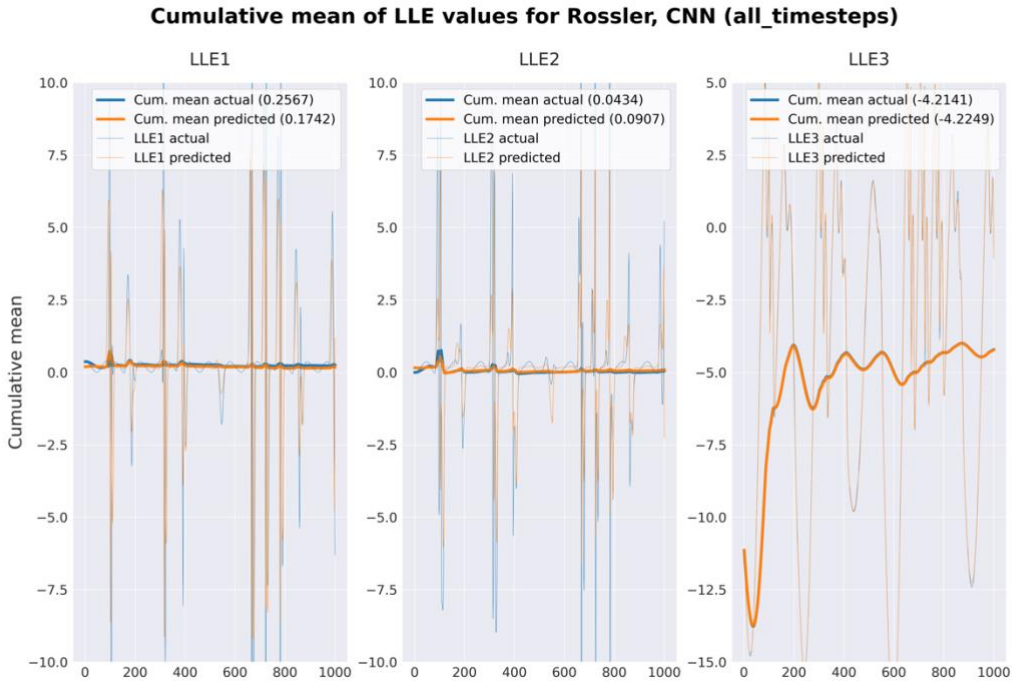

Figure 24: Plot of cumulative mean of LLE values of CNN for Rössler system, all time-step case. The 3 columns represent the three LLEs

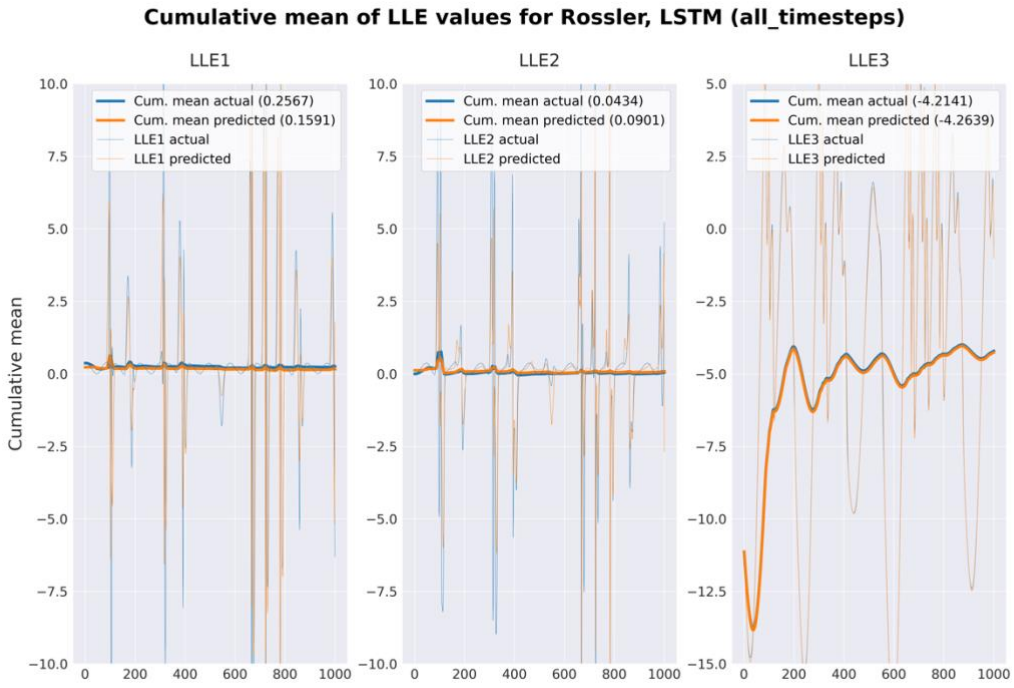

Figure 25: Plot of cumulative mean of LLE values of LSTM for Rössler system, all time-step case. The 3 columns represent the three LLEs

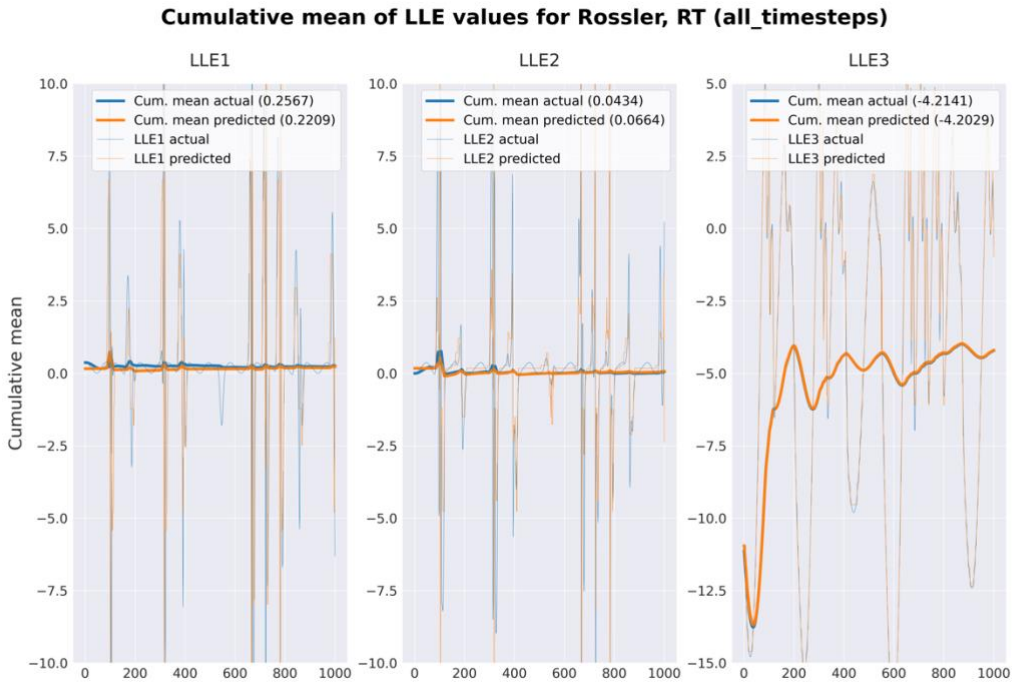

Figure 26: Plot of cumulative mean of LLE values of RT for Rössler system, all time-step case. The 3 columns represent the three LLEs

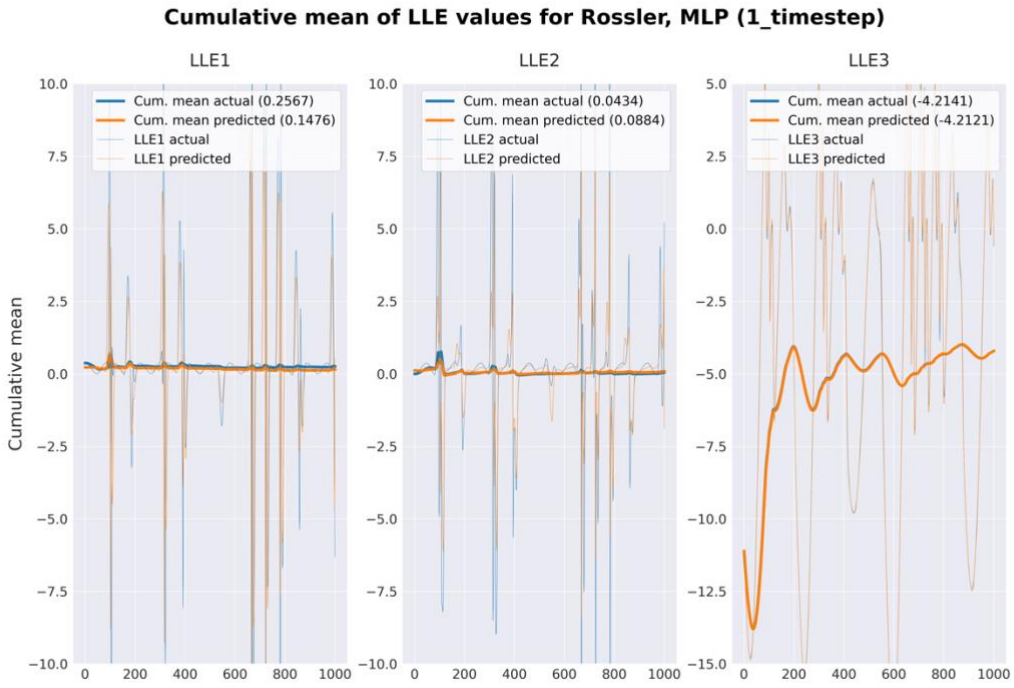

Figure 27: Plot of cumulative mean of LLE values of MLP for Rössler system, one time-step case. The 3 columns represent the three LLEs

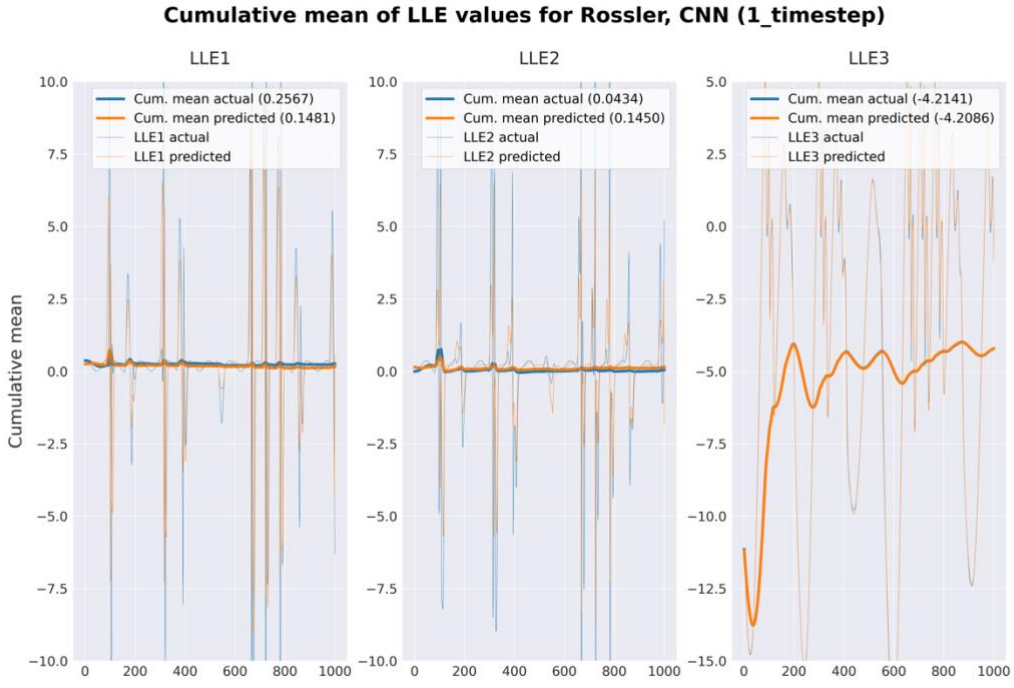

Figure 28: Plot of cumulative mean of LLE values of CNN for Rössler system, one time-step case. The 3 columns represent the three LLEs

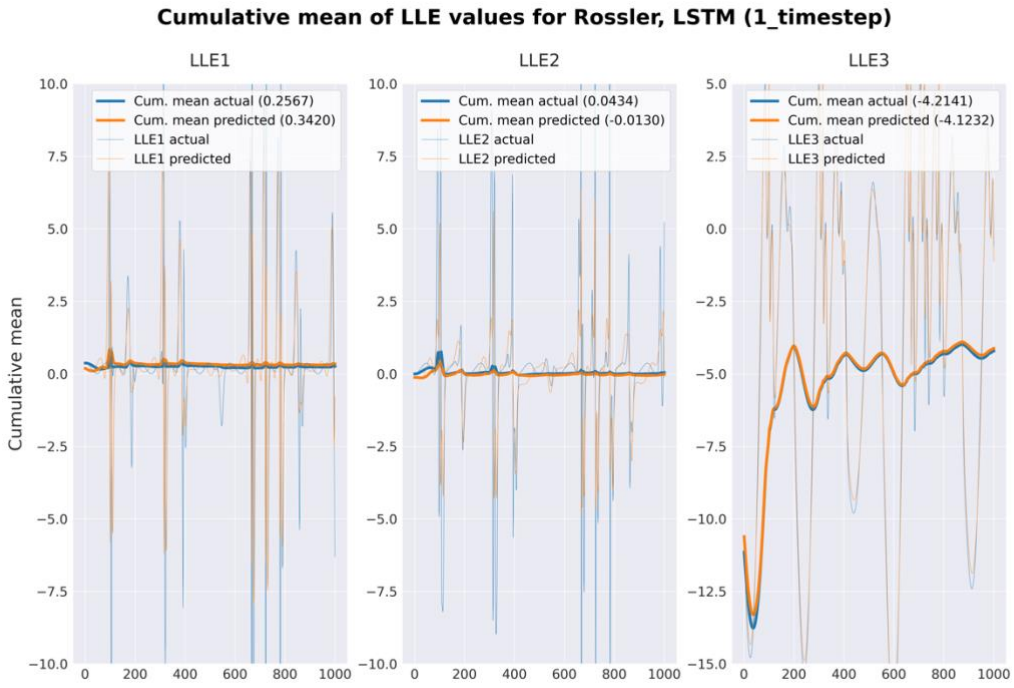

Figure 29: Plot of cumulative mean of LLE values of LSTM for Rössler system, one time-step case. The 3 columns represent the three LLEs

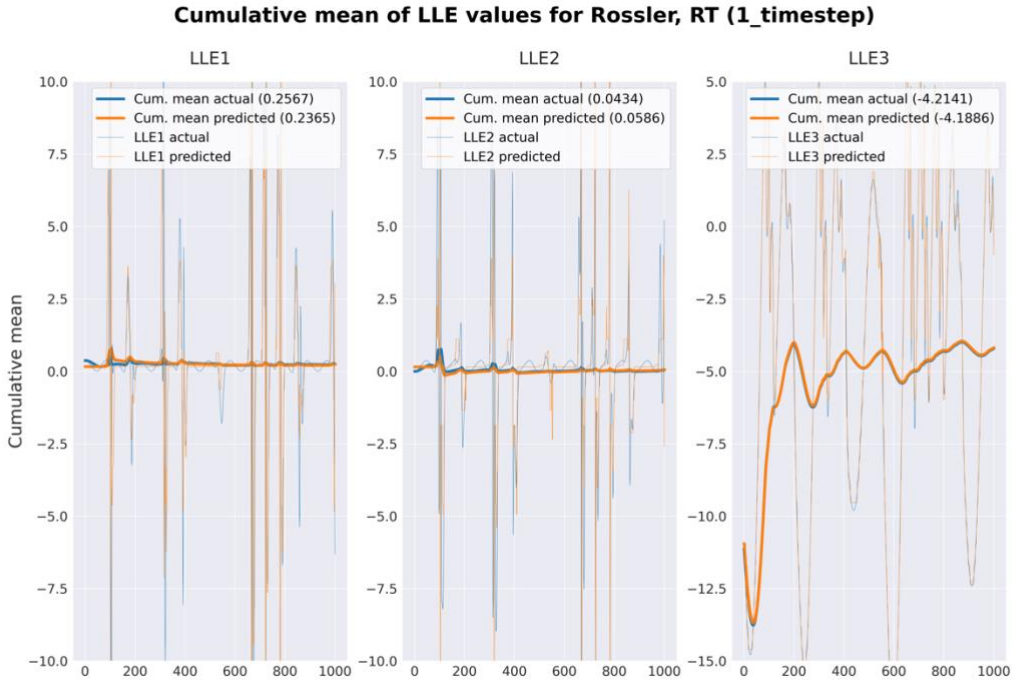

Figure 30: Plot of cumulative mean of LLE values of RT for Rössler system, one time-step case. The 3 columns represent the three LLEs

A.3.2 Lorenz-63

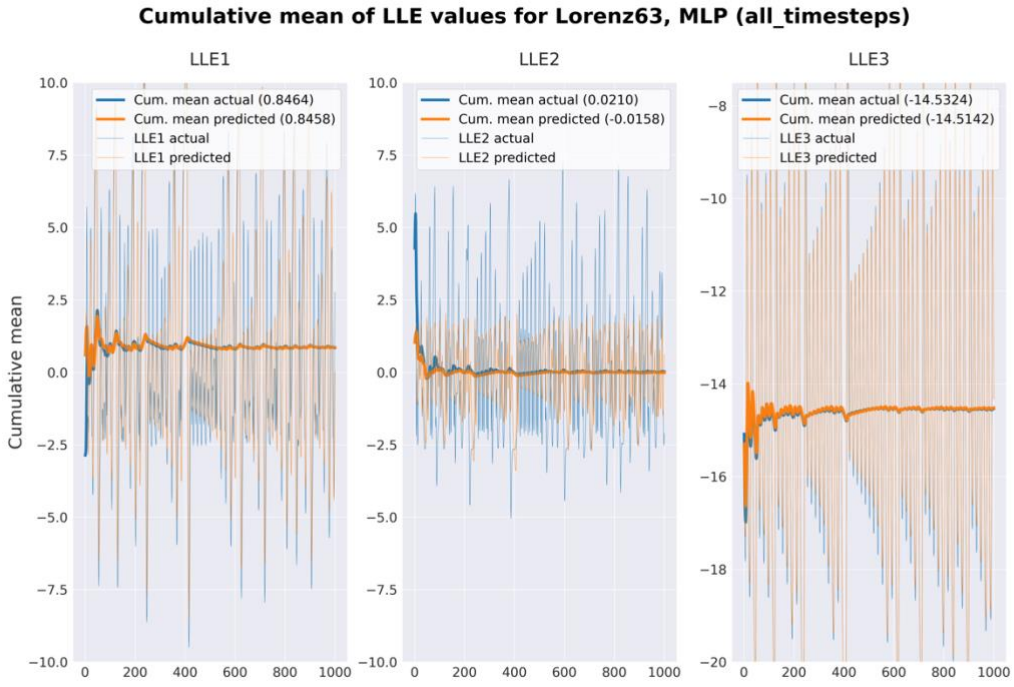

Figure 31: Plot of cumulative mean of LLE values of MLP for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

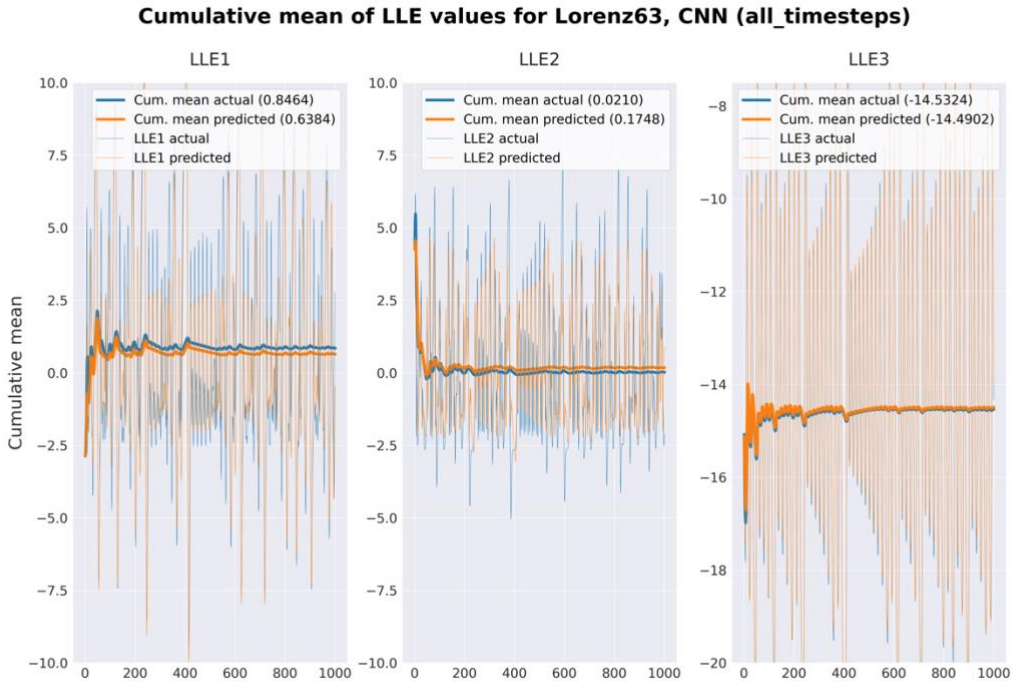

Figure 32: Plot of cumulative mean of LLE values of CNN for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

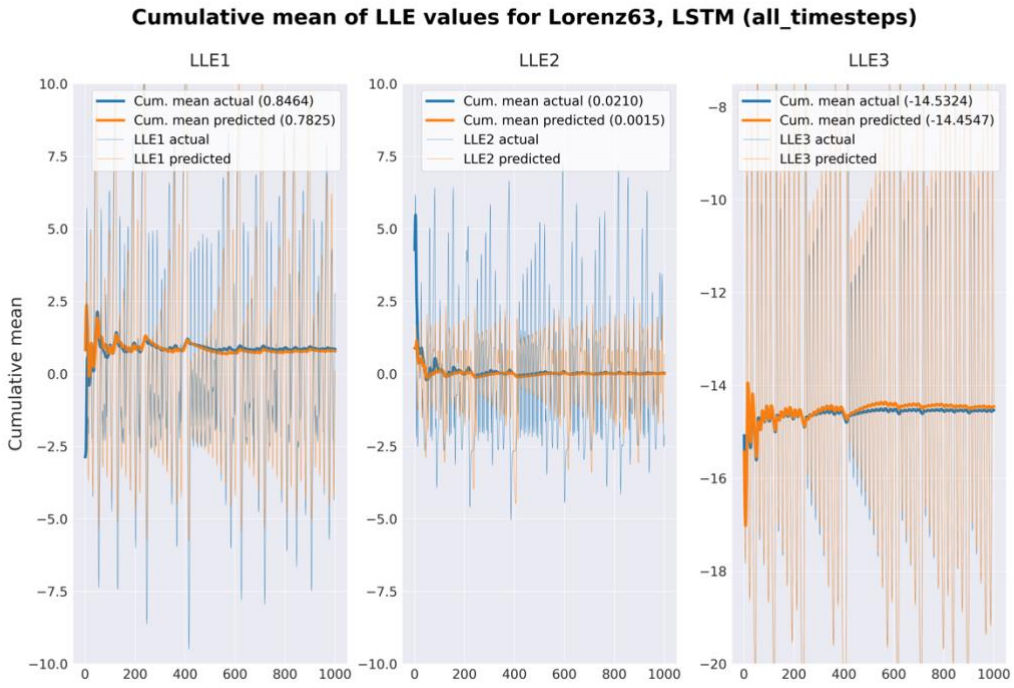

Figure 33: Plot of cumulative mean of LLE values of LSTM for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

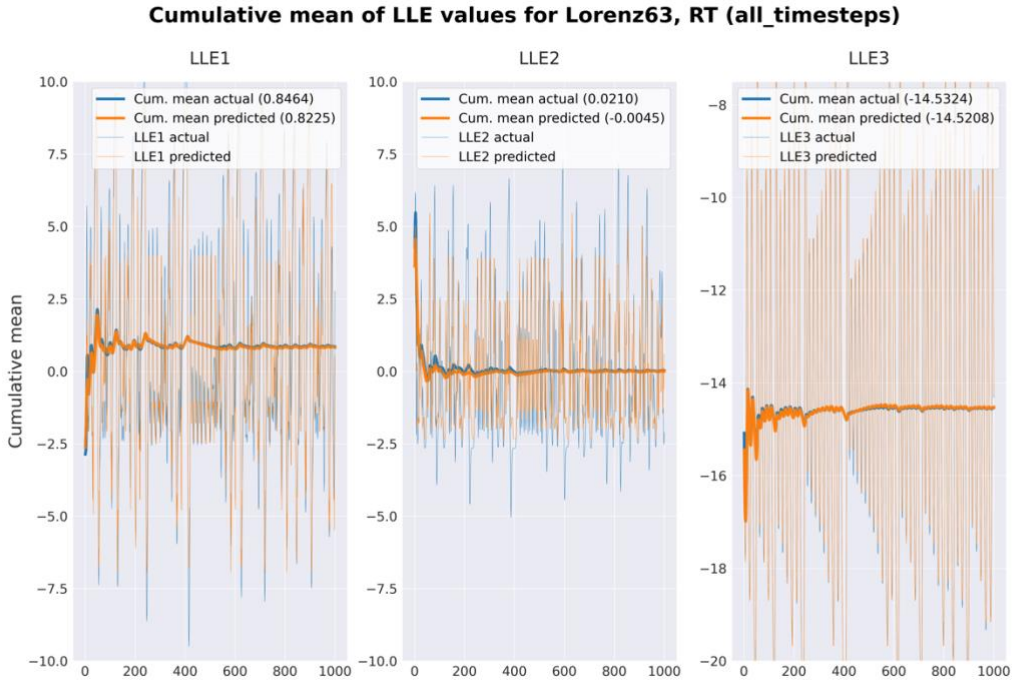

Figure 34: Plot of cumulative mean of LLE values of RT for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

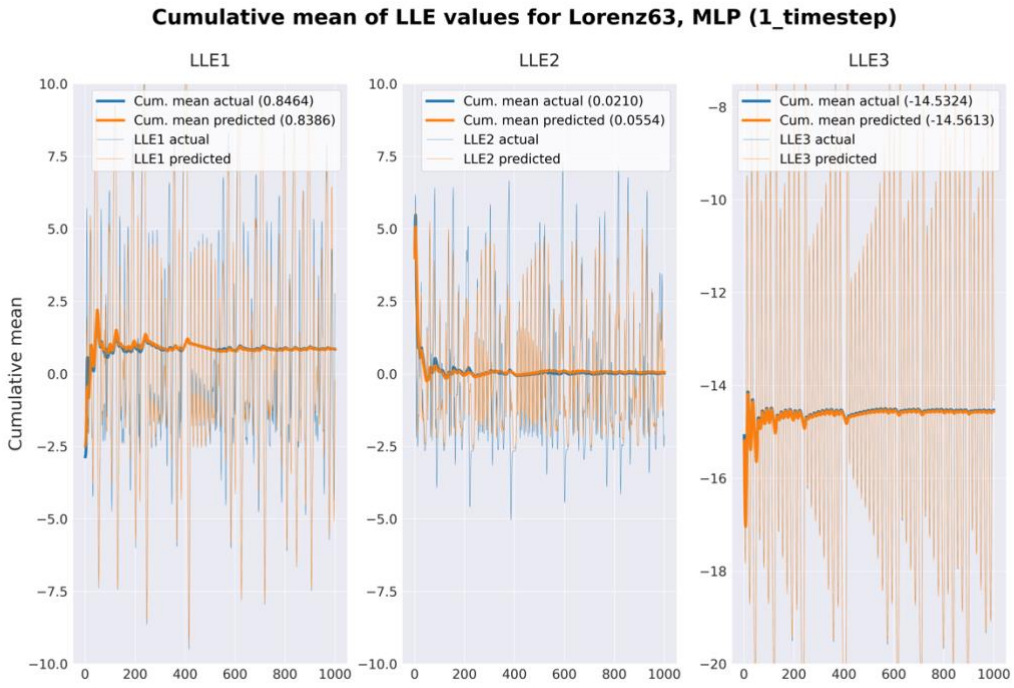

Figure 35: Plot of cumulative mean of LLE values of MLP for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

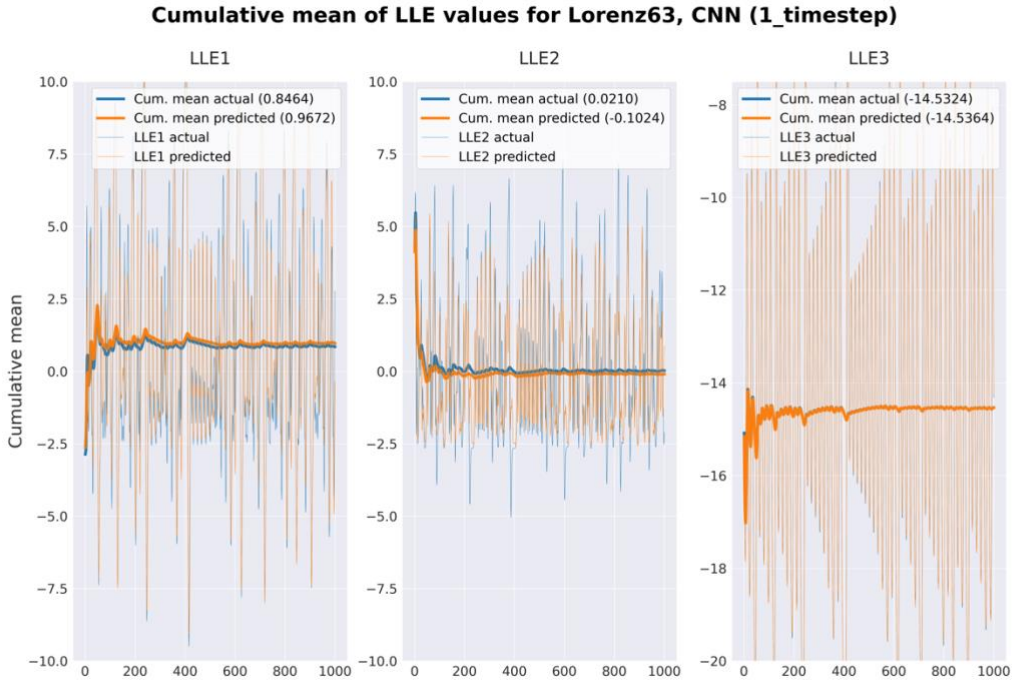

Figure 36: Plot of cumulative mean of LLE values of CNN for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

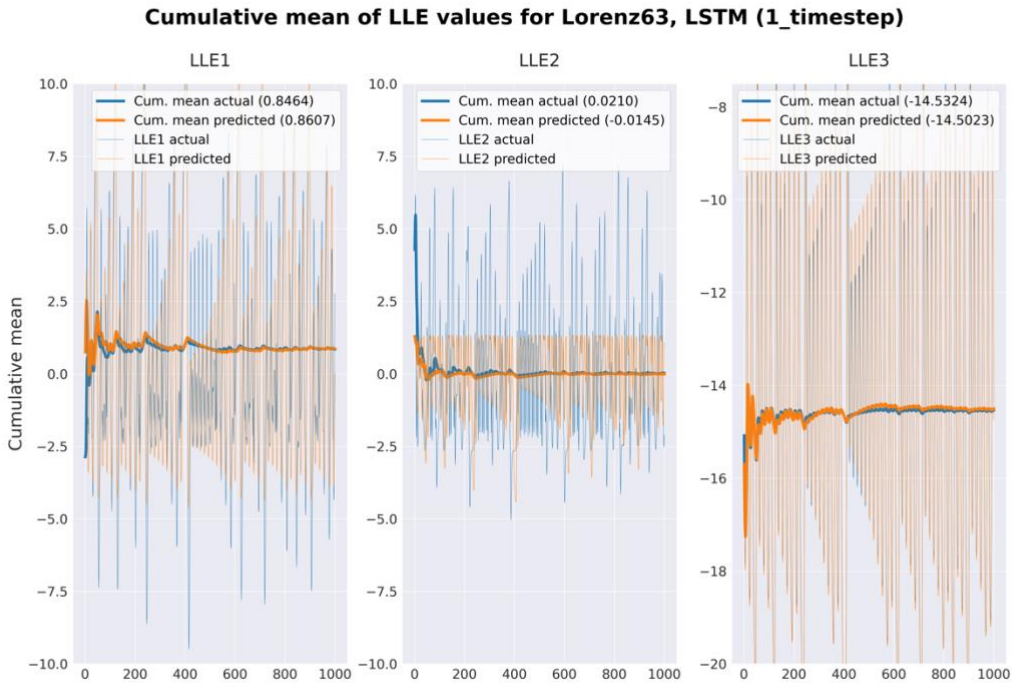

Figure 37: Plot of cumulative mean of LLE values of LSTM for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

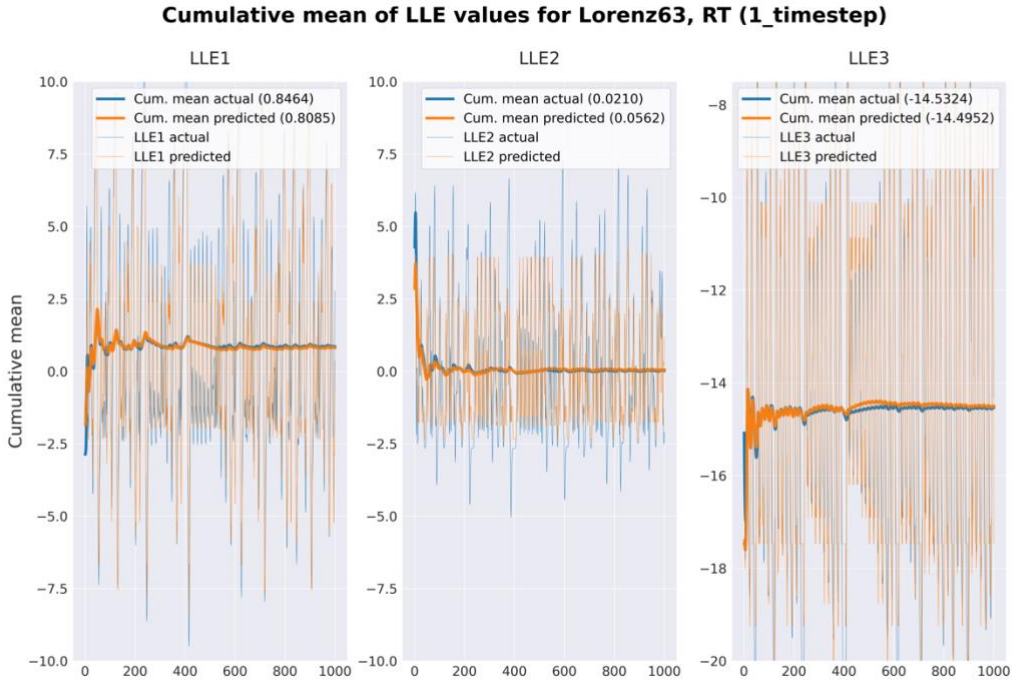

Figure 38: Plot of cumulative mean of LLE values of RT for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

A.4 Moving mean plots of LLEs

A.4.1 Rössler

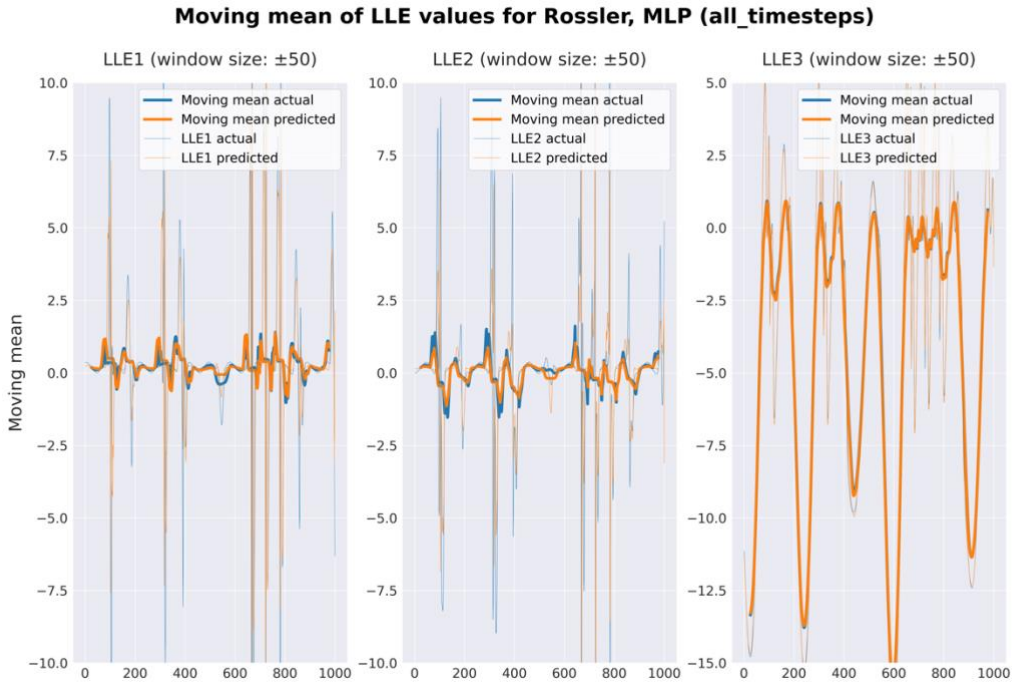

Figure 39: Plot of moving mean of LLE values of MLP for Rössler system, all time-steps case. The 3 columns represent the three LLEs

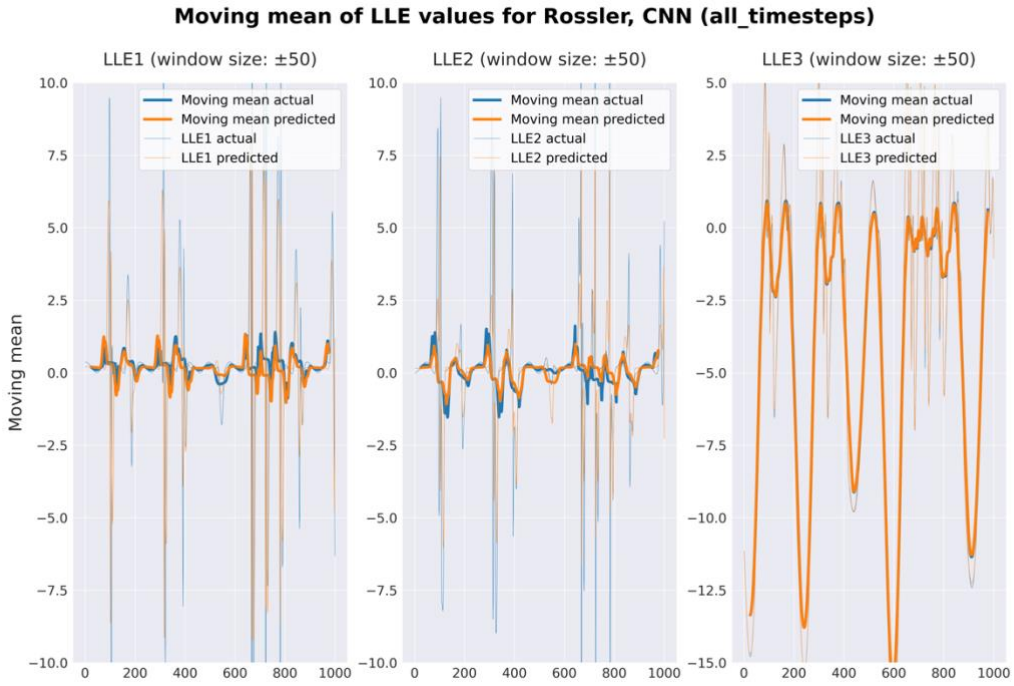

Figure 40: Plot of moving mean of LLE values of CNN for Rössler system, all time-steps case. The 3 columns represent the three LLEs

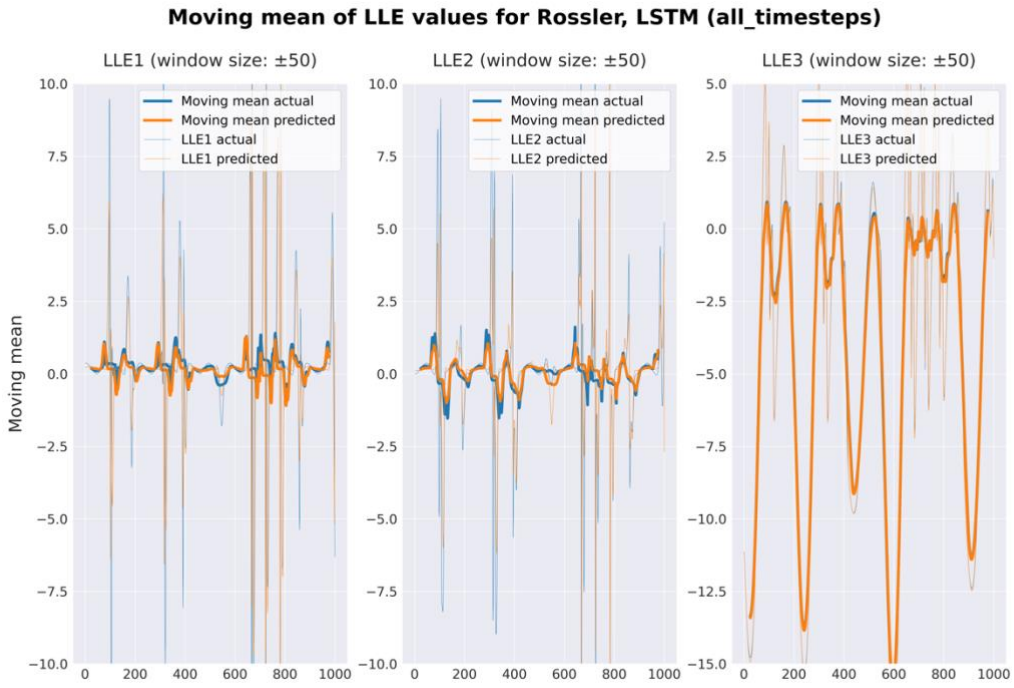

Figure 41: Plot of moving mean of LLE values of LSTM for Rössler system, all time-steps case. The 3 columns represent the three LLEs

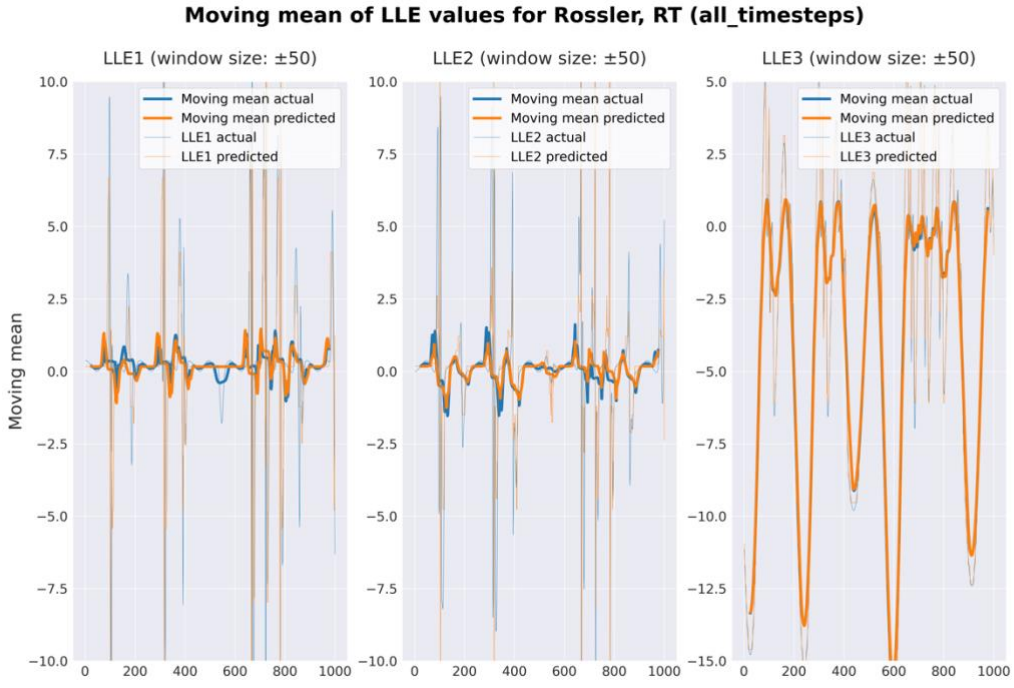

Figure 42: Plot of moving mean of LLE values of RT for Rössler system, all time-steps case. The 3 columns represent the three LLEs

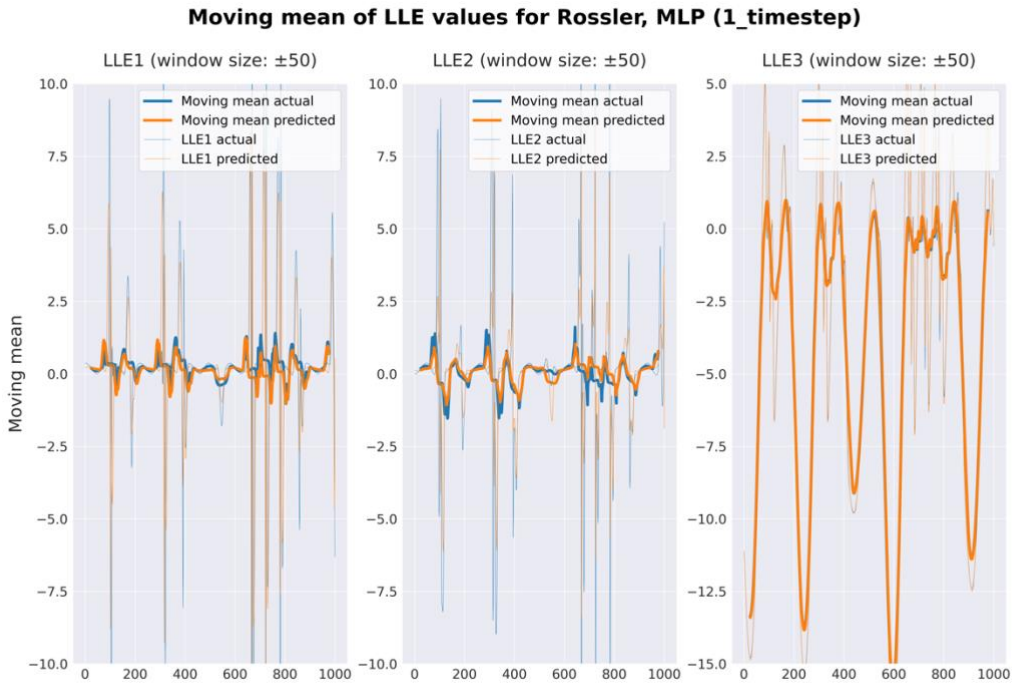

Figure 43: Plot of moving mean of LLE values of MLP for Rössler system, one time-step case. The 3 columns represent the three LLEs

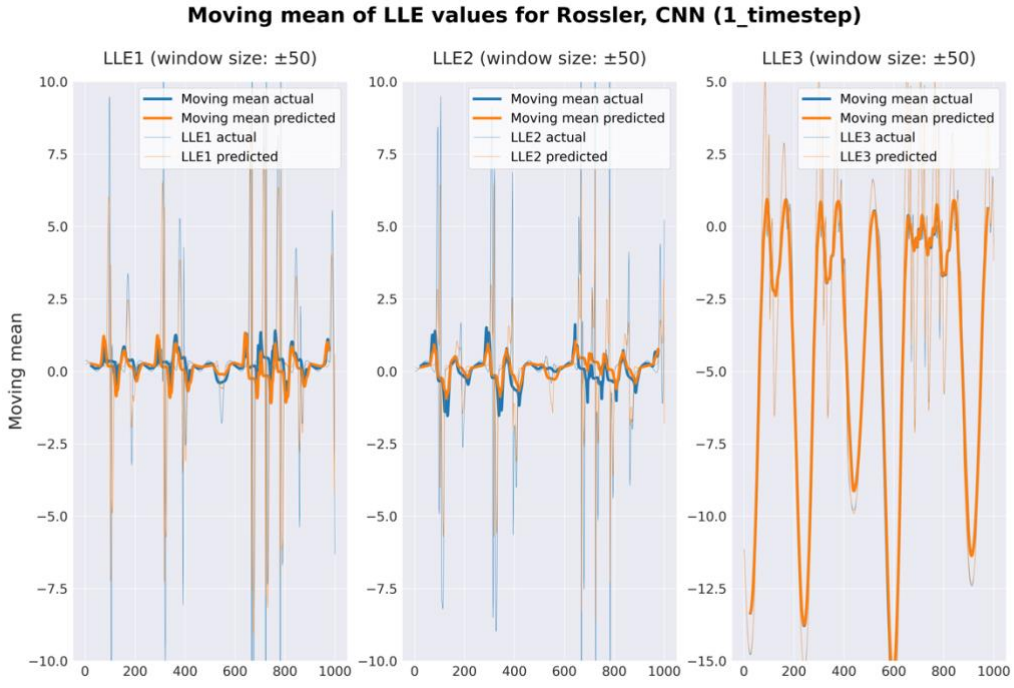

Figure 44: Plot of moving mean of LLE values of CNN for Rössler system, one time-step case. The 3 columns represent the three LLEs

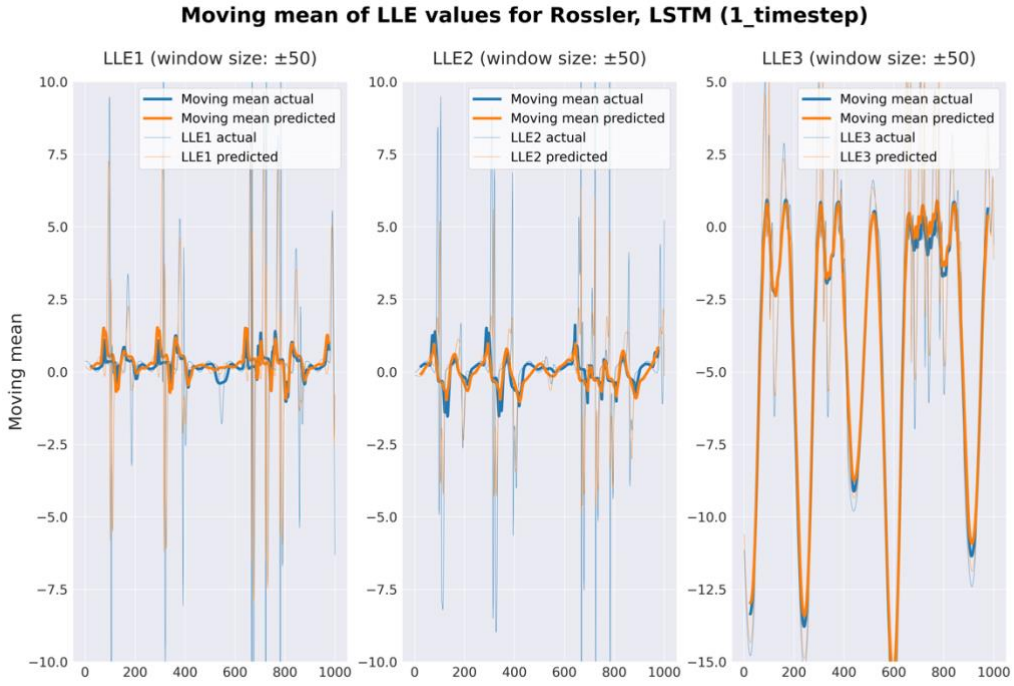

Figure 45: Plot of moving mean of LLE values of LSTM for Rössler system, one time-step case. The 3 columns represent the three LLEs

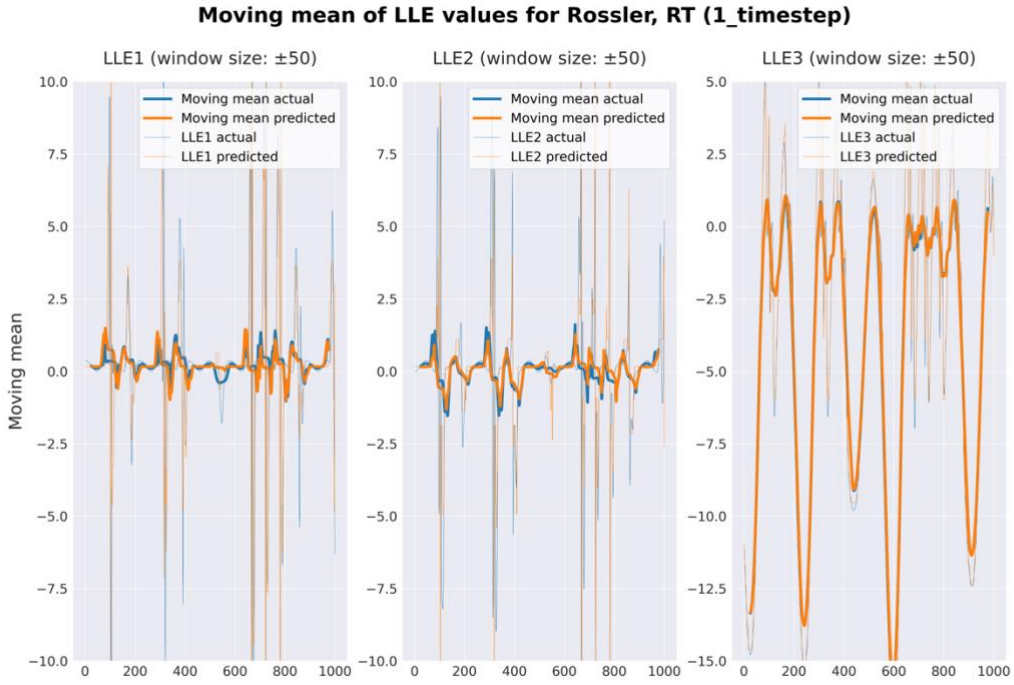

Figure 46: Plot of moving mean of LLE values of RT for Rössler system, one time-step case. The 3 columns represent the three LLEs

A.4.2 Lorenz-63

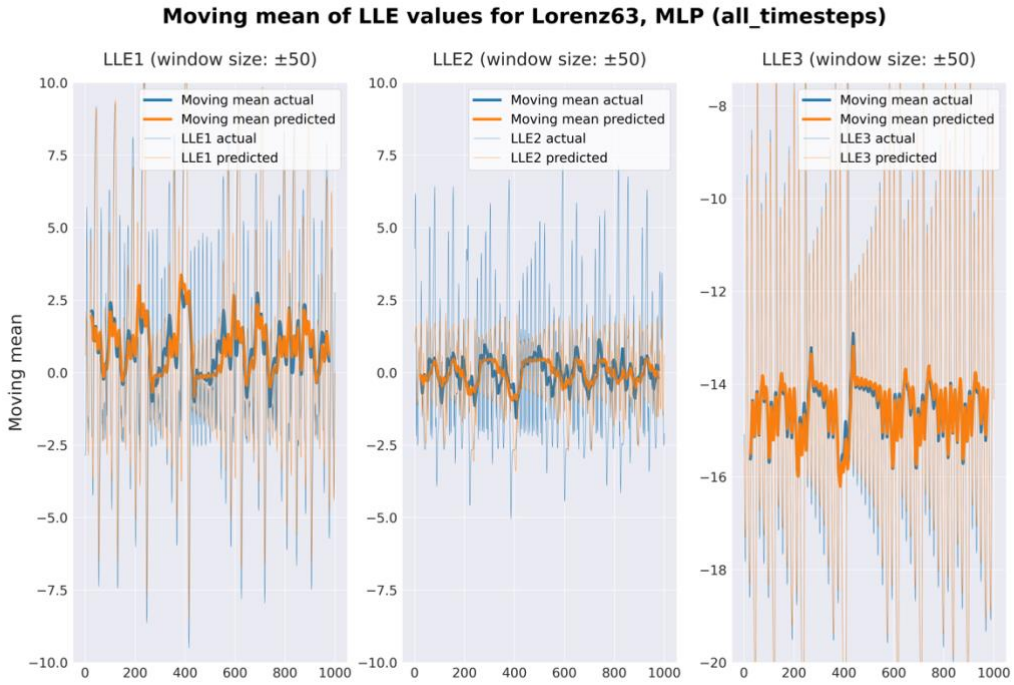

Figure 47: Plot of moving mean of LLE values of MLP for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

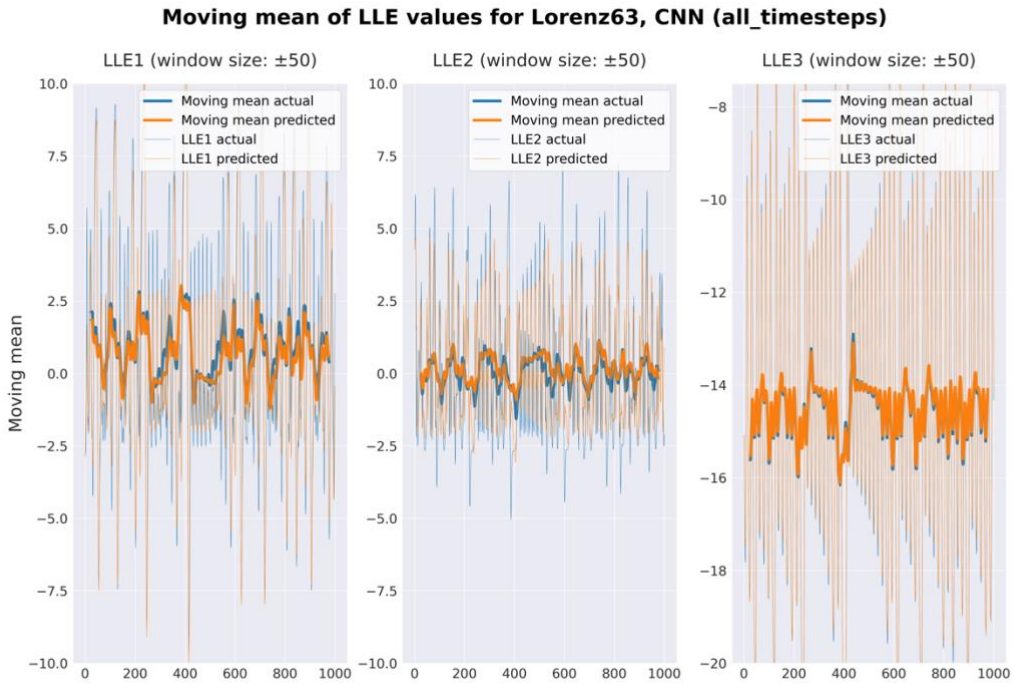

Figure 48: Plot of moving mean of LLE values of CNN for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

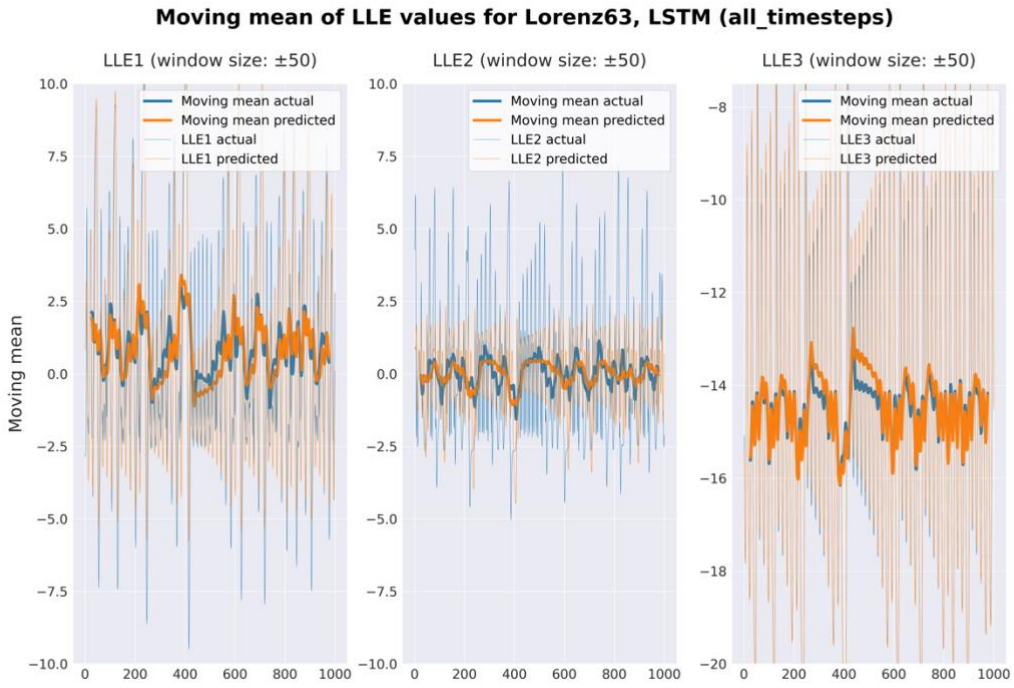

Figure 49: Plot of moving mean of LLE values of LSTM for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

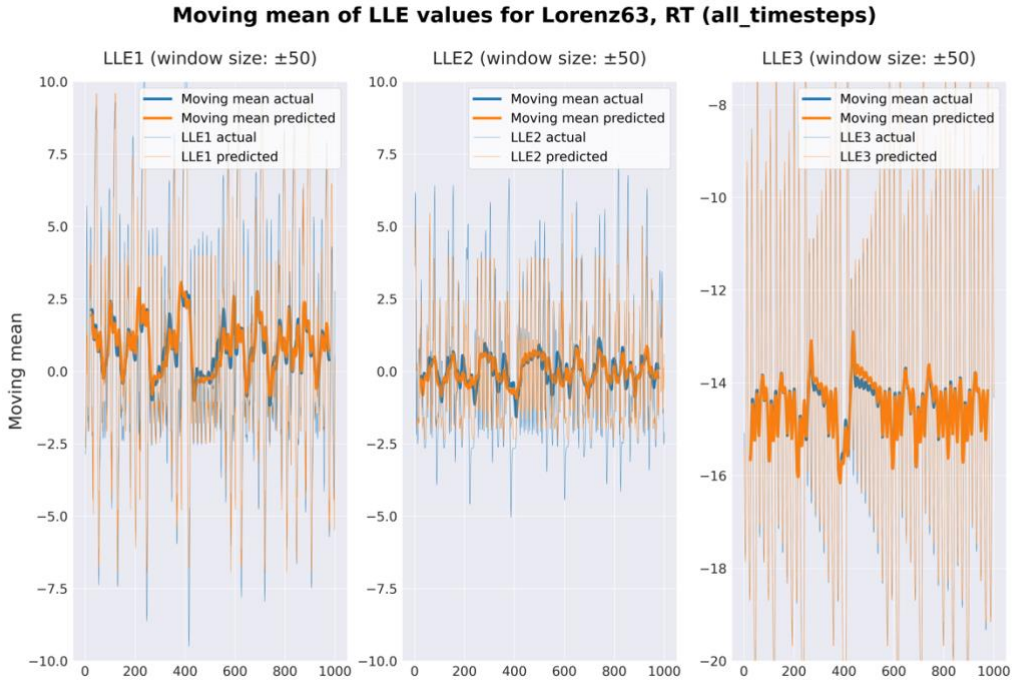

Figure 50: Plot of moving mean of LLE values of RT for Lorenz-63 system, all time-steps case. The 3 columns represent the three LLEs

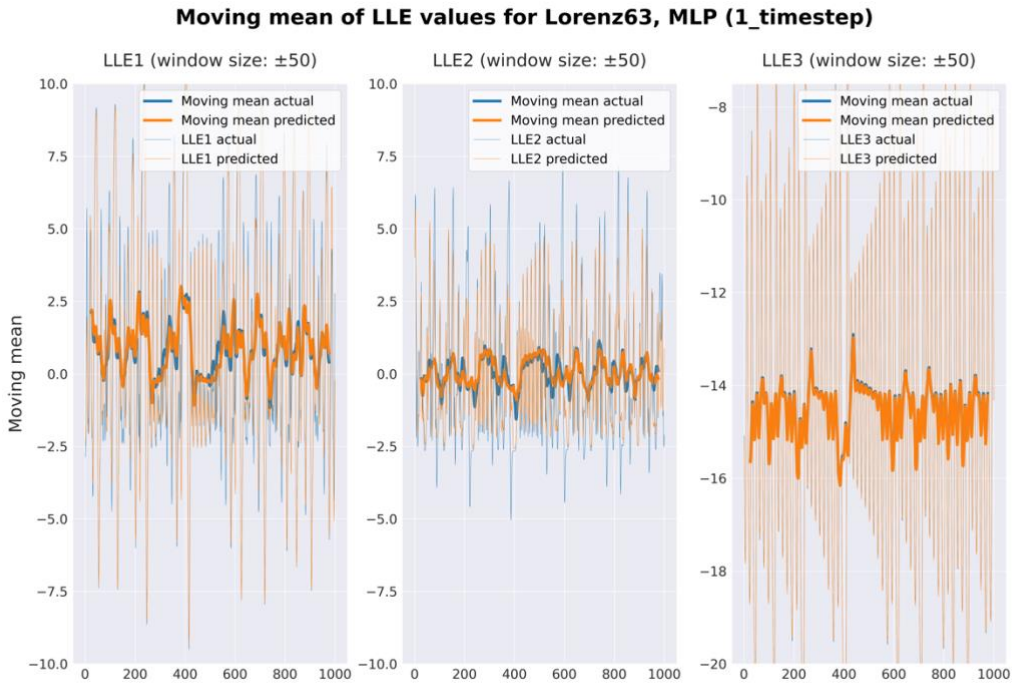

Figure 51: Plot of moving mean of LLE values of MLP for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

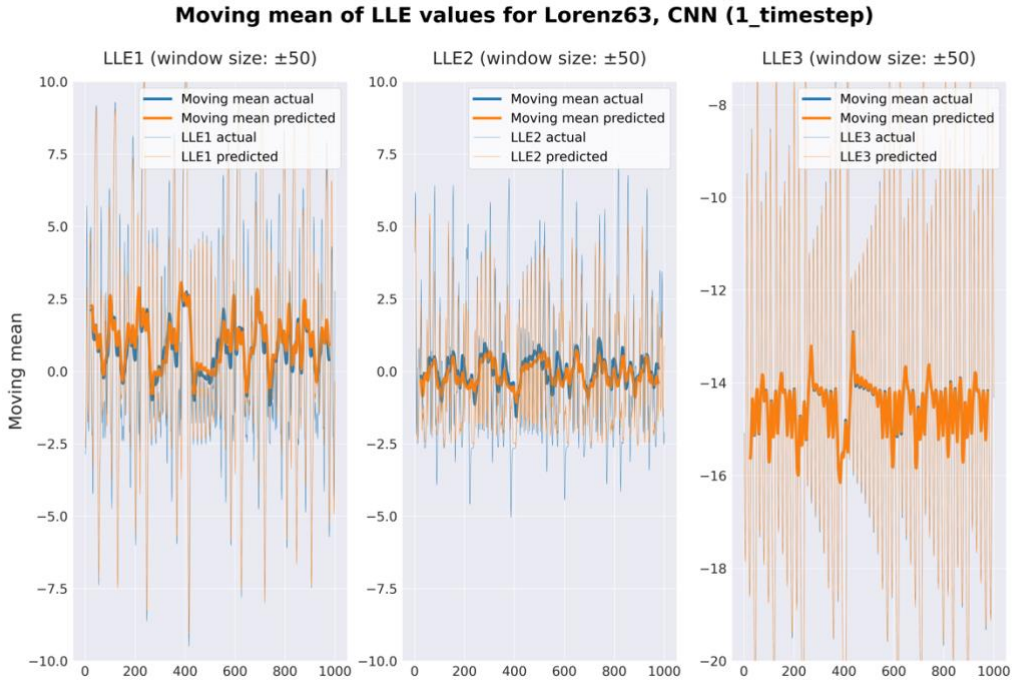

Figure 52: Plot of moving mean of LLE values of CNN for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

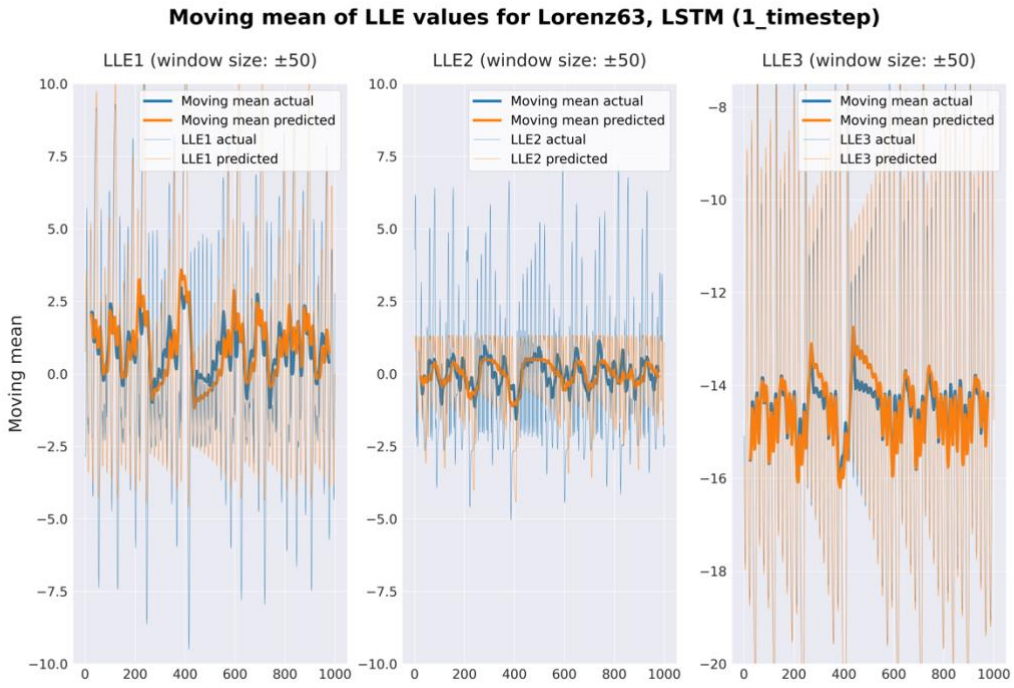

Figure 53: Plot of moving mean of LLE values of LSTM for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

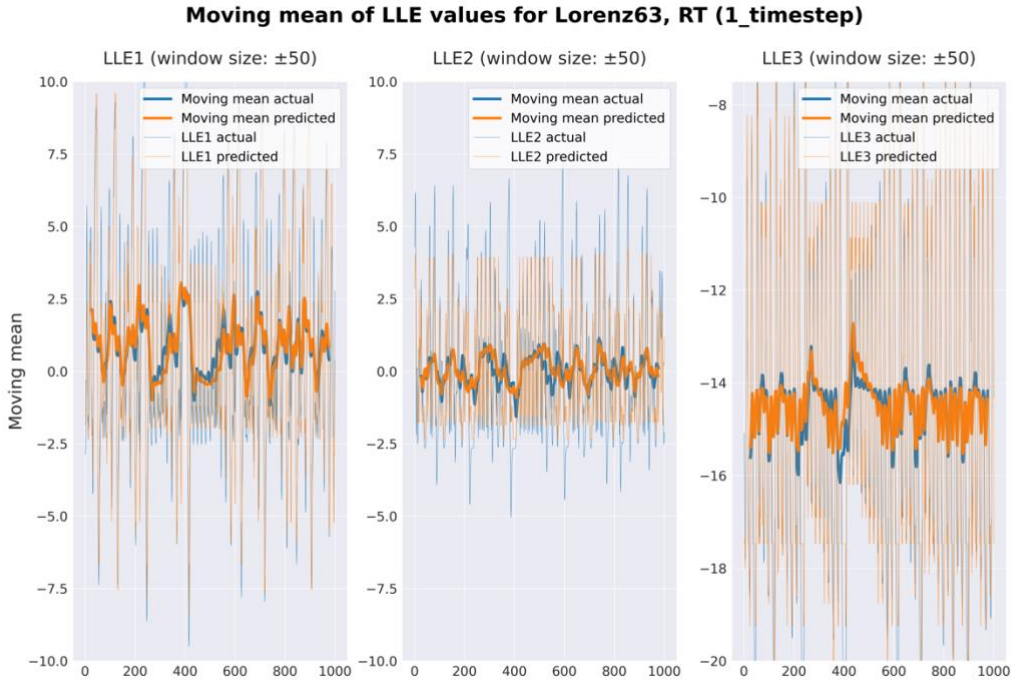

Figure 54: Plot of moving mean of LLE values of RT for Lorenz-63 system, one time-step case. The 3 columns represent the three LLEs

Appendix B. Hyperparameter Tuning

B.1 Bayesian optimisation search space

Table 8: Bayesian optimisation search space

ML Model	Hyperparameter	Search range / values
CNN	Learning rate	1e-6 to 0.9
	No. of layers	1 to 6
	No. of neurons per layer	10 to 200
	Regulariser	L1, L2, None
	No. of filters	1 to 100
MLP	Learning rate	1e-6 to 0.9
	No. of layers	1 to 10
	No. of neurons per layer	1 to 200
	Regulariser	L1, L2, None
LSTM	Learning rate	1e-6 to 0.9
	No. of layers	1 to 3
	No. of LSTM units per layer	1 to 200
	Regulariser	L1, L2, None
RT	ccp_alpha	1e-6 to 100
	Max. depth	1 to 100
	Max. features	auto, log2, sqrt, None
	Max. leaf nodes	5 to 100
	Min. samples in leaf	1 to 20
	Min. weight fraction leaf	0, 0.5
	Splitter	best, random

B.2 Durations of running Bayesian optimisation

Table 9: Durations of running Bayesian optimisation

ML Model	System	No. of time-steps	Duration (hh:mm)
CNN	Rössler	One time-step	4:14
		All time-steps	4:23
	Lorenz-63	One time-step	3:11
		All time-steps	5:49
MLP	Rössler	One time-step	5:28
		All time-steps	11:23
	Lorenz-63	One time-step	5:11
		All time-steps	4:41
LSTM	Rössler	One time-step	8:05
		All time-steps	19:21
	Lorenz-63	One time-step	8:10
		All time-steps	21:39
RT1	Rössler	One time-step	0:03
		All time-steps	0:03
	Lorenz-63	One time-step	0:03
		All time-steps	0:05
RT2	Rössler	One time-step	0:03
		All time-steps	0:04
	Lorenz-63	One time-step	0:03
		All time-steps	0:06
RT3	Rössler	One time-step	0:03
		All time-steps	0:05
	Lorenz-63	One time-step	0:03
		All time-steps	0:05
		Total duration:	**102:21**

B.3 Optimal hyperparameters

Table 10: Optimal hyperparameter values for CNN obtained from Bayesian optimisation

CNN	Learning rate	No. of layers	No. of neurons per layer	Regulariser	No. of filters
Rössler, One time-step	0.000108	6	200	None	100
Rössler, All time-steps	0.000159	6	200	L1	37
Lorenz-63, One time-step	0.000329	3	122	L1	29
Lorenz-63, All time-steps	0.000102	6	200	L1	59

Table 11: Optimal hyperparameter values for LSTM obtained from Bayesian optimisation

LSTM	Learning rate	No. of layers	No. of LSTM units per layer	Regulariser
Rössler, One time-step	0.002857	3	62	L1
Rössler, All time-steps	0.001874	1	100	L1
Lorenz-63, One time-step	0.002722	2	100	L1
Lorenz-63, All time-steps	0.000111	2	100	None

Table 12: Optimal hyperparameter values for MLP obtained from Bayesian optimisation

MLP	Learning rate	No. of layers	No. of neurons per layer	Regulariser
Rössler, One time-step	0.000073	7	200	L2
Rössler, All time-steps	0.000122	6	182	L2
Lorenz-63, One time-step	0.000076	8	165	L2
Lorenz-63, All time-steps	0.000019	10	200	L1

Table 13: Optimal hyperparameter values for RT1 (for LLE_1) obtained from Bayesian optimisation

RT1	ccp_alpha	Max. depth	Max. features	Max. leaf nodes	Min. samples in leaf	Min. weight fraction leaf	Splitter
Rössler, One time-step	0.000002	78	auto	100	20	0	best
Rössler, All time-steps	0.000076	58	auto	100	13	0	random
Lorenz-63, One time-step	0.000001	100	auto	100	15	0	best
Lorenz-63, All time-steps	0.000003	100	None	100	16	0	best

Table 14: Optimal hyperparameter values for RT2 (for LLE_2) obtained from Bayesian optimisation

RT2	ccp_alpha	Max. depth	Max. features	Max. leaf nodes	Min. samples in leaf	Min. weight fraction leaf	Splitter
Rössler, One time-step	0.000001	100	auto	100	20	0	best
Rössler, All time-steps	0.000001	24	auto	100	20	0	random
Lorenz-63, One time-step	0.000581	100	auto	100	1	0	best
Lorenz-63, All time-steps	0.000462	78	None	100	1	0	best

Table 15: Optimal hyperparameter values for RT3 (for LLE_3) obtained from Bayesian optimisation

RT3	ccp_alpha	Max. depth	Max. features	Max. leaf nodes	Min. samples in leaf	Min. weight fraction leaf	Splitter
Rössler, One time-step	0.000114	17	auto	100	2	0	best
Rössler, All time-steps	0.000003	48	None	100	1	0	best
Lorenz-63, One time-step	0.000001	100	sqrt	100	14	0	best
Lorenz-63, All time-steps	0.000002	72	auto	100	20	0	best

Appendix C. Source repository of the project

All the source codes, input data files, results files and plots are available in GitHub:

https://github.com/Dangyers/SL-estimation-of-LLEs-L63-Rössler