Master Thesis

Uncertainty-Aware Neural Network for Data-Driven Modeling of A Foam-Damped System

Lowie De Malsche

Delft University of Technology

TU Delft

REDWIRE

Master Thesis

Uncertainty-Aware Neural Network for Data-Driven Modeling of A Foam-Damped System

by

Lowie De Malsche

Responsible Supervisor:	Dr. V. Yaghoubi Nasrabadi
External Supervisor:	Ir. E. Lemmens
Project Duration:	August, 2024 - June, 2025
Faculty:	Faculty of Aerospace Engineering, Delft

Cover: Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under CC BY-NC 2.0 (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld [1]

TU Delft

Preface

This report was written as part of my Master thesis at the Delft University of Technology. The project was carried out in collaboration with Redwire Space NV., and builds on the work that was done during my internship where I tried to find a method that could be used to simulate the response of a foam-packed item to a shock. The results of the internship project were a promising proof of concept but lacked accuracy due to the absence of a model that could accurately represent a foam packed system. In this thesis, three models are evaluated for their viability as a surrogate model to represent the foam-damped system.

While writing this report the reader was assumed to have basic engineering knowledge as well as basic knowledge about machine learning and artificial-intelligence. A basic explanation of the concepts that were used can be found in Chapter 2.

Readers that are interested in how the models were chosen can find the trade-off in Chapter 3. Readers that are interested in how the data-set that was used to train the models as well as how the models were trained can find the methodology in Chapter 4. The results as well as the discussion of the results can be found in Chapters 5 and 6. These chapters have the same structure and are meant to be read side-by-side. Recommendations for future work on this subject are made in Chapter 8.

I would like to thank the supervisor of this thesis, Vahid Yaghoubi Nasrabadi, for his close involvement and guidance throughout the project. I would also like to thank my external supervisor, Els Lemmens at Redwire Space, for her continued support during both my internship and this thesis. Finally, I would like to thank Amalia Macali who helped me augment the models with a GP, making them uncertainty-aware.

Lowie De Malsche
Delft, June 2025

Contents

Glossary

back-propagation A method used to train a neural network by reducing the difference between the predicted output and the actual output [2]

concatenated Chained together or appended

densification A state in which foam starts behaving like a solid

epoch A complete pass through the entire training dataset [3]

hysteresis A lag in response exhibited by a system reacting to change [4]

sandwich panel A panel consisting of two face plates that are spaced apart using a core material

surrogate model A simplified approximation of a more complex model or system

Acronyms

CPU Central Processing Unit
ECSSMET European Conference On Spacecraft Structures, Materials And Environmental Testing
FC Fully-Connected
GP Gaussian Process
GPU Graphics Processing Unit
LSTM Long Short-Term Memory
MSE Mean Squared Error
NASA National Aeronautics and Space Administration
NMSE Normalised Mean Squared Error
NN Feed Forward Neural Network
NRMSE Normalised Root Mean Squared Error
PCA Principal Component Analysis
PSD Power Spectral Density
RMSE Root Mean Squared Error
RNN Recurrent Neural Network
SRS Shock Response Spectrum
SS State Space

Symbols

β Non-Dimensional Non-Linear Spring Constant
c Damping Coefficient
f Frequency
F Non-Dimensional Force
F_a Applied Force
k Spring Constant
k_3 Non-Linear Spring Constant
m Mass
ω Angular Velocity
ω_0 Resonance Frequency of the System
r Non-Dimensional Damping Coefficient
R^2 Coefficient of Determination
t Time
y_i True Value at Point i
$\bar{y}_i$ Average Value of y_i
$\hat{y}_i$ Predicted Value at Point i

1

Introduction

Launching sensitive equipment into space poses a series of challenges that must be overcome. One of these challenges is surviving in the harsh environment that can be found in the capsule on a launching rocket. To protect sensitive equipment from this environment, it is often packed in foam. This foam packaging is meant to isolate its contents from vibration, and shock loads that can be found in the launch environment. In order to assess if the foam packaging is sufficient, tests need to be performed. These tests take up time and resources, and are difficult to perform accurately according to the specification. Therefore a method was proposed to simulate the response of an object packed in foam to shock, and possibly vibration loads. The findings of this method can be found in a paper presented at ECSSMET 2024 [5]. The proposed method is to model the motion of the object packed in foam as a single degree of freedom mass-spring-damper system. A specified input shock or vibration can then be applied to this model, and from the simulated response, a SRS or PSD can be taken to determine what the internal loads on the object will be. Using this method, it can be determined what the expected loads inside the packaging will be. This can be used to either determine if the packaging is sufficient or if the shock or vibration loads are critical.

The results of this method were promising. However, the simple mass-spring-damper system was not able to model the response of a foam-packed object accurately at high amplitudes due to the non-linear nature of foam. To overcome this, a new model that accurately takes this non-linearity into account needs to be made.

The aim of this thesis is to find a model that can accurately model the behaviour of an object packed in foam through a wide range of amplitudes and for different amounts of foam. To achieve this a complex model of the system could be made where the foam and the interaction between the foam and the object is modelled in detail. However, this would require a solver that can solve for systems with extremely large deformations and complex interactions between the object in the foam and the foam itself. Moreover, to achieve this, a detailed model must be custom made for every object that is packaged in foam. Doing this is impractical as it is both time-consuming and expensive.

To overcome this issue, this thesis proposes the creation of a surrogate model to model the damping of foam packaging for a wide range of foam thicknesses and objects. The aim of this surrogate model is to make it both easy and computationally inexpensive to model the behaviour of an object that is packed in foam when subjected to a shock or vibration. While a range of methods to fit a model like this exist, for this thesis an AI-based surrogate model is made. The rationale behind this choice is that it is difficult to determine the underlying equations of motion as the system is highly non-linear and dependent on a large range of variables as is described in further detail in Chapters 2 and 4. A sufficiently complex AI-model is able to learn the way these variables are connected in a data-driven way. To achieve this, a range of AI-models is used as a solver and finally, the performance of these solvers is compared for both accuracy as well as the computational resources required.

The report is presented in the following structure. Chapter 2 contains the literature review. It contains the required background information on the concepts that are discussed in this thesis. The preliminary

work that was done to determine what models would be used as solvers for the final comparison is shown in Chapter 3. Chapter 4 details how the training dataset is obtained, what models are compared, how these models are structured, how the models are trained, and how the results are compared. The results of the different models are shown in Chapter 5. These results are discussed in more detail in Chapter 6 where the models are compared and a trade-off is made to evaluate what model is best suited. The findings of this thesis are summarised the conclusion in Chapter 7. Finally, recommendations on how these results should be used in future work are given in Chapter 8.

2

Literature Review

This chapter provides background information about what variables influence the dynamic behaviour of foam as well as background information about what methods and models can be used to create a surrogate model and how the performance can be evaluated. Background information about the variables that influence the behaviour of foam can be found in Section 2.1. Background information on how a neural network can be used to create a surrogate model can be found in Section 2.2.

2.1. What Variables Characterise the Behaviour of Foam

A shock test has a lot of variables which can influence the results of the test. To characterise the behaviour of the foam, these variables must be examined to see which have a meaningful impact and which can safely be ignored.

2.1.1. Foam Density

Foam Density varies a lot in different foams and has a large influence on the behaviour of the foam. The main effect of changing the foam density is found in the stiffness of the foam. As the density of the foam increases the stiffness of the foam increases with it as less of the volume of the foam is air [6, 7]. Additionally, the foam density has an influence on the deformation mechanics of the foam.

As foams compress their cells get loaded in compression and eventually buckle. In general, the lower the foam density the larger the cell size. The larger cell size of the low density foams means the cells of the foam will buckle at a lower strain. When the cells of a foam start to buckle, the apparent stiffness of the foam decreases dramatically, and the stress plateaus as the cells lose their ability to resist further compression until nearly all cells have buckled. A denser foam has smaller cells which can resist a higher stress before buckling [6].

Additionally, increasing the density of the foam decreases the amount of empty volume inside of the foam. This means that the densification strain, which is the strain at which the foam starts behaving like a solid, decreases [7, 8].

2.1.2. Temperature

Temperature can have an influence on the behaviour of foam. Many polymer foams are made from thermoplastic materials which can soften well below their melting temperature [9]. This has a large influence on the compression behaviour. While the foam does get softer, the buckling strain of the foam cells is not significantly altered given that the foam cells do not become brittle or degrade. As a result, the stress plateau starts at the same strain, regardless of temperature [6].

2.1.3. Strain Rate

Many polymeric foams are sensitive to strain rate. Hence, the effect of strain rate must be considered. When polymeric foams are exposed to a large sudden deformation, they become much stiffer. This happens for several reasons. First, the deformation of the polymer itself is largely dependent on the

strain rate. Secondly, the air inside of the foam acts as a cushion. When the foam is compressed, the air is either pushed out or compressed depending on the type of foam. These effects not only influence the stiffness but also cause hysteresis, which further increases the complexity of the system.

Although the stiffness increases with strain rate, the strain at which the stiffness plateaus remains roughly constant. As the strain rate increases, the strain at which densification occurs starts to decrease [8, 10].

2.1.4. Pre-Load and Pre-Strain

When objects are packed in foam, the foam is often pre-compressed, either from the object pressing against the foam or the the foam being strapped down. This pre-load has an influence on the ability of the foam to absorb shocks. When a foam is pre-strained, less energy is dissipated and more of the shock is transmitted [11]. Additionally, the pre-strain can have an effect on the eigenfrequency of an object that is packed in foam.

2.1.5. Fatigue Behaviour of Foam

The mechanical properties of polymer foam can vary significantly due to the effects of fatigue. As described in the section about foam density, polymeric foam usually has a stress plateau where the stress no longer increases due to increased strain. This plateau is caused by the foam cells buckling when the strain increases. When a foam has been strained up to this point, the cells that have buckled buckle at a lower strain the next compression cycle. Depending on the type of foam, this can dramatically reduce the stiffness of the foam [12].

Depending on the type of foam, this effect can be more or less severe. Foams like polystyrene or phenolic foam suffer severely from this fatigue effect, losing their stiffness even after the first compression. Meanwhile, polyethylene foams barely suffer from this effect, showing only very subtle signs of degradation [12].

2.2. How can a neural network based dynamic model be made

In recent years, machine learning has widely been used to model a multitude of complex processes in a wide range of fields [13, 14, 15, 16]. This section provides background information on what simulation methods are used, what types of AI-models can be used, and how their performance can be evaluated.

2.2.1. Simulation Methods

When it comes to modelling the behaviour of foam, two main methods show promise. The first method is to train a neural network to directly calculate the output to a given input motion. This is called a direct-solution model. The second method is to train a neural network to calculate the current motion given the current state of the system for each time-step. This can either be the motion directly or a derivative of the motion which is then integrated. This second method is called the time-stepper model [16].

A direct-solution model has the advantage that it can directly calculate the output of system without the need of running the model multiple times. This makes it relatively simple to implement and much faster to run. However, this comes at the downside that the model has a fixed simulation length, and requires constant initial conditions [16].

Time-stepper models are more versatile in the sense that they do not have a fixed length. This is an advantage as model training can be done on shorter timespans. Furthermore, different solvers can be used which can be favourable for different dynamical systems. For example, a direct solver can be used to directly get the required output for each step. Alternatively, a neural network can be trained to find the derivative of the desired output which can be integrated to find the output. Furthermore, if the system that is to be modelled has a known linear part, the linear part can be combined with the neural network to obtain a result that is based in part on a linear solution [16].

2.2.2. Types of AI Model

There are a large number of AI models which are used for various tasks. This section describes the models that were considered in this thesis.

The first type of AI model is the **NN** this is a basic form of neural network. In this form, the input is passed through a layer of nodes. The output of each node is the sum of the product of the inputs of the layer and a set of weights. Multiple layers can be combined in sequence to create a deep neural network. Activation functions are often used between the layers of the neural network. These activation functions are mathematical functions that are called on the output of each node in a layer. They are often used to introduce non-linearities into the model which can improve model performance when solving complex problems [17, 18].

The second type of AI model is the **RNN** this is a type of neural network that saves the output of a layer and uses it as an input the next time the model is run. This makes it very well suited to simulate a time domain function like a shock or vibration on foam [19].

The problem with training an RNN on a long time domain series is the vanishing gradient problem. This is a phenomenon where the gradients which are used for back-propagation vanish for the early samples in a long time domain series. A **LSTM** model can be used to overcome this issue [20]. An LSTM achieves this by storing information outside of the neural network. This allows it to be used for time domain series of over 1000 steps [21, 22].

The last type of AI model that is considered is the **GP**. This model works differently to both NN and RNN models. Instead of using nodes, a GP is a probabilistic mapping from an input to an output. The advantage of this for simulating a time domain series is that a GP is able to return a standard deviation. Using this standard deviation, a confidence interval can be made where the output of the real value is expected to be in. On the other hand, a GP is computationally very intensive. Calculating the variance takes $\mathcal{O}(n^2)$ time which makes it impractical for large datasets [23].

These proposed models can be used with different complexities. Special care should be taken when selecting the correct model complexity. When the model complexity increases, the error rate of the training data decreases. However, this may not be the same for the validation data. As the model complexity increases, the model starts to model the error and noise from the training data and becomes over-fitted. When an over-fitted neural network is used to model the validation data, an increase in the error will be observed. Special care must be taken to find the model complexity which optimises both the error rate of the training data, and the validation data [24].

2.2.3. Model Performance Metrics

Coefficient of determination (R^2)

The coefficient of determination, R^2, is a metric that can be interpreted as "the proportion of total variation about the mean that can be explained by regression" [25]. It is calculated using equation 2.1 [26]. The R^2 score ranges from 1 for a perfect fit to $-\infty$ for the worst fit. When R^2 is 0, the fitted model is not correlated with the data. When R^2 is smaller than 0, the fitted model is worse at predicting the data than a horizontal line [27, 28].

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.1}$$

Mean Squared Error

The MSE is a metric that is used to assess the accuracy of the predicted value compared to the true value. It is the average of the square of the error at every data point and can be calculated using equation 2.2 [26, 29]. The MSE ranges for 0 for a perfect fit to $+\infty$. Due to the square term of the MSE, a single very bad prediction can influence the MSE to a very large degree [27].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

A major downside to using the MSE is that it cannot be interpreted without context because it is dependent on the magnitude of the data. The value of the MSE should always be interpreted in context and cannot be used to directly compare the accuracy of two different predictions [27].

Root Mean Squared Error

The RMSE is obtained by taking the square root of the MSE as is shown in equation 2.3. The main difference between the MSE and the RMSE is that the RMSE has the same unit as the data. This makes the RMSE easier to interpret than the MSE [27]. The RMSE suffers from the same issue as the MSE where it should always be interpreted in context.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.3}$$

Normalised Mean Squared and Root Mean Squared Error

Based on the MSE and RMSE, the NMSE and NRMSE solve the problem with the MSE and RMSE of not being comparable without context by normalising the values [30]. There are multiple different ways to normalise the MSE and RMSE. However, for this thesis they are normalised by the mean square and root mean square of the real measurement respectively as is shown in equations 2.4 and 2.5.

$$NMSE = \frac{MSE}{MS} \tag{2.4}$$

$$NRMSE = \frac{RMSE}{RMS} \tag{2.5}$$

3

Preliminary Evaluation and Model Selection

Before creating a dataset to train a model, it is important to know what this dataset must consist of. To determine this, a range of models were trained on a synthetic dataset that could be created and altered easily. Going through this process reveals what variables are required to make the models before creating a dataset. Moreover, it reveals what types of model have the potential to serve as an accurate surrogate model for a foam-damped system. The chapter is structured in the following way. The synthetic dataset that is used to train the models is described in Section 3.1. What types of models are used is shown in Section 3.2. Finally, the individual results of these models and a comparison between them can be found in Section 3.3.

3.1. Synthetic Data

To train the preliminary models, a dataset that is representative of a non-linear foam-damped system is required. The system that is used for this purpose is described in Section 3.1.1. The dataset that is synthesised using this system is described in Section 3.1.2.

3.1.1. The Used System

In order to asses what models are best suited to model a non-linear systems a dataset of a non-linear system is needed. The non-linear system used for this purpose was a Duffing-oscillator. A Duffing oscillator was used for several reasons. Firstly, this is used by many other papers to compare the performance of AI-models, making it easier to verify if a simulation works correctly as the performance can be compared to other resources. Secondly, using a synthetic dataset with known parameters allows for complete control of the parameters of the system. This way, the influence of parameters like damping and non-linear damping on the model performance can be assessed.

A Duffing oscillator is a damped and driven oscillator that is represented by the second-order differential equation shown in Equation 3.1 [31]. It differs from a more classical driven damped oscillator by adding the non-linear term βx^3 which introduces non-linearity into the system. As the β term relates to the displacement of the system it serves the role as a non-linear spring constant. This non-linear term makes the Duffing oscillator very suitable for evaluating the performance of an AI-model on a non-linear system [32].

$$\ddot{x} + r\dot{x} + \omega_0^2 x + \beta x^3 = F\cos\omega t \tag{3.1}$$

The force, F from equation 3.1 is a non-dimensional force. This means that it is normalised to be massless, effectively making the force an acceleration. To return the dimensionality to the system, the equation should be multiplied by the mass, m, resulting in Equation 3.2 where F_a is the applied force

and k_3 is the non-linear spring constant where $m\beta = k_3$. Making the equation dimensional allows for a synthetic dataset to be made with parameters that have a physical meaning.

$$m\ddot{x} + c\dot{x} + kx + k_3x^3 = F_a \cos \omega t \tag{3.2}$$

A schematic representation of a Duffing oscillator with dimensional parameters is shown in Figure 3.1. The figure shows a mass connected to two springs, k and k_3, and a damper c. The applied force F_a is supplied through an imposed ground motion, z.

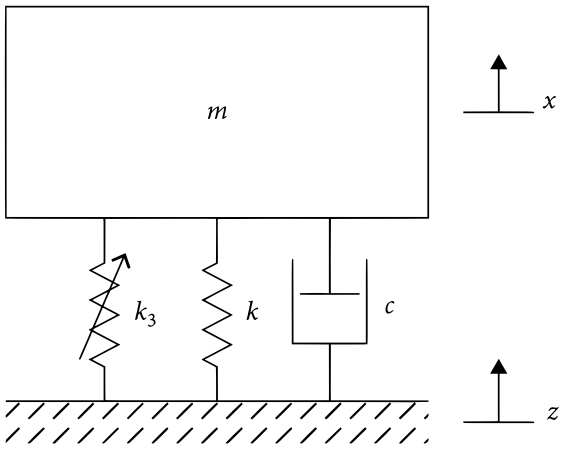


Figure 3.1: Schematic representation of a Duffing oscillator [33]

An additional benefit of creating a synthetic dataset is that it is much faster and more flexible than creating a real dataset. When the need arises to create a new dataset of a different size or with different parameters, this can be done in minutes where performing real tests might take days to do the testing and to process the data. Using a synthetic dataset before making a real world dataset ensures that when the real world dataset is created, the required variables are measured at the correct frequency for the correct amount of time.

3.1.2. The Synthesised Dataset

The synthetic data which was used to evaluate the performance of each model consists of a Duffing-oscillator with a varying input signal. This input signal is chosen to be a linear chirp signal. This is a frequency-swept cosine signal with a constant amplitude where the frequency, $f(t)$ follows Equation 3.3 where f_0 is the start frequency, f_1 is the end frequency, and t_1 is the end time.

$$f(t) = f_0 + \frac{f_1 - f_0}{t_1}t \tag{3.3}$$

The reason that a chirp signal is used over the signal with a single frequency seen in Equations 3.1 and 3.2 is that the response of the system to a varying forcing frequency shows the transmissibility of the system for each frequency. As a result, the AI models need to be able to model the behaviour of the system across a range of frequencies instead of at a single frequency. Furthermore, if the eigenfrequency of the system is within the sweep frequency range of the chirp signal, the chirp will excite the system before and after resonance. This makes it easier to assess how well the models are able to model the behaviour of the system around the resonance frequency of the system.

To create a full dataset, the amplitude of the applied force is incremented from $0.2\,\mathrm{N}$ to $10\,\mathrm{N}$ with $0.2\,\mathrm{N}$ increments. Then the non-linear spring constant is varied from a value of $0.0\,\mathrm{N\,m^{-3}}$ to $2.0\,\mathrm{N\,m^{-3}}$ with $0.2\,\mathrm{N\,m^{-3}}$ increments. This results in a total of 550 time domain signals. The parameters of the Duffing-oscillator dataset are summarised in table 3.1. It shows the the mass, m, the damping coefficient, c, the spring constant, k, the non-linear spring constant, k_3, the range of the applied force, F_a, and the

Frequency range of the chirp signal. This dataset is then used to train the AI models. The goal of these models is to determine the displacement of this system.

Table 3.1: Duffing Oscillator Dataset Parameters

m	c	k	k_3	F_a	Frequency Range
$1\,\mathrm{kg}$	$0.2\,\mathrm{N\,s\,m^{-1}}$	$1\,\mathrm{N\,m^{-1}}$	$0.0\,\mathrm{N\,m^{-3}}$-$2.0\,\mathrm{N\,m^{-3}}$	0.2 N-10 N	0.05 Hz-2.0 Hz

3.2. Evaluated models

To determine what models should be used to make the final models, several models are evaluated and their performance is compared qualitatively to determine what model should be used as the final model. Each of the models are used as a solver for a time-stepper method. The goal of these time-steppers is to find the displacement of the system based only on the input acceleration of the system. The used models can be broadly sub-divided into two categories NNs and RNNs. The used NN models are described in Section 3.2.1. The RNN models that are evaluated are described in Section 3.2.2. The models that were considered but not evaluated for the purpose of this thesis are described in Section 3.2.3.

3.2.1. NN Models

NN models are considered as a model for the time stepper solvers. While the goal of these solvers is to determine the displacement of the system, multiple approaches are possible. The first possibility is to use the NN to directly solve for the displacement of the system. The advantage of using this method is that no integrator is needed to solve the system. However, for a normal NN with no feedback this is not feasible. Because the system has inertia, there is no way to determine directly from the input acceleration what the displacement of the system is.

To solve this, the NN can be used to find the derivative of the system. With this approach an integrator can be used to integrate the velocity to find the displacement of the system. However, this approach suffers from inertia of the system as well as there is no way to find the velocity of a system using only the acceleration of a single time step as an input.

Finally, the NN can be used to find the acceleration, or the second derivative of the system. This second derivative can then be integrated using a double integral to obtain the displacement. This solves the inertial problems completely and allows for the NN to be used to translate the input acceleration of the system to the response acceleration of the system without having any information about the state of the system.

The main benefit of using a NN instead of an RNN is that the entire system can be solved in parallel, massively speeding up the required time to solve the model. The downside of using these models is that they have no information about the state of the system. It is therefore expected that the results of a time stepper using a NN as a model will be much less accurate than the results of a time stepper that uses a RNN as a model.

3.2.2. RNN Models

The main difference between a RNN model and a NN is that the RNN has information about the previous time steps it has been run. In practice this means that the model has information about the state of the system. Two forms of RNN are considered, a NN with the outputs of the previous time-step returned as the input of the current time-step, and a LSTM model. This LSTM model does not require the input of the previous time-step as it has an internal memory that learns through training what should be remembered and what should be forgotten.

The RNN models can be used in similar time-stepper set-ups than were described for the NN. However, in contrast to the NN models, the RNN models have information about the current state of the model. This means that these models can be used to directly calculate the output of variables that require information about the state of the system. Because of this a RNN can be used as a model for a direct solver time-stepper, single integral time-steppers as well as double integral time-steppers.

3.2.3. Models That Were Considered but Not Evaluated

Other models that have been mentioned in the literature review in Chapter 2 but are not considered are the direct-solution models as well as the SS+NN models. Direct-solution models can be a very good way to model the behaviour of a system. However, their major disadvantage is that they are not able to model the response of the system beyond the length of the direct-solution model. Furthermore, these models must be trained with constant initial conditions. If a situation arises where the initial conditions vary, a new model must be trained for the new initial conditions. This makes them not well suited for the intended use-case of this thesis as the length of the input shock or vibration can vary depending on the situation. Hence, direct-solution models are not used.

The other model, the SS+NN model, is a model that can be used as a model for a time-stepper solver. These models would combine the linear part of the response of a system in a SS model with the non-linear part of the response of the system represented by a NN model. This would reduce the amount of information the NN needs to learn about the system, allowing it to be less complex. Furthermore, the NN can be replaced by an RNN to improve the performance of the model further. While this method is promising and could provide great results, the implementation is more complex. Therefore this model is not used in the scope of this Thesis.

3.3. Results From Preliminary Evaluation

The results from the preliminary testing are shown in this section. Two types of results are considered. First is the results of the models when they are used to model a single system. This system is a Duffing-oscillator with a set value for k_3 of $1\,\mathrm{N\,m^{-3}}$. Out of the validation results, a result with a low amplitude, a middle high amplitude, and a high amplitude are shown. These amplitudes are chosen to show how the accuracy of the model varies with a changing amplitude. The goal of this model for a system with a single k_3 parameter is to show how well each model performs at modelling the system. Separate models are made for the system with a varying value of k_3 to compare how the performance changes when they are made to be parameter-conditioned. For the comparison of the performance of these parameter-conditioned versions of the models, a result with a middle high amplitude is shown for a system with a k_3 value of 0, 1, and 2. The parameter-conditioned version of the models is important as the intent of the thesis is to find one model that can be used to simulate foam of different thicknesses and surface pressures.

To maintain clarity in this chapter, plots of the time-domain results are moved to Appendix A, and the results for the middle amplitude of the model trained for a single value of k_3, and the results for the middle amplitude with a k_3 value of 1.0 for the parameter-conditioned models are repeated in this chapter. The results are shown per model in Sections A.1, A.2, A.3, A.4, and A.5 for the second derivative NN, direct RNN, first derivative RNN, second derivative RNN, and direct LSTM models respectively. As the difference in performance of the models is very large, the preliminary results are compared qualitatively rather than quantitatively.

3.3.1. Second Derivative NN

The second derivative NN model is the both the simplest and the fastest model of the models considered in this chapter. It works by using the time-signal as an input for every time-step, and calculating the second derivative, or in this case the acceleration, of the system as an output. This second derivative can then be integrated twice to obtain the displacement of the system. As a result, the only input of this model is the input signal. For the parameter-conditioned version of this model, the input signal is concatenated with the non-linear stiffness, k_3, for every time-step. A diagram of this model can be seen in Figure 3.2. Because of the model's structure, the second derivative for every time-step is independent of the other time-steps. This means that every time-step can be calculated in parallel making this model significantly faster than the other models seen in this chapter.

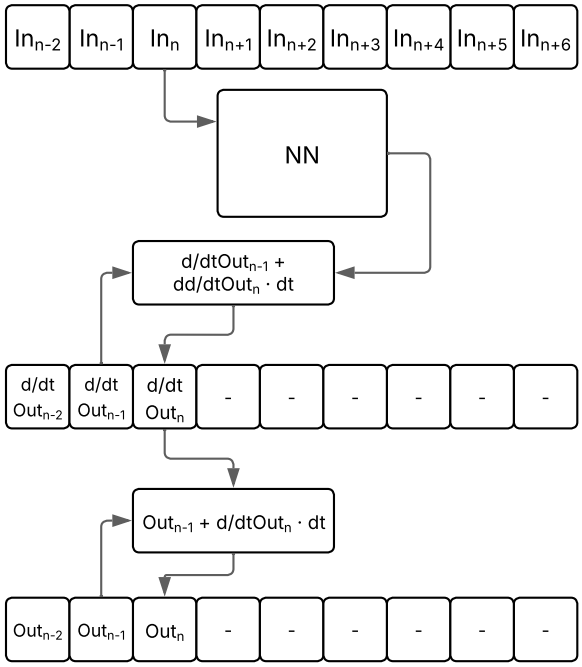


Figure 3.2: Diagram of second derivative NN model

Time domain plots of the response of the second derivative NN model can be seen in Figure A.1. The figure for the result with a middle amplitude is repeated in this section in Figure 3.3. The figures show that the displacement of the system is unstable. Each amplitude shows the displacement rising to a high positive amplitude, followed by a continuing decrease in the amplitude that shows no sign of slowing down. This result is an indication that this model is not usable to model the behaviour of a non-linear system.

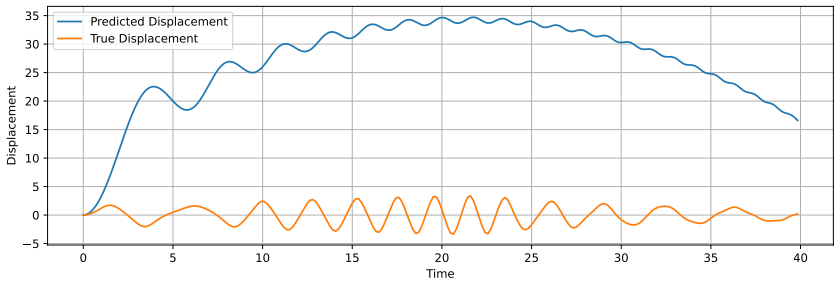


Figure 3.3: Middle amplitude time-domain result for second derivative NN model trained on a system with a single k_3

It should be noted that these results were expected as the model has no information about the current displacement or velocity of the model. This makes it impossible for the model to make corrections based on the current state of the simulated system, or to reliably return to the neutral point when the input acceleration is removed. The simplest way to solve this issue is to provide the NN with context about the state of the system. However, this would turn the model into a RNN. This exact model setup is discussed in Sections 3.3.2, 3.3.3, and 3.3.4. The downside to making this change is that it makes the output of the NN dependent on the previous time-steps meaning the time-steps must be run in sequence instead of in parallel, drastically slowing down the model.

The time domain plots of the parameter-conditioned second derivative NN model can be seen in Figure A.2. The figure for the result with a k_3 value of 1.0 is repeated in this section in Figure 3.4. The results show a very similar behaviour to the single parameter version of the model where the displacement that is predicted by the model first increases to a large positive amplitude and then decreases as time goes

on. While the model models the increasing resonance frequency of the system with an increasing value of k_3 to some extend by delaying the highest point of its displacement, showing that it does have some parameter-conditioning capabilities, the bad model behaviour that was seen for the single parameter model makes the model unusable for modelling a non-linear system.

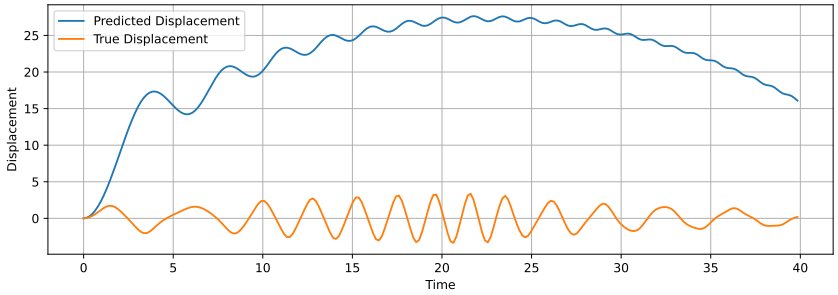


Figure 3.4: Time-domain results for parameter-conditioned second derivative NN model with k_3: 1.0

3.3.2. Direct RNN

The direct RNN model is the simplest of the RNN models. For every time-step it combines the input signal, that was used for the NN model, with the displacement of the previous time-step and uses those inputs to directly calculate the output displacement. A diagram of this model setup is shown in Figure 3.5. In contrast to the NN model, the RNN model does have information about the current state of the model. When used correctly this should solve the problem that was observed with the second derivative NN model where the displacement does not oscillate around 0. The downside of this is that the RNN model must be run in sequence per time step meaning the calculation cannot be parallelised.

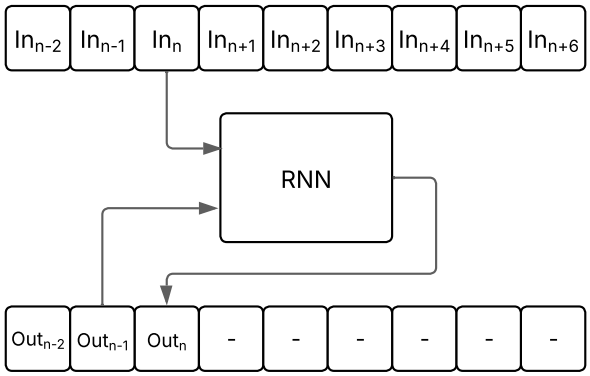


Figure 3.5: Diagram of direct RNN model

The results of the direct RNN model for the system with a single non-linear stiffness is shown in Figure A.3. The figure for the result with a middle amplitude is repeated in this section in Figure 3.6. The results show that the model is not able to correctly model the response of the system for all amplitudes. The model's performance for the lowest amplitude is the worst, and improves as the amplitude increases. It can be seen that the model performs best for the first few oscillations but quickly loses accuracy as the frequency increases above the resonance frequency of the system.

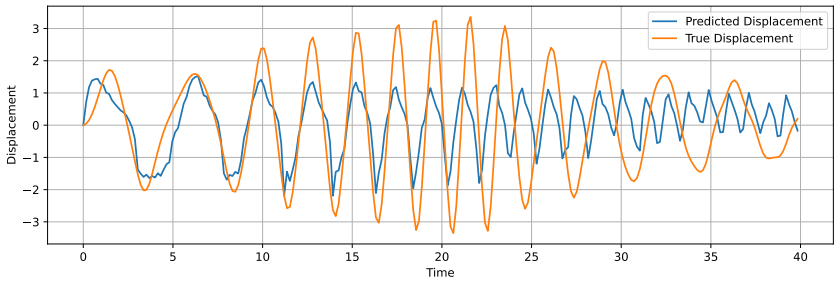


Figure 3.6: Middle amplitude time-domain result for direct RNN model trained on a system with a single k_3

One of the issues the direct RNN still suffers from in this configuration is that it has no information about the first derivative of the system. This makes it impossible for the model to know what direction the system is currently oscillating in. When the frequency of the forced oscillation is lower than the resonance frequency of the system, the system oscillates at the forcing frequency. This makes it easy for the model to model the response of the system as the system is oscillating at the same frequency the input signal is oscillating at. As the forcing frequency, and as a result the frequency of the input signal, increases above the system's resonance frequency, the oscillations of forcing frequency and the system's frequency diverge, and the model can no longer correctly model the oscillation of the system. This issue can be mitigated by providing the model with the system's velocity for every time-step which is used in the first, and second derivative RNN models in Sections 3.3.3, and 3.3.4 respectively.

The results of the parameter-conditioned direct RNN model can be seen in Figure A.4. The figure for the result with a k_3 value of 1.0 is repeated in this section in Figure 3.7. The results show a similar trend to the single parameter model which loses accuracy as the forcing frequency increases above the resonance frequency of the system. While the parameter-conditioned version of the model shows a slight change in amplitude between the response of the systems with different values of k_3, there is no major change in the system's behaviour across different values of the non-linear stiffness.

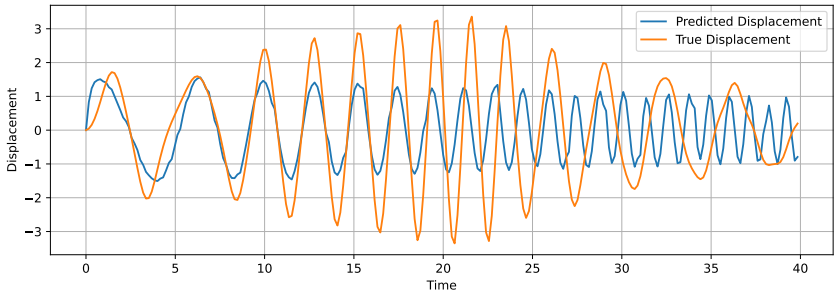


Figure 3.7: Time-domain results for parameter-conditioned direct RNN model with k_3: 1.0

3.3.3. First Derivative RNN

The first derivative RNN model is more complex than the direct RNN model. Instead of directly outputting the displacement of the system as its result, it outputs the first derivative. The inputs of this model are similar to the inputs of the direct RNN model with the addition of the first derivative, or velocity, to the input, combining it with the displacement. A diagram of this setup can be seen in Figure 3.8. The advantage of this approach is that the system's inertia is directly factored into the model's response. This makes the first derivative RNN, in theory, very well suited for simulating a dynamic system.

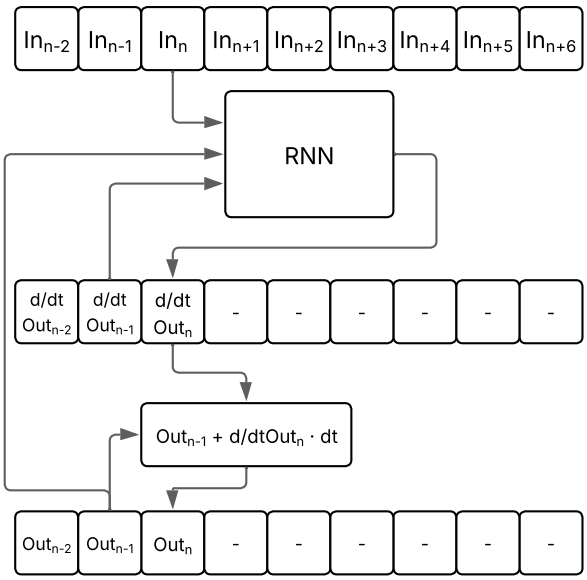


Figure 3.8: Diagram of first derivative RNN model

The time domain results of the model for a system with a single non-linear stiffness can be seen in Figure A.5. The figure for the result with a middle amplitude is repeated in this section in Figure 3.9. The results show that the system performs better than the direct RNN particularly with respect to the amplitude with increasing frequency. Like the direct RNN the model's accuracy improves with increasing amplitude. However, contrary to expectation, the first derivative RNN is still unable to correctly model the behaviour of the system as the forcing frequency increases above the resonance frequency.

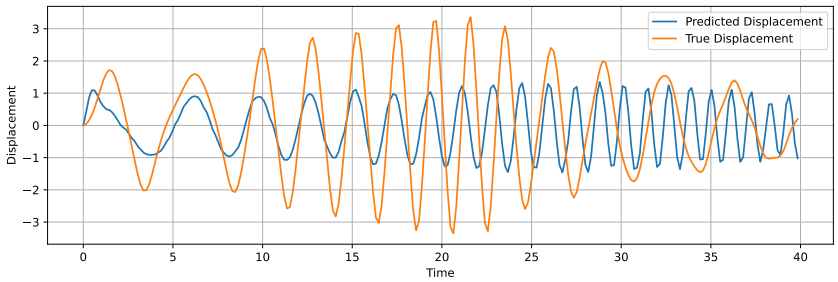


Figure 3.9: Middle amplitude time-domain result for first derivative RNN model trained on a system with a single k_3

The reason for this might be due to a lack of samples beyond this frequency. Given the system's increasing resonance frequency with increasing amplitude and the model's tendency to prioritise the model's accuracy for high-amplitude oscillations it is possible that there is not enough high amplitude data with a forcing frequency higher than the resonance frequency of the system. Moreover, the response of the system to an input signal with a forcing frequency that is higher than the resonance frequency of the system happens towards the end of the time domain simulation because the forcing frequency increases over time. The RNN that is used as a model is worse at optimising time-steps as the simulation gets longer due to the vanishing gradient problem which makes it difficult for models to find long-term dependencies. One of the solutions to make models more accurate over a longer time is to make them physics informed. For a physics informed model, the model is not just optimised for the data it represents but also to minimise residuals for the differential equation that represents the system. This makes the model better suited for long time extrapolation.

The results of the parameter-conditioned version of the first derivative RNN model can be seen in Figure A.6. The figure for the result with a k_3 value of 1.0 is repeated in this section in Figure 3.10. Similar to the results of the single parameter model, the model is more accurate for the cases where

the system has a higher resonance frequency. For the parameter-conditioned model this corresponds to the systems with a higher non-linear stiffness, k_3. For this model, the model not only changes the average amplitude for the different systems but the model shows a transmissibility that changes slightly as the forcing frequency increases for the system with the highest non-linear stiffness.

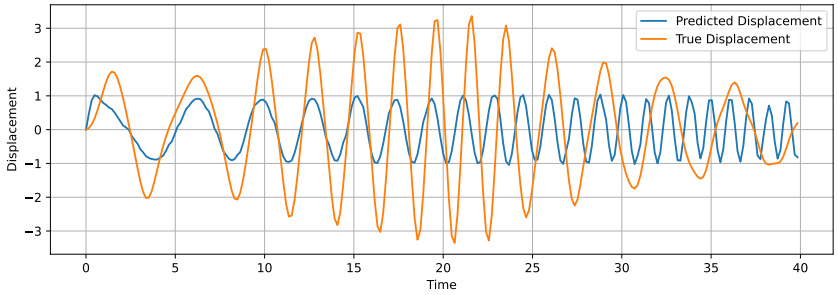


Figure 3.10: Time-domain results for parameter-conditioned first derivative RNN model with k_3: 1.0

3.3.4. Second Derivative RNN

The second derivative RNN goes one derivative lower than the first derivative RNN predicting the second derivative of the system or in this case, the acceleration. The model integrates the second derivative similarly to what was done with the first derivative RNN by adding the first derivative of the previous time-step with the integral of the second derivative of the current time-step. The same method is used to go from the first derivative to the displacement itself. The inputs of this model are the same inputs used in the first derivative RNN, combining the input signal with the displacement and velocity of the previous time-step as inputs to the model. A diagram of this set up is shown in Figure 3.11. It is expected that this model, while being the most complex of the RNN models also performs the best as it does not need to model the system's inertia and instead can directly model the system's acceleration.

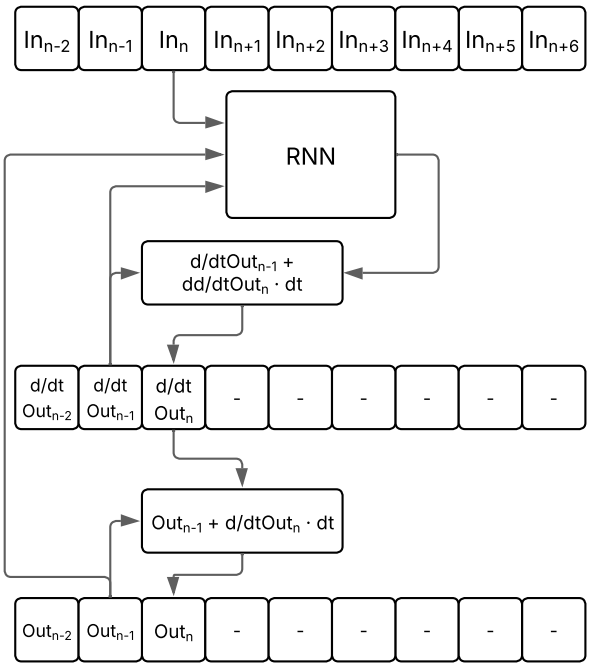


Figure 3.11: Diagram of second derivative RNN model

The time-domain results of the second derivative RNN for a system with a single non-linear stiffness, k_3 are shown in Figure A.7. The figure for the result with a middle amplitude is repeated in this section in Figure 3.12. The results of this system do not show a clear improvement over the single derivative

RNN. The model is still unable to capture the response of the system at a forcing frequency above the resonance frequency of the system. This is despite the fact that the model does not need to model the system's inertia which reduces the complexity of the information it needs to learn. A possible cause for the lack of improvement is the step-size of the time domain data. As the step-size increases the double integration of the second derivative becomes less accurate. It is possible that the model would become more accurate if it was trained with a smaller step-size. Similar to the first derivative RNN the model could be made more accurate over time by making it physics-informed.

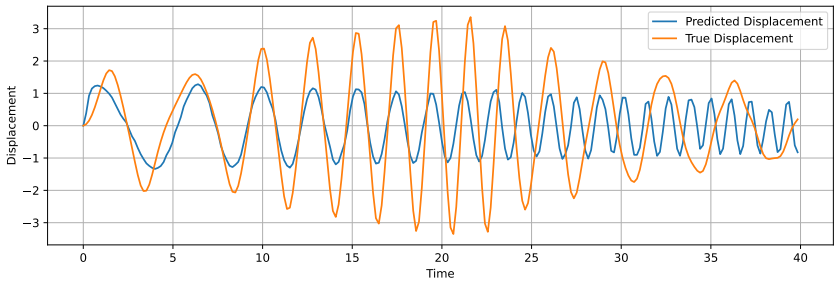


Figure 3.12: Middle amplitude time-domain result for second derivative RNN model trained on a system with a single k_3

The time-domain result of the parameter-conditioned second derivative RNN can be seen in Figure A.8. The figure for the result with a k_3 value of 1.0 is repeated in this section in Figure 3.13. When compared to the single parameter model, a notable difference can be found in the results of the parameter-conditioned model. Where the amplitude of the displacement of the single parameter model decreases as the forcing frequency increases, the amplitude of the displacement of the parameter-conditioned model stays roughly constant throughout the time domain, reducing the accuracy of the model.

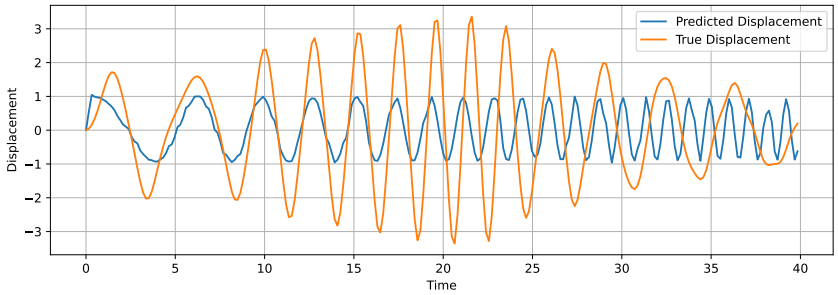


Figure 3.13: Time-domain results for parameter-conditioned second derivative RNN model with k_3: 1.0

3.3.5. Direct LSTM

The direct LSTM model is similar to the direct RNN model in the sense that it directly outputs the system's displacement, and that an LSTM model is a subtype of RNN. The LSTM's memory comes from the model's internal memory terms. This is in contrast to the other models that were used as RNN which get their memory from returning the output of the previous time step back as input for the current time-step. The advantage of this internal memory of the LSTM is that it is trained with the model meaning that the model learns what information is important and what information is not important. Additionally the LSTM can store memory for longer allowing the model to find long term relations that the other RNN models could not. This makes the LSTM model very well suited for simulating time-domain series. The set-up for this model is relatively simple. The input signal is provided with every time-step, and the output of the model is the displacement of the Duffing-oscillator. For the parameter-conditioned version of this the non-linear stiffness, k_3 is concatenated every time-step with the input signal. A diagram of this setup is shown in Figure 3.14.

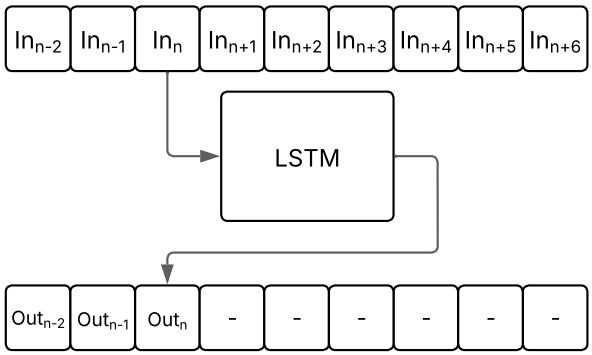


Figure 3.14: Diagram of direct LSTM model

The time-domain results of the direct LSTM model are shown in Figure A.9. The figure for the result with a middle amplitude is repeated in this section in Figure 3.15. The results show that the LSTM is able to capture the oscillation of the system at both low and high frequencies relatively accurately. Furthermore, in contrast with the other models, the LSTM model is able to model the behaviour of the system at forcing frequencies above the resonance frequency. Additionally, it correctly models the reduction in the transmissibility as the difference between the forcing frequency and the resonance frequency of the system becomes larger.

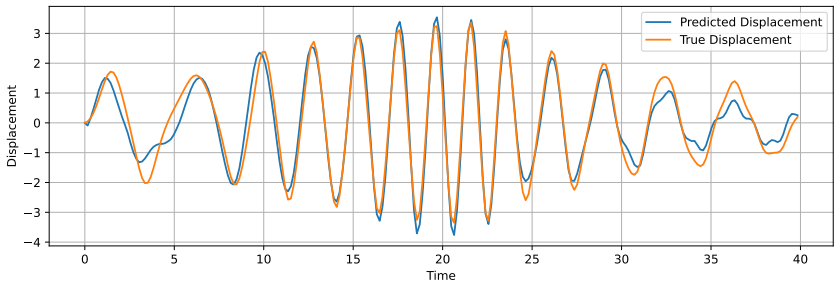


Figure 3.15: Middle amplitude time-domain result for direct LSTM model trained on a system with a single k_3

There are two large factors that contribute to the LSTM's ability to model the system's response at forcing frequencies above the resonance frequency of the system. First, the LSTM has dedicated memory terms which are trained with the model. As the model is trained these terms contain information about the system that can last for multiple time-steps, allowing the model to learn long-term dependencies. Secondly, the LSTM suffers much less from the vanishing gradient problem which makes it better suited at optimising longer time-domain simulations as well as variables that have long-term dependencies. These factors contribute to the model's ability to model the system very well.

The time-domain results of the parameter-conditioned direct LSTM model can be seen in Figure A.10. The figure for the result with a k_3 value of 1.0 is repeated in this section in Figure 3.16. The results of this show that the ability of the LSTM to model the system has not significantly been altered by the additional parameters. The LSTM is able to model the changing behaviour of the system with a changing non-linear stiffness very accurately. This result shows that the direct LSTM model is very well suited to be used as a parameter-conditioned model.

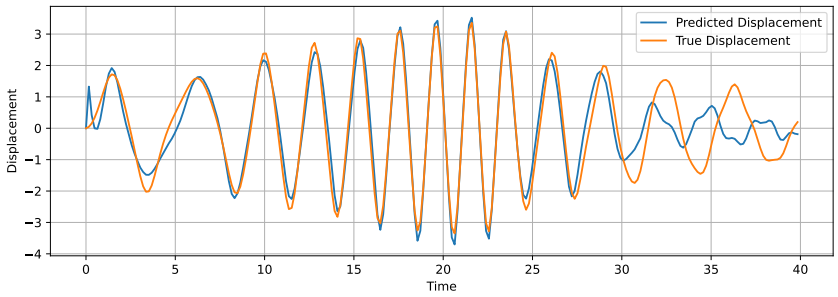


Figure 3.16: Time-domain results for parameter-conditioned direct LSTM model with k_3: 1.0

3.3.6. Comparison

When comparing the results of the different models it becomes clear that a simple NN is not suitable to model the behaviour of a non-linear system. Among the four RNN models that are considered the direct RNN performs the worst, often having large jumps and highly-irregular accelerations in the displacement. Additionally it is unable to model the behaviour of the system when the forcing frequency is higher than the resonance frequency. The first derivative RNN improves on the direct RNN by having the system's inertia built into the integrator. Furthermore, this model shows the transmissibility of the system increasing before the resonance frequency of the system and decreasing after the resonance frequency of the system. However, it is still unable to model the system's behaviour when the forcing frequency exceeds the resonance frequency of the system. Despite the expectations, the second derivative RNN performs worse than the first derivative RNN, losing the ability to model the transmissibility when the forcing frequency is around the system's resonance frequency. Finally, the direct LSTM performs the best by far, showing a continuous oscillation that has the correct amplitude at every frequency, even when the forcing frequency exceeds the system's resonance frequency.

The difference between direct LSTM and the other models becomes even more apparent with the parameter-conditioned models. Where all the other models perform even worse when the system gets another parameter, the LSTM model is able to model the system with a very high accuracy across all different values of the non-linear stiffness.

While it is still possible to improve the first derivative RNN model, some of the biggest improvements like making the model physics informed can also be applied to the LSTM model. Given the size of the performance difference between the LSTM model and the other RNN models, especially for the parameter-conditioned models, the increased accuracy of the LSTM model is well worth the slower performance of the model. Therefore, the LSTM model is used as a basis for all other models for the rest of the thesis.

4

Methodology

This chapter describes the methodology that is used to create the dataset used to train the models as well as the methodology used to make and train the models. The methodology used to create the dataset can be found in Section 4.1. How the models are structured with respect to the considered variables as well as the variants that are trained can be found in Section 4.2. How these models were implemented is described in Section 4.3. Different versions of these models were trained to determine how they differ. These different versions can be found in Section 4.4.

4.1. Obtaining The Dataset

In order to train the models, a relevant dataset is required. As no relevant data relating to foam was available, a series of tests had to be performed. This section describes the test set-up, what tests were performed, and the problems encountered during the tests.

4.1.1. Test Set-Up

As the simulation model is meant to represent shock tests on foam-packed items, the tests must be performed in a way that is representative of this case. In a real scenario, an object is typically mounted vertically on a large shaker. This shaker then performs a half-sine shock with a period of $3\,\mathrm{ms}$ with an amplitude depending on the test scenario.

Performing a test like this requires a shaker with a large range of motion that is powerful enough to produce these shocks. This typically requires a specialised testing facility with specialised equipment. However, making use of these test facilities can be very expensive. Given that the thesis has no budget, making use of a specialised facility like this is out of the question. As a result, all the tests were performed at the TU Delft DASML Dynamics Lab.

The largest shaker that is available at the Dynamics Lab as of writing is the Model 2025E modal shaker from The Modal Shop [34]. This shaker has a limited stroke length of $18\,\mathrm{mm}$ making it unable to perform high amplitude low frequency shock tests. Additionally, the shaker is not very large and is not suited to carry a mass vertically.

To compensate for these shortcomings the test set-up is changed from a vertical orientation to a horizontal orientation. Additionally, the test is changed from a half sine shock test to a vibration test. While this means that the data is not entirely representative of the real test set-up, it allows for high amplitudes to be tested due to the shorter required stroke length for a vibration test compared to a shock test.

As tests on multiple foam thicknesses are required, the test sample is made as follows. Two steel plates with a thickness of approximately $14\,\mathrm{mm}$ are placed between foam plates with a thickness of $28\,\mathrm{mm}$ each. To vary the total foam thickness, these foam plates can be added or removed. Tests were performed on a configuration with one, two, and three foam plates on both sides of the steel plates. To keep the foam and steel plates together, and to transfer a load, these plates are lightly pressed

between two aramid-aramid sandwich panels. These two sandwich panels are connected with four threaded rods. Two on the top side, and two on the bottom side. The rods on the bottom side are placed closer towards the centre of the sandwich panels to support the foam and steel plates. This bundle is suspended via a wire and aligned with the shaker. The shaker is connected to one of the sandwich panels through a load-cell/accelerometer combination. To measure the response of the steel plates in the foam, an accelerometer is placed at the centre of the steel plate closest to the shaker. An additional accelerometer was also placed directly on the shaker to be used as a reference. This described test set-up with a configuration with three foam plates is shown in Figure 4.1.

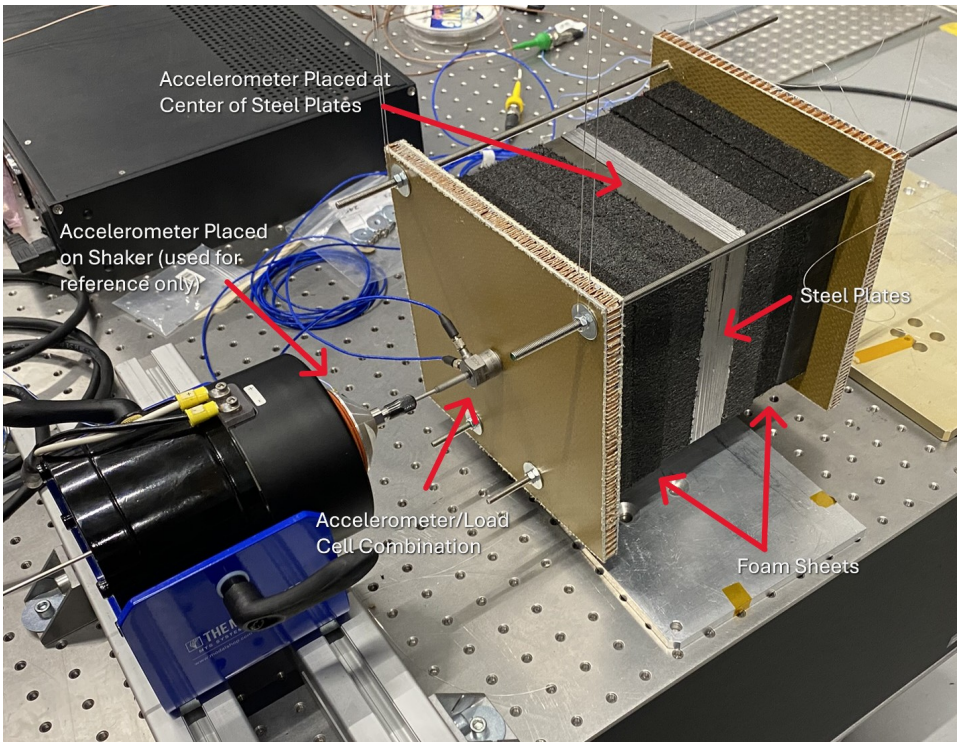


Figure 4.1: Test set-up in a configuration with three foam plates

4.1.2. Tests Performed

On this test setup, two types of tests were performed. The first is a chirp test. In this test, a sine motion with an increasing frequency over a given range is supplied to the shaker. The advantage of this test is that is excites all the frequencies in the sweep range ensuring the resonance frequency is found and excited. The second test is a random burst test. In this test, a white noise signal is supplied to the shaker with a given frequency range. The advantage of this test is that the noise is random. This avoids the possibility that an AI model is able to learn and anticipate the input signal.

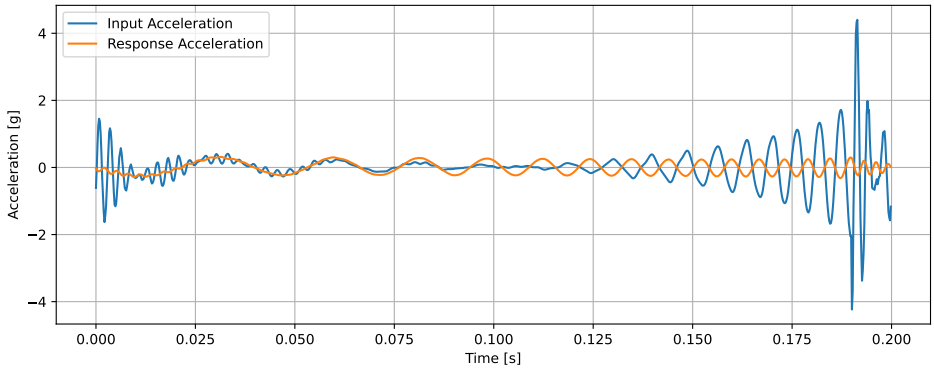
Because a time-stepper model must run through every time-step in sequence, these steps can for the most part not be run in parallel. As a result, making tests longer has a very large influence on the training time of the model. For this reason the tests are limited to a duration of $0.2\,\mathrm{s}$ and 500 time-steps. In order to obtain desirable results from these short tests, the chirp signal is divided into two tests. One test ranging between $20\,\mathrm{Hz}$ and $200\,\mathrm{Hz}$, and one test ranging between $200\,\mathrm{Hz}$ and $2000\,\mathrm{Hz}$. The random vibration test is performed four times per amplitude in the range of $20\,\mathrm{Hz}$ and $2000\,\mathrm{Hz}$. Both the chirp and random burst tests were performed at increasing target maximum amplitudes ranging from $5\,\mathrm{g}$ to up to $25\,\mathrm{g}$ with four steps in-between. Exact step sizes between these values are difficult to determine as the shaker amplitude scales non-linearly with an increasing supplied voltage. These tests are repeated for each of the tested foam thicknesses.

As a way to reduce the amount of data that is used in the GP layer of the model, a reduced dataset was created where only two out of the four random burst tests per amplitude were kept. Furthermore,

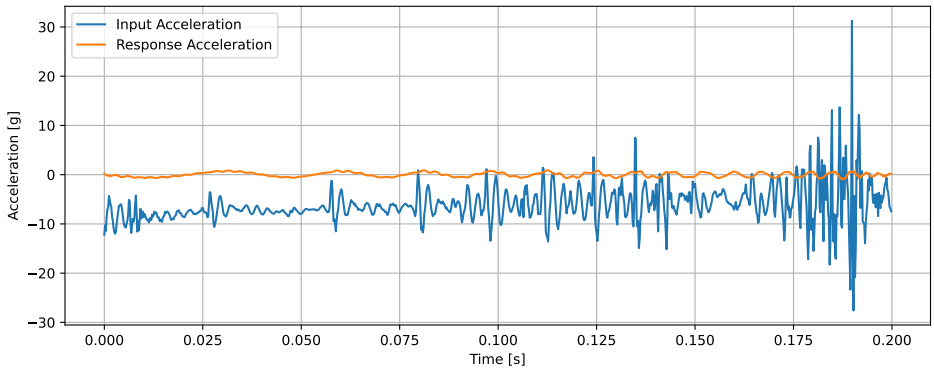
it was decided to only train the models on the random burst dataset as this input signal is completely random and the risk of problems related to learning the input signal can be avoided.

4.1.3. Issues During Testing

While performing tests for the smallest foam thickness, a metallic noise was heard during testing. It is unclear what caused this noise, however, it is observable in the measured input data as noise. Figure 4.2 shows an example of this. Figure 4.2a shows a chirp signal at a low amplitude that has no observable issues. Meanwhile, Figure 4.2b shows a chirp signal that was performed at a higher amplitude. The shape of the high amplitude chirp signal is supposed to be identical to the shape of the first signal.



(a) Low-amplitude chirp with little noise



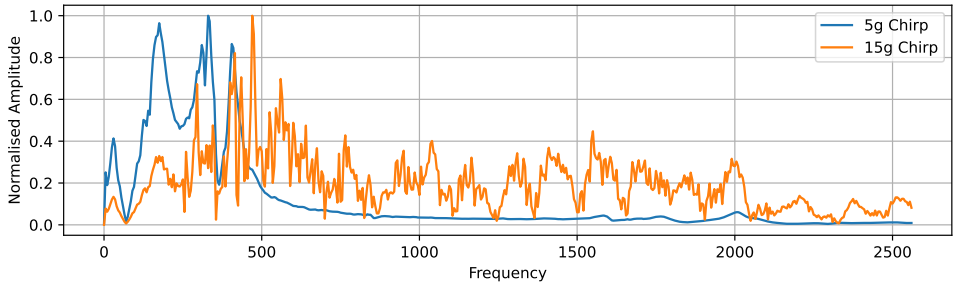
(b) High-amplitude chirp with high amount of noise

Figure 4.2: Example of low- and high-amplitude chirp signal for the test sample with 1 foam plate

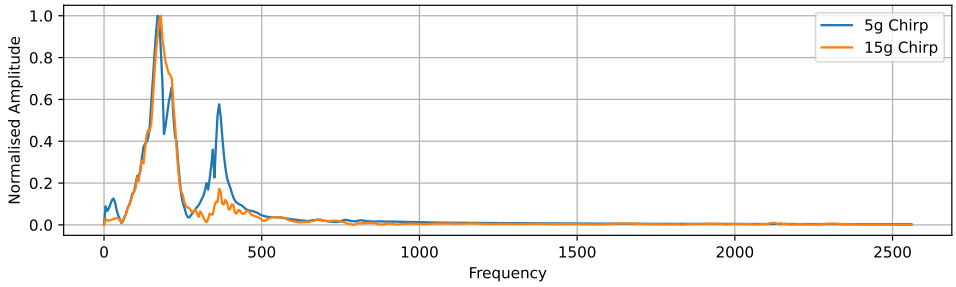
Not only is the signal of the high amplitude chirp signal heavily distorted, the acceleration is entirely skewed towards an amplitude between $-10\,\mathrm{g}$ and $-5\,\mathrm{g}$. Furthermore, the response acceleration of the mass between the foam plates does not show this skew and seems to only be lightly affected by this phenomenon. The applied acceleration is consistent with a measuring error. However, there seems to be some effect on the response acceleration which is also distorted.

The frequency domain of the input acceleration of four of the chirp tests can be seen in Figure 4.3. This input acceleration is supposed to scale linearly with in increasing amplitude, only being influenced by the resonance of the shaker. Figure 4.3a shows the normalised amplitude of the frequency domain of two chirp tests with $5\,\mathrm{g}$ and $15\,\mathrm{g}$ amplitudes for the test case with a single sheet of foam. Figure 4.3b shows the normalised amplitude of the frequency domain of two chirp tests with $5\,\mathrm{g}$ and $15\,\mathrm{g}$ amplitudes for the test case with three sheets of foam.

The figures show that the shape of the frequency stays roughly the same across amplitudes for the test case with three sheets of foam. This means that for this test case the input scales linearly as is expected. However, for the test case with only a single sheet of foam the shape of the frequency spectrum of the inputs with two different amplitudes is drastically different. The lowest amplitude has



(a) Frequency domain of test case with $28\,\mathrm{mm}$ of foam



(b) Frequency domain of test case with $84\,\mathrm{mm}$ of foam

Figure 4.3: Frequency domain of the measured input acceleration of the shaker during chirp tests with a maximum amplitude of $5\,\mathrm{g}$ and $15\,\mathrm{g}$

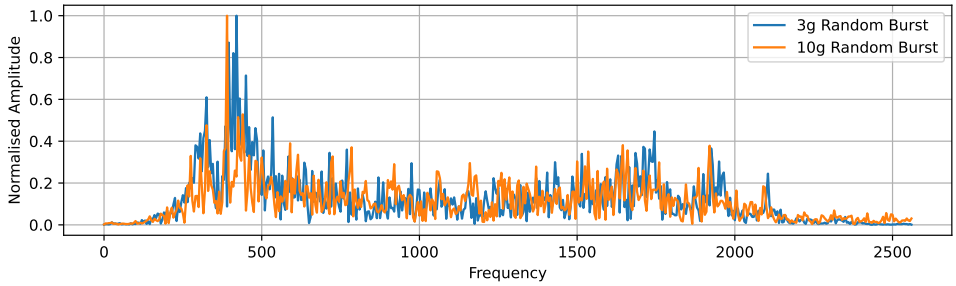
a shape that is comparable to the shape of the frequency domain of the input acceleration of the test case with three sheets of foam. Furthermore, the amplitude of the frequency domain of input with the lowest amplitude shows very few large jumps. Meanwhile, the amplitude of the frequency domain of the input with the highest amplitude is spread across a wide range of frequencies with large jumps in amplitudes between frequencies. This shows that the observed noise has a wide frequency range and is not a tonal noise which would be visible as a large peak around a single frequency.

To complete the comparison, the frequency domain of the input acceleration of four of the random burst input accelerations is shown in Figure 4.4. The input accelerations of a random burst with an amplitude of $3\,\mathrm{g}$ and $10\,\mathrm{g}$ is shown for the test case with a single sheet of foam and three sheets of foam in Figures 4.4a and 4.4b respectively.

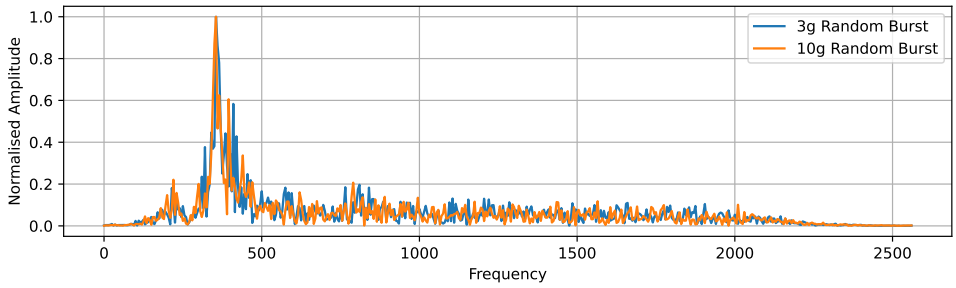
Unlike for the chirp signal, the shape of the low amplitude random burst an the high amplitude random burst is very similar within the same foam thickness. However, there are still differences across the test case with a single foam sheet and the test case with three foam sheets. While the amplitudes of the frequency spectrum of the input acceleration for the test case with three sheets of foam has large jumps in the amplitudes, they are smaller than the jumps that can be seen for the test case with a single foam sheet. Moreover, for the test case with three foam sheets the relative amplitude of the higher frequencies above $500\,\mathrm{Hz}$ stays below $20\,\%$ of the amplitude around the resonance frequency of the shaker. Meanwhile, for the test case with a single sheet of foam, the amplitude in this same frequency range reaches as high as $40\,\%$ of the amplitude around the resonance frequency. This shows that the noise that was observed in the chirp tests is also present in the random burst tests across all amplitudes.

To solve this issue, the tests with the single sheet of foam were re-done. However, the same problem was observed in the new test data, indicating that this issue was not random chance and has to do with the test set-up. To mitigate this issue, chirp tests that show this issue were removed from the dataset. However, random burst results are kept as the noise is present for all amplitudes. How this affects model performance is discussed in more detail in chapters 5 and 6.

A possible cause of this error is that the pre-compression of the foam caused by the load-cell connected to the sandwich panel and the accelerometer connected to the mass clamped inside the foam, illustrated in Figure 4.1, causes the foam to compress to a point approaching densification resulting in the load cell and the accelerometer touching or almost touching. However, if the load-cell and the accelerometer



(a) Frequency domain of test case with $28\,\mathrm{mm}$ of foam



(b) Frequency domain of test case with $84\,\mathrm{mm}$ of foam

Figure 4.4: Frequency domain of the measured input acceleration of the shaker during random burst tests with a maximum amplitude of $3\,\mathrm{g}$ and $10\,\mathrm{g}$

on the mass inside the foam did indeed touch during the vibration a transient spike would be visible on the response measurement which is not visible in Figure 4.2b. This is therefore unlikely to be the real cause and the true cause is still unknown.

4.2. Model Structure

In Sections 3.2 and 3.3 a range of models are presented and their performance to simulate a non-linear dynamic system evaluated. The results of these preliminary models show that an LSTM model should preferably be used. In order to use the LSTM model to simulate the response of an object packed in foam, what variables are considered and how they are considered must first be determined. In this section, it is discussed if the variables from Section 2.1 are implemented in the model and how they are considered.

4.2.1. Foam Variables

The first variable is the **Density of the Foam**. The foam density from Section 2.1.3 is an important variable that is strategically used to tune the eigenmodes of foam-packed items. Therefore, it must be considered. In practice, only a limited number of foam types and densities are used at Redwire Space NV. Because the foam density is not continuously distributed, different models should be made for each foam type. This reduces the required model complexity and training time per model while allowing for larger variation in the foam types in case they have a different chemical composition.

The next Variable is the **Environmental Temperature** from Section 2.1.2. While temperature can have an effect on the mechanical properties of the polymer the foam is made of, in the real world, foam packed items are continuously kept in a controlled environment where the temperature should be within a range of no more than $12\,\mathrm{K}$ [35]. Because of this, the temperature can be assumed to be roughly constant and the temperature must not be considered if the experimental data is obtained in a room within the same temperature range as the launch environment.

The **Strain Rate** from Section 2.1.3 is a very important variable and cannot be assumed to have no effect [5]. Therefore a model must be chosen that takes the effect of this strain rate into account. This is discussed in further detail in section 4.2.3.

The **Pre-Strain** from Section 2.1.4 has a large effect on both how much of the applied shocks and vibrations can be absorbed by the foam, and on the eigenfrequency of a foam packed item. While it can't be ignored, it is impractical to measure the pre-strain. Therefore, the pre-strain cannot be included in the model. To compensate for this oversight in the model, it is vital that a foam packed item is mounted in the test setup in a way that is consistent, and representative of the real world environment. If this is not the case, the test cannot be considered valid.

Lastly, the **Fatigue Behaviour** discussed in Section 2.1.5 can have a significant impact on the behaviour of foam. However, as was discussed in Section 2.1.5, the effect of this fatigue is much lower or even negligible in polyethylene foams. As NASA suggests the use of polyethylene foam in the requirements for ISS payloads, the effect of fatigue can be ignored [35].

4.2.2. Other Variables

In addition to the variables discussed in Sections 2.1 and 4.2.1, The thickness of the foam packaging, the mass of the object packed in the foam, and the surface area of the foam and the object must be considered in the model as well.

As the **thickness of the foam** increases, the perceived stiffness of the foam packaging decreases. As a result this makes the eigenfrequency of the foam packed item decrease. Furthermore, due to the increased thickness, the relative strain for a given displacement decreases. This reduces the non-linear effects related to strain and strain-rate that were discussed in sections 2.1 and 4.2.1.

As the **mass of the object** packed in foam increases, the inertia of the object increases. As a result, the eigenfrequency decreases as well. However, because of this, the object has more energy which must be damped which results in a higher strain, increasing non-linear effects. This means more foam might be required to sufficiently damp the object.

The **surface area of the foam** plays an important role in damping. It is used in combination with the mass of the object to determine the surface pressure. If for a constant object mass, the surface area of the foam increases, the surface pressure of the object on the foam decreases and the foam will be perceived as stiffer and will be able to damp more mass.

4.2.3. Model Structure With Respect to the Variables

The variables discussed in Sections 4.2.1 and 4.2.2 are used in the following way as shown in Figure 4.5. For every time step, the model receives the applied acceleration, the foam thickness, and the surface pressure on the foam as an input. This surface pressure combines the object mass and the surface area of the foam. It is a measure of how much mass is spread over the contact area of the foam. While the applied acceleration varies for each time-step, the Foam thickness and surface pressure are constant. As such, the foam thickness and surface pressure act as state parameters which provide context for the model. This way the same model can be trained as a parameter-conditioned model and used across a range of multiple thicknesses and surface pressures.

The strain rate of the foam is considered by the model itself. By training the LSTM it learns how to correctly represent the strain rate of the system in its memory. The foam type is a combination of the foam density and the material the foam is made of. As there are only a limited number of foam types used by Redwire Space NV., the foam density cannot be considered a continuous variable and using this as a parameter from the model is illogical and would only make the model more complex. Therefore separate models should be trained for every foam type.

The environmental temperature and pre-strain of the foam are not considered in the model itself as is discussed in Section 4.2.1. Instead, they are considered when selecting training data. During a test to obtain training data, it must be ensured that the temperature and the pre-strain of the foam are consistent for the environment the foam is expected to be used in.

Lastly, the acceleration of the object in the foam as a response to the applied acceleration is the output of the model. Using this model structure, the model can be run without knowledge of the real packaging or resonance frequency of the system that is to be analysed. The advantage of this method is that this model could be used to determine if a proposed type of packaging will be sufficient or what type of packaging should be used.

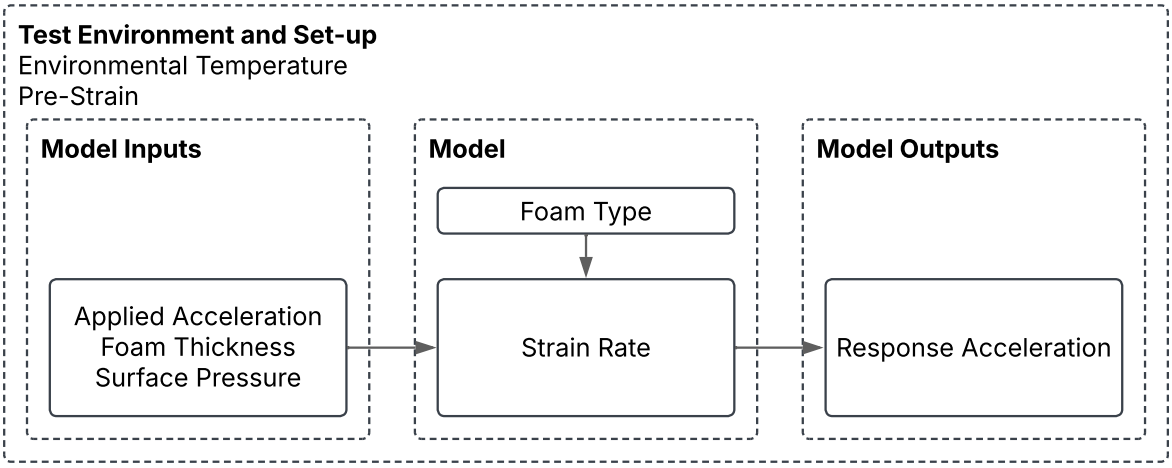


Figure 4.5: The structure of the model with respect to the different variables that are considered

To reduce the model complexity for the purpose of this thesis, only the foam thickness is considered as a state parameter. This was done because every additional state parameter increases the amount of test data exponentially. For example, if for one surface pressure five foam thicknesses must be tested, adding five surface pressures increases the required number of tests from five to 25. Having only one state parameter instead of two reduces the complexity of the model drastically while still proving that the model is able to determine the meaning of a state parameter within a single model.

4.2.4. Model Variants

An additional benefit to using a LSTM model as solver is that the internal memory terms of the LSTM means that the recurrence of the model is not dependent on obtaining the model's final result. This means that any step after the LSTM layer of the model can be run in parallel after the LSTM layer has gone through every time-step. The benefit of this is that a GP layer can be used after the LSTM layer to obtain the final acceleration. The effect of this is that the GP layer not only returns the acceleration but a variance as well, giving an indication in what range the result is expected to be.

The reason why a GP is not considered as a solver by itself is because during training, the inputs and outputs of the GP must all be fitted at the same time. This makes it impossible to use the GP layer as a solver for a time-stepper model on its own. Instead the GP must be used for a direct-solution which has the disadvantage that the length of the time domain cannot vary, and thus the model cannot be used for extrapolation.

As was discussed in Section 2.2.2 using a GP is very computationally expensive. To mitigate this problem the input dimension of the GP can be reduced. In this case, the output of the LSTM layer can be reduced before being used as an input for the GP which in this case works as a layer. For this thesis, two methods to reduce the dimension of the output of the LSTM layer were used. The first method is to use a FC layer. This layer must be trained with the model in order to accurately reduce the dimension of the output of the LSTM layer. The second method is the use a PCA as a layer on the output of the LSTM layer to reduce the input of the GP layer. Unlike the FC layer, a PCA is not trained and is instead fitted every epoch. In practice, the largest difference between using a FC layer and a PCA layer is that the FC reduces the dimension in a way where potentially no information in the data is lost. Meanwhile the PCA layer reduces the dimension by finding a new set of orthogonal vectors representing the data, and then only maintaining the most important dimensions, discarding the least important dimensions. This means that the PCA reduces the dimension in a way that reduces the amount of information in the system.

The three variants of the model that result from this are the LSTM model which starts with a LSTM layer followed by a FC layer that reduces the output of the LSTM to the output dimension. The second type is the LSTM-FC-GP model which starts with an LSTM layer, followed by a FC layer which reduces the output dimension of the LSTM layer to the input dimension of the GP layer which directly outputs the

output of the model. Finally, there is the LSTM-PCA-GP model which is similar to the LSTM-FC-GP model but uses a PCA layer to reduce the output dimension of the LSTM instead of a FC layer.

4.3. Training the Models

To make the results form the trained models comparable, the models must be trained in a standardised way. This section describes how the different models were implemented and how they were trained.

4.3.1. Model Implementation

To make training and running the models that were presented in Section 4.2.4 easier, the models are stored in an object that stores the models, as well as the relevant parameters such such as dimensions as well as input and output scalers. This makes it easier to store, load, and use the models. This is structured as follows. Each model has a model container. This model container stores information about the model itself such as input dimensions, output dimensions, input scale, and output scale. The source code of this model container can be found in Appendix Section B.1.

This model container serves as a base class for each of the three model variants. The LSTM, LSTM-FC-GP, and LSTM-PCA-GP models each add their own relevant variables well as a forward function which is called every time the model runs. The source code for each of the model classes can be found in Appendix Section B.2. Storing the models in this way ensures that when saved, everything needed to run the model, including the scalers, can be found in one object.

Finally, every model variant has a corresponding model executor. The model executor contains the instructions to train and run the models. Providing functions that directly train and run the models ensures that a program used to train or run the models only needs to provide the data, reducing the possibility for errors when training different models on the same data. Additionally, the model executor has a save and a load function allowing for other programs to easily load the executor, and obtain results from the models simply by providing the run function with the desired time domain input. The source code of this model executor can be found in Appendix section B.3.

4.3.2. Training Setup

The models were trained and tested on a machine running windows 11 with an Intel i7-8700k, $64\,\mathrm{GB}$ of ddr4 memory and an NVIDIA GTX 1070Ti with $8\,\mathrm{GB}$ of video memory.

In order to compare the models that are presented in Section 4.2.4 they must be trained on the same dataset. This limits the use of the test data for all models to the reduced dataset described in Section 4.1.2. Due to this small dataset, a traditional train-test split in the data is not feasible and K-fold cross-validation is used.

A K-fold cross validation is performed by splitting the dataset into k folds. Then, k iterations of training and validation are performed where for each iteration 1 fold is used for validation and the other folds are used for training [36]. This reduces the possibility that favourable or unfavourable results are due to a favourable or unfavourable train-validation split of the dataset.

To train the models, the reduced test dataset is first loaded and split into 5 folds. These splits are done per foam thickness to avoid an uneven distribution between different foam thicknesses across different folds. Furthermore, the data is shuffled between every foam thickness to avoid folds having similar amplitudes across the different foam thicknesses. The number of 5 folds was chosen because this ensures a $20\,\%$-$80\,\%$ validation-train split for every time the model is trained.

The models are trained for every iteration using the model executors described in Section 4.3.1. As the training progresses, the learning rate of the optimiser is reduced at a set interval. This interval is carefully chosen to allow for the model to continue training at a reasonable pace without running the risk of increasing the learning rate too soon. Using this approach ensures the model can train at a learning rate small enough to get a high accuracy while also optimising at a fast rate at the start of training.

Unfortunately, training the GP models with the reduced dataset still exceeds the memory limit of the available GPU. This means that the model must be trained on the the CPU instead. This drastically increases the amount of time required to train the model. To overcome this issue, an optimisation is used where only a smaller section of the outputs of the LSTM layer is used to train the GP layer. In this

case, one out of every four time steps is used during training for the GP layer. Which time step is used is iterated in sequence every epoch. This optimisation drastically speeds up the training process while maintaining a high accuracy.

4.4. Different Models Trained

To evaluate how the performance of the models changes with changing parameters, different models were trained with changing parameters. The results of these models are then presented and compared to the default model in Chapter 5.

4.4.1. Default Model

To compare the results with changing parameters, a baseline is required. The default models are trained as follows, The LSTM model is trained over 4000 epochs. During this training the learning rate starts at 0.0025 and is divided by $\sqrt{2}$ every 400 epochs.

The LSTM-FC-GP and LSTM-PCA-GP models are trained over 2000 epochs, which was chosen as the training time at 2000 epochs is similar to 4000 epochs for an LSTM model. The input dimension of the GP layer is chosen at 8 which during preliminary testing proved to be a good compromise between model accuracy and complexity. The models were trained on $1/4$ of the time steps, skipping $3/4$ of the outputs of the LSTM layer. This number was chosen because it allowed the models utilising a GP layer to be trained on the system's GPU.

4.4.2. Models Varying Number of Skipped Time Steps

To determine the influence of the amount of steps skipped by the GP layer, the amount of time steps skipped was changed to $1/2$ and the models with a GP layer were re-trained for the first fold. The number $1/2$ was chosen as it is expected that an increase in the number of time steps available to the GP layer during training will allow the models to fit the data better. The results of this model will show if this is true and how much better the models will be fitted. Training the models on this increased amount of time steps increased the total training time by a factor of 5. Increasing the fraction of time steps used further would result in training times that are unacceptable. Therefore, the models were not re-trained with all the time-steps used as an input for the GP layer.

4.4.3. Models Trained Without Noisy Data

To determine if the noisy test data that was discussed in Section 4.1.3 has an impact on the accuracy of the models, the models were re-trained excluding the data from the test case with a single sheet of foam. As this data is not replaced a consequence of this is that the dataset is reduced in size and that the models are trained with less training data. To maintain consistency between the results of different models, the numbers used to label the validation data for each fold are kept the same. As a result, the validation data for these models is numbered 3 to 6.

4.4.4. Models Trained With Varying GP Input Dimension

To determine the effect of the input dimension of the GP layer of the models, the models with a GP layer were re-trained using a reduced GP input dimension of 6 and 4. These numbers are chosen because they reduce the amount of memory required to run the model while showing the effect of reducing the input dimension of the GP layer. Increasing the GP input dimension was not possible as it would increase the required video memory to run the model by a too large amount.

4.4.5. Models Trained With Early Stopping

In order to get an accurate measure of the best potential performance of the model, and not just the behaviour during training, the models were re-trained using early stopping. Here, the performance of the models was evaluated using the validation loss. Each time the validation loss reaches a new lowest value, the model weights are saved. If the validation loss does not decrease for 600 epochs for the LSTM model or 100 epochs for the models with a GP layer, the training is stopped and the model weights with the lowest validation loss are restored.

The reason these models are not used as the default model is to identify how the models behave when they keep training. To compare the effects of the different variables on performance aspects

like over-fitting the default models are trained for often significantly longer than the models using early stopping.

Finally, to determine the effect of the noisy data on the total accuracy of the models, the models are re-trained with both early stopping and having the noisy test data removed, as was described in Section 4.4.3. The results of these models are then compared to the performance of the other models that employ early stopping.

4.4.6. Models Trained for Half the Duration of the Training Data

The main reason for using a time-stepper method is that it can be used to simulate a system for a varying duration. To determine if the models retain their accuracy when they are used to simulate a time-domain response that is longer than the duration of the training data, the models are re-trained for only half the duration of the training data. Then, the models are used to simulate the response of the validation data for the full duration. The accuracy of the first half of the simulated response of the validation data is then compared to the accuracy of the second half of the simulated response of the validation data.

5 Results

This chapter includes the results produced by the different models introduced in Section 4.4. It shows different metrics such as NRMSE, and the size and accuracy of the confidence interval for the models utilising a GP layer. The results in this chapter are presented with a minimal amount of comparison, and without a comment commenting on if the obtained results are good or bad. The discussion of the results presented in this chapter can be found in Chapter 6 which follows the same structure. Readers are advised to reach this chapter and Chapter 6 side by side.

The chapter is structured in the following way. The accuracy of the default models is shown in Section 5.1. The effects of varying the amount of output steps of the LSTM layer the GP layer skips during training is shown in Section 5.2. The effect of the noise in the dataset that was observed in Section 4.1.3 is shown in Section 5.3. How varying the input dimension of the GP layer influences the models is shown in Section 5.4. The accuracy of the models when over-fitting is prevented is shown in Section 5.5. How the models perform when they are used to extrapolate past the duration of the training data is shown in Section 5.6. Finally, the computational performance of the models is shown in Section 5.7.

5.1. Default Model Accuracy

To establish a baseline for the accuracy of the models, the results of the default models described in Section 4.4.1 are shown first. Table 5.1 shows the summarised results of the K-Fold cross validation for each of the models. The NRMSE score of every model is shown for each fold in the columns of the table. The last column shows the average for each model across all folds. The average score of the LSTM model is the highest at 0.681. The LSTM-FC-GP model has the middle highest average at 0.643, followed by the LSTM-PCA-GP model with a NRMSE of 0.607.

Table 5.1: NRMSE of the K-Fold Cross validation of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
LSTM	0.604	0.782	0.684	0.689	0.647	0.681
LSTM-FC-GP	0.607	0.601	0.645	0.718	0.644	0.643
LSTM-PCA-GP	0.622	0.574	0.579	0.652	0.610	0.607

The full NRMSE results of the K-fold cross validation can be seen in table 5.3. The columns of the table correspond to the validation data of each fold. Validation results 1 and 2 correspond to the case with the least amount of foam. Validation results 3 and 4 correspond to the case where there are two sheets of foam. Validation results 5 and 6 correspond to the case where there are three sheets of foam. With each foam sheet having a thickness of $28\,\mathrm{mm}$ this corresponds to a foam thickness of $28\,\mathrm{mm}$, $56\,\mathrm{mm}$, and $84\,\mathrm{mm}$ respectively. A summary of these thicknesses can be seen in table 5.2.

The results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models are shown in sub-tables 5.3a, 5.3b, and 5.3c respectively. These full results provide a notable nuance to the results shown in table 5.1. The average NRMSE of validation results 1 and 2 is notably higher than the NRMSE of the other validation

Table 5.2: Foam thickness corresponding to the validation results

Result	Corresponding Foam Thickness
Validation Result 1	$28\,\mathrm{mm}$
Validation Result 2	$28\,\mathrm{mm}$
Validation Result 3	$56\,\mathrm{mm}$
Validation Result 4	$56\,\mathrm{mm}$
Validation Result 5	$84\,\mathrm{mm}$
Validation Result 6	$84\,\mathrm{mm}$

results. This effect is more pronounced for the models that have a GP layer than for the model without a GP layer. Moreover, the LSTM-PCA-GP model has a higher NRMSE than the LSTM-FC-GP model for validation results 1 and 2 despite having a lower overall average NRMSE. As can be seen in table 5.2 validation results 1 and 2 correspond to the test case with only a single sheet of foam. This is consistent with the noise that was noted in Section 4.1.3.

Taking the average NRMSE of the models without the case with only a single sheet of foam results in a NRMSE of 0.656 for the LSTM model, 0.576 for the LSTM-FC-GP model, and 0.503 for the LSTM-PCA-GP model. This is an indication that the different models respond to noise in the input signal in a different way. How this noisy data effects the model accuracy is shown in more detail in Sections 5.3 and 5.5.3.

The confidence interval of the LSTM-FC-GP, and LSTM-PCA-GP is based on the output of the LSTM layer during training. The confidence interval should contain $95\,\%$ of the measured real data. However, it can be observed visually that this is not the case. Figure 5.1 is an extract from validation data result 4 of fold 4 of the LSTM-FC-GP model. The measured acceleration is shown as a green line labelled ”true acceleration” it can be seen that this line is outside of the $95\,\%$ confidence interval more frequently than $5\,\%$ of all data points.

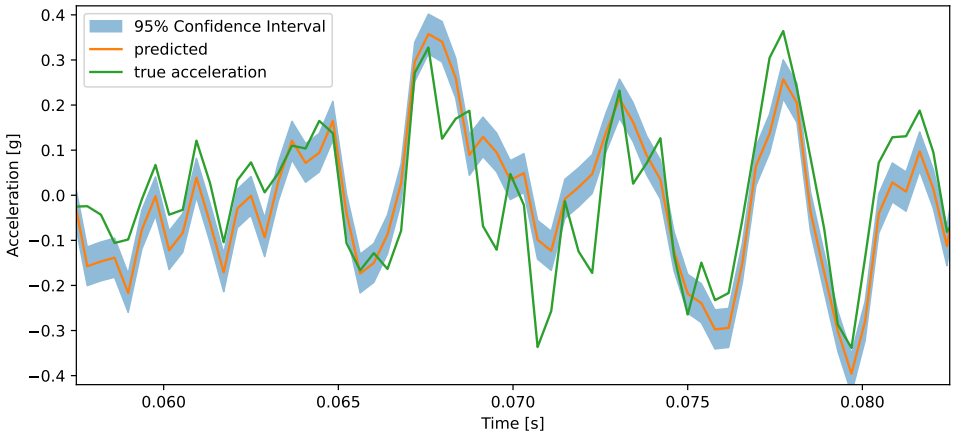


Figure 5.1: Example from LSTM-FC-GP fold 4, validation data result 4

This effect is illustrated more clearly in table 5.4 which shows the percentage of the real data points that are outside of the $95\,\%$ confidence interval. The results for the LSTM-FC-GP and LSTM-PCA-GP models are shown in sub-tables 5.4a and 5.4b respectively. In an ideal scenario this percentage should be $5\,\%$ for every fold.

The percentages outside the confidence show a similar trend to the NRMSE. The case with the lowest foam thickness shown in validation results 1 and 2 show significantly worse performance than the two cases with more foam. Furthermore, the model using PCA performs significantly better than the model using a FC layer, showing both an average percentage that is closer to $5\,\%$ and a lower standard deviation. A notable difference between these results and the results from the NRMSE is that for the percentage of the real measurements outside the $95\,\%$ confidence interval is lower across all validation

Table 5.3: NRMSE result for every test data result for every model

(a) LSTM

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.727	0.852	0.457	0.463	0.522	0.601	0.604	0.144
Fold 2	0.688	0.616	0.815	0.78	0.888	0.906	0.782	0.104
Fold 3	0.813	0.669	0.771	0.618	0.59	0.646	0.684	0.081
Fold 4	0.78	0.703	0.674	0.658	0.676	0.644	0.689	0.044
Fold 5	0.799	0.664	0.713	0.477	0.569	0.659	0.647	0.102
Average	0.762	0.701	0.686	0.599	0.649	0.691	0.681	
Std	0.047	0.081	0.124	0.118	0.13	0.109		0.117

(b) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.758	0.827	0.442	0.543	0.522	0.547	0.607	0.137
Fold 2	0.768	0.634	0.45	0.514	0.642	0.6	0.601	0.101
Fold 3	0.801	0.699	0.713	0.554	0.546	0.555	0.645	0.098
Fold 4	0.807	0.844	0.586	0.643	0.727	0.703	0.718	0.089
Fold 5	0.84	0.798	0.569	0.48	0.554	0.625	0.644	0.131
Average	0.795	0.761	0.552	0.547	0.598	0.606	0.643	
Std	0.029	0.081	0.1	0.054	0.076	0.056		0.12

(c) LSTM-PCA-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.739	1.011	0.491	0.478	0.502	0.51	0.622	0.195
Fold 2	0.796	0.737	0.447	0.453	0.512	0.5	0.574	0.139
Fold 3	0.773	0.745	0.59	0.461	0.422	0.482	0.579	0.137
Fold 4	0.83	0.869	0.479	0.645	0.55	0.537	0.652	0.149
Fold 5	0.847	0.82	0.463	0.442	0.511	0.577	0.61	0.164
Average	0.797	0.837	0.494	0.496	0.499	0.521	0.607	
Std	0.039	0.1	0.05	0.076	0.042	0.033		0.161

results for the LSTM-PCA-GP model compared to the LSTM-FC-GP model. This points to a difference in the width of the confidence interval.

The width of the confidence interval can be seen in table 5.5 with the interval for the LSTM-FC-GP and LSTM-PCA-GP being shown in tables 5.5a and 5.5b. The results from this table show that the average size of the confidence interval of the LSTM-FC-GP model is approximately half of the size of the confidence interval of the LSTM-PCA-GP model. The smaller size of the confidence interval of the LSTM-FC-GP model is consistent with a higher degree of over-fitting for the LSTM-FC-GP model compared to the LSTM-PCA-GP model.

Figure 5.2 shows the negative marginal log likelihood of the training data of the LSTM-FC-GP and LSTM-PCA-GP models that were used as a loss function while training. The figure shows the loss of both models reducing at a similar rate up to epoch 370. At this point the loss of the LSTM-PCA-GP model plateaus at a value of -1.2. Meanwhile the loss of the LSTM-FC-GP model keeps decreasing until below a value of -2.5.

The much lower value of the negative marginal log likelihood of the LSTM-FC-GP model suggests that it is more accurate at predicting the training data the the LSTM-PCA-GP model. However, the lower accuracy proves this is not the case for the validation data. This is an indicator that the LSTM-FC-GP model is over-fitted and thus has a reduced accuracy.

Table 5.4: The percentage of points that are outside of the 95 % confidence interval

(a) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	66.0 %	82.0 %	1.4 %	56.8 %	33.2 %	71.0 %	51.73 %	27.06 %
Fold 2	79.6 %	82.4 %	29.2 %	72.2 %	73.6 %	79.0 %	69.33 %	18.29 %
Fold 3	77.4 %	81.6 %	8.0 %	20.6 %	37.0 %	29.8 %	42.4 %	27.71 %
Fold 4	13.4 %	16.0 %	47.4 %	65.0 %	1.2 %	2.0 %	24.17 %	23.83 %
Fold 5	59.0 %	60.0 %	48.8 %	63.4 %	66.2 %	72.0 %	61.57 %	7.14 %
Average	59.08 %	64.4 %	26.96 %	55.6 %	42.24 %	50.76 %	49.84 %	
Std	24.05 %	25.66 %	19.56 %	18.17 %	25.89 %	29.92 %		27.16 %

(b) LSTM-PCA-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	45.0 %	70.8 %	0.4 %	20.4 %	9.6 %	49.6 %	32.63 %	24.55 %
Fold 2	21.4 %	24.4 %	0.0 %	18.8 %	16.2 %	17.4 %	16.37 %	7.8 %
Fold 3	39.2 %	45.6 %	0.0 %	0.0 %	0.2 %	0.2 %	14.2 %	20.03 %
Fold 4	7.2 %	9.2 %	25.8 %	57.2 %	0.0 %	0.2 %	16.6 %	20.09 %
Fold 5	48.2 %	50.8 %	24.2 %	53.2 %	47.8 %	63.8 %	48.0 %	11.91 %
Average	32.2 %	40.16 %	10.08 %	29.92 %	14.76 %	26.24 %	25.56 %	
Std	15.56 %	21.39 %	12.19 %	21.89 %	17.61 %	26.04 %		22.17 %

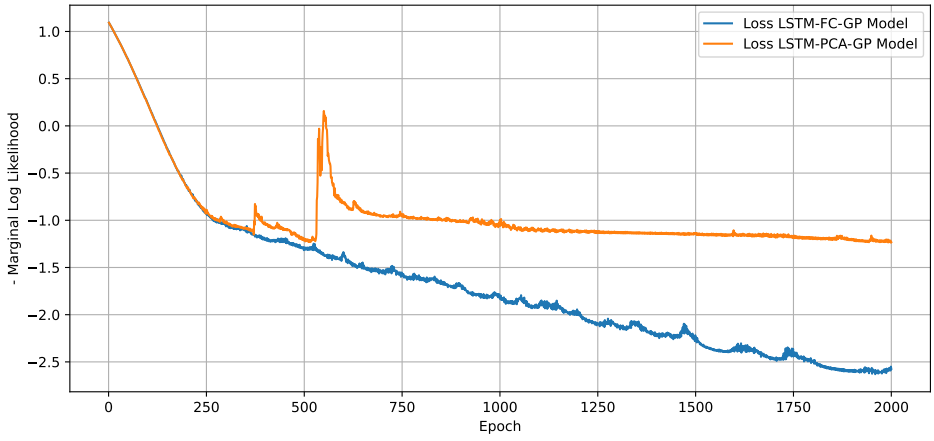


Figure 5.2: Marginal log likelihood of the LSTM-FC-GP and LSTM-PCA-GP models of fold 2 compared

Table 5.5: The average width of the $95\,\%$ confidence interval of every test data

(a) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$7.11 \times 10^{-2}\,\mathrm{g}$	$7.23 \times 10^{-2}\,\mathrm{g}$	$6.82 \times 10^{-2}\,\mathrm{g}$	$6.87 \times 10^{-2}\,\mathrm{g}$	$6.84 \times 10^{-2}\,\mathrm{g}$	$7.02 \times 10^{-2}\,\mathrm{g}$	$6.98 \times 10^{-2}\,\mathrm{g}$	$1.51 \times 10^{-3}\,\mathrm{g}$
Fold 2	$5.02 \times 10^{-2}\,\mathrm{g}$	$5.13 \times 10^{-2}\,\mathrm{g}$	$4.84 \times 10^{-2}\,\mathrm{g}$	$4.96 \times 10^{-2}\,\mathrm{g}$	$5.01 \times 10^{-2}\,\mathrm{g}$	$5.07 \times 10^{-2}\,\mathrm{g}$	$5.00 \times 10^{-2}\,\mathrm{g}$	$9.00 \times 10^{-4}\,\mathrm{g}$
Fold 3	$5.77 \times 10^{-2}\,\mathrm{g}$	$5.83 \times 10^{-2}\,\mathrm{g}$	$5.60 \times 10^{-2}\,\mathrm{g}$	$5.60 \times 10^{-2}\,\mathrm{g}$	$5.62 \times 10^{-2}\,\mathrm{g}$	$5.61 \times 10^{-2}\,\mathrm{g}$	$5.67 \times 10^{-2}\,\mathrm{g}$	$9.11 \times 10^{-4}\,\mathrm{g}$
Fold 4	$8.66 \times 10^{-2}\,\mathrm{g}$	$8.67 \times 10^{-2}\,\mathrm{g}$	$8.71 \times 10^{-2}\,\mathrm{g}$	$8.76 \times 10^{-2}\,\mathrm{g}$	$8.63 \times 10^{-2}\,\mathrm{g}$	$8.63 \times 10^{-2}\,\mathrm{g}$	$8.68 \times 10^{-2}\,\mathrm{g}$	$4.47 \times 10^{-4}\,\mathrm{g}$
Fold 5	$5.22 \times 10^{-2}\,\mathrm{g}$	$5.22 \times 10^{-2}\,\mathrm{g}$	$5.16 \times 10^{-2}\,\mathrm{g}$	$5.26 \times 10^{-2}\,\mathrm{g}$	$5.23 \times 10^{-2}\,\mathrm{g}$	$5.35 \times 10^{-2}\,\mathrm{g}$	$5.24 \times 10^{-2}\,\mathrm{g}$	$5.83 \times 10^{-4}\,\mathrm{g}$
Average	$6.36 \times 10^{-2}\,\mathrm{g}$	$6.41 \times 10^{-2}\,\mathrm{g}$	$6.23 \times 10^{-2}\,\mathrm{g}$	$6.29 \times 10^{-2}\,\mathrm{g}$	$6.27 \times 10^{-2}\,\mathrm{g}$	$6.34 \times 10^{-2}\,\mathrm{g}$	$6.32 \times 10^{-2}\,\mathrm{g}$	
Std	$1.36 \times 10^{-2}\,\mathrm{g}$	$1.36 \times 10^{-2}\,\mathrm{g}$	$1.41 \times 10^{-2}\,\mathrm{g}$	$1.39 \times 10^{-2}\,\mathrm{g}$	$1.34 \times 10^{-2}\,\mathrm{g}$	$1.33 \times 10^{-2}\,\mathrm{g}$		$1.37 \times 10^{-2}\,\mathrm{g}$

(b) LSTM-PCA-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$1.17 \times 10^{-1}\,\mathrm{g}$	$1.29 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.13 \times 10^{-1}\,\mathrm{g}$	$1.15 \times 10^{-1}\,\mathrm{g}$	$6.30 \times 10^{-3}\,\mathrm{g}$
Fold 2	$2.58 \times 10^{-1}\,\mathrm{g}$	$3.30 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.77 \times 10^{-1}\,\mathrm{g}$	$1.72 \times 10^{-1}\,\mathrm{g}$	$1.74 \times 10^{-1}\,\mathrm{g}$	$2.14 \times 10^{-1}\,\mathrm{g}$	$6.06 \times 10^{-2}\,\mathrm{g}$
Fold 3	$1.59 \times 10^{-1}\,\mathrm{g}$	$1.79 \times 10^{-1}\,\mathrm{g}$	$1.37 \times 10^{-1}\,\mathrm{g}$	$1.37 \times 10^{-1}\,\mathrm{g}$	$1.37 \times 10^{-1}\,\mathrm{g}$	$1.37 \times 10^{-1}\,\mathrm{g}$	$1.48 \times 10^{-1}\,\mathrm{g}$	$1.63 \times 10^{-2}\,\mathrm{g}$
Fold 4	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.12 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$1.11 \times 10^{-1}\,\mathrm{g}$	$2.79 \times 10^{-4}\,\mathrm{g}$
Fold 5	$7.39 \times 10^{-2}\,\mathrm{g}$	$7.41 \times 10^{-2}\,\mathrm{g}$	$7.37 \times 10^{-2}\,\mathrm{g}$	$7.41 \times 10^{-2}\,\mathrm{g}$	$7.38 \times 10^{-2}\,\mathrm{g}$	$7.41 \times 10^{-2}\,\mathrm{g}$	$7.40 \times 10^{-2}\,\mathrm{g}$	$1.77 \times 10^{-4}\,\mathrm{g}$
Average	$1.44 \times 10^{-1}\,\mathrm{g}$	$1.65 \times 10^{-1}\,\mathrm{g}$	$1.21 \times 10^{-1}\,\mathrm{g}$	$1.22 \times 10^{-1}\,\mathrm{g}$	$1.21 \times 10^{-1}\,\mathrm{g}$	$1.22 \times 10^{-1}\,\mathrm{g}$	$1.32 \times 10^{-1}\,\mathrm{g}$	
Std	$6.31 \times 10^{-2}\,\mathrm{g}$	$8.94 \times 10^{-2}\,\mathrm{g}$	$3.20 \times 10^{-2}\,\mathrm{g}$	$3.41 \times 10^{-2}\,\mathrm{g}$	$3.25 \times 10^{-2}\,\mathrm{g}$	$3.28 \times 10^{-2}\,\mathrm{g}$		$5.47 \times 10^{-2}\,\mathrm{g}$

5.2. Varying the Steps Skipped by GP Layer

To evaluate the effect of leaving out a large chunk of the outputs of the LSTM layer before providing it to the GP layer during training, the first fold of both models was retrained with double the amount of time steps provided to the GP layer, increasing the amount of inputs to the GP layer from one every four time steps to one every two time steps. As was noted in Section 2.2.2, calculating the variance takes $\mathcal{O}(n^2)$ time meaning this doubling of the amount of data results in a quadrupling of computation time. Additionally, the increased amount of memory exceeds the video memory of the system that was used to train the model on. This meant that the GP layer could not be run on the GPU effectively and had to be run on the CPU instead. This increased the overall training time of fold 1 to the same amount of time it took to perform the entire k-fold cross-validation on the default model. Therefore the results are only provided for fold 1.

The NRMSE of fold 1 of the new models that were trained with double the amount of time steps inputted to the GP layer is shown in table 5.6. Comparing the average NRMSE of the results to those of the default models, the average NRMSE is higher for both the LSTM-FC-GP and LSTM-PCA-GP models. When excluding validation results 1 and 2 like was done for the default models, the average NRMSE becomes 0.622 for the LSTM-FC-GP model and 0.581 for the LSTM-PCA-GP model which are both higher than the results for the default models.

Table 5.6: NRMSE for each test result of fold 1 when doubling the amount of time steps supplied to the GP layer

Model	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	0.756	0.823	0.658	0.594	0.606	0.629	0.678	0.084
LSTM-PCA-GP	0.768	0.846	0.493	0.606	0.625	0.601	0.656	0.116

The percentage of real data points that are outside of the $95\,\%$ confidence interval is shown in table 5.7. This percentage is higher not just on average but for every test result for both models, having an average of over $60\,\%$ for both models. This percentage should ideally be $5\,\%$.

Table 5.7: Percentage of real data points outside of the 95% confidence interval of fold 1 for models retrained with $1/2$ time-steps as GP input

Model	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	$70.0\,\%$	$85.8\,\%$	$15.0\,\%$	$65.0\,\%$	$55.2\,\%$	$80.6\,\%$	$61.93\,\%$	$23.24\,\%$
LSTM-PCA-GP	$79.4\,\%$	$88.4\,\%$	$8.6\,\%$	$71.8\,\%$	$57.8\,\%$	$81.4\,\%$	$64.57\,\%$	$26.78\,\%$

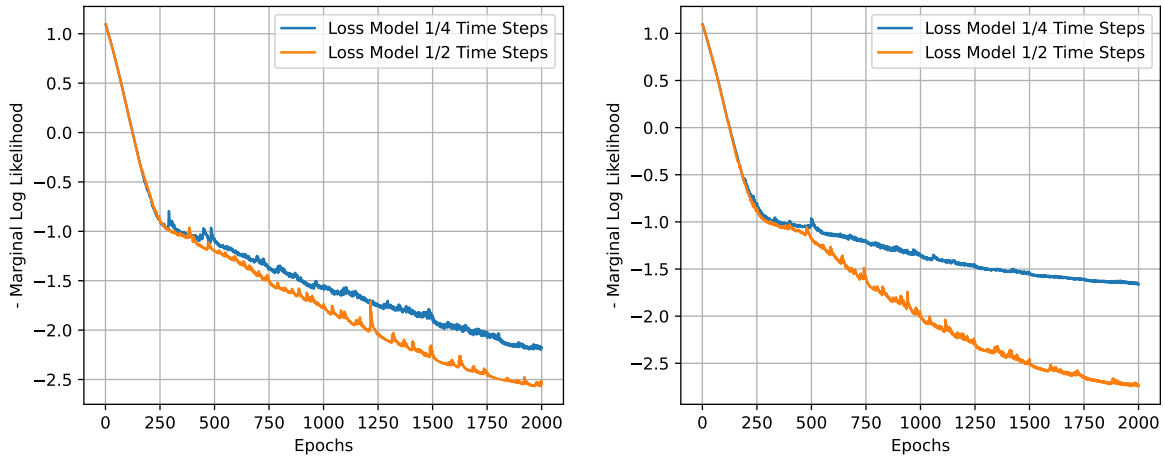
Supporting the result from table 5.7, table 5.8 shows the average width of the $95\,\%$ confidence interval for every test result for both models. Comparing these results to the results of the default models, it can be seen that the confidence interval is narrower for both models for all test results.

Table 5.8: Width of the $95\,\%$ confidence interval of fold 1 for models retrained with $1/2$ time-steps as GP input

Model	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	$5.36 \times 10^{-2}\,\mathrm{g}$	$5.48 \times 10^{-2}\,\mathrm{g}$	$5.28 \times 10^{-2}\,\mathrm{g}$	$5.29 \times 10^{-2}\,\mathrm{g}$	$5.29 \times 10^{-2}\,\mathrm{g}$	$5.41 \times 10^{-2}\,\mathrm{g}$	$5.35 \times 10^{-2}\,\mathrm{g}$	$7.34 \times 10^{-4}\,\mathrm{g}$
LSTM-PCA-GP	$4.68 \times 10^{-2}\,\mathrm{g}$	$4.70 \times 10^{-2}\,\mathrm{g}$	$4.66 \times 10^{-2}\,\mathrm{g}$	$4.67 \times 10^{-2}\,\mathrm{g}$	$4.67 \times 10^{-2}\,\mathrm{g}$	$4.70 \times 10^{-2}\,\mathrm{g}$	$4.68 \times 10^{-2}\,\mathrm{g}$	$1.62 \times 10^{-4}\,\mathrm{g}$

Figure 5.3 shows the training loss of the models that were trained with both one fourth and half the time steps inputted into the GP layers. The losses for the LSTM-FC-GP and LSTM-PCA-GP models are shown in Sub-Figures 5.3a and 5.3b respectively.

The training loss shows the loss was approximately equal until around epoch 400. After epoch 400, the training loss decreases at a rate that is higher than it is for the other epochs. This effect is much more pronounced in the LSTM-PCA-GP model. Furthermore, the models that were trained with a higher number of time steps have a final negative marginal log likelihood that is lower than the models that were trained with a smaller number of time steps. This is consistent with the smaller width of the confidence interval shown in table 5.8.



(a) LSTM-FC-GP model

(b) LSTM-PCA-GP model

Figure 5.3: Comparison between the training loss of the LSTM-FC-GP and LSTM-PCA-GP models using 1/4 time steps and 1/2 time steps for the GP layer during training

5.3. The Effect of Noisy Data

To determine to what degree the noisy data associated with the test case with only a single sheet of foam, which was discussed in Section 4.1.3, influences the accuracy of the model, all three models for were re-trained for fold 1 with only the data from the test case with two and three sheets of foam. This test data corresponds to validation results 3 to 6 from the previous sections. To remain consistent the validation results in this section are numbered 3 to 6.

The NRMSE of the validation results is shown in table 5.9. The results show that removing the test data with a single sheet of foam does not have an exclusively positive effect on the test results. While the average NRMSE of the LSTM-FC-GP and LSTM-PCA-GP reduces compared to fold 1, the NRMSE for the LSTM model increases. Furthermore, when taking the average NRMSE of fold 1 excluding validation results 1 and 2, the NRMSE increases by 0.095, 0.030, and 0.046 for the LSTM, LSTM-FC-GP, and LSTM-PCA-GP model respectively.

Table 5.9: NRMSE of the models retrained on the dataset excluding the data from the case with a single sheet of foam for fold 1

Model	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM	0.553	0.507	0.745	0.621	0.606	0.09
LSTM-FC-GP	0.467	0.557	0.542	0.608	0.543	0.05
LSTM-PCA-GP	0.57	0.526	0.548	0.518	0.541	0.02

The percentage of real data points that fall outside of the $95\,\%$ confidence interval is shown in table 5.10. Here, a similar trend can be observed to the NRMSE where the results improve for the LSTM-FC-GP model but become worse for the LSTM-PCA-GP model.

Table 5.10: Percentage of real data points that fall outside of the $95\,\%$ confidence interval for the models that were trained without the data from the test-case with a single sheet of foam

Model	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	$0.4\,\%$	$49.2\,\%$	$26.6\,\%$	$65.8\,\%$	$35.5\,\%$	$24.58\,\%$
LSTM-PCA-GP	$0.6\,\%$	$27.2\,\%$	$14.2\,\%$	$48.0\,\%$	$22.5\,\%$	$17.47\,\%$

The decreasing percentage of the LSTM-FC-GP and increasing percentage for the LSTM-PCA-GP model is consistent with the width of the confidence interval which is shown in table 5.11. The width of this confidence interval increases for the LSTM-FC-GP model but decreases for the LSTM-PCA-GP model.

Table 5.11: Width of the $95\,\%$ confidence interval for the models trained without the data from the test-case with a single sheet of foam

Model	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	$7.79 \times 10^{-2}\,\mathrm{g}$	$7.82 \times 10^{-2}\,\mathrm{g}$	$7.80 \times 10^{-2}\,\mathrm{g}$	$8.03 \times 10^{-2}\,\mathrm{g}$	$7.86 \times 10^{-2}\,\mathrm{g}$	$9.63 \times 10^{-4}\,\mathrm{g}$
LSTM-PCA-GP	$1.06 \times 10^{-1}\,\mathrm{g}$	$1.07 \times 10^{-1}\,\mathrm{g}$	$1.06 \times 10^{-1}\,\mathrm{g}$	$1.08 \times 10^{-1}\,\mathrm{g}$	$1.07 \times 10^{-1}\,\mathrm{g}$	$6.97 \times 10^{-4}\,\mathrm{g}$

5.4. Effect of GP Input Dimension

To determine the effect of varying the input dimension of the GP layer the models were re-trained for fold 1 with the input dimension of the GP layer reduced from 8 to 6 and 4 inputs.

The NRMSE of the models with an input dimension of 6 can be seen in table 5.12. The results show that the NRMSE is similar to the results with a higher input dimension. For the LSTM-FC-GP model, the NRMSE is reduced by$1\,\%$. Meanwhile, the NRMSE of the LSTM-FC-GP model is reduced by $1.3\,\%$.

Table 5.12: NRMSE for each test result of fold 1 when reducing the input dimension of the GP layer to 6

Model	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	0.712	0.788	0.445	0.538	0.542	0.584	0.601	0.115
LSTM-PCA-GP	0.739	0.797	0.474	0.528	0.535	0.611	0.614	0.117

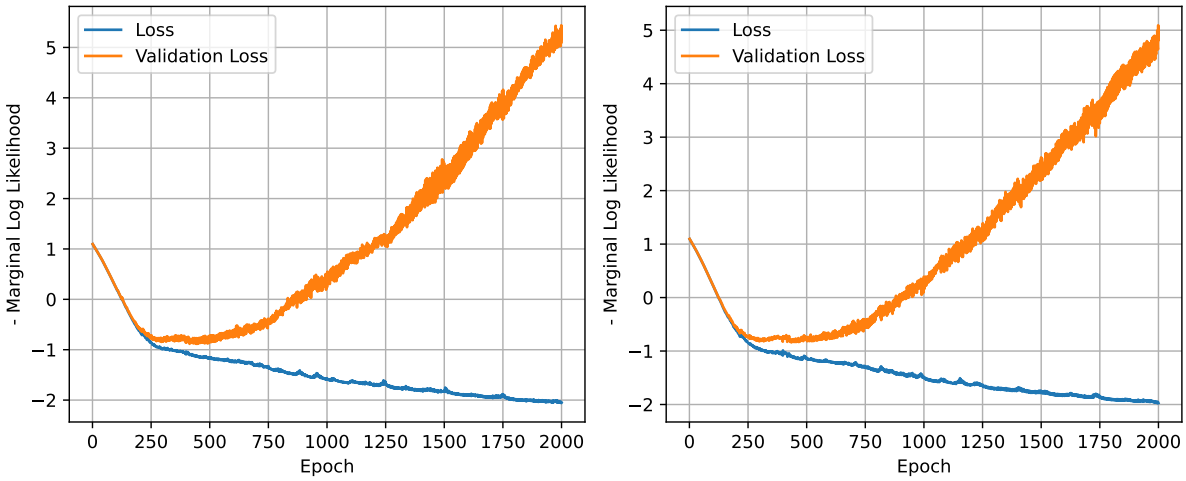
The NRMSE of the models with a GP input dimension of 4 can be seen in table 5.13. The results show that the NRMSE of the LSTM-FC-GP model are higher than the NRMSE of the model with a GP input dimension of 6 and than the default model. Meanwhile, the NRMSE of the LSTM-PCA-GP model with a GP input dimension of 4 is nearly identical to the NRMSE of the model with a GP input dimension of 6. However, the standard deviation of the model with a GP input dimension of 4 is somewhat higher.

The training and validation loss of the models with a GP input dimension of 6 is shown in figure 5.4. The training and validation loss of the LSTM-FC-GP and LSTM-PCA-GP models can be seen in Subfigures 5.4a and 5.4b respectively. The results show the training and validation loss decreasing at the

Table 5.13: NRMSE for each test result of fold 1 when reducing the input dimension of the GP layer to 4

Model	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
LSTM-FC-GP	0.808	0.865	0.489	0.588	0.545	0.57	0.644	0.14
LSTM-PCA-GP	0.772	0.794	0.471	0.5	0.52	0.621	0.613	0.129

same rate until approximately epoch 200. After epoch 200, the validation loss decreases slower than the training loss up to epoch 500 where the validation loss starts to increase again.



(a) LSTM-FC-GP

(b) LSTM-PCA-GP

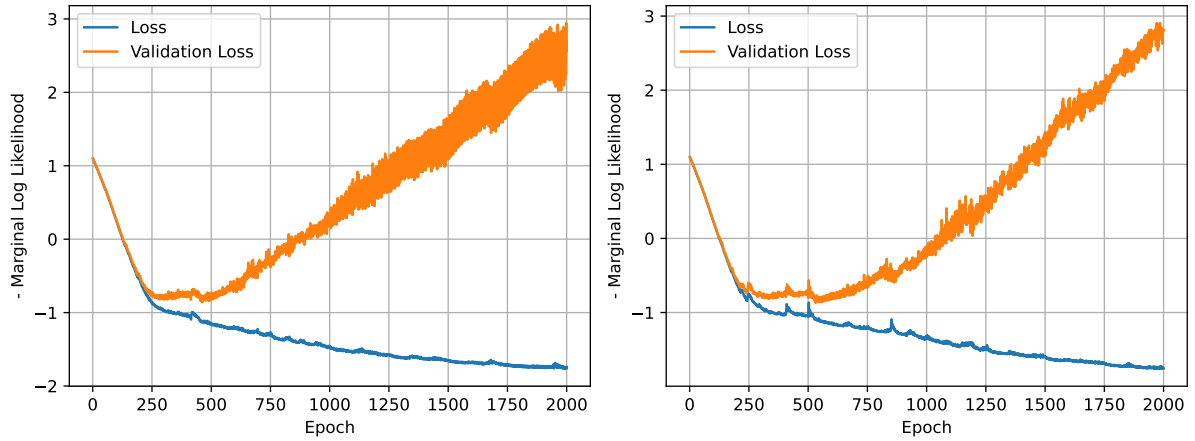
Figure 5.4: Training and validation loss of models trained with a GP input dimension of 6

The training and validation loss of the models with a GP input dimension of 4 is shown in figure 5.5. The training loss and validation of the LSTM-FC-GP and LSTM-PCA-GP models is shown in Sub-figures 5.5a and 5.5b respectively. The results show a similar trend to the results of the models with a larger GP input dimension. Here, the training loss and validation loss decrease together up to epoch 200 where the validation loss plateaus while the training loss keeps decreasing. The difference between the models with a GP input dimension of 4 and 6 is that the training loss for the models with a GP input dimension of 4 does not decrease to the same level as it does for the models that were trained with a GP input dimension of 6. Furthermore, the validation loss increases to a lower negative marginal log likelihood for both models.

A comparison is made between the validation loss of the models with and without a reduced GP layer input dimension. This comparison can be seen in Figure 5.6 which shows the validation loss of the models with a GP input dimension of 8, 6, and 4. The comparison for the LSTM-FC-GP and LSTM-PCA-GP models can be seen in Sub-figures 5.6a and 5.6b respectively.

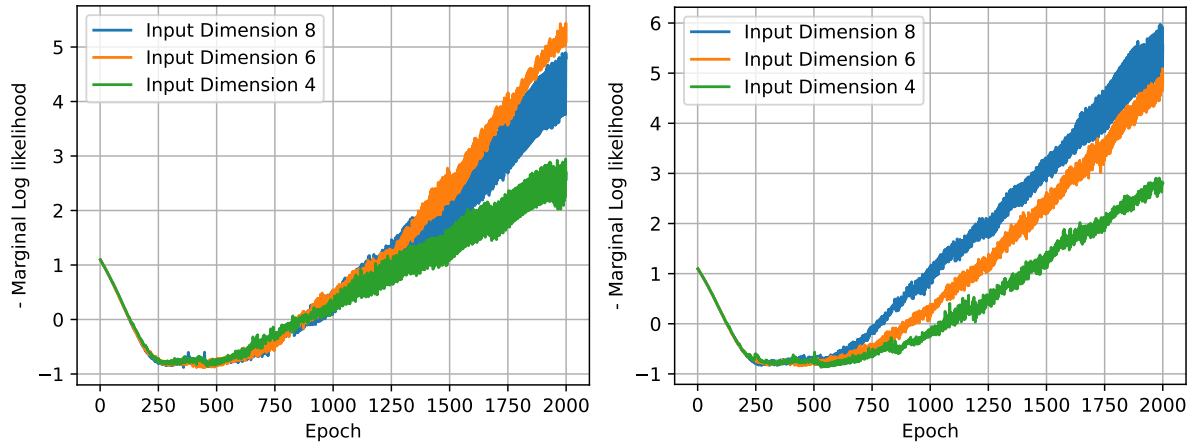
For the LSTM-FC-GP model, the comparison shows that the validation loss is nearly indistinguishable up to epoch 750. After epoch 750, the validation loss for the model with a an input dimension of 6 increases faster than the validation loss of the model with the original input dimension while the model with a GP input dimension of 4 increases slower.

The LSTM-PCA-GP model shows the validation loss is nearly indistinguishable up to epoch 600. After epoch 600 the loss of the models with a reduced input dimension increases slower than the model with the original input dimension. After epoch 1000, the model with a GP input dimension of 6 increases at nearly the same rate as the model with an input dimension of 8 while the model with a GP input dimension of 4 keeps increasing at a slower speed.



(a) LSTM-FC-GP **(b)** LSTM-PCA-GP

Figure 5.5: Training and validation loss of models trained with a GP input dimension of 4



(a) LSTM-FC-GP **(b)** LSTM-PCA-GP

Figure 5.6: Comparison between validation loss of models trained with a reduced GP input dimension and models trained without a reduced GP input dimension

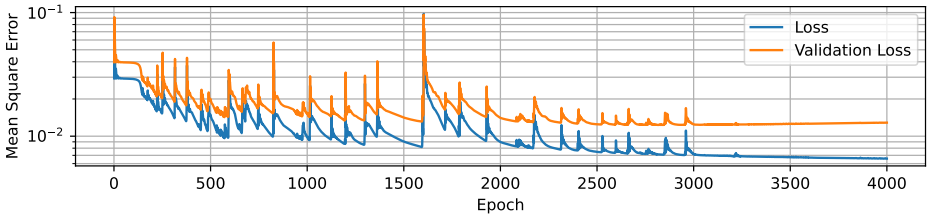
5.5. Models Without Over-Fitting

To compare the real performance of the models, the models were re-trained with early stopping to ensure the models are not over-fitted. The results of the default models are shown in Section 5.5.1. Time-domain plots of these results are shown in Section 5.5.2. The results of the models trained without data from the test case with a single sheet of foam can be seen in section 5.5.3. Finally, a comparison between the PSD of the measured response and the simulated response can be found in Section 5.5.4.

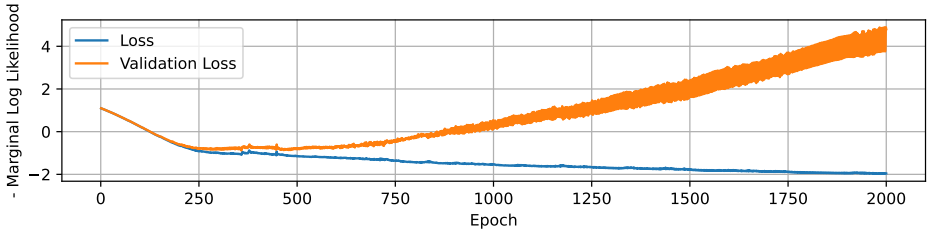
5.5.1. Results Without Over-Fitting

To find out to what extend and from what point each model is over-fitted, the training loss and validation loss for each of the models are shown in Figure 5.7. The losses of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models are shown in Figures 5.7a, 5.7b, and 5.7c respectively.

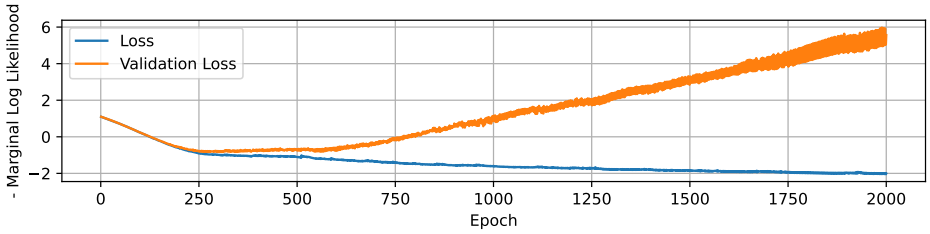
The results shown in Figure 5.7 show that for the LSTM model the validation loss decreases proportional to the training loss up to the last epoch. Furthermore, it can be seen that neither training loss nor validation loss change in a meaningful way beyond epoch 3000. The LSTM-FC-GP and LSTM-PCA-GP models show results that are very similar to each other. The training and validation loss are roughly equal up to epoch 250. Beyond this point, the training loss continues to decrease while the validation loss stops decreasing and starts to increase again beyond epoch 600.



(a) LSTM



(b) LSTM-FC-GP



(c) LSTM-PCA-GP

Figure 5.7: Training and validation loss for Fold 1 of all Models

These results point to little to no over-fitting for the LSTM model, and significant over-fitting beyond epoch 600 for the LSTM-FC-GP and LSTM-PCA-GP models. A more detailed discussion about these results can be found in Section 6.5. To evaluate the performance of the models without over-fitting, the LSTM-FC-GP and LSTM-PCA-GP models are re-trained up to epoch 600. The results of these models are shown alongside the original results from section 5.1 for the LSTM model to make comparison easier.

The NRMSE of the re-trained models can be seen in table 5.14. The results for the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models can be seen in tables 5.14a, 5.14b, and 5.14c respectively. The results show that the LSTM-PCA-GP model has the lowest average NRMSE, followed by the LSTM-FC-GP, and the LSTM model.

The percentage of real data points that are outside of the $95\,\%$ confidence interval for the re-trained models can be seen in table 5.15. The percentages for the LSTM-FC-GP and LSTM-PCA-GP models can be found in tables 5.15a, and 5.15b respectively. The results show that the average percentage of real data points outside of the $95\,\%$ confidence interval is $11.44\,\%$ for the LSTM-FC-GP model, and $9.62\,\%$ for the LSTM-PCA-GP model. Factoring the percentages of the test results of the case with only a single sheet of foam out of the average results in an average percentage of $6.59\,\%$ for the LSTM-FC-GP model and $5.8\,\%$ for the LSTM-PCA-GP model.

The width of the confidence interval of the re-trained models is shown for every fold in table 5.16. The results for the LSTM-FC-GP and LSTM-PCA-GP models are shown in tables 5.16a and 5.16b respectively. The results show that the width of the confidence interval of the LSTM-FC-GP model is nearly three times larger the width of the confidence interval of the default model. The width of the confidence interval of the LSTM-PCA-GP model is wider by less than half.

Table 5.14: NRMSE result for every test data result for every model

(a) LSTM

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.727	0.852	0.457	0.463	0.522	0.601	0.604	0.144
Fold 2	0.688	0.616	0.815	0.78	0.888	0.906	0.782	0.104
Fold 3	0.813	0.669	0.771	0.618	0.59	0.646	0.684	0.081
Fold 4	0.78	0.703	0.674	0.658	0.676	0.644	0.689	0.044
Fold 5	0.799	0.664	0.713	0.477	0.569	0.659	0.647	0.102
Average	0.762	0.701	0.686	0.599	0.649	0.691	0.681	
Std	0.047	0.081	0.124	0.118	0.13	0.109		0.117

(b) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.577	0.824	0.458	0.425	0.498	0.43	0.535	0.139
Fold 2	0.776	0.715	0.409	0.436	0.533	0.478	0.558	0.139
Fold 3	0.777	0.697	0.626	0.491	0.455	0.467	0.586	0.123
Fold 4	0.746	0.687	0.464	0.491	0.581	0.556	0.587	0.101
Fold 5	0.776	0.663	0.472	0.405	0.415	0.466	0.533	0.138
Average	0.73	0.717	0.486	0.45	0.496	0.479	0.56	
Std	0.077	0.056	0.073	0.035	0.058	0.042		0.131

(c) LSTM-PCA-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.58	0.758	0.492	0.426	0.514	0.44	0.535	0.112
Fold 2	0.762	0.699	0.417	0.41	0.517	0.484	0.548	0.135
Fold 3	0.783	0.637	0.644	0.519	0.427	0.484	0.582	0.119
Fold 4	0.716	0.683	0.454	0.525	0.612	0.581	0.595	0.089
Fold 5	0.751	0.643	0.456	0.382	0.449	0.475	0.526	0.128
Average	0.718	0.684	0.493	0.452	0.504	0.493	0.557	
Std	0.072	0.044	0.079	0.059	0.065	0.047		0.121

Table 5.15: The percentage of points that are outside of the $95\,\%$ confidence interval

(a) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	15.6 %	41.0 %	0.0 %	3.8 %	0.6 %	19.0 %	13.33 %	14.34 %
Fold 2	46.2 %	52.0 %	0.0 %	26.0 %	22.2 %	24.8 %	28.53 %	17.02 %
Fold 3	21.8 %	32.0 %	0.0 %	0.0 %	0.0 %	0.0 %	8.97 %	13.02 %
Fold 4	0.0 %	0.0 %	4.4 %	13.0 %	0.0 %	0.0 %	2.9 %	4.79 %
Fold 5	1.6 %	1.2 %	0.4 %	6.2 %	2.4 %	9.0 %	3.47 %	3.09 %
Average	17.04 %	25.24 %	0.96 %	9.8 %	5.04 %	10.56 %	11.44 %	
Std	16.76 %	21.1 %	1.73 %	9.14 %	8.62 %	9.99 %		15.07 %

(b) LSTM-PCA-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	8.6 %	31.8 %	0.0 %	1.0 %	0.4 %	13.0 %	9.13 %	11.22 %
Fold 2	36.0 %	45.8 %	0.0 %	19.6 %	19.2 %	21.4 %	23.67 %	14.4 %
Fold 3	19.0 %	26.2 %	0.0 %	0.4 %	0.0 %	0.0 %	7.6 %	10.81 %
Fold 4	0.0 %	0.0 %	2.8 %	12.2 %	0.0 %	0.0 %	2.5 %	4.46 %
Fold 5	3.0 %	2.2 %	0.4 %	7.2 %	6.0 %	12.4 %	5.2 %	3.94 %
Average	13.32 %	21.2 %	0.64 %	8.08 %	5.12 %	9.36 %	9.62 %	
Std	13.06 %	17.62 %	1.09 %	7.21 %	7.4 %	8.28 %		12.31 %

Table 5.16: The average width of the $95\,\%$ confidence interval of every test data

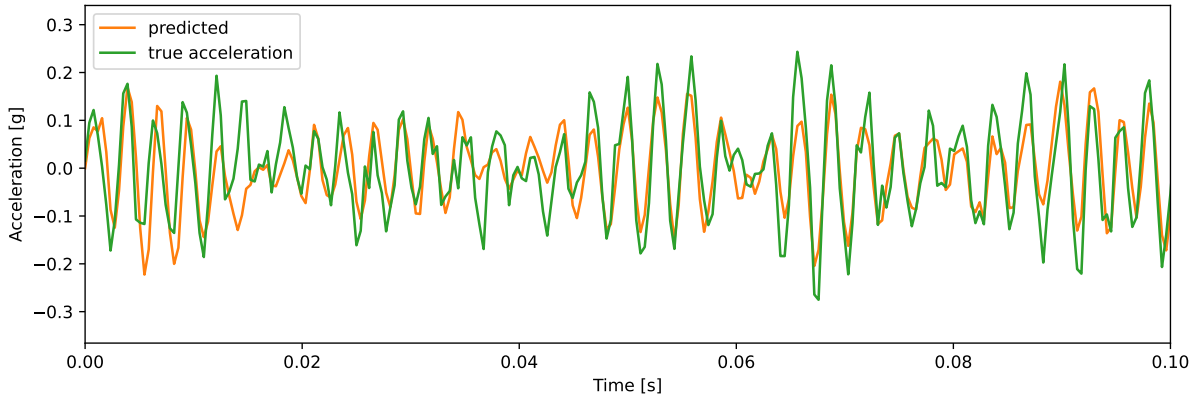
(a) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$1.87 \times 10^{-1}\,\mathrm{g}$	$2.33 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.72 \times 10^{-1}\,\mathrm{g}$	$1.84 \times 10^{-1}\,\mathrm{g}$	$2.30 \times 10^{-2}\,\mathrm{g}$
Fold 2	$1.59 \times 10^{-1}\,\mathrm{g}$	$1.66 \times 10^{-1}\,\mathrm{g}$	$1.45 \times 10^{-1}\,\mathrm{g}$	$1.46 \times 10^{-1}\,\mathrm{g}$	$1.45 \times 10^{-1}\,\mathrm{g}$	$1.46 \times 10^{-1}\,\mathrm{g}$	$1.51 \times 10^{-1}\,\mathrm{g}$	$8.29 \times 10^{-3}\,\mathrm{g}$
Fold 3	$1.93 \times 10^{-1}\,\mathrm{g}$	$2.08 \times 10^{-1}\,\mathrm{g}$	$1.87 \times 10^{-1}\,\mathrm{g}$	$1.88 \times 10^{-1}\,\mathrm{g}$	$1.88 \times 10^{-1}\,\mathrm{g}$	$1.88 \times 10^{-1}\,\mathrm{g}$	$1.92 \times 10^{-1}\,\mathrm{g}$	$7.58 \times 10^{-3}\,\mathrm{g}$
Fold 4	$1.94 \times 10^{-1}\,\mathrm{g}$	$1.94 \times 10^{-1}\,\mathrm{g}$	$1.95 \times 10^{-1}\,\mathrm{g}$	$1.95 \times 10^{-1}\,\mathrm{g}$	$1.94 \times 10^{-1}\,\mathrm{g}$	$1.94 \times 10^{-1}\,\mathrm{g}$	$1.94 \times 10^{-1}\,\mathrm{g}$	$3.72 \times 10^{-4}\,\mathrm{g}$
Fold 5	$1.90 \times 10^{-1}\,\mathrm{g}$	$1.90 \times 10^{-1}\,\mathrm{g}$	$1.90 \times 10^{-1}\,\mathrm{g}$	$1.91 \times 10^{-1}\,\mathrm{g}$	$1.90 \times 10^{-1}\,\mathrm{g}$	$1.90 \times 10^{-1}\,\mathrm{g}$	$1.90 \times 10^{-1}\,\mathrm{g}$	$2.86 \times 10^{-4}\,\mathrm{g}$
Average	$1.84 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$1.77 \times 10^{-1}\,\mathrm{g}$	$1.78 \times 10^{-1}\,\mathrm{g}$	$1.77 \times 10^{-1}\,\mathrm{g}$	$1.78 \times 10^{-1}\,\mathrm{g}$	$1.82 \times 10^{-1}\,\mathrm{g}$	
Std	$1.32 \times 10^{-2}\,\mathrm{g}$	$2.21 \times 10^{-2}\,\mathrm{g}$	$1.83 \times 10^{-2}\,\mathrm{g}$	$1.79 \times 10^{-2}\,\mathrm{g}$	$1.80 \times 10^{-2}\,\mathrm{g}$	$1.77 \times 10^{-2}\,\mathrm{g}$		$1.96 \times 10^{-2}\,\mathrm{g}$

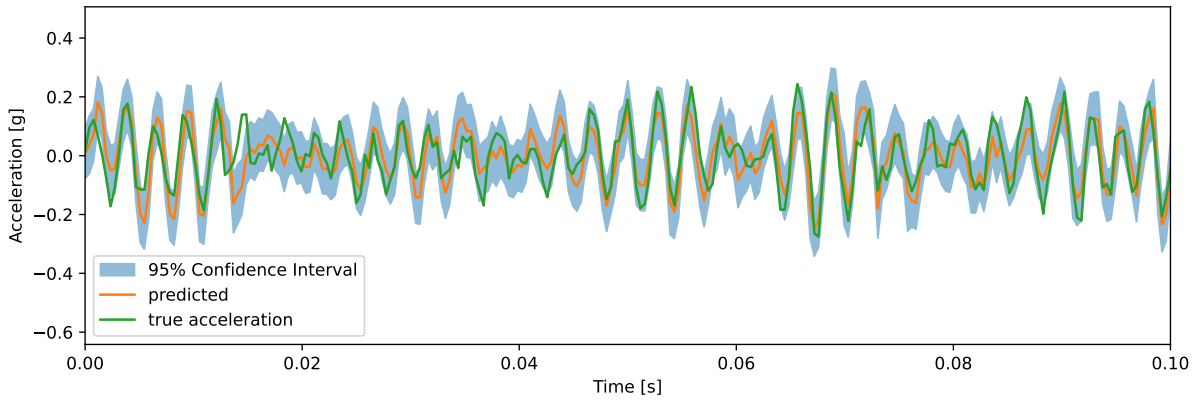
(b) LSTM-PCA-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$2.04 \times 10^{-1}\,\mathrm{g}$	$2.49 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$1.99 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$2.04 \times 10^{-1}\,\mathrm{g}$	$2.09 \times 10^{-1}\,\mathrm{g}$	$1.83 \times 10^{-2}\,\mathrm{g}$
Fold 2	$1.77 \times 10^{-1}\,\mathrm{g}$	$1.99 \times 10^{-1}\,\mathrm{g}$	$1.53 \times 10^{-1}\,\mathrm{g}$	$1.56 \times 10^{-1}\,\mathrm{g}$	$1.54 \times 10^{-1}\,\mathrm{g}$	$1.55 \times 10^{-1}\,\mathrm{g}$	$1.66 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-2}\,\mathrm{g}$
Fold 3	$2.00 \times 10^{-1}\,\mathrm{g}$	$2.04 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$1.98 \times 10^{-1}\,\mathrm{g}$	$1.99 \times 10^{-1}\,\mathrm{g}$	$2.26 \times 10^{-3}\,\mathrm{g}$
Fold 4	$2.13 \times 10^{-1}\,\mathrm{g}$	$2.13 \times 10^{-1}\,\mathrm{g}$	$2.14 \times 10^{-1}\,\mathrm{g}$	$2.14 \times 10^{-1}\,\mathrm{g}$	$2.13 \times 10^{-1}\,\mathrm{g}$	$2.13 \times 10^{-1}\,\mathrm{g}$	$2.13 \times 10^{-1}\,\mathrm{g}$	$4.85 \times 10^{-4}\,\mathrm{g}$
Fold 5	$1.75 \times 10^{-1}\,\mathrm{g}$	$1.77 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.72 \times 10^{-1}\,\mathrm{g}$	$1.71 \times 10^{-1}\,\mathrm{g}$	$1.72 \times 10^{-1}\,\mathrm{g}$	$1.73 \times 10^{-1}\,\mathrm{g}$	$2.44 \times 10^{-3}\,\mathrm{g}$
Average	$1.94 \times 10^{-1}\,\mathrm{g}$	$2.08 \times 10^{-1}\,\mathrm{g}$	$1.86 \times 10^{-1}\,\mathrm{g}$	$1.88 \times 10^{-1}\,\mathrm{g}$	$1.87 \times 10^{-1}\,\mathrm{g}$	$1.88 \times 10^{-1}\,\mathrm{g}$	$1.92 \times 10^{-1}\,\mathrm{g}$	
Std	$1.50 \times 10^{-2}\,\mathrm{g}$	$2.36 \times 10^{-2}\,\mathrm{g}$	$2.17 \times 10^{-2}\,\mathrm{g}$	$2.11 \times 10^{-2}\,\mathrm{g}$	$2.13 \times 10^{-2}\,\mathrm{g}$	$2.14 \times 10^{-2}\,\mathrm{g}$		$2.23 \times 10^{-2}\,\mathrm{g}$

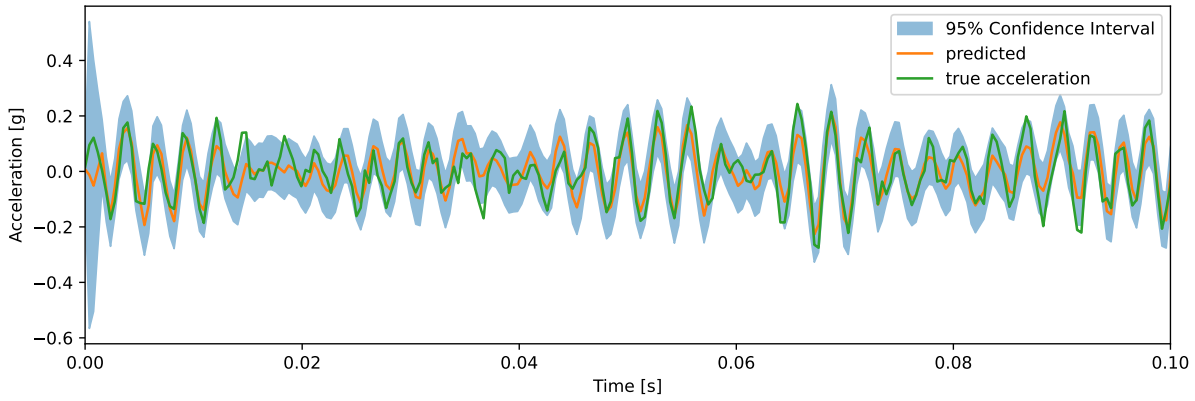
5.5.2. Time-Domain Results



(a) LSTM

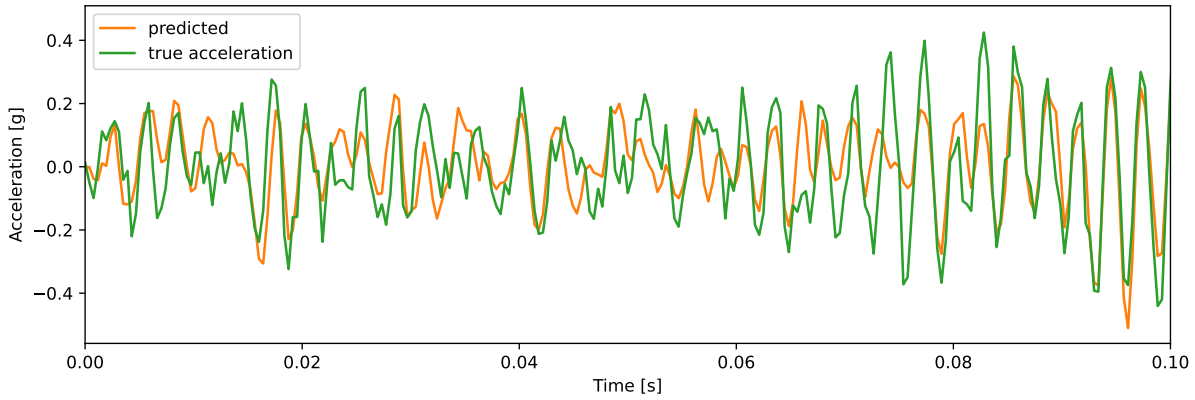


(b) LSTM-FC-GP

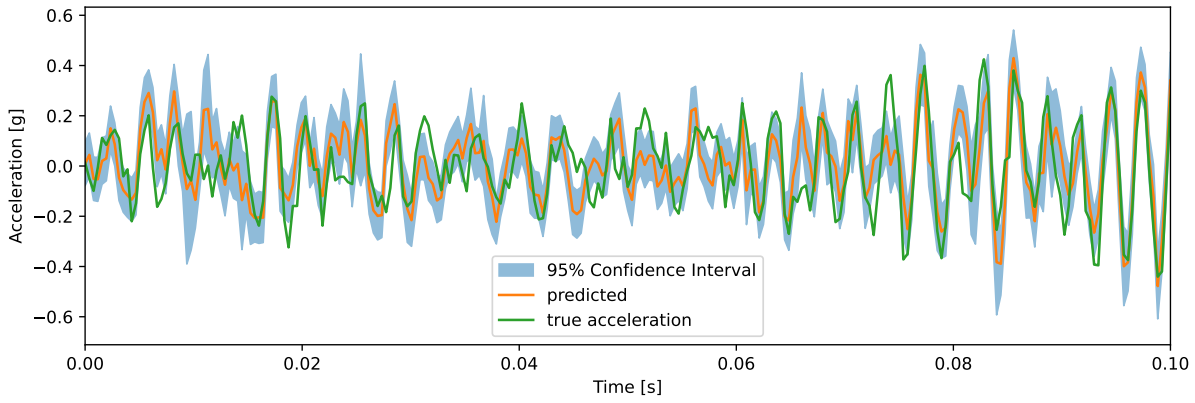


(c) LSTM-PCA-GP

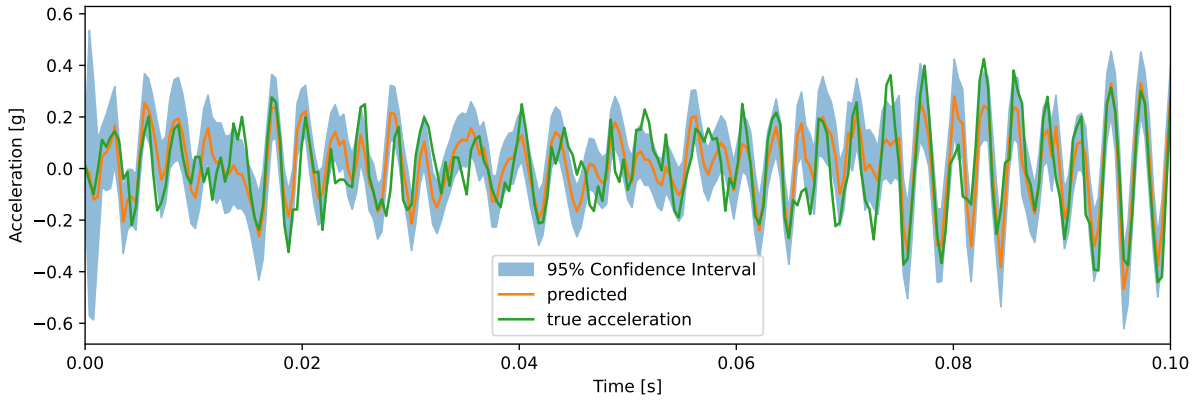
Figure 5.8: Time domain of Validation Result 1 of Fold 1



(a) LSTM

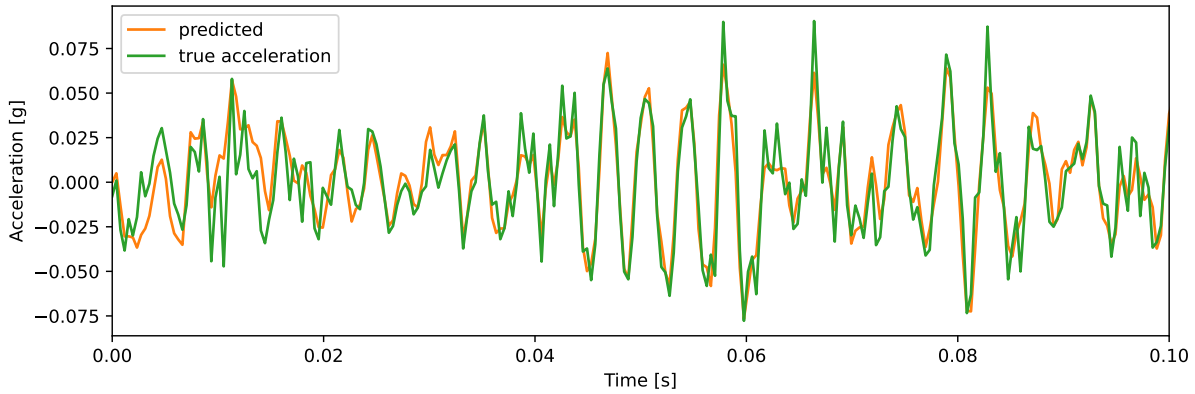


(b) LSTM-FC-GP

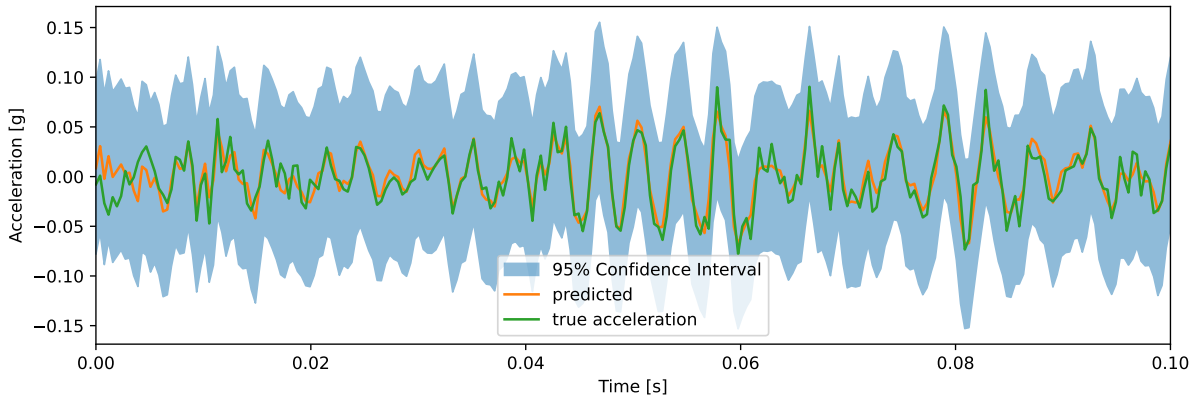


(c) LSTM-PCA-GP

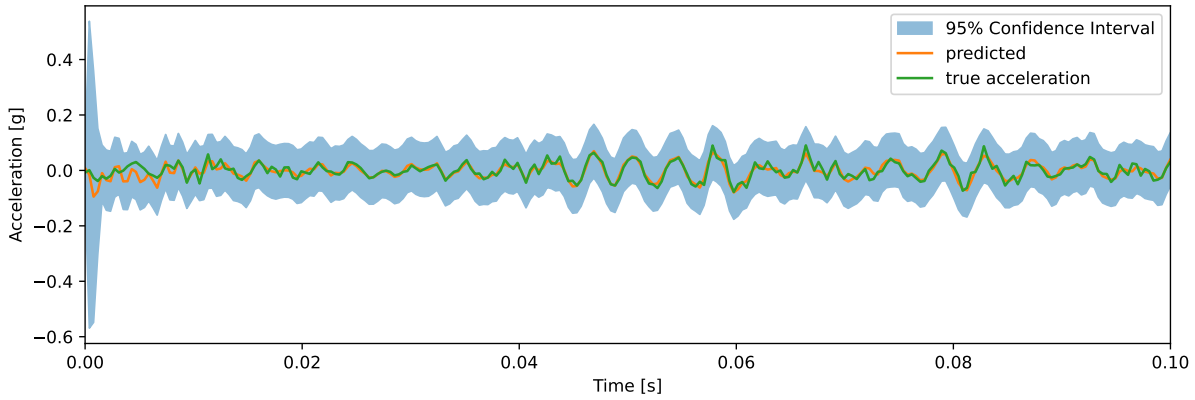
Figure 5.9: Time domain of Validation Result 2 of Fold 1



(a) LSTM

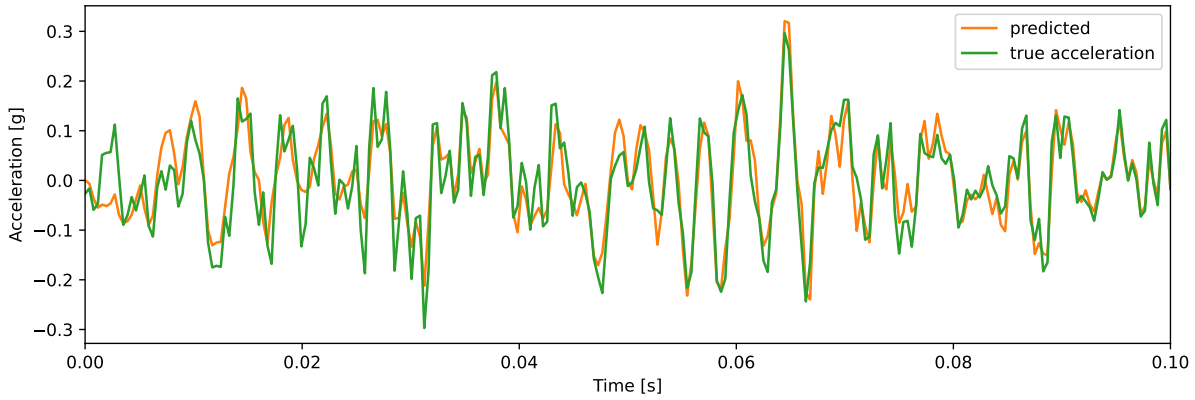


(b) LSTM-FC-GP

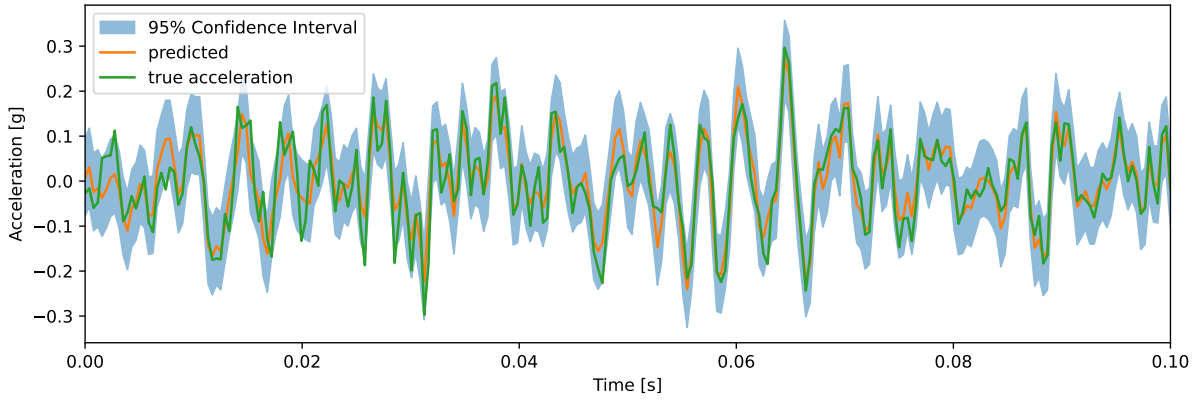


(c) LSTM-PCA-GP

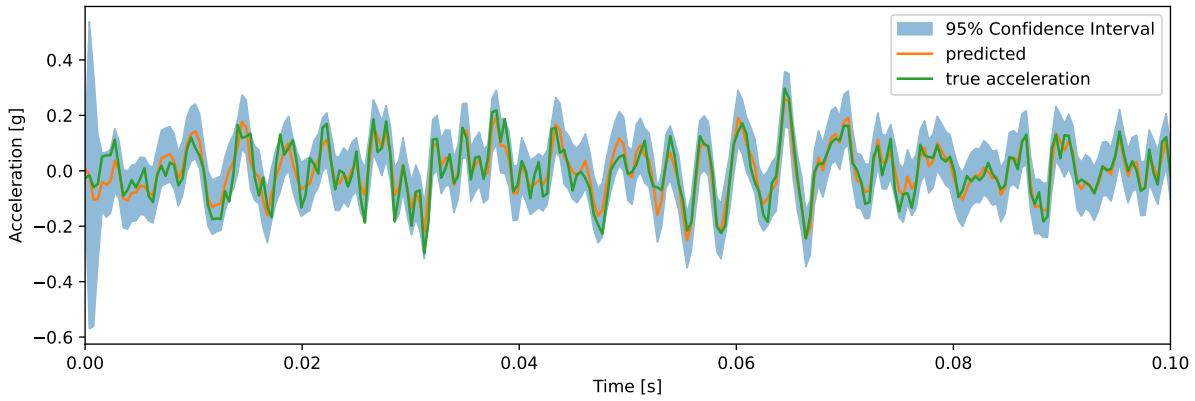
Figure 5.10: Time domain of Validation Result 3 of Fold 1



(a) LSTM

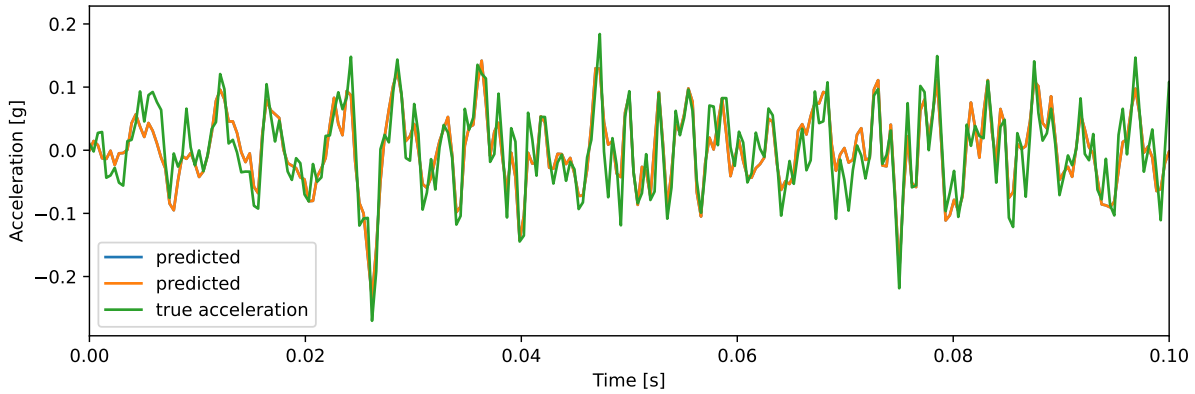


(b) LSTM-FC-GP

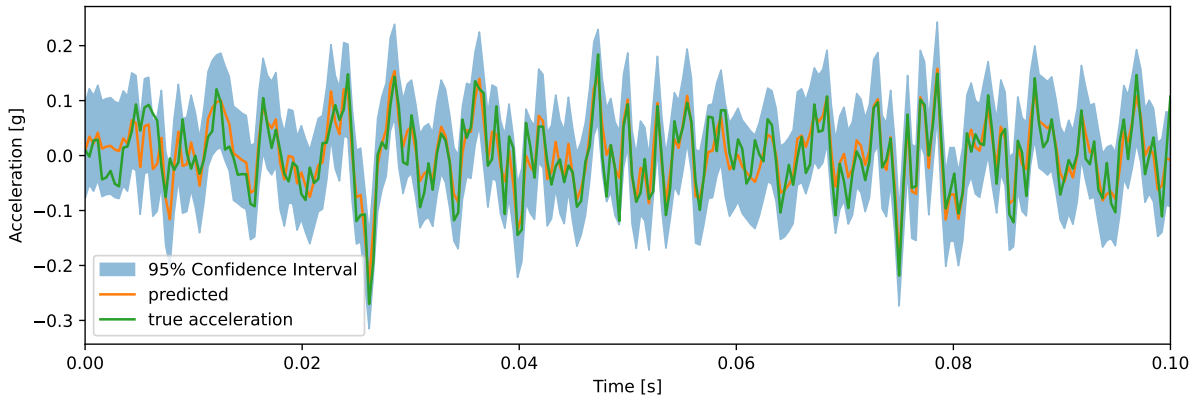


(c) LSTM-PCA-GP

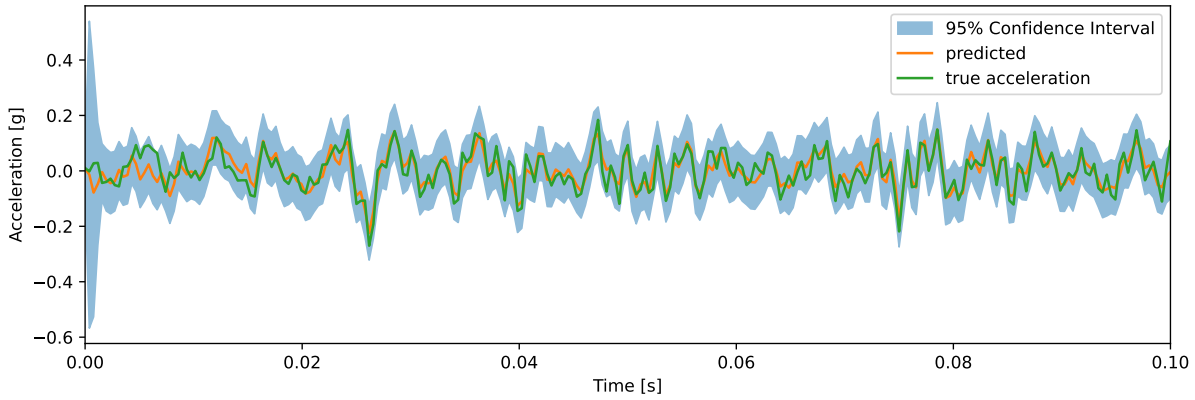
Figure 5.11: Time domain of Validation Result 4 of Fold 1



(a) LSTM

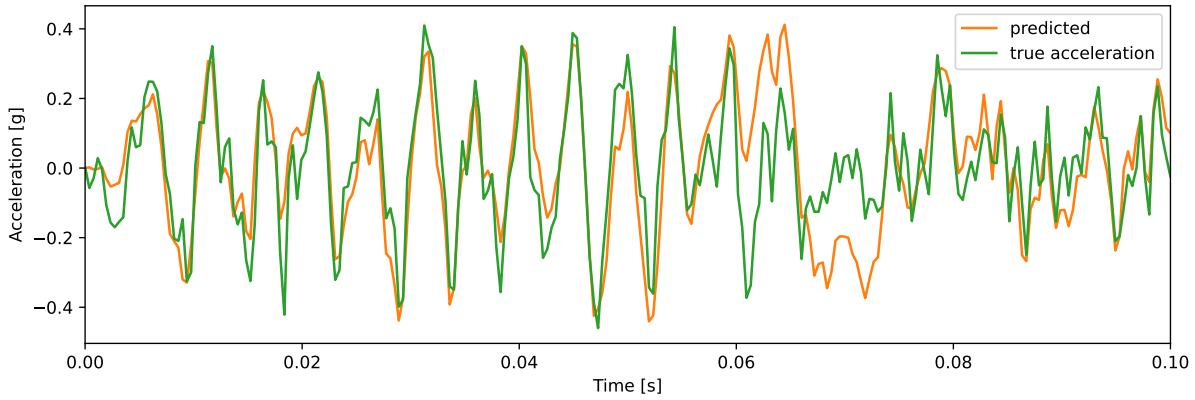


(b) LSTM-FC-GP

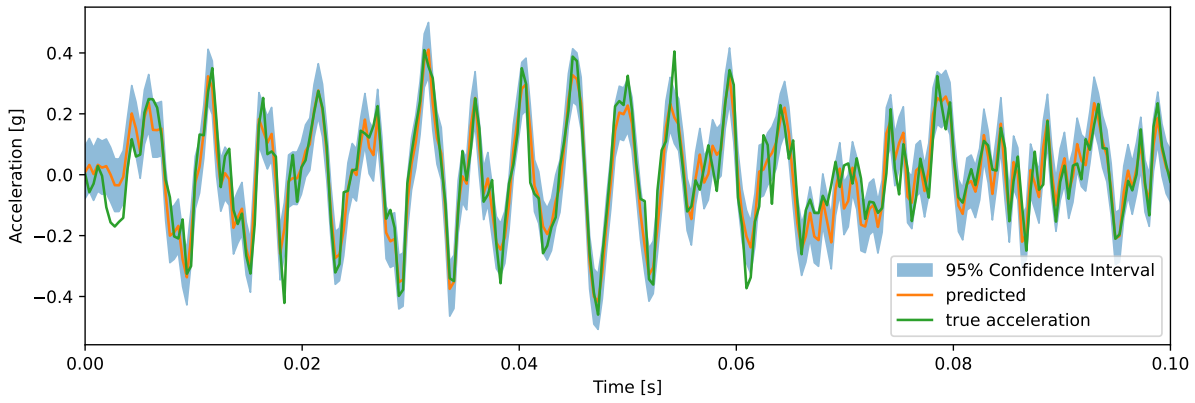


(c) LSTM-PCA-GP

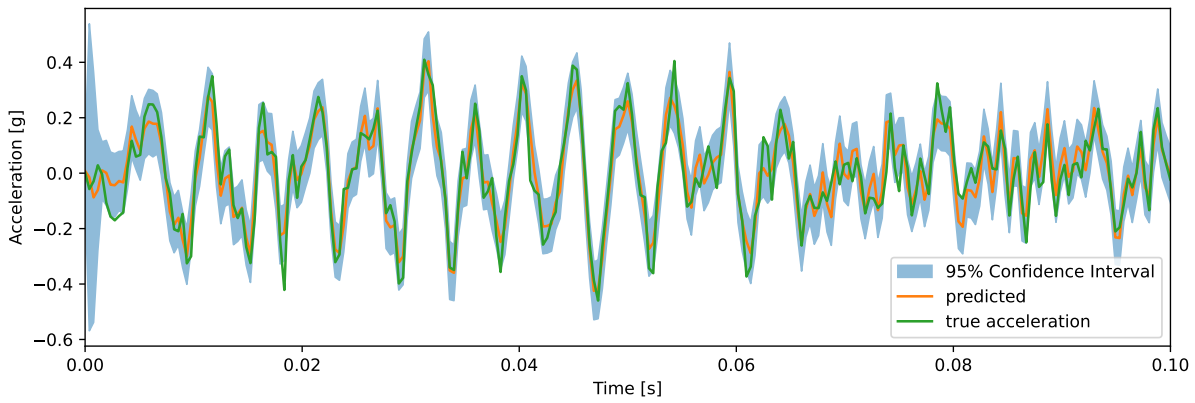
Figure 5.12: Time domain of Validation Result 5 of Fold 1



(a) LSTM



(b) LSTM-FC-GP



(c) LSTM-PCA-GP

Figure 5.13: Time domain of Validation Result 6 of Fold 1

5.5.3. The Effect of Noisy Data

To compare the effect of the noisy data on the optimal model results, the models were re-trained on the dataset excluding the test sample with a single sheet of foam. The models were re-trained using early stopping ensuring the model with the lowest validation loss is used for the results.

the NRMSE of the models can be seen in table 5.17. The results for the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models can be seen in tables 5.17a, 5.17b, and 5.17c respectively. The results show that for the LSTM and LSTM-FC-GP models the average NRMSE is lower than the average NRMSE of the default models that were trained without over-fitting the data. However, the results are still worse compared to the average of the default models that where trained without over-fitting the data when validation results 1 and 2 are not included in the average. Meanwhile, the average result of the LSTM-PCA-GP model is less accurate than the average result of the default model that is trained without over-fitting the data.

Table 5.17: NRMSE result for every test data result for every model

(a) LSTM

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.536	0.525	0.749	0.649	0.615	0.091
Fold 2	0.551	0.499	0.662	0.633	0.586	0.065
Fold 3	0.73	0.599	0.621	0.641	0.648	0.05
Fold 4	0.596	0.679	0.683	0.66	0.655	0.035
Fold 5	0.656	0.46	0.541	0.624	0.57	0.076
Average	0.614	0.552	0.651	0.641	0.615	
Std	0.072	0.078	0.069	0.012		0.074

(b) LSTM-FC-GP

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.484	0.491	0.467	0.427	0.467	0.025
Fold 2	0.58	0.519	0.714	0.658	0.618	0.074
Fold 3	0.754	0.615	0.561	0.625	0.639	0.071
Fold 4	0.444	0.492	0.796	0.74	0.618	0.152
Fold 5	0.654	0.43	0.519	0.599	0.551	0.085
Average	0.583	0.51	0.611	0.61	0.578	
Std	0.113	0.06	0.124	0.103		0.111

(c) LSTM-PCA-GP

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	0.567	0.543	0.749	0.62	0.62	0.08
Fold 2	0.815	0.754	0.976	0.974	0.88	0.098
Fold 3	0.62	0.486	0.457	0.465	0.507	0.066
Fold 4	0.544	0.574	0.693	0.659	0.618	0.061
Fold 5	0.863	0.694	0.69	0.736	0.746	0.07
Average	0.682	0.61	0.713	0.691	0.674	
Std	0.131	0.099	0.166	0.167		0.148

The percentage of data points that fall outside of the $95\,\%$ confidence interval is shown in table 5.18. The results for the LSTM-FC-GP, and LSTM-PCA-GP models is shown in tables 5.18a and 5.18b respectively. The results show that the percentage of data points that are out of bounds of the confidence interval is almost exactly $5\,\%$ which is the desired target.

The width of the confidence interval of the models retrained without the data from the test sample with a single sheet of foam is shown in table 5.19. The results from the LSTM-FC-GP and LSTM-PCA-GP models can be seen in tables 5.19a and 5.19b respectively.

Table 5.18: The percentage of points that are outside of the $95\,\%$ confidence interval

(a) LSTM-FC-GP

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$0.0\,\%$	$6.8\,\%$	$1.2\,\%$	$21.0\,\%$	$7.25\,\%$	$8.34\,\%$
Fold 2	$0.0\,\%$	$8.8\,\%$	$10.8\,\%$	$12.8\,\%$	$8.1\,\%$	$4.89\,\%$
Fold 3	$0.0\,\%$	$0.2\,\%$	$0.0\,\%$	$0.0\,\%$	$0.05\,\%$	$0.09\,\%$
Fold 4	$6.0\,\%$	$19.0\,\%$	$0.8\,\%$	$0.6\,\%$	$6.6\,\%$	$7.48\,\%$
Fold 5	$0.8\,\%$	$6.4\,\%$	$3.4\,\%$	$14.8\,\%$	$6.35\,\%$	$5.27\,\%$
Average	$1.36\,\%$	$8.24\,\%$	$3.24\,\%$	$9.84\,\%$	$5.67\,\%$	
Std	$2.34\,\%$	$6.1\,\%$	$3.94\,\%$	$8.25\,\%$		$6.61\,\%$

(b) LSTM-PCA-GP

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$0.0\,\%$	$2.8\,\%$	$1.0\,\%$	$21.8\,\%$	$6.4\,\%$	$8.95\,\%$
Fold 2	$0.0\,\%$	$4.8\,\%$	$4.6\,\%$	$8.0\,\%$	$4.35\,\%$	$2.85\,\%$
Fold 3	$0.0\,\%$	$0.4\,\%$	$0.4\,\%$	$0.0\,\%$	$0.2\,\%$	$0.2\,\%$
Fold 4	$5.2\,\%$	$13.0\,\%$	$0.0\,\%$	$0.0\,\%$	$4.55\,\%$	$5.32\,\%$
Fold 5	$0.8\,\%$	$11.4\,\%$	$4.0\,\%$	$11.4\,\%$	$6.9\,\%$	$4.64\,\%$
Average	$1.2\,\%$	$6.48\,\%$	$2.0\,\%$	$8.24\,\%$	$4.48\,\%$	
Std	$2.02\,\%$	$4.9\,\%$	$1.91\,\%$	$8.12\,\%$		$5.76\,\%$

Comparing the results from the models that were re-trained without the data from the test sample with a single sheet of foam with the results from the models that were trained on all the test samples, it can be seen that the width of the confidence interval of the models that were trained on the dataset without the test sample with a single sheet of foam is wider than that of the other models. This is consistent with the reduced number of real data points that are outside of the confidence interval.

Table 5.19: The average width of the $95\,\%$ confidence interval of every test data

(a) LSTM-FC-GP

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$1.57 \times 10^{-1}\,\mathrm{g}$	$1.58 \times 10^{-1}\,\mathrm{g}$	$1.57 \times 10^{-1}\,\mathrm{g}$	$1.59 \times 10^{-1}\,\mathrm{g}$	$1.58 \times 10^{-1}\,\mathrm{g}$	$8.51 \times 10^{-4}\,\mathrm{g}$
Fold 2	$2.40 \times 10^{-1}\,\mathrm{g}$	$2.45 \times 10^{-1}\,\mathrm{g}$	$2.42 \times 10^{-1}\,\mathrm{g}$	$2.44 \times 10^{-1}\,\mathrm{g}$	$2.43 \times 10^{-1}\,\mathrm{g}$	$1.82 \times 10^{-3}\,\mathrm{g}$
Fold 3	$2.06 \times 10^{-1}\,\mathrm{g}$	$2.06 \times 10^{-1}\,\mathrm{g}$	$2.07 \times 10^{-1}\,\mathrm{g}$	$2.07 \times 10^{-1}\,\mathrm{g}$	$2.06 \times 10^{-1}\,\mathrm{g}$	$4.65 \times 10^{-5}\,\mathrm{g}$
Fold 4	$1.71 \times 10^{-1}\,\mathrm{g}$	$1.71 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.70 \times 10^{-1}\,\mathrm{g}$	$1.71 \times 10^{-1}\,\mathrm{g}$	$5.21 \times 10^{-4}\,\mathrm{g}$
Fold 5	$2.02 \times 10^{-1}\,\mathrm{g}$	$2.03 \times 10^{-1}\,\mathrm{g}$	$2.02 \times 10^{-1}\,\mathrm{g}$	$2.03 \times 10^{-1}\,\mathrm{g}$	$2.03 \times 10^{-1}\,\mathrm{g}$	$5.44 \times 10^{-4}\,\mathrm{g}$
Average	$1.95 \times 10^{-1}\,\mathrm{g}$	$1.97 \times 10^{-1}\,\mathrm{g}$	$1.96 \times 10^{-1}\,\mathrm{g}$	$1.97 \times 10^{-1}\,\mathrm{g}$	$1.96 \times 10^{-1}\,\mathrm{g}$	
Std	$2.92 \times 10^{-2}\,\mathrm{g}$	$3.03 \times 10^{-2}\,\mathrm{g}$	$2.99 \times 10^{-2}\,\mathrm{g}$	$3.01 \times 10^{-2}\,\mathrm{g}$		$2.99 \times 10^{-2}\,\mathrm{g}$

(b) LSTM-PCA-GP

	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$2.07 \times 10^{-1}\,\mathrm{g}$	$2.08 \times 10^{-1}\,\mathrm{g}$	$2.07 \times 10^{-1}\,\mathrm{g}$	$2.10 \times 10^{-1}\,\mathrm{g}$	$2.08 \times 10^{-1}\,\mathrm{g}$	$1.16 \times 10^{-3}\,\mathrm{g}$
Fold 2	$3.51 \times 10^{-1}\,\mathrm{g}$	$3.54 \times 10^{-1}\,\mathrm{g}$	$3.53 \times 10^{-1}\,\mathrm{g}$	$3.54 \times 10^{-1}\,\mathrm{g}$	$3.53 \times 10^{-1}\,\mathrm{g}$	$1.06 \times 10^{-3}\,\mathrm{g}$
Fold 3	$1.57 \times 10^{-1}\,\mathrm{g}$	$1.58 \times 10^{-1}\,\mathrm{g}$	$1.58 \times 10^{-1}\,\mathrm{g}$	$1.58 \times 10^{-1}\,\mathrm{g}$	$1.58 \times 10^{-1}\,\mathrm{g}$	$7.52 \times 10^{-5}\,\mathrm{g}$
Fold 4	$2.12 \times 10^{-1}\,\mathrm{g}$	$2.13 \times 10^{-1}\,\mathrm{g}$	$2.10 \times 10^{-1}\,\mathrm{g}$	$2.10 \times 10^{-1}\,\mathrm{g}$	$2.11 \times 10^{-1}\,\mathrm{g}$	$1.51 \times 10^{-3}\,\mathrm{g}$
Fold 5	$2.45 \times 10^{-1}\,\mathrm{g}$	$2.46 \times 10^{-1}\,\mathrm{g}$	$2.46 \times 10^{-1}\,\mathrm{g}$	$2.47 \times 10^{-1}\,\mathrm{g}$	$2.46 \times 10^{-1}\,\mathrm{g}$	$6.74 \times 10^{-4}\,\mathrm{g}$
Average	$2.34 \times 10^{-1}\,\mathrm{g}$	$2.36 \times 10^{-1}\,\mathrm{g}$	$2.35 \times 10^{-1}\,\mathrm{g}$	$2.36 \times 10^{-1}\,\mathrm{g}$	$2.35 \times 10^{-1}\,\mathrm{g}$	
Std	$6.49 \times 10^{-2}\,\mathrm{g}$	$6.56 \times 10^{-2}\,\mathrm{g}$	$6.55 \times 10^{-2}\,\mathrm{g}$	$6.56 \times 10^{-2}\,\mathrm{g}$		$6.54 \times 10^{-2}\,\mathrm{g}$

5.5.4. PSD Accuracy

An important property of a vibration is its frequency spectrum. A PSD is often used to determine in what frequency range the vibration is most energetic. As another metric of comparison, the PSD of the measured response and the result of the default models that were trained not to over-fit the data are plotted together. This plot can be seen for every validation result in Appendix C. Moreover, the plots for fold 1 are repeated in Figures 5.14 to 5.19.

The results provide several insights. First, it can be seen that the general trend of all models is to very accurately model the PSD of the real system up to $500\,\mathrm{Hz}$. For folds 2 to 5 the LSTM model has a PSD that is up to two magnitudes lower than the measured PSD above $500\,\mathrm{Hz}$ while the LSTM-FC-GP and LSTM-PCA-GP models accurately show the amplitude of the PSD up to $700\,\mathrm{Hz}$ to $800\,\mathrm{Hz}$.

For the other validation results, the LSTM-FC-GP and LSTM-PCA-GP models provide an accurate result up to $700\,\mathrm{Hz}$ to $800\,\mathrm{Hz}$. Between $800\,\mathrm{Hz}$ and $1000\,\mathrm{Hz}$ there is a pronounced decrease in the accuracy of the models' PSD. For these same validation results, the LSTM model loses accuracy above $600\,\mathrm{Hz}$ and can be up to two magnitudes below the measured PSD. The exception to this observation is fold 1 where the LSTM model does not have a reduced accuracy compared to the LSTM-FC-GP or LSTM-PCA-GP models.

1 Foam sheet

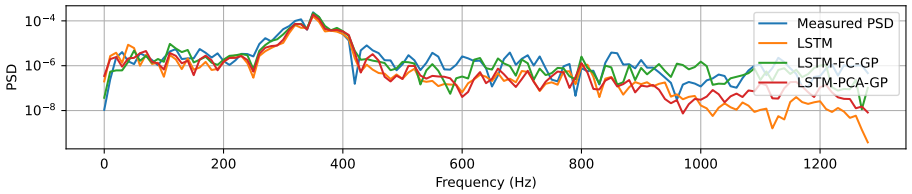


Figure 5.14: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 1 of fold 1

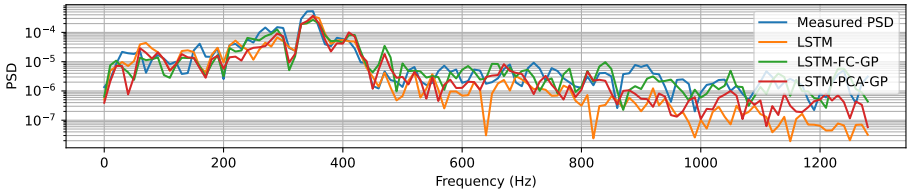


Figure 5.15: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 2 of fold 1

2 Foam sheets

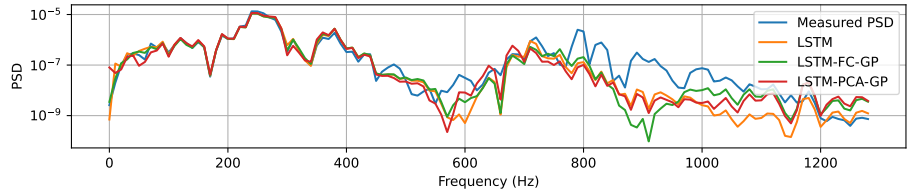


Figure 5.16: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 3 of fold 1

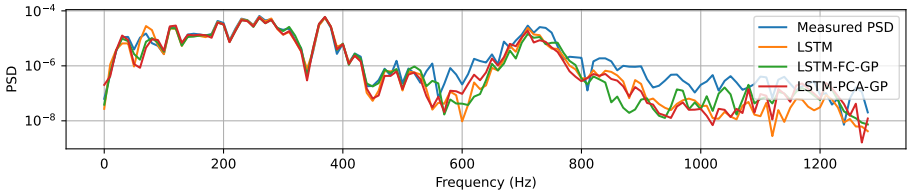


Figure 5.17: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 4 of fold 1

3 Foam sheets

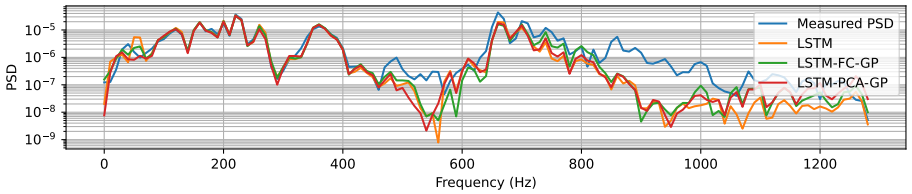


Figure 5.18: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 5 of fold 1

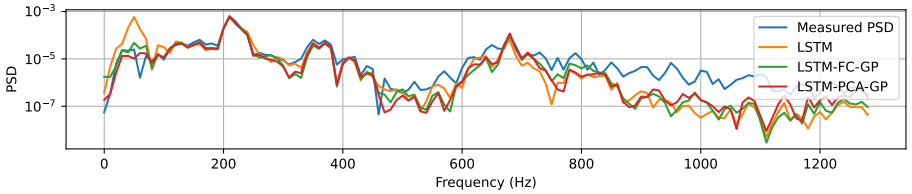


Figure 5.19: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 6 of fold 1

5.6. Model Extrapolation

One of the main reasons a time-stepper solver was chosen is because a time-stepper is able to extrapolate beyond the duration of the training data. To determine the performance of the models when extrapolating the data beyond the training duration, the models were re-trained up to time-step 250. Afterwards, the models were used to simulate the response of the validation data up to time-step 500. The accuracy of the first 250 time-steps of the validation data is then compared to the accuracy of the last 250 time-steps of the validation data. This change in accuracy can be seen in Section 5.6.1. To make a qualitative comparison between the first and second half of the validation results, the time-domain plots can be found in Section 5.6.2.

5.6.1. Percent Change in NRMSE

To compare the performance of the models, the NRMSE of the first part and the part that is extrapolated beyond the training length of the model of the validation results is taken. The percent change of the NRMSE of the extrapolated part with respect to the first part is then taken. This serves as an indication of how much the accuracy of the models changes as the time domain simulation is run longer than the training duration. These results can be found in table 5.20 with the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models in sub-tables 5.20a, 5.20b, and 5.20c respectively.

Table 5.20: Percent change in NRMSE between trained duration and extrapolated duration of validation results

(a) LSTM

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$5.7\,\%$	$-10.1\,\%$	$1.8\,\%$	$-20.7\,\%$	$-21.0\,\%$	$24.3\,\%$	$-3.3\,\%$	$16.0\,\%$
Fold 2	$0.4\,\%$	$33.8\,\%$	$-8.6\,\%$	$65.2\,\%$	$-20.0\,\%$	$1.3\,\%$	$12.0\,\%$	$28.9\,\%$
Fold 3	$-6.7\,\%$	$-4.6\,\%$	$14.4\,\%$	$-45.1\,\%$	$4.6\,\%$	$-2.2\,\%$	$-6.6\,\%$	$18.6\,\%$
Fold 4	$5.2\,\%$	$-18.6\,\%$	$-27.9\,\%$	$-1.3\,\%$	$22.7\,\%$	$8.5\,\%$	$-1.9\,\%$	$16.9\,\%$
Fold 5	$-23.9\,\%$	$19.7\,\%$	$8.4\,\%$	$2.3\,\%$	$27.7\,\%$	$-14.9\,\%$	$3.2\,\%$	$18.1\,\%$
Average	$-3.9\,\%$	$4.1\,\%$	$-2.4\,\%$	$0.1\,\%$	$2.8\,\%$	$3.4\,\%$	$0.7\,\%$	
Std	$11.0\,\%$	$19.6\,\%$	$14.9\,\%$	$36.7\,\%$	$20.5\,\%$	$12.9\,\%$		$21.3\,\%$

(b) LSTM-FC-GP

	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$-24.3\,\%$	$-6.4\,\%$	$-1.6\,\%$	$-27.3\,\%$	$-25.0\,\%$	$61.5\,\%$	$-3.8\,\%$	$30.8\,\%$
Fold 2	$0.5\,\%$	$27.1\,\%$	$-10.8\,\%$	$24.6\,\%$	$-23.0\,\%$	$-1.3\,\%$	$2.8\,\%$	$18.0\,\%$
Fold 3	$-15.0\,\%$	$-8.5\,\%$	$11.5\,\%$	$-40.0\,\%$	$-5.2\,\%$	$-3.4\,\%$	$-10.1\,\%$	$15.6\,\%$
Fold 4	$20.9\,\%$	$-14.3\,\%$	$-23.3\,\%$	$-3.8\,\%$	$-12.9\,\%$	$-27.1\,\%$	$-10.1\,\%$	$15.7\,\%$
Fold 5	$-14.6\,\%$	$18.8\,\%$	$5.8\,\%$	$-15.6\,\%$	$26.6\,\%$	$5.7\,\%$	$4.4\,\%$	$15.6\,\%$
Average	$-6.5\,\%$	$3.4\,\%$	$-3.7\,\%$	$-12.4\,\%$	$-7.9\,\%$	$7.1\,\%$	$-3.3\,\%$	
Std	$15.8\,\%$	$16.4\,\%$	$12.3\,\%$	$22.1\,\%$	$18.7\,\%$	$29.4\,\%$		$21.0\,\%$

(c) LSTM-PCA-GP

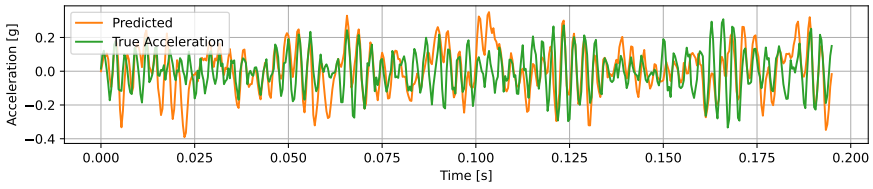
	Validation Result 1	Validation Result 2	Validation Result 3	Validation Result 4	Validation Result 5	Validation Result 6	Average	Std
Fold 1	$-27.5\,\%$	$-5.3\,\%$	$-3.7\,\%$	$-23.9\,\%$	$-22.5\,\%$	$48.8\,\%$	$-5.7\,\%$	$26.0\,\%$
Fold 2	$8.6\,\%$	$23.4\,\%$	$-9.4\,\%$	$22.2\,\%$	$-20.7\,\%$	$0.6\,\%$	$4.1\,\%$	$16.0\,\%$
Fold 3	$-13.0\,\%$	$-14.9\,\%$	$11.7\,\%$	$-36.9\,\%$	$-1.7\,\%$	$-1.8\,\%$	$-9.4\,\%$	$15.0\,\%$
Fold 4	$22.9\,\%$	$-17.8\,\%$	$-9.1\,\%$	$-17.5\,\%$	$-14.7\,\%$	$-21.9\,\%$	$-9.7\,\%$	$15.1\,\%$
Fold 5	$-19.3\,\%$	$11.9\,\%$	$4.1\,\%$	$-15.6\,\%$	$19.0\,\%$	$-2.7\,\%$	$-0.4\,\%$	$13.8\,\%$
Average	$-5.7\,\%$	$-0.5\,\%$	$-1.3\,\%$	$-14.3\,\%$	$-8.1\,\%$	$4.6\,\%$	$-4.2\,\%$	
Std	$18.6\,\%$	$15.9\,\%$	$8.1\,\%$	$19.7\,\%$	$15.4\,\%$	$23.5\,\%$		$18.5\,\%$

The results show that the the LSTM model has an increase in the average NRMSE of less than $1\,\%$. The LSTM-FC-GP and LSTM-PCA-GP models show a reduced average NRMSE with the LSTM-PCA-GP model having the highest decrease. Furthermore, the models with a GP layer have a lower overall standard deviation with the LSTM-PCA-GP model having the lowest value.

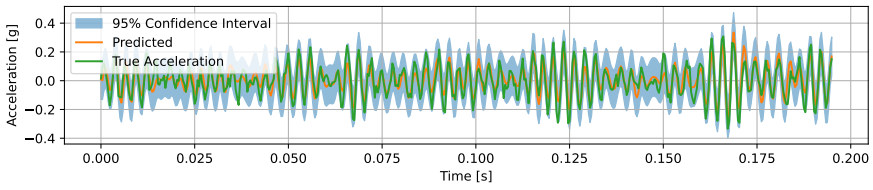
5.6.2. Time Domain Results

To find out any qualitative differences between the first half and the second half of the simulation, the time-domain plots are shown here. To make the comparison easier, the shown time domain plots are all validation results from fold 1 which are also shown for the default models in Section 5.5.2.

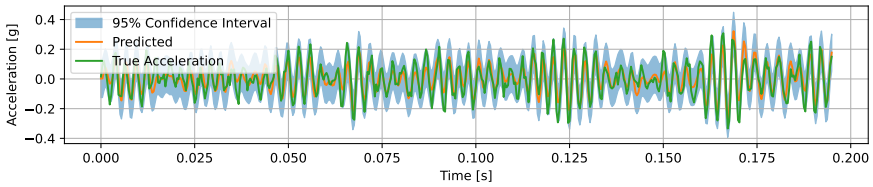
1 Foam Sheet



(a) LSTM

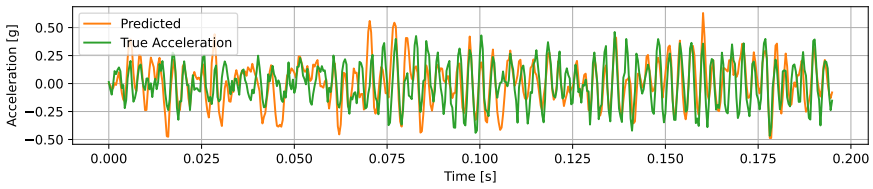


(b) LSTM-FC-GP

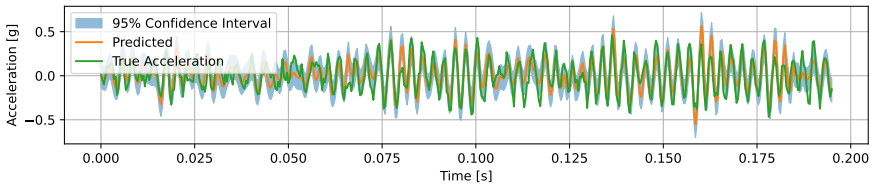


(c) LSTM-PCA-GP

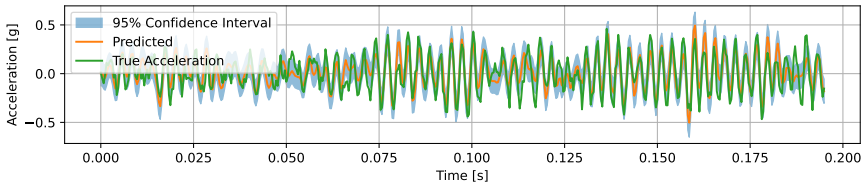
Figure 5.20: Time domain of Validation Result 1 of Fold 1 of the models trained for half the duration



(a) LSTM



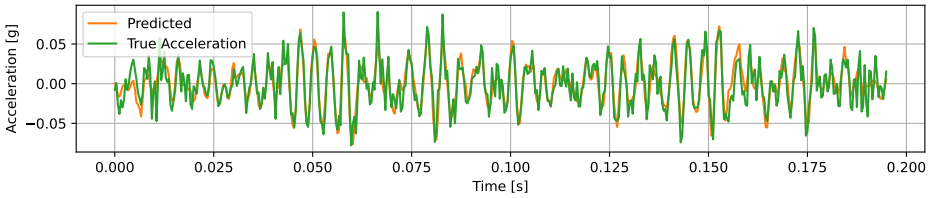
(b) LSTM-FC-GP



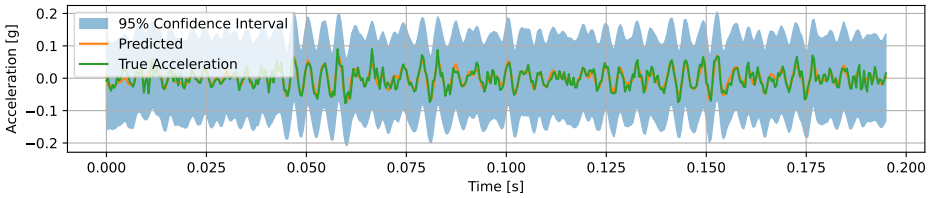
(c) LSTM-PCA-GP

Figure 5.21: Time domain of Validation Result 2 of Fold 1 of the models trained for half the duration

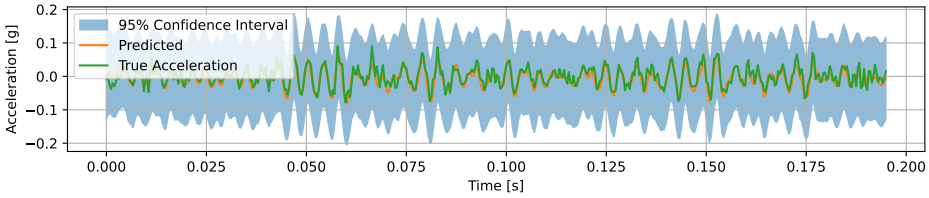
2 Foam Sheets



(a) LSTM

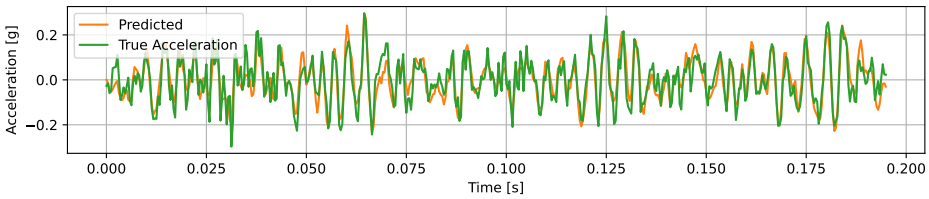


(b) LSTM-FC-GP

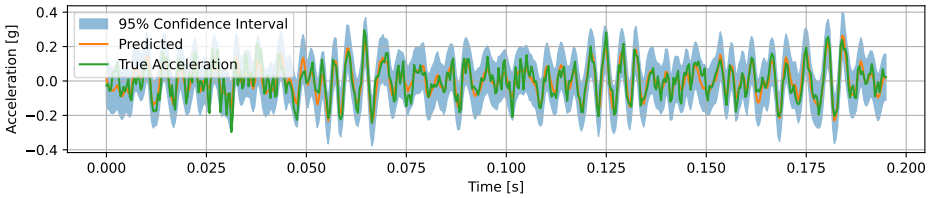


(c) LSTM-PCA-GP

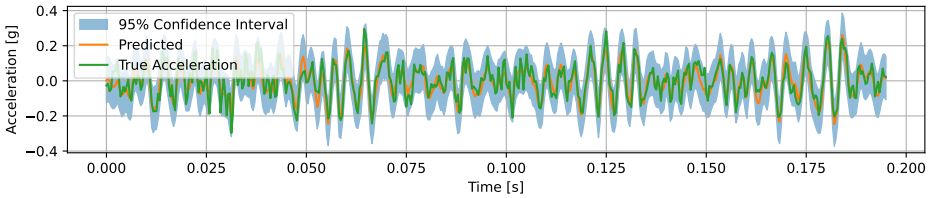
Figure 5.22: Time domain of Validation Result 3 of Fold 1 of the models trained for half the duration



(a) LSTM



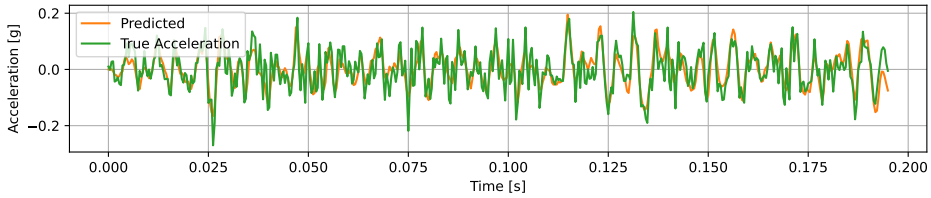
(b) LSTM-FC-GP



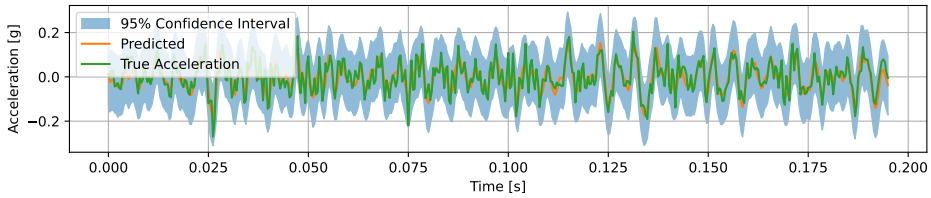
(c) LSTM-PCA-GP

Figure 5.23: Time domain of Validation Result 4 of Fold 1 of the models trained for half the duration

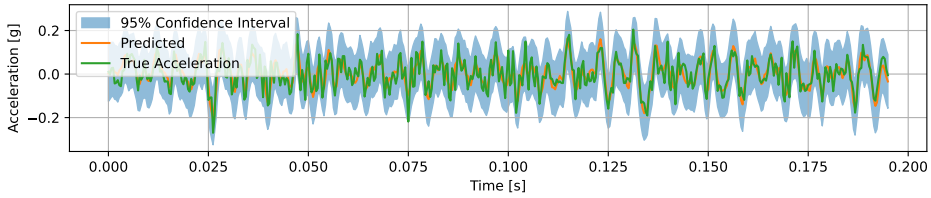
3 Foam Sheets



(a) LSTM

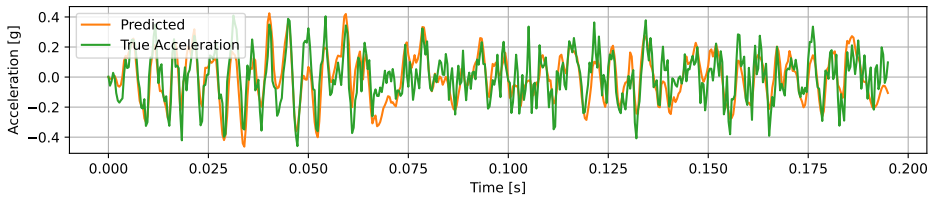


(b) LSTM-FC-GP

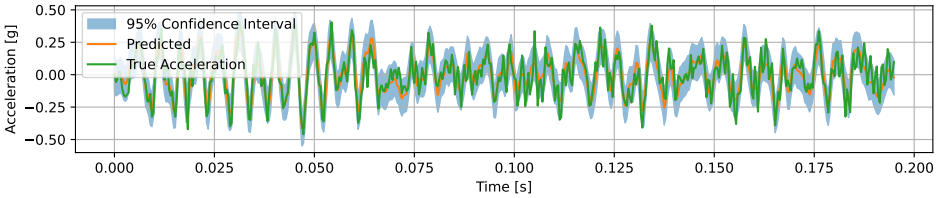


(c) LSTM-PCA-GP

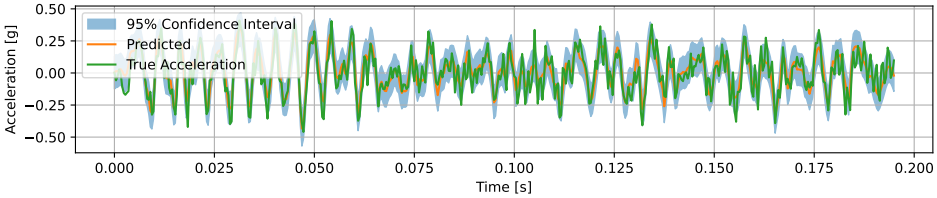
Figure 5.24: Time domain of Validation Result 5 of Fold 1 of the models trained for half the duration



(a) LSTM



(b) LSTM-FC-GP



(c) LSTM-PCA-GP

Figure 5.25: Time domain of Validation Result 6 of Fold 1 of the models trained for half the duration

5.7. Model Performance

To make a fair trade-off between the models a benchmark was performed to see how much resources each model uses in order to run. This benchmark was done on the default models on a windows machine with all but the critical background processes and task manager closed. Both the training and run test were performed on time domain signals with 500 time-steps. The average time per epoch was measured after 50 epochs while the time to run was averaged over 10 runs. During this test, the total memory usage as well as the video memory usage were measured during the run test and reported. As the training memory usage depends on the training dataset size, this memory usage was not included.

The results of the benchmark can be seen in table 5.21. The results show that the LSTM model has the lowest time per epoch, time to run, and memory usage of all the models. Furthermore, as it does not utilise CUDA, the model does not require any video memory to run. Meanwhile the LSTM-FC-GP and LSTM-PCA-GP models perform nearly identically. Both the training time and the time to run are several times longer than the LSTM while the required memory is over $80\,\%$ larger. Meanwhile the models use more than$1500\,\mathrm{MiB}$ of video memory to run.

Table 5.21: Benchmark results of all models

Model	Time per Epoch	Time to Run	Memory Usage	Video Memory Usage
LSTM	$2.36\,\mathrm{s}$	$1.054\,\mathrm{s}$	$366\,\mathrm{MiB}$	Not Applicable
LSTM-FC-GP	$7.26\,\mathrm{s}$	$2.484\,\mathrm{s}$	$668\,\mathrm{MiB}$	$1538\,\mathrm{MiB}$
LSTM-PCA-GP	$7.24\,\mathrm{s}$	$2.342\,\mathrm{s}$	$668\,\mathrm{MiB}$	$1559\,\mathrm{MiB}$

Because the models using a GP layer can also be run on the CPU instead of on CUDA, the models were re-run on the CPU. Training the models using a CPU instead of a GPU has proven to take impractically long. Therefore, the Time per epoch results are omitted. The results of this benchmark are shown in table 5.22. The results show that the time to run increases to over $3\,\mathrm{s}$. While the video memory usage has shifted to system memory, the total value is lower than video memory and system memory combined of the models run on a GPU

Table 5.22: Benchmark results of models with a GP layer run on CPU

Model	Time to run	Memory usage
LSTM-FC-GP	$3.05\,\mathrm{s}$	$1873\,\mathrm{MiB}$
LSTM-PCA-GP	$3.03\,\mathrm{s}$	$1605\,\mathrm{MiB}$

6 Discussion

The results that are presented in Chapter 5 are discussed in this chapter. The chapter follows the same structure that is found in Chapter 5. Readers are advised to read Chapter 6 and this chapter side by side. In addition to discussing the results from Chapter 5, a trade-off between the different models is made in Section 6.8.

6.1. Default Model Accuracy

This section discusses the results of the default models shown in Section 5.1. The summarised results are discussed in Section 6.1.1. More context is given to the summarised results by discussing the full results in 6.1.2. Finally, the difference between the results of the LSTM-FC-GP and LSTM-PCA-GP models is discussed in Section 6.1.3.

6.1.1. Summarised Results

The NRMSE shown in table 5.1 shows that the LSTM-PCA-GP model has the lowest NRMSE, followed by the LSTM-FC-GP, and LSTM models. Because the models using a GP layer use the LSTM model as a basis, it is expected that the models that use a GP layer perform better at the downside of being computationally more expensive. As is expected, the LSTM model performs the worst of the models. However, the LSTM-PCA-GP has a lower NRMSE than the LSTM-FC-GP model. This is an unexpected result as the PCA layer reduces the input dimension of the GP layer in a way that reduces the amount of information available to the GP layer while the FC layer reduces the input dimension of the GP in a way that can potentially maintain all the information that the LSTM layer has in its output. This leads to the expectation that the LSTM-FC-GP provides better results than the LSTM-PCA-GP which is the opposite of what is seen.

6.1.2. Full Results

To give more context to the summarised results discussed in Section 6.1.1, the full results of each validation result for each fold are evaluated. Examining the full results for the NRMSE shown in table 5.3 shows that despite the lower average NRMSE the average standard deviation of the LSTM-FC-GP and LSTM-PCA-GP is higher than the standard deviation of the NRMSE of the LSTM model. Moreover, the NRMSE is noticeably higher for validation results 1 and 2. This effect is even stronger with the models that have a GP layer where validation results 1 and 2 have an even higher NRMSE despite the lower overall average NRMSE. This effect is likely the cause of the higher standard deviation seen in the LSTM-FC-GP and LSTM-PCA-GP models.

Validation results 1 and 2 correspond to the test case with only a single sheet of foam. Considering this, the phenomenon of the worse performance can be explained by noting the issues that were encountered while performing tests to create the dataset noted in Section 4.1.3. Here, a high amount of noise was observed when performing chirp tests at low frequency and high amplitude. Examining the frequency domain of the random vibration tests reveals that the noise that is clearly visible in the

chirp tests is also present in the random vibration tests. The NRMSE results presented in table 5.3 show that this noise has a significant impact on the accuracy of the model.

The finding that the difference in the NRMSE between noisy test data and less noisy test data is greater for models using a GP layer suggests that the models with a GP layer are better at distinguishing between noise and data during training. However, it makes them less suited for simulations with a noisy input signal. Given that the goal of the models is to predict the vibration and shock response before doing a real test, distinguishing between noise and data during training is a more beneficial trait as the simulation's input is meant to be a synthetic signal without input noise.

A summary of the average NRMSE of the models when removing validation results 1 and 2 from the average is shown in table 6.1. Examining these results reveals how the noise affects the accuracy of the different models. The LSTM model only sees a small decrease in the NRMSE of 0.025. Meanwhile the LSTM-FC-GP model has a much larger decrease of 0.067. Finally, despite already having the lowest NRMSE the LSTM-PCA-GP model has the largest decrease of 0.104. This suggests that the models with a GP layer not only provide better overall results but are better at distinguishing between noise and real data during training, resulting in better performance for the models with a GP layer.

Table 6.1: Average results of the K-fold cross validation without considering the score of the test data from the case with only a single sheet of foam

	LSTM	**LSTM-FC-GP**	**LSTM-PCA-GP**
NRMSE	0.656	0.576	0.503

While the scores of both the LSTM-FC-GP model, and the LSTM-PCA-GP model improve drastically when leaving out the scores from test data 1 and 2, the LSTM-PCA-GP still performs better than the LSTM-FC-GP model despite the expectation being the opposite. The reason to why this happens is discussed in Section 6.1.3.

6.1.3. Difference Between LSTM-FC-GP and LSTM-PCA-GP models

Comparing the accuracy of the LSTM-FC-GP and LSTM-PCA-GP models reveals a number of key insights. First, when comparing the NRMSE results of the K-fold cross-validation there is a key difference in the way the two models are influenced by the effect of the noisy data. While the overall accuracy of the LSTM-PCA-GP model is better than the accuracy of the LSTM-FC-GP model, the accuracy of the LSTM-FC-GP model is higher for validation results 1 and 2. This implies that the LSTM-FC-GP model was optimised in a way that balances the accuracy of the model for the noisy training data and the training data with less noise.

In contrast, the LSTM-PCA-GP model has a lower accuracy for validation results 1 and 2 but has a comparatively higher accuracy for the other validation results. This means that this model was optimised in a way that considers the noisy data to a much lesser extend and instead focusses on getting a high accuracy for the other test results. This makes the LSTM-PCA-GP model score much higher when the first two test results are left out of the average scores.

While the PCA layer filtering out the noise explains why the the LSTM-PCA-GP performs this way, it does not explain entirely why the LSTM-FC-GP model scores lower on average than the LSTM-PCA-GP model despite having more information available to the GP layer. To explain this behaviour the training loss of both models for fold 2, which is shown in Figure 5.2 should be compared. This figure shows the LSTM-FC-GP model scoring a much lower negative log likelihood score compared to the LSTM-PCA-GP model. This lower training loss combined with the lower accuracy very strongly implies that the LSTM-FC-GP model is over-fitting the data, at least to a higher extend than the LSTM-PCA-GP model.

This observation is confirmed when evaluating the confidence interval of these two models. As noted in Section 5.1 and shown visualised in Figure 5.1, the amount of real data points that are outside of the $95\,\%$ confidence interval is much higher than $5\,\%$. Table 5.4 shows a precise breakdown of the percentage, showing that for the LSTM-FC-GP model almost $50\,\%$ of the real data points fall outside of the confidence interval while for the LSTM-PCA-GP model only $25\,\%$ of the real data points are outside of the confidence interval. Furthermore, the LSTM-PCA-GP model has a lower percentage of data

points outside of the confidence interval across all validation results suggesting the confidence interval is more accurate even for the cases with only a single sheet of foam.

This result is only possible if the width of the confidence interval is larger for the LSTM-PCA-GP model than it is for the LSTM-FC-GP model. This is confirmed in table 5.5. This table shows the confidence interval for each validation result for both models. It shows that the average width of the confidence interval of the LSTM-PCA-GP model is approximately double that of the LSTM-FC-GP model. Furthermore, it shows that the average confidence interval of each test result is approximately the same for every fold for both models, showing that the GP layer has difficulty distinguishing between the data points it can accurately predict and the data points it cannot accurately predict. This effect is particularly visible when looking at the percentage of points outside of the confidence interval for the LSTM-PCA-GP model as some of the test results have a percentage of points of almost $0\,\%$ while ideally this should be $5\,\%$. This implies that the performance of the models can be improved drastically if the GP layers were given a measure with which they could evaluate the quality of the data points.

Comparing all the information presented in this section the conclusion can be made that the LSTM-PCA-GP model performs better than the LSTM-FC-GP model because the PCA layer reduces the amount of information presented to the GP layer, effectively filtering the noise, and preventing the model from over-fitting the training data. Meanwhile the LSTM-FC-GP model reduces the data size without reducing the information in the model, allowing the GP layer to over-fit the data, reducing the accuracy of the model.

6.2. Varying the Steps Skipped by GP Layer

To determine how leaving out a large portion of the training time steps during training influences performance, the models using a GP layer were re-trained on fold 1 with double the amount of time steps used during training, going from one in four to one in two time steps. This change significantly impacted the models' performance, increasing both the training time and the required memory by a substantial amount. This increase raised the required amount of video memory to run the model above the $8\,\mathrm{GB}$ available, forcing the model to be run on the system's CPU. As a result, training this single fold took approximately the same amount of time as the entire k-fold cross validation with one in four time steps used as input for the GP layer. Because of this, the models were only re-trained for the first fold to evaluate the effect of skipping time steps during training.

Comparing the NRMSE values of the models with more and fewer time steps, shown in table 5.6, shows that considering more time steps during training does not favourably influence the over-fitting problem that was seen in the case with fewer time steps as the NRMSE is higher for both models compared to fold 1 of the default models. Looking at the individual validation results shows that the accuracy for validation results 1 and 2 is practically unchanged or even slightly improved while the accuracy of validation results 3 to 6 is significantly worse. While the standard deviation of the models with more time steps is lower, this likely only due to the fact that the NRMSE of validation results 3 to 6 is closer to the accuracy of validation results 1 to 2 due to the decreased overall accuracy of the models.

These results imply that the additional information that is supplied to the GP layer worsens the over fitting problem by a considerable amount. This is confirmed by evaluating the training loss that is shown in Figure 5.3 which shows the the models with a higher number of time steps decreasing at a higher rate to a lower negative marginal log likelihood.

This is is also visible in the percentage of points that are outside of the $95\,\%$ confidence interval, as well as the width of the confidence interval which can be seen in tables 5.7 and 5.8. Adding a larger number of time steps during training increases the percentage of real data points that are outside of the confidence interval by a large amount. This combined with the decreasing width of the confidence interval, which by itself is a good thing, clearly indicates that the model is severely over-fitted.

This over-fitting problem has a much larger impact on the LSTM-PCA-GP than it has on the LSTM-FC-GP model. The loss of information caused by the PCA layer which prevented the model from over-fitting to the same degree as the LSTM-FC-GP model no longer has the same effect. As a result, the LSTM-PCA-GP model is able to over-fit the training data to the same degree that was seen in the LSTM-FC-GP model.

6.3. The Effect of Noisy Data

To determine the effect of the noisy data from the case with only a single sheet of foam on the performance of the models, the models were re-trained for fold 1 without the data from the testcase with only a single sheet of foam. The results of this can be seen in Section 5.3.

Comparing the NRMSE of the models that are re-trained without the data from test case with a single sheet of foam to the results of the default models the results are mixed. The average results for the LSTM-FC-GP and LSTM-PCA-GP improve. However, the average NRMSE of the LSTM model is higher than the average NRMSE of the default LSTM model. Furthermore, when comparing the average results of the models that are trained without the data from the test case with a single sheet of foam to the average of the default models when excluding the results from validation results 1 and 2, the NRMSE of all models becomes significantly worse.

Examining the average width of the $95\,\%$ confidence interval, it can be seen that the width of the confidence interval of the LSTM-FC-GP model increases. Meanwhile the width of the confidence interval of the LSTM-PCA-GP is slightly reduced. For the LSTM-FC-GP model this directly translates into an improvement to the accuracy of the confidence interval. Moreover, despite the decrease in the width, the accuracy of the confidence interval of the LSTM-PCA-GP model is also increased.

When calculating the average percentage of real data points outside the $95\,\%$ confidence interval of the default models when excluding validation results 1 and 2 the average becomes $40.6\,\%$ for the LSTM-FC-GP model, and $20.0\,\%$ for the LSTM-PCA-GP model. Comparing this result to the result of the models trained without data from test case 1, the LSTM-FC-GP still has a percentage that is closer to the target $5\,\%$. However, the LSTM-PCA-GP model now has a percentage that is further away from the $5\,\%$ target which is consistent with the smaller width of its $95\,\%$ confidence interval. These results point to the LSTM-FC-GP and LSTM-PCA-GP models still over-fitting the training data.

These results show that removing the noisy data from the test case with a single sheet of foam did not prevent the models with a GP layer from over-fitting the data. Instead the model performance is worse on average for the LSTM model and worse when only considering validation results 3 to 6 for the LSTM-FC-GP and LSTM-PCA-GP models. How the noisy data influences model accuracy when over-fitting is prevented is discussed in more detail in section 5.5.3

6.4. Effect of GP Input Dimension

To determine the effect of changing the input dimension of the GP layer of the LSTM-FC-GP and LSTM-PCA-GP models, the input dimension of the GP layer was reduced from 8 to 6, and 4, and the models were re-trained. The result of this can be seen in Section 5.4.

The two different models respond to a decrease in the GP input dimension in a slightly different way. When decreasing the GP input dimension from 8 to 6, both models have a slight reduction in their NRMSE and standard deviation. However, when the GP input dimension is further reduced to 4, the NRMSE increases above the level of the default model with a higher standard deviation for the LSTM-FC-GP model. However, for the LSTM-PCA-GP model, the average NRMSE decreases further slightly. This slight decrease in NRMSE is accompanied by a moderate increase in the standard deviation suggesting that the overall result could be considered worse than for the model with a GP input dimension of 6.

The difference in the behaviour of the two models to the reduction in the GP input dimension can be explained by evaluating how they reduce the GP input dimension. The LSTM-FC-GP model reduces the GP input dimension by means of a fully connected layer. While this layer needs to be trained in order to accurately reduce the dimension of the system, it is connected to all output nodes of the LSTM layer, reducing the GP input dimension in a way that can potentially retain all the information in LSTM output. This means that reducing the GP input dimension does not prevent the model from over-fitting the training data.

Meanwhile, the LSTM-PCA-GP model uses principal component analysis to reduce the output dimension of the LSTM model. principal component analysis finds a new coordinate system where all axes are orthogonal. These axes are ranked for how much they correlate with the desired output. Principal component analysis reduces the dimension by only keeping the most important axes and discarding the

remaining axes. This means that information is lost when using principal component analysis. When the input dimension of the GP layer is reduced, the PCA layer discards more axes of its coordinate system, further reducing the amount of information that is retained in the output. This reduction in information hinders the GP layer's ability to over-fit the training data, reducing the degree to which the model over-fits the training data.

The difference between this behaviour can be observed by examining the validation loss of both models for different input dimensions shown in Figure 5.6. For the LSTM-FC-GP model, the negative marginal log likelihood of the validation data coincides until epoch 1000. After this epoch the negative marginal log likelihood of the model with a GP input dimension of 4 increases slower than the negative marginal log likelihood of the models with a larger GP input dimension. In contrast, the negative marginal log likelihood of validation data of the LSTM-PCA-GP models diverges after epoch 500. After this point, the negative marginal log likelihood increases at a slower rate as the GP input dimension of the model suggesting the smaller GP input dimension prevents the model from over-fitting by forcing the PCA to discard more data.

For all the models it can be observed that before epoch 500 all negative marginal log likelihoods are roughly identical implying the peak accuracy of the models is expected to be nearly identical regardless of the GP input dimension. This means that there is some margin to reduce the GP input dimension, reducing the amount of memory and time required to run and train the models.

6.5. Model Performance Without Over-Fitting

In the results discussed in the previous sections, over-fitting was observed. To determine the degree of this over-fitting as well as from what epoch over-fitting negatively affects the results, the validation loss during training was measured for fold 1 and shown in Figure 5.7. This figure shows the LSTM model was not over-fitted to a degree that significantly impacts the performance of the model. Meanwhile, the LSTM-FC-GP and LSTM-PCA-GP models are severely over-fitted beyond epoch 600. To determine the real performance of the LSTM-FC-GP and LSTM-PCA-GP models, the models were re-trained up to epoch 600 for every fold. The detailed results of this can be found in Section 5.5.

6.5.1. Model Accuracy

Comparing the results of the NRMSE between the three models shows that the LSTM-PCA-GP model has the lowest NRMSE of the three models. The LSTM-FC-GP model has a NRMSE that is less than $1\,\%$ higher than the LSTM-PCA-GP model. Meanwhile the NRMSE of the LSTM model is over $22\,\%$ higher than that of the LSTM-PCA-GP model. Moreover the LSTM-PCA-GP model has a standard deviation that is lower than the standard deviation of the LSTM-FC-GP model.

Comparing the results from the test cases with a single sheet of foam to the results from the test cases with more foam, the trend that was observed when over-fitting the model where the NRMSE is higher for the models using a GP layer compared to the LSTM model is no longer clearly present. However, it is still the case that the NRMSE of validation results 1 and 2 is higher than the NRMSE of validation results 3 to 6 for every model.

The improvement in the NRMSE of the models with a GP layer is accompanied by a corresponding in the improvement of the accuracy of the $95\,\%$ confidence interval. While there are still more than $5\,\%$ of real data points outside of the confidence interval, the amount of data points outside the confidence interval has dropped by $38.40\,\%$ for the LSTM-FC-GP model and by $15.94\,\%$ for the LSTM-PCA-GP model to $11.44\,\%$ for the LSTM-FC-GP model, and $9.62\,\%$ for the LSTM-PCA-GP model. Furthermore, the standard deviation of these percentages has significantly decreased to $15.07\,\%$ for the LSTM-FC-GP model and $12.31\,\%$.

The percentages of the individual validation results are unevenly distributed among the validation results, being much higher for validation results 1 and 2. Factoring these percentages out of the total average results in an average of $6.59\,\%$ for the LSTM-FC-GP model and $5.8\,\%$ for the LSTM-PCA-GP model. These percentages are both very close to the target value of $5\,\%$. However good these percentages are, it should be noted that the model was trained including the data from the test case with a single sheet of foam. As a consequence, leaving out validation results 1 and 2 is not entirely

representative in this case as the confidence interval was intended to match the total percentage of all data.

While the values of $11.44\,\%$ and $9.62\,\%$ are both higher than the $5\,\%$ target, the model is expected to perform worse on the validation data than on the training data. Considering this fact these results are quite good and the confidence interval serves as a good indicator for in what range the real results can be expected to be which is entirely absent for the LSTM model.

A consequence of not over-fitting the models utilising a GP layer is that the confidence interval has a much larger size. The size of this confidence interval is shown in table 5.16. While the confidence interval of the LSTM-PCA-GP model is only $45\,\%$ larger than the confidence interval of the over-fitted model, this increase is much larger for the LSTM-FC-GP model for which the size of the confidence interval has increased with nearly $200\,\%$. This serves as a very good indicator to how much more the LSTM-FC-GP model was over-fitting compared to the LSTM-PCA-GP model. For the models that were not over-fitted, the width of the confidence interval is within $6\,\%$ of each other, leading to similar results for the two models.

6.5.2. Time-Domain Results

The time-domain results for fold 1 of the models that were trained not to over-fit the data are shown in section 5.5.2. To make the figures more readable, only the first half of the validation results is shown. In these figures, the LSTM model is shown in figure a, the LSTM-FC-GP model is shown in figure b, and the LSTM-PCA-GP model is shown in figure c. Looking at the qualitative aspects of the figures, several observations are made.

The first observation is that the models tend to under-estimate the maximum amplitude of the oscillation shown in figures 5.8 and 5.9 which correspond to validation results 1 and 2 respectively. For many applications this has implications for the validity of the results where higher amplitudes tend to be conservative. This same issue is not visible in figures 5.10 to 5.13 which correspond to validation results 3 to 6.

Another observation is that the models tend to be much less accurate in the first few data points. This can be attributed to the LSTM layer of the model still needs to initialise its memory. To address this is possible to discard the first few data points. This would allow the LSTM layer to provide better outputs but it would also prevent the model from trying to over-fit the start of the training data accelerations.

Next, as was shown before in the tables showing the confidence intervals of the models, there is very little variation in the variance of the output of the GP layer across the time domain and across samples. This means that irrespective of the amplitude of the oscillation the GP model will provide a similar result. Looking at Figure 5.10, it can be seen that the oscillation has a very low amplitude. However, the $95\,\%$ confidence interval is still the same size resulting in a model that despite very accurately modelling the time domain response, still has a confidence interval that is approximately as wide as the maximum amplitude of the oscillation itself.

This implies that the models lack the context to determine across the time domain how much individual data points should vary. If this context was provided to the model it is possible that the accuracy of the models could be further improved.

6.5.3. The Effect of Noisy Data

To determine the effect of the noisy data on the optimal model results, the models were re-trained with early stopping on the dataset excluding the noisy input data associated with the test case with a single sheet of foam.

The results show that the average NRMSE is only lower for the LSTM model. While the NRMSE of the LSTM-FC-GP model is only slightly higher, the NRMSE of the LSTM-PCA-GP model is more than $21\,\%$ higher than the average NRMSE of the models that were trained without over-fitting, including the data from the test case with a single sheet of foam. Comparing the models among each other, the LSTM model now performs better than the LSTM-PCA-GP model.

As the LSTM-PCA-GP model suffers the least from the noisy test data, it is expected that the LSTM model and LSTM-FC-GP models benefit more from removing the data from the test case with a single

sheet of foam. However, the expectation is that the average NRMSE improves for all models which in reality is not the case.

The cause of this phenomenon is not clear. However, there are multiple possibilities. First is the fact that there is less training data available. This makes it easier for complex models to over-fit the training data. However, given the use of early stopping this is unlikely the root cause. Another possibility is that the early stopping erroneously stopped training the models prematurely resulting in a higher NRMSE. While this is possible, the percentage of the points that are outside of the $95\,\%$ confidence interval suggests that the models were correctly fitted as both confidence intervals have an effective average of very close to $5\,\%$. Another possibility is that the learning rate or other hyper-parameters are not correctly tuned for the model to converge on this different data set. To determine what the cause of the reduced accuracy is, these possibilities must be tested to determine the cause of the reduced accuracy.

6.5.4. PSD Accuracy

At Redwire Space NV. vibration tests are evaluated based on their PSD. Therefore the models must have a high accuracy when comparing the PSD of the real measurement with the PSD of the model results. This comparison can be seen in the plots shown in Appendix C. Across all models the PSD of the results is very accurate up to a frequency of $500\,\mathrm{Hz}$. Above this frequency in general the LSTM model performs the worst of all models.

The PSD of most validation results has a peak between $600\,\mathrm{Hz}$ and $900\,\mathrm{Hz}$. For all folds except for fold 1, the LSTM model does not model this frequency range accurately. However, the LSTM-FC-GP and LSTM-PCA-GP models do model this frequency range accurately. At a frequency range between $800\,\mathrm{Hz}$ and $1000\,\mathrm{Hz}$ the LSTM-FC-GP and LSTM-PCA-GP models lose their accuracy and quickly drop to the same level as the LSTM model.

These findings imply that the higher accuracy of the models with a GP layer can at least in part be found in the model's ability to model oscillations in the frequency range of $600\,\mathrm{Hz}$ to $900\,\mathrm{Hz}$. The frequency response of the models can potentially be used to improve the accuracy of the models. Using a metric like the PSD as a loss function combined with the error in the time domain can be a possible way to improve the model's performance.

6.6. Model Extrapolation

To determine how well every model performed when extrapolating beyond the amount of time steps used in the training data, the models were re-trained up to time-step 250. These models were then used to simulate the validation data up to time-step 500. To determine the accuracy of the models when extrapolating beyond the amount of time-steps used in the training data, the percent change in the last 250 time-steps when compared to the first 250 time-steps is shown in table 5.20. In addition to this, the time domain results of fold 1 are shown in Figures 5.20 to 5.25 to determine any qualitative changes between the first half and the second half of the validation results.

6.6.1. Percent Change in NRMSE

The results show that none of the models show a significant overall reduction in the accuracy. Furthermore, an improvement in the average accuracy of the LSTM-FC-GP and LSTM-PCA-GP models was observed with the LSTM-PCA-GP model showing the highest improvement. A possible reason for this improvement is the observation noted in Section 6.5.2 where the first few time-steps show a high inaccuracy while the memory terms of the LSTM layer do not yet have a large amount of information.

While the models with a GP layer have both a higher overall accuracy and a lower standard deviation, The difference in relative accuracy is not very large and all models can be said to perform well at simulating the data beyond the duration of the training data. Based on this finding, extrapolation performance is not a major consideration for the model trade-off.

6.6.2. Time Domain Results

Examining the time-domain results shows that there is no obvious visual difference between the first half of the validation results and the second half of the validation results. Overall, there is no obvious change in the accuracy which is confirmed by the percentual change in NRMSE.

Moreover, examining the results of the models with a GP layer shows that the confidence interval does not get wider when the model is simulating beyond the duration of the training data. While this sounds unintuitive, it should be noted that according to the NRMSE the accuracy of the model is not reduced beyond the duration of the training data. Therefore the confidence interval should not be wider as the accuracy is not lower.

6.7. Model Performance

In order to make a fair comparison between the models, the performance of the default models was benchmarked. The results of this benchmark can be seen in Section 5.7. The performance of the models with a GP layer are tested in two ways. First, they were tested using a GPU that utilises CUDA to run the model. Next, these models were run on the system's CPU.

Comparing the results, it can be seen that the LSTM model performs much better than the LSTM-FC-GP and LSTM-PCA-GP models in all metrics. This is to be expected as the LSTM model is a sub-component of the LSTM-FC-GP and LSTM-PCA-GP models. Meanwhile, the performance of the LSTM-FC-GP and LSTM-PCA-GP models is nearly identical in every test.

Comparing the training time it can be seen that the time per epoch of the LSTM model is approximately 3 times lower than the time per epoch of the LSTM-FC-GP and LSTM-PCA-GP models. This lower time per epoch does however not translate directly into faster training times. When using early stopping on the models using a GP layer it was found that the longest training fold lasted for 900 epochs. Meanwhile, the LSTM model was trained over 4000 epochs. This results in a total training time of $2\,\mathrm{h}37\,\mathrm{min}$ for the LSTM model, and $1\,\mathrm{h}48\,\mathrm{min}$ for the models with a GP layer. However, most folds for the LSTM-FC-GP and LSTM-PCA-GP models trained for only 500 epochs which corresponds to a training time of just over $1\,\mathrm{h}$. While this training time is much lower than the training time for the LSTM model, it should be noted that these results can only be obtained when utilising a GPU for the GP layer. Training the models on a CPU results in significantly longer training times.

The difference between the runtime of the models is smaller than the relative difference between the time per epoch. The models with a GP layer take less than 2.5 times longer to run compared to the LSTM model. While this increase is still significant the total time to run a time domain simulation with 500 time-steps is under $2.5\,\mathrm{s}$. This time is short enough for the time to run to not be a large obstacle during use.

The difference between the total memory usage is much larger. The total memory usage of the LSTM model when running the model is only $366\,\mathrm{MiB}$. This is very low and does not pose an issue for any modern computer. Meanwhile the LSTM-FC-GP and LSTM-PCA-GP models use $668\,\mathrm{MiB}$ of memory and more than $1500\,\mathrm{MiB}$ of system memory. While this is substantially more than the LSTM model, this could still be run on any modern system with a modern discrete GPU that supports CUDA and thus shows that memory usage is not a large obstacle to running the models on a modern system.

Because not every computer has a discrete graphics card that supports CUDA the models using a GP layer were re-run on CPU only. While this is impractical during training, running the model is far less computationally intensive and possible on a CPU. The results of this show that running these models on a CPU only increases the runtime to an average of $3\,\mathrm{s}$. For the intended application this is not a restrictive amount of time. Meanwhile the memory that was previously stored in the system's video memory shifts to the system memory increasing the total system memory usage to close to $1900\,\mathrm{MiB}$. This is more than five times higher than the LSTM model. While by itself this is not a restrictive amount this can be a substantial restriction on a system that is running other software alongside the model or when the model is used as part of a larger program.

6.8. Trade-Off Between Models

To determine what model should be used a final comparison is made that considers all aspects discussed in this chapter. When comparing the LSTM model to the models using a GP layer, the behaviour of the two models using a GP layer is often very similar when contrasted to the LSTM model. Therefore they are compared together unless they are individually specified.

When compared to the original models used in the internship leading up to the thesis, all models achieve extremely accurate results. All models are able to follow the main oscillation of the system they are modelling with a very high accuracy showing little to no phase shift across every foam thickness. However when the models are compared among each other, the models that utilise a GP layer provide a noticeably higher accuracy when early stopping is used. Furthermore, the accuracy of the models with a GP layer are influenced to a much lower degree by noisy input data. This behaviour is present more strongly in the LSTM-PCA-GP model than in the LSTM-FC-GP model.

Moreover, the models with a GP layer provide a confidence interval that, when the models are trained correctly on representative data, is accurately able to predict in what range the simulated data will be. Knowing this confidence interval is an enormous advantage considering the goal is to simulate a physical system. In order to make the simulation conservative the models must either over-estimate the real oscillation or provide another way to determine what the maximum expected amplitude of the oscillation can be. The confidence interval provides the models using a GP layer with a way to determine this maximum expected amplitude. Meanwhile, the LSTM model has been shown to systematically under-estimate the maximum amplitude. Furthermore, the frequency response of the LSTM model shows that it is less accurate at modelling the response of higher frequency oscillations. This means that in its current form, the LSTM model could not be used and a way to determine its accuracy on real tests must be found.

This, combined with the shorter training times of the models with a GP layer makes the models with a GP layer a very good choice for the model. However, there are exceptions to this. The computational resources required to train and run the models with a GP layer can be a very limiting factor and increase with an increasing number of data points to train the model. As the dataset gets larger a model with a GP layer may no longer be viable and an LSTM model must be used. For the simplicity of this thesis, only a single variable was used as an input for the LSTM layer. when more variables are added the amount of data required to train the model increases exponentially. This could very quickly increase the resource requirements beyond anything that is practical. However, when the available hardware is able to train a model with a GP layer on the training dataset, a model with a GP layer is very strongly recommended.

Another aspect that needs to be considered is that the LSTM model is much easier to train. Something that is very clear from training the models is that the models with a GP layer have room for improvement. In order to train the models with a GP layer to a desirable level, many hyper-parameters must be evaluated and tuned. This might require a considerable amount of time and effort which in a commercial setting might not make sense.

Comparing the LSTM-PCA-GP model and the LSTM-FC-GP model, the differences become very small for both the model performance and accuracy. Both models react in a similar way to the effect of input dimension, leaving out the noisy test data, and varying the amount of steps skipped when training the GP layer. Furthermore, both models have a frequency response that is very similar. However, the LSTM-PCA-GP model suffers less from noise in the test data. Therefore, the LSTM-PCA-GP model is recommended based on its better performance with noisy test data. However, the LSTM-PCA-GP model was only made possible by the torch-pca python package which allows gradients to be stored through the PCA layer. [37] As this package is maintained by a single person it is possible that it will loose support in the future. While the package is very simple and can easily be modified, as was done for this thesis, this may not be worth the risk and effort for a production environment and the LSTM-FC-GP model which relies only on PyTorch could be a better alternative based on this.

The conclusions of this section are summarised in Table 6.2.

Table 6.2: Summary of Best Use Cases for Models

Model	Use Case
LSTM	Best for use with large datasets
LSTM-FC-GP	Best for long term support
LSTM-PCA-GP	Best performance for noisy data

7

Conclusion

The aim of this thesis was to find a surrogate model that can accurately model the behaviour of an object packed in foam packaging of varying thickness when it is subjected to shocks and vibrations of both high and low amplitudes. To achieve this, a dataset was made by performing tests on a mass that is clamped between a varying number of foam plates which were used to achieve varying foam thicknesses. This test setup is shown in figure 7.1. The test setup was then subjected to random vibration bursts of varying amplitudes. Both the applied acceleration as well as the response of the mass packed between the foam plates was measured to create the dataset. This dataset was then used to train AI models in a data-driven way to act as a surrogate model to simulate the response of this foam-damped mass.

The proposed LSTM, LSTM-FC-GP, and LSTM-PCA-GP AI-models were then trained on this dataset using a k-fold cross-validation with 5 folds. The models were compared based on their overall accuracy, their sensitivity to input noise, their accuracy in the frequency domain, their ability to run simulations that exceed the duration of the training data, and the computational resources required to run the models. Additionally, the effect of several parameters influencing the GP layer was evaluated.

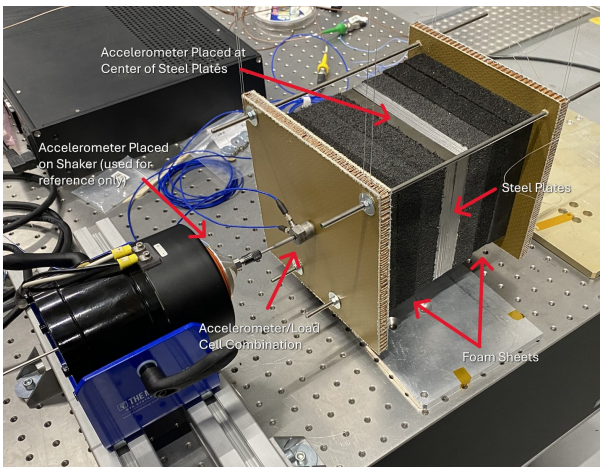


Figure 7.1: The test-setup used to create the dataset in a configuration with three foam sheets on either side of the mass.

The comparison between these models shows that the two models with a GP layer performed nearly identically to each other and significantly out-performed the LSTM model at the cost of more computational resources. Furthermore, the models using a GP layer provide a confidence interval which gives an indication of how much the results are expected to vary. Despite the larger amount of computational resources required, the models with a GP layer trained in both fewer epochs, and less time than the LSTM model. However, the models with a GP layer were more prone to over-fitting necessitating the use of early stopping during training.

Based on the comparison made between the AI-models the LSTM-PCA-GP model is recommended to be used based on its superior accuracy when using noisy test data. However, as this model depends on a package maintained by a single person, the LSTM-FC-GP model is a very competitive alternative when long term support is preferred. Finally, as the required computational resources for the GP layer increase with an increasing dataset size, the use of a model using this GP layer can become impossible. In that case an LSTM model should be used instead. The biggest downside to this is model is the absence of a confidence interval. As a result, there is no indication how much the output of the model can vary, and the accuracy of the result becomes difficult to determine.

8

Future Works

In this thesis, a set of models that can be used to model the behaviour of a foam-packed item to a shock or vibration is found. These models were trained on a dataset consisting of vibration data obtained in the TU Delft DASML Dynamics Lab. While the models trained on these datasets show promising results, the test setup, and the reduced number of variables makes this data, and as a consequence the models, not representative of the real response of a foam-packed item to a shock. This chapter describes the steps that would need to be taken to develop these models to make them able to accurately model a shock test on a foam-packed item.

8.1. Test Articles

In order to make a model representative, a representative dataset must be made. Making this representative dataset starts with a representative test article. The mass, foam thickness, and variation of this test article are described in Sections 8.1.1, 8.1.2, and 8.1.3 respectively.

8.1.1. Test Article Mass

To meet the requirements set by NASA, the surface pressure the test object exerts on the foam at a loading of $1\,\mathrm{g}$ shall be between $0.2\,\mathrm{psi}$ and $0.7\,\mathrm{psi}$. [35]

Using a density of $7850\,\mathrm{kg\,m^{-3}}$ and $2700\,\mathrm{kg\,m^{-3}}$ for steel and aluminium [38, 39] respectively, this minimum and maximum thickness can be converted to the thickness of a block. For the steel block this thickness would range between approximately $17.9\,\mathrm{mm}$ and $62.7\,\mathrm{mm}$ while for the aluminium block this thickness should range between approximately $52.1\,\mathrm{mm}$ and $182\,\mathrm{mm}$.

In this thesis a contact area of $15\,\mathrm{cm}$ by $15\,\mathrm{cm}$ was used to perform the tests. To avoid possible unexpected effects from having a small foam area, it is recommended to not go below this surface area.

8.1.2. Test Article Foam

In order to make a representative dataset, the foam packaging of the test article must be representative of the real scenario. To achieve this, foam bags similar to the bags used during a real flight must be made. These bags must have a varying thickness. Allowing for sufficient tests to be performed with foam of increasing thickness.

The exact thickness range of the foam in the bags must be determined by an expert and is dependent on what the thickness range of foam in real foam bags is. This thickness range determines in what range of thicknesses the model will be valid.

8.1.3. Varying the Foam and Block Thickness

To obtain a reliable model, the thickness of the aluminium or steel block, and the thickness of the foam in the foam bag must be varied with a sufficiently small increment. If the increment is too large, there is a risk that the model will not be able to model the effect of a varying thickness accurately.

The exact maximum increment size is difficult to determine in advance. However, more increments will result in a more accurate model. The downside of increasing the number of increments between the minimum and maximum thickness is that the training dataset will become much larger. The larger this dataset is, the more resources the model will need during the training phase, and possibly during the evaluation phase if a model with a GP layer is used.

To combat this issue, the exact increments can be varied within the dataset. For example, if a thickness between $5\,\mathrm{cm}$ and $20\,\mathrm{cm}$ is required, one block thickness could have foam thicknesses of $5\,\mathrm{cm}$ and $15\,\mathrm{cm}$ while another block thickness could have a foam thickness of $10\,\mathrm{cm}$ and $20\,\mathrm{cm}$. In this way, a larger number of increments can be achieved without increasing the amount of collected data.

8.2. Test Setup

The test setup to acquire the dataset is as follows. An accelerometer is placed on the mass that is described in section 8.1.1. The test mass is placed in the foam bag that is described in Section 8.1.2. The foam bag is closed, and placed onto the shaker. The foam bag must be secured to the shaker surface without pre-compressing the foam. The acceleration of the shaker surface must be measured.

When the foam bag is secured on the shaker, the desired tests shall be performed. These tests depend on the desired type of model and are described in Section 8.3. After the tests have been performed, the mass shall be placed in a bag with a different thickness and the tests shall be repeated until the tests have been performed with each foam thickness. This shall be done with every mass until all combinations have been tested.

8.3. Test Sequence

The tests that shall be performed on the test articles depend on the desired results of the model. For a model that is meant to model a shock, a shock test must be performed. In order to ensure that the model does not learn to anticipate the input shock, a large variation of amplitudes must be used. Furthermore, the shock duration, the type of shock, and the amount of time before the shock can all be varied to avoid the model over fitting.

If the model is intended to represent random vibration behaviour, the test should be a vibration test. As vibration tests can be very long it is preferable to do a larger number of short vibration tests instead of a long vibration test. In this thesis a random burst test was used to achieve this goal. Additionally, a large number of amplitudes is preferred to more tests at the same amplitude.

The number of samples taken during the test is directly proportional to the training duration. The more samples taken, the longer the training will last. Therefore the number of time steps used must be minimised. The two factors to consider are the duration of the test, and the sampling frequency. The duration of the test is dependent on the type of model that is being created. For example, if the performed test is a shock test, a shorter duration with a higher sampling frequency can be used. However, if a model is created with a vibration test data, lower frequency longer duration data can be used.

It is important to consider that the step size of the model must be consistent between the training datasets. As a direct solution model was chosen, the LSTM has no way to know the step size of the model. Therefore, a consistent step size must be used and the model cannot be used to do simulations at higher frequencies. It may be possible to use the step size as an input to the model, however this was not tested.

8.4. Training the Model

The model shall be trained on the dataset described in Section 8.3. In order to ensure accuracy, the training should be done over a sufficient number of epochs. Furthermore, special care should be taken

when selecting hyper-parameters for the model. If the learning rate is chosen too low, the model may not converge fast enough. However, if the learning rate is chosen too high, the model may not be able to achieve a high accuracy. Additionally, the dataset must be split into a training and validation dataset. After the model has been trained on the training dataset, the accuracy must be evaluated on the validation dataset to determine the model's accuracy. Special care must be taken when training the models to prevent over-fitting. It is advised to use early stopping to prevent over-fitting the model.

References

[1] *Report/Thesis Template | LaTeX × TU Delft*. URL: https://dzwaneveld.github.io/report/.

[2] *Backpropagation in Neural Network - GeeksforGeeks*. URL: https://www.geeksforgeeks.org/machine-learning/backpropagation-in-neural-network/.

[3] *Epoch in Machine Learning - GeeksforGeeks*. URL: https://www.geeksforgeeks.org/epoch-in-machine-learning/.

[4] *HYSTERESIS Definition & Meaning | Dictionary.com*. URL: https://www.dictionary.com/browse/hysteresis.

[5] Lowie De Malsche and Els Lemmens. *Usefulness of Shock Tests on Foam Packed Items: Standardized Simulation Tool for Verification Approach*. Tech. rep. Kruibeke: Redwire Space NV, Sept. 2024.

[6] K. C. Rusch. "Load–compression behavior of flexible foams". In: *Journal of Applied Polymer Science* 13.11 (Nov. 1969), pp. 2297–2311. ISSN: 1097-4628. DOI: 10.1002/APP.1969.070131106. URL: https://onlinelibrary.wiley.com/doi/10.1002/app.1969.070131106.

[7] E Linul et al. "Study of factors influencing the mechanical properties of polyurethane foams under dynamic compression". In: *Journal of Physics: Conference Series* (2013). DOI: 10.1088/1742-6596/451/1/012002.

[8] Zhiying Zhao et al. "Study on the Mechanical Properties and Energy Absorbing Capability of Polyurethane Microcellular Elastomers under Different Compressive Strain Rates". In: *Polymers 2023, Vol. 15, Page 778* 15.3 (Feb. 2023), p. 778. ISSN: 2073-4360. DOI: 10.3390/POLYM15030778. URL: https://www.mdpi.com/2073-4360/15/3/778/htm%20https://www.mdpi.com/2073-4360/15/3/778.

[9] M. F. Ashby, Hugh Shercliff, and David Cebon. *Materials: engineering, science, processing and design*. Third Edition. Oxford: Butterworth-Heinemann, 2014.

[10] Ying Li et al. "Molecular simulation guided constitutive modeling on finite strain viscoelasticity of elastomers". In: *Journal of the Mechanics and Physics of Solids* 88 (Mar. 2016), pp. 204–226. ISSN: 0022-5096. DOI: 10.1016/J.JMPS.2015.12.007.

[11] Brett Sanborn, Bo Song, and Scott Smith. "Pre-strain Effect on Frequency-Based Impact Energy Dissipation through a Silicone Foam Pad for Shock Mitigation". In: *Journal of Dynamic Behavior of Materials* 2.1 (Mar. 2016), pp. 138–145. ISSN: 21997454. DOI: 10.1007/S40870-015-0043-1/FIGURES/14. URL: https://link.springer.com/article/10.1007/s40870-015-0043-1.

[12] M. H. Ozkul and J. E. Mark. "The effect of preloading on the mechanical properties of polymeric foams". In: *Polymer Engineering & Science* 34.10 (May 1994), pp. 794–798. ISSN: 1548-2634. DOI: 10.1002/PEN.760341003. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/pen.760341003.

[13] Alvaro Sanchez-Gonzalez et al. "Learning to Simulate Complex Physics with Graph Networks". In: *37th International Conference on Machine Learning, ICML 2020* PartF168147-11 (Feb. 2020), pp. 8428–8437. URL: https://arxiv.org/abs/2002.09405v2.

[14] Keith T. Butler et al. "Machine learning for molecular and materials science". In: *Nature 2018 559:7715* 559.7715 (July 2018), pp. 547–555. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0337-2. URL: https://www.nature.com/articles/s41586-018-0337-2.

[15] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52.Volume 52, 2020 (Jan. 2020), pp. 477–508. ISSN: 00664189. DOI: 10.1146/ANNUREV-FLUID-010719-060214/CITE/REFWORKS. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-fluid-010719-060214.

[16] Christian Legaard et al. “Constructing Neural Network Based Models for Simulating Dynamical Systems”. In: *ACM Computing Surveys* 55.11 (Feb. 2023). ISSN: 15577341. DOI: 10.1145/3567591/ASSET/1B6B98D6-9E2E-4FCA-9AF6-37A9AD05E2F8/ASSETS/GRAPHIC/CSUR-2021-0704-F23.JPG. URL: https://dl-acm-org.tudelft.idm.oclc.org/doi/10.1145/3567591.

[17] Mohaiminul Islam, Guorong Chen, and Shangzhu Jin. “An Overview of Neural Network”. In: *American Journal of Neural Networks and Applications* 5.1 (2019), pp. 7–11. ISSN: 2469-7419. DOI: 10.11648/j.ajnna.20190501.12. URL: http://www.sciencepublishinggroup.com/j/ajnna.

[18] Serhat Kılıçarslan and Kemal Adem. “An overview of the activation functions used in deep learning algorithms”. In: *Journal of New Results in Science* 10.3 (2021), pp. 75–88. DOI: 10.54187/jnrs.1011739. URL: https://doi.org/10.54187/jnrs.1011739.

[19] Robin M. Schmidt. “Recurrent Neural Networks (RNNs): A gentle Introduction and Overview”. In: (Nov. 2019). URL: http://arxiv.org/abs/1912.05911.

[20] Yuhuang Hu et al. “Overcoming the vanishing gradient problem in plain recurrent networks”. In: (July 2019). URL: http://arxiv.org/abs/1801.06105.

[21] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/NECO.1997.9.8.1735.

[22] Sepp Hochreiter. “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06.02 (1998), pp. 107–116. ISSN: 0218-4885. DOI: S0218488598000094.

[23] Kalvik Jakkala. “Deep Gaussian Processes: A Survey”. In: ().

[24] Deep Ray, Orazio Pinti, and Assad A. Oberai. “Deep Learning and Computational Physics”. In: *Deep Learning and Computational Physics* (2024). DOI: 10.1007/978-3-031-59345-1.

[25] Norman Richard Draper and Harry Smith. *Applied regression analysis*. New York : Wiley, 1998, xvii, 706 pages : ISBN: 0471170828. URL: http://www.citeulike.org/group/450/article/155094.

[26] *3.4. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.7.0 documentation*. URL: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error.

[27] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. “The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation”. In: *PeerJ Computer Science* 7 (July 2021), pp. 1–24. ISSN: 23765992. DOI: 10.7717/PEERJ-CS.623/SUPP-1. URL: https://peerj.com/articles/cs-623.

[28] Alessandro Di Bucchianico. *Coefficient of Determination (R2) - Knovel*. 2007. DOI: 10.1002/9780470061572.eqr173. URL: https://app.knovel.com/hotlink/pdf/id:kt007NTF9C/encyclopedia-statistics/structure-coherent-system.

[29] Hossein Pishro-Nik. *Introduction to probability, statistics, and random processes*. Kappa Research, LLC, 2014.

[30] Alexei Botchkarev. “Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology”. In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14 (Sept. 2018), pp. 45–76. DOI: 10.28945/4184. URL: http://arxiv.org/abs/1809.03006%20http://dx.doi.org/10.28945/4184.

[31] Georg. Duffing. *Erzwungene schwingungen bei veränderlicher eigenfrequenz und ihre technische bedeutung.* Braunschweig, F. Vieweg & Sohn, 1918, vi, 134 pages.

[32] Alvaro H. Salas. “An Elementary Solution to a Duffing Equation”. In: *Scientific World Journal* 2022 (2022). ISSN: 1537744X. DOI: 10.1155/2022/2357258.

[33] Saeideh Khatiry Goharoodi et al. “Evolutionary-Based Sparse Regression for the Experimental Identification of Duffing Oscillator”. In: *Mathematical Problems in Engineering* 2020.1 (Jan. 2020), p. 7286575. ISSN: 1563-5147. DOI: 10.1155/2020/7286575. URL: /doi/pdf/10.1155/2020/7286575%20https://onlinelibrary.wiley.com/doi/abs/10.1155/2020/7286575%20https://onlinelibrary.wiley.com/doi/10.1155/2020/7286575.

[34] *Modal Shaker 13 lbf*. URL: https://www.modalshop.com/vibration-test/products/modal-shakers/13-lbf-exciter.

[35] NASA. *Pressurized Payloads Interface Requirements Document International Space Station Program Revision R*. English. Tech. rep. Houston,Texas: NASA, Oct. 2015.

[36] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-Validation". In: *Encyclopedia of Database Systems* (2009), pp. 532–538. DOI: 10.1007/978-0-387-39940-9{_}565. URL: https://link.springer.com/rwe/10.1007/978-0-387-39940-9_565.

[37] *valentingol/torch_pca: Principal Component Anlaysis (PCA) in PyTorch.* URL: https://github.com/valentingol/torch_pca.

[38] *Density of Steel - Mild and Carbon Steel Density lb/in3 or kg/m3*. URL: https://www.amardeepsteel.com/blog/density-of-steel.html.

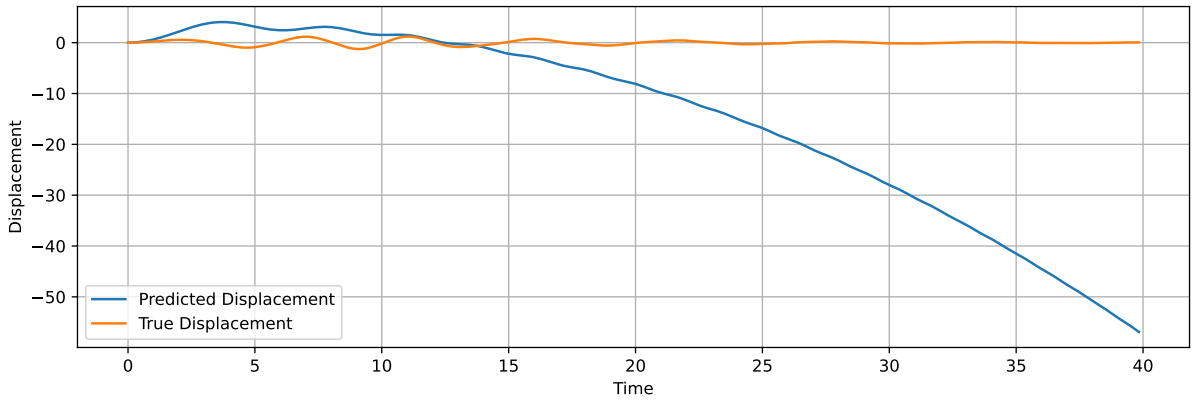
[39] *The Density of Aluminium and its Alloys - thyssenkrupp Materials (UK)*. URL: https://www.thyssenkrupp-materials.co.uk/density-of-aluminium.html.

A

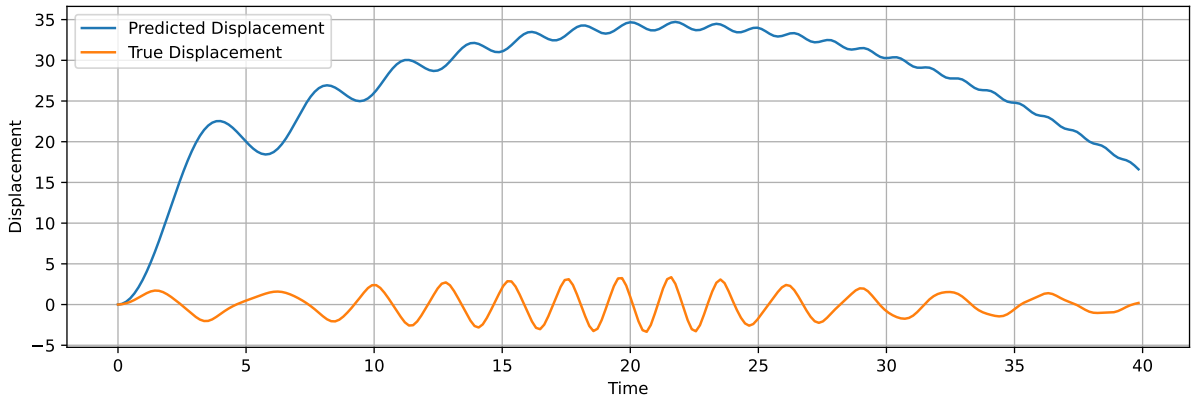
Time Domain Results of Preliminary Tests

A.1. Second Derivative NN

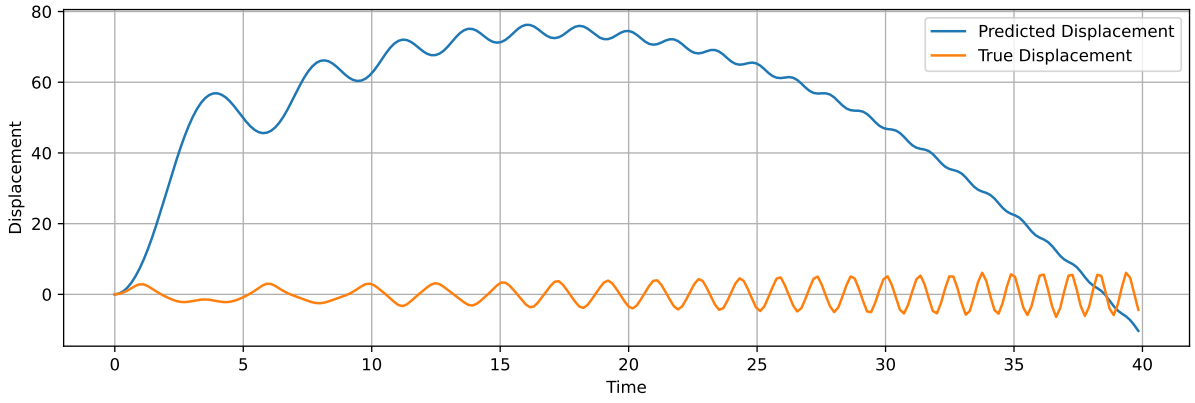
A.1.1. Single Parameter Model



(a) Result for signal with amplitude: 0.6



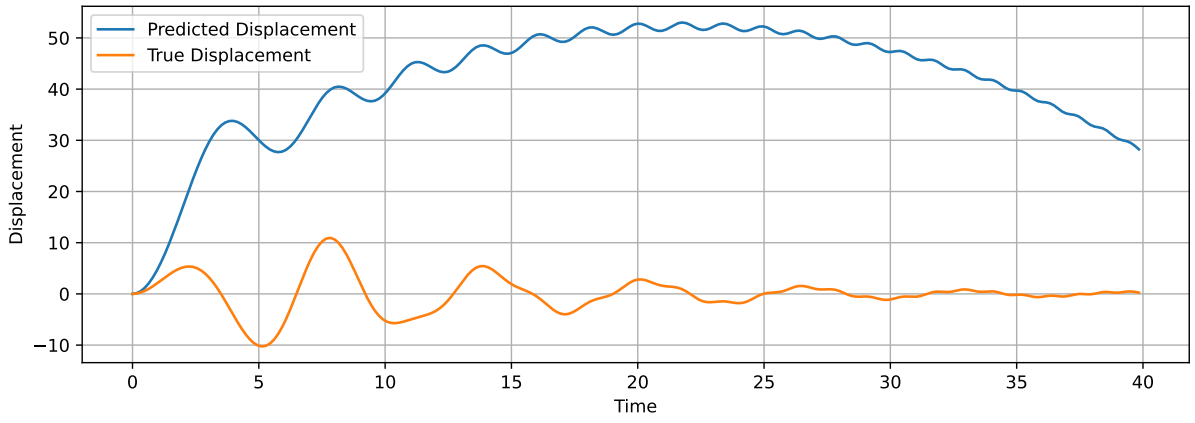
(b) Result for signal with amplitude: 3.0



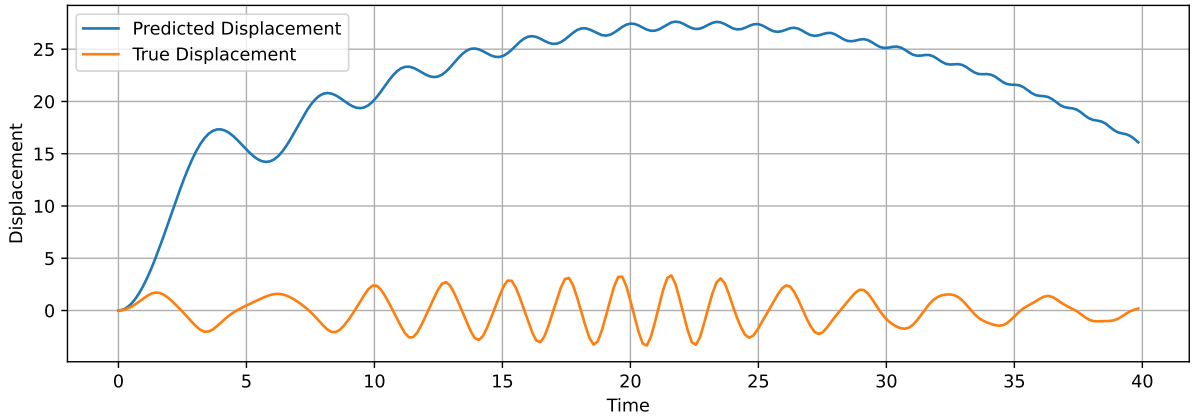
(c) Result for signal with amplitude: 9.4

Figure A.1: Time-domain results for second derivative NN model on single system with k_3: 1.0

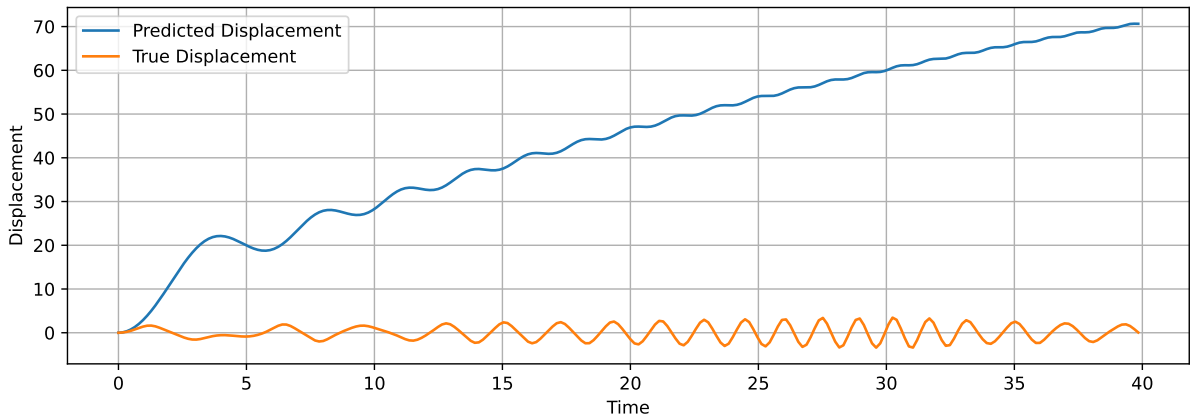
A.1.2. Parameter-Conditioned Model



(a) Result with k_3=0



(b) Result with k_3=1

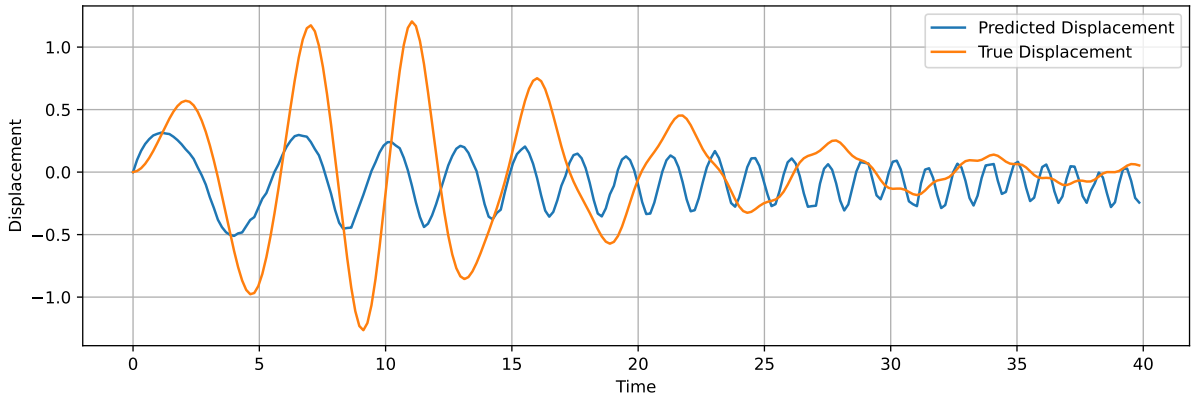


(c) Result with k_3=2

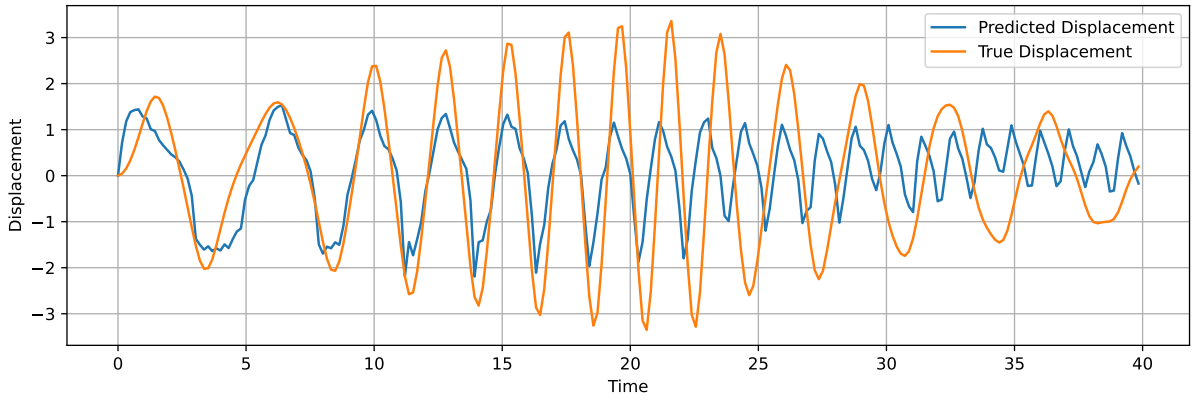
Figure A.2: Time-domain results for second derivative parameter-conditioned NN model

A.2. Direct RNN

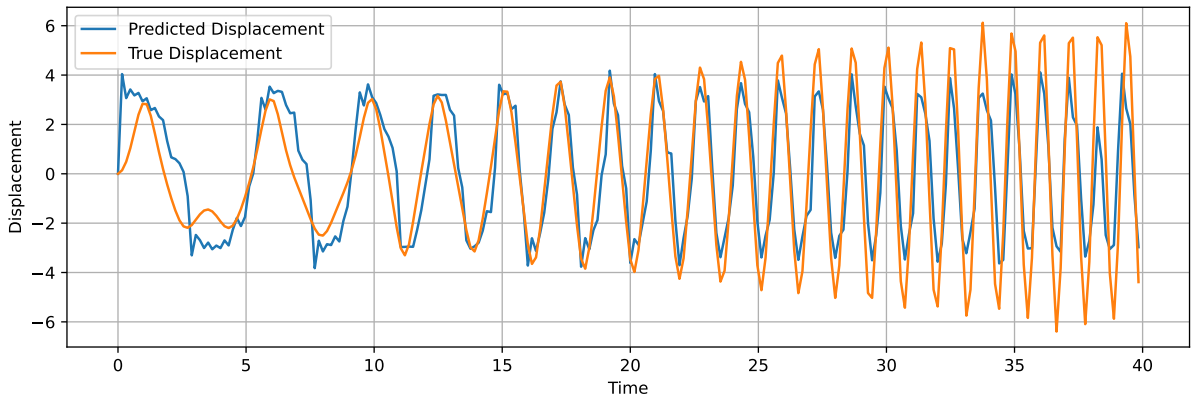
A.2.1. Single Parameter Model



(a) Result for signal with amplitude: 0.6



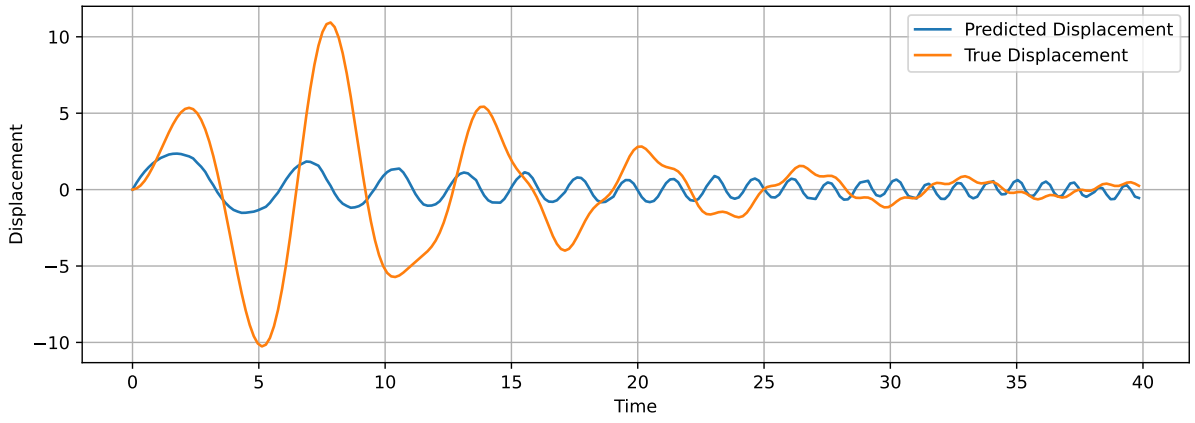
(b) Result for signal with amplitude: 3.0



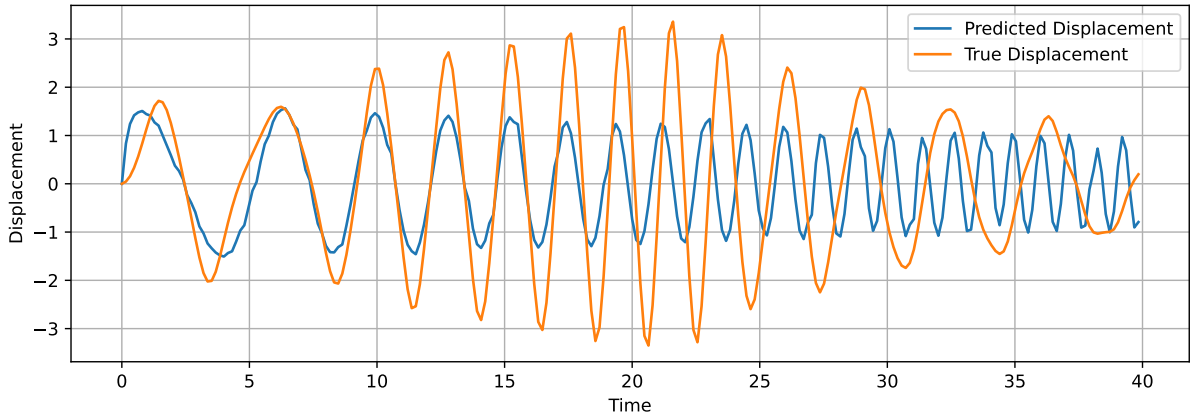
(c) Result for signal with amplitude: 9.4

Figure A.3: Time-domain results for direct RNN model on single system with k_3: 1.0

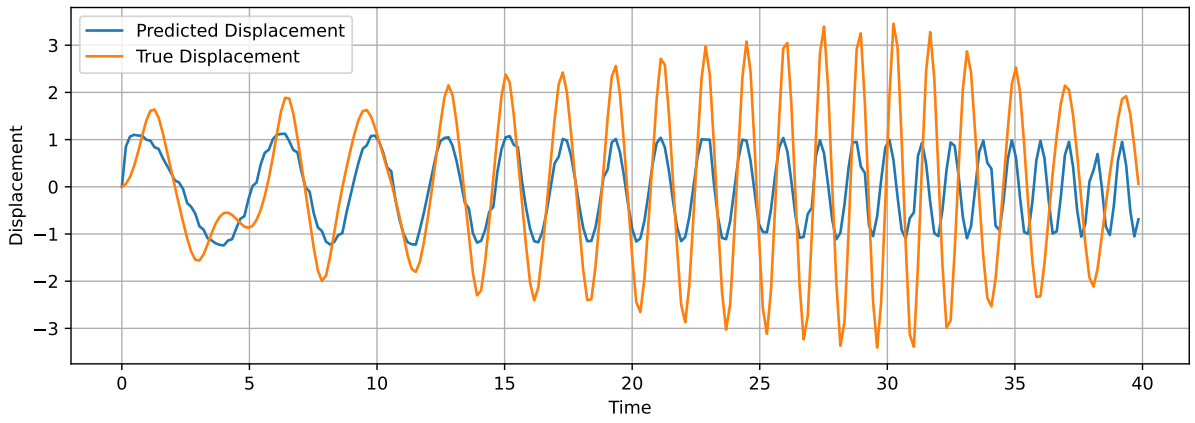
A.2.2. Parameter-Conditioned Model



(a) Result with k_3=0



(b) Result with k_3=1

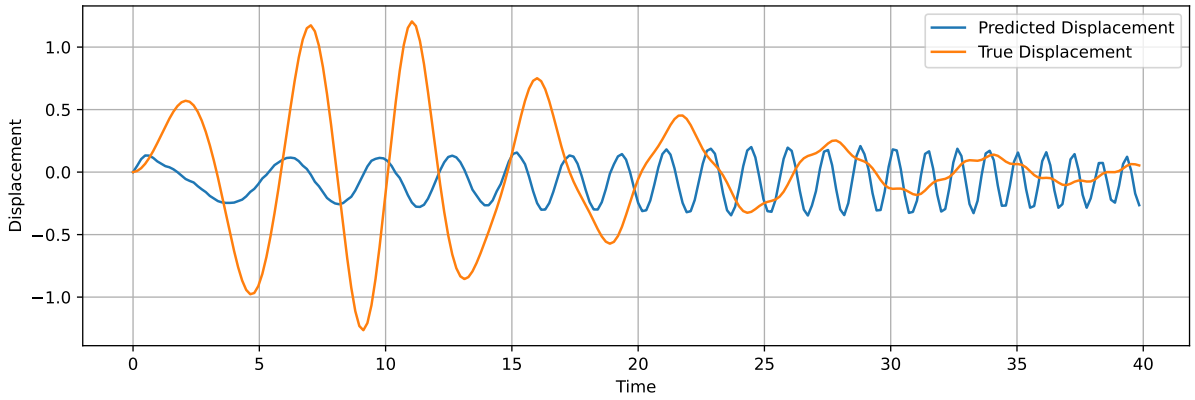


(c) Result with k_3=2

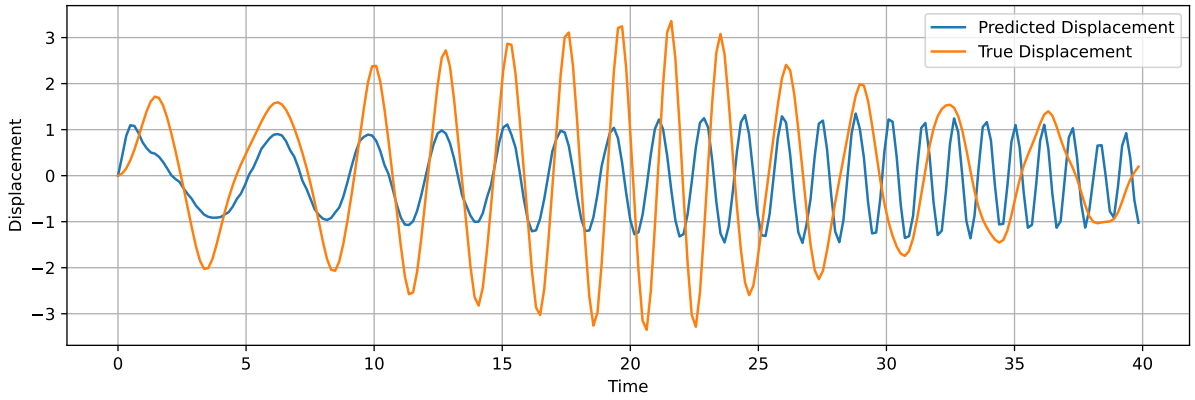
Figure A.4: Time-domain results for direct parameter-conditioned RNN model

A.3. First Derivative RNN

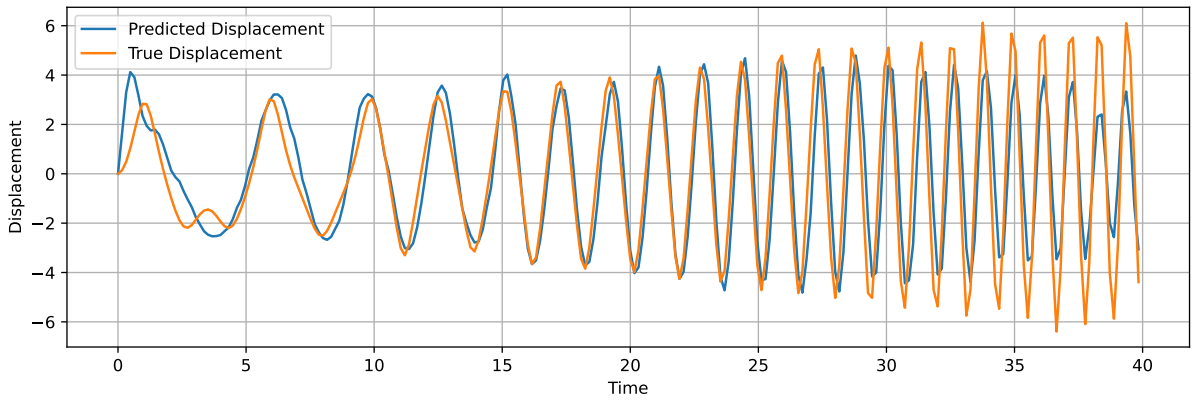
A.3.1. Single Parameter Model



(a) Result for signal with amplitude: 0.6



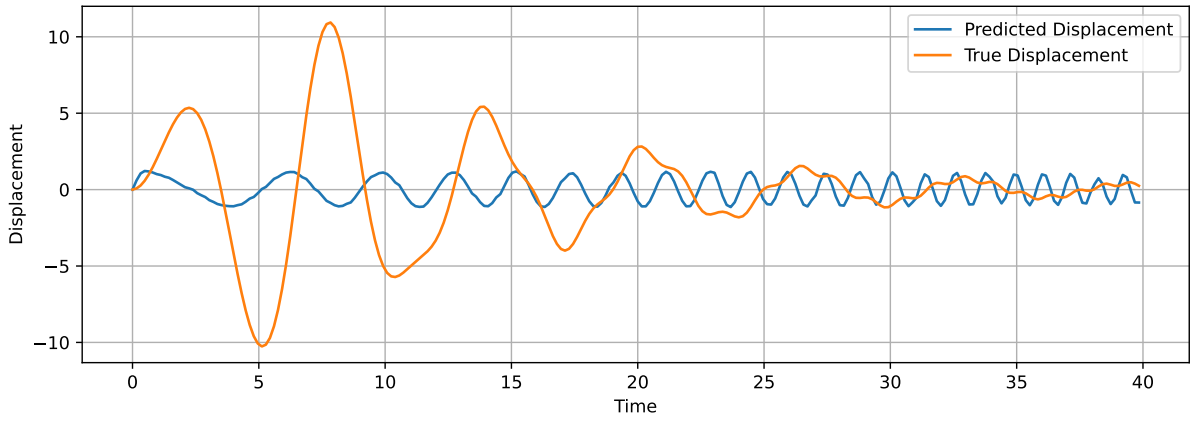
(b) Result for signal with amplitude: 3.0



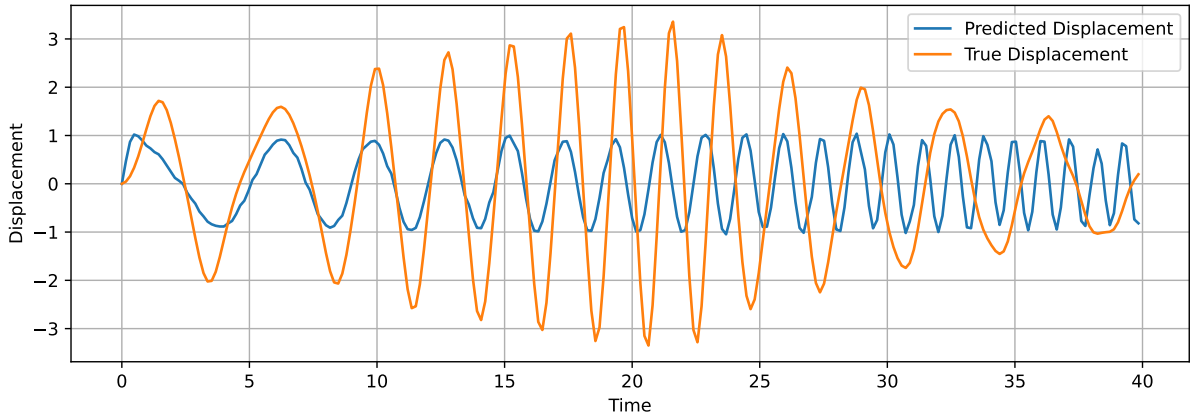
(c) Result for signal with amplitude: 9.4

Figure A.5: Time-domain results for first derivative RNN model on single system with k_3: 1.0

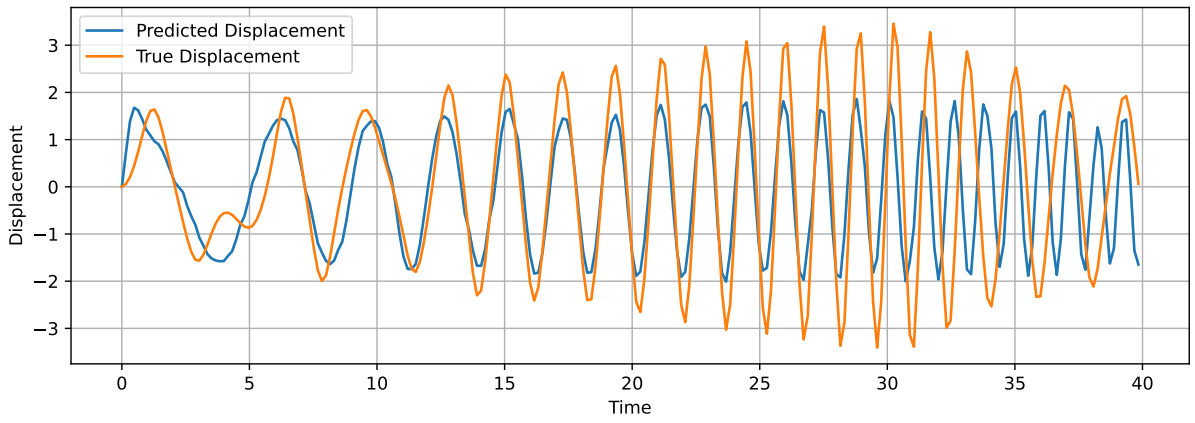
A.3.2. Parameter-Conditioned Model



(a) Result with k_3=0



(b) Result with k_3=1

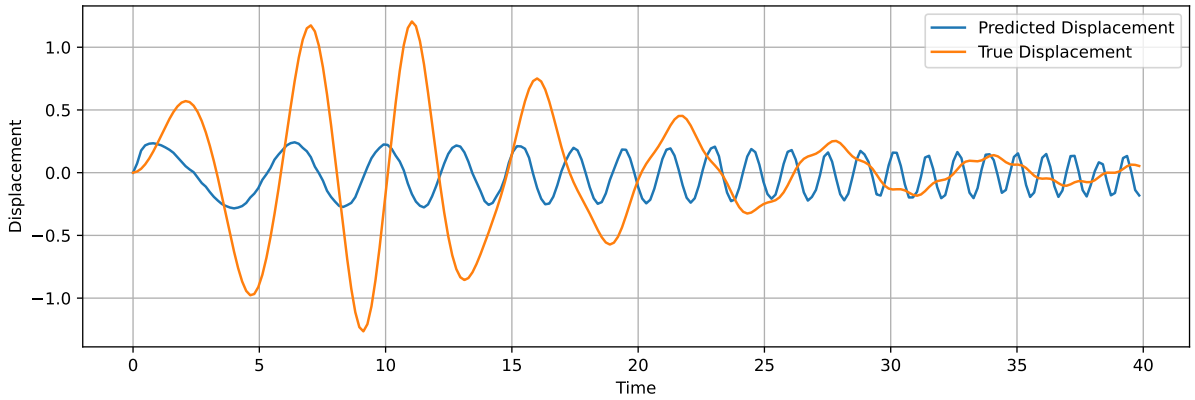


(c) Result with k_3=2

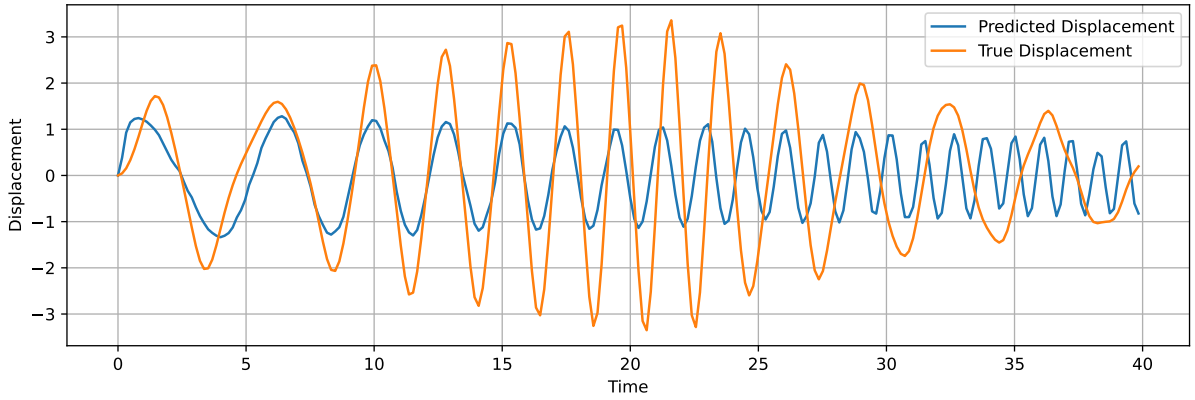
Figure A.6: Time-domain results for first derivative parameter-conditioned RNN model

A.4. Second Derivative RNN

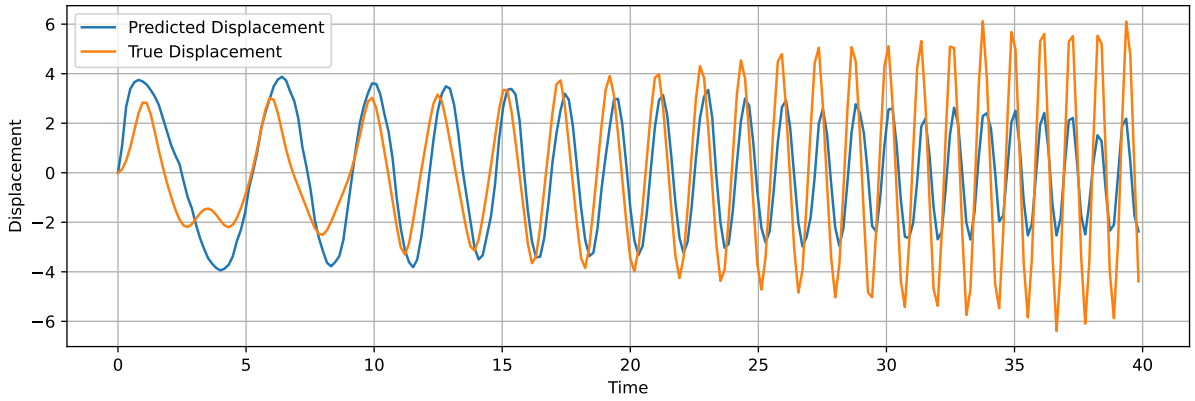
A.4.1. Single Parameter Model



(a) Result for signal with amplitude: 0.6



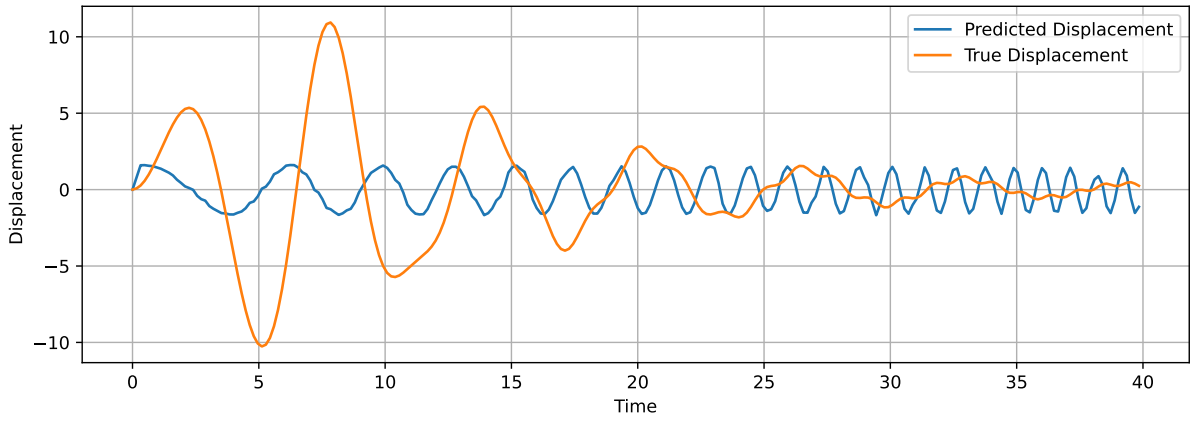
(b) Result for signal with amplitude: 3.0



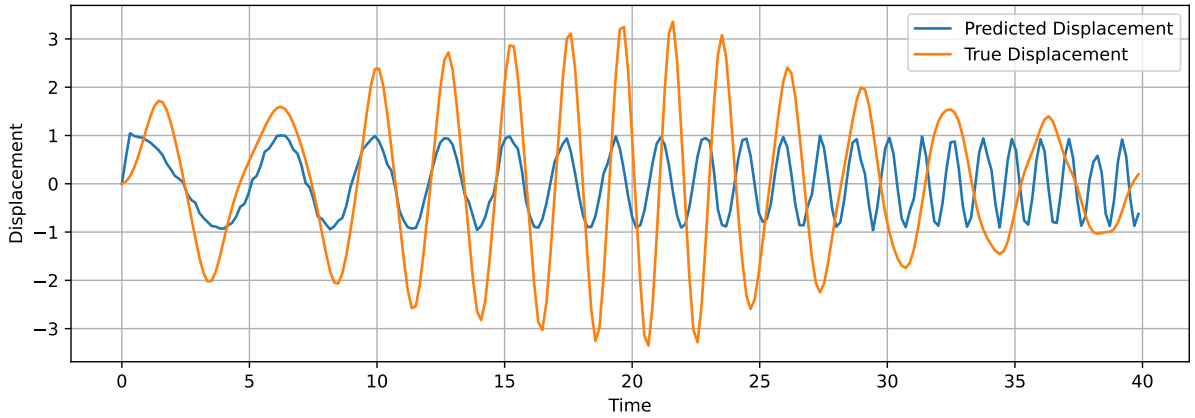
(c) Result for signal with amplitude: 9.4

Figure A.7: Time-domain results for second derivative RNN model on single system with k_3: 1.0

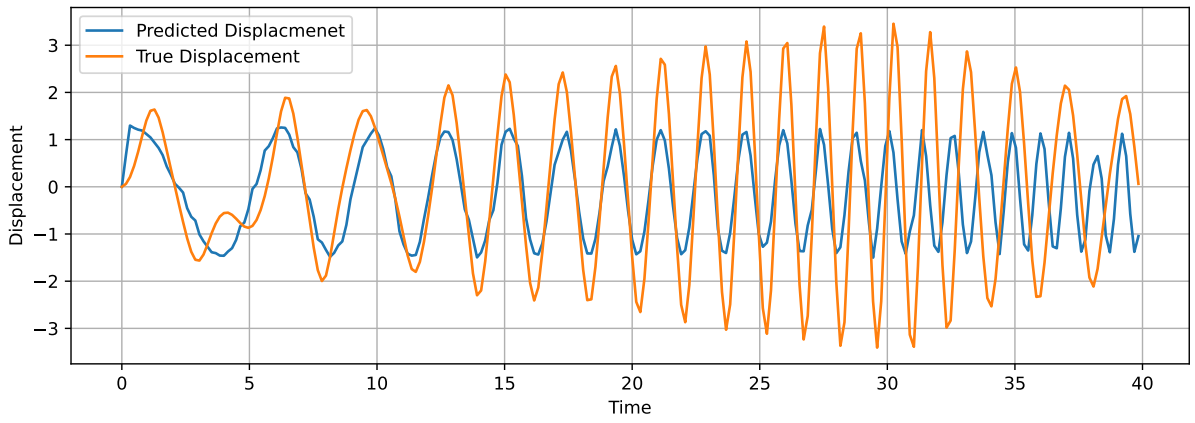
A.4.2. Parameter-Conditioned Model



(a) Result with k_3=0



(b) Result with k_3=1

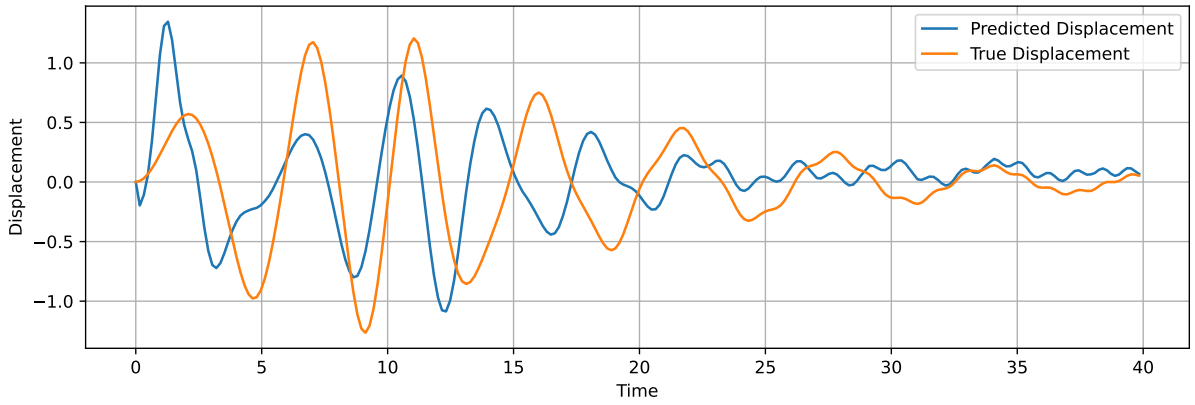


(c) Result with k_3=2

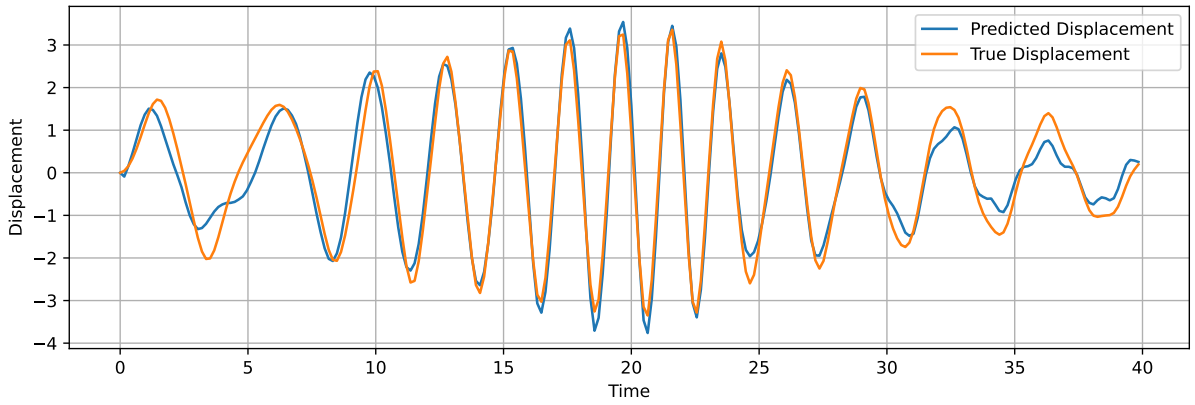
Figure A.8: Time-domain results for second derivative parameter-conditioned RNN model

A.5. Direct LSTM

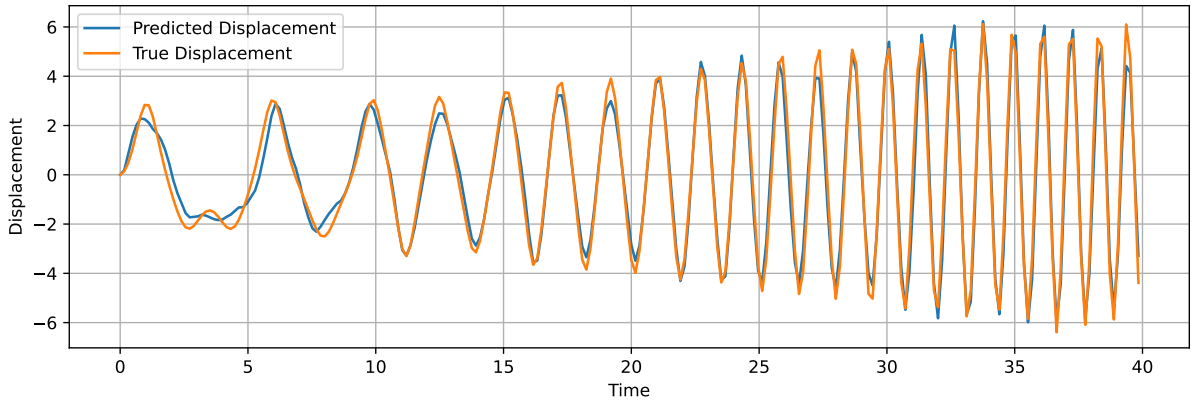
A.5.1. Single Parameter Model



(a) Result for signal with amplitude: 0.6



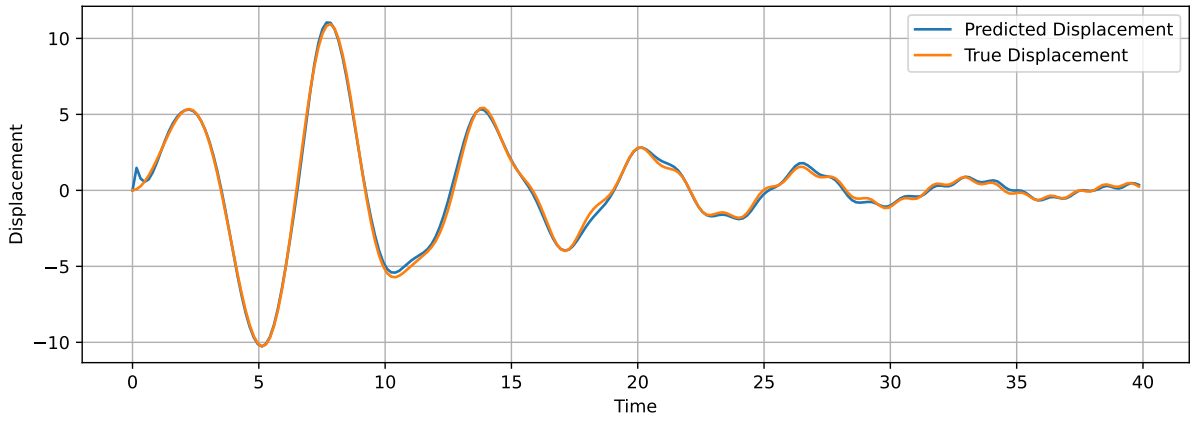
(b) Result for signal with amplitude: 3.0



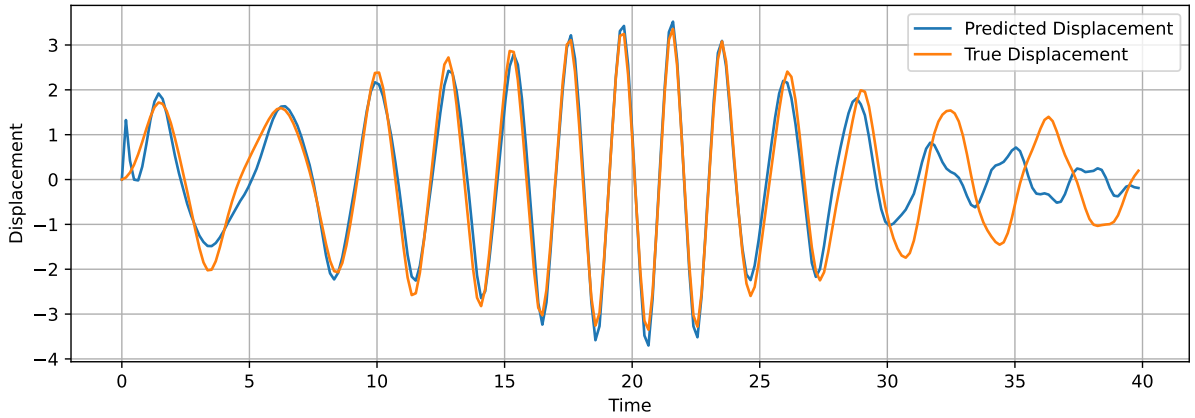
(c) Result for signal with amplitude: 9.4

Figure A.9: Time-domain results for direct LSTM model on single system with k_3: 1.0

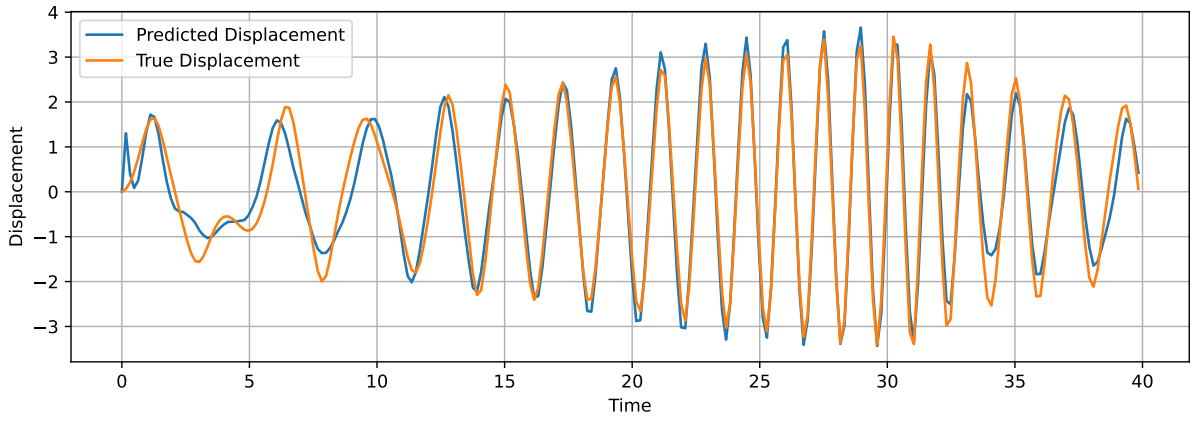
A.5.2. Parameter-Conditioned Model



(a) Result with k_3=0



(b) Result with k_3=1



(c) Result with k_3=2

Figure A.10: Time-domain results for direct parameter-conditioned LSTM model

B

Source Code

B.1. Model Container

```python
import copy
from sklearn.preprocessing import MaxAbsScaler, MinMaxScaler
import torch
from torch import nn


class ModelContainer(nn.Module):
    __input_scaler: MinMaxScaler
    __output_scaler: MinMaxScaler
    __input_dim: int
    __output_dim: int

    def __init__(self, input_dim = None, output_dim = None, device="cpu"):
        super().__init__()
        self.__input_dim = input_dim
        self.__output_dim = output_dim
        scaler_range = (-1,1)
        self.__input_scaler = MaxAbsScaler()
        self.__output_scaler = MaxAbsScaler()
        self.__device = device

    @property
    def input_dim(self):
        return self.__input_dim

    @property
    def output_dim(self):
        return self.__output_dim

    @property
    def scale_input(self):
        return self.__input_scaler.scale_[0]

    @property
    def scale_output(self):
        return self.__output_scaler.scale_[0]

    @property
    def device(self):
        return self.__device

    @device.setter
    def device(self, device):
        self.__device = device
        self.to(self.device)

    def apply_input_scaler(self, x, fit=False, inverse=False):
```

```
48          x = apply_scaler(self.__input_scaler, x, fit=fit, inverse=inverse, dtype=torch.
                float32)
49          return x
50
51      def apply_output_scaler(self, x, fit=False, inverse=False):
52          x = apply_scaler(self.__output_scaler, x, fit=fit, inverse=inverse, dtype=torch.
                float32)
53          return x
54
55
56  def apply_scaler(scaler: MaxAbsScaler, data: torch.Tensor, inverse=False, fit=False,
57                   dtype=None) -> torch.tensor:
58      #data = copy.deepcopy(data)
59      datashape = data.shape
60      if len(datashape) > 1:
61          data = data.reshape((datashape[0] * datashape[1], 1))
62      else:
63          data = data.reshape((datashape[0], 1))
64      if inverse:
65          if type(data) is torch.Tensor:
66              data = data.cpu().detach().numpy()
67          data = scaler.inverse_transform(data)
68      else:
69          if fit:
70              data = scaler.fit_transform(data)
71          else:
72              data = scaler.transform(data)
73
74      data = data.reshape(datashape)
75
76      if dtype is not None:
77          data = torch.tensor(data, dtype=dtype)
78      else:
79          data = torch.tensor(data)
80
81      return data
```

B.2. Models

```
1   import torch
2   from linear_operator import LinearOperator
3   from torch import nn, Tensor
4   import gpytorch
5   from torch_pca import PCA
6   from torch.distributions import Distribution
7
8   from AI_Models import ModelContainer
9
10  class LSTMModel(ModelContainer):
11      def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
12          super(LSTMModel, self).__init__(input_dim, output_dim)
13          self.hidden_dim = hidden_dim
14          self.layer_dim = layer_dim
15          self.lstm = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)
16          self.fc = nn.Linear(hidden_dim, output_dim)
17
18      def forward(self, x, h0=None, c0=None):
19          # If hidden and cell states are not provided, initialize them as zeros
20          if h0 is None or c0 is None:
21              h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(x.device)
22              c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(x.device)
23
24          # Forward pass through LSTM
25          out, (hn, cn) = self.lstm(x, (h0, c0))
26          out = self.fc(out[:, -1, :])  # Selecting the last output
27          return out, hn, cn
28
29  class GPModel(gpytorch.models.ExactGP):
30      def __init__(self, input_size, output_size, likelihood, num_tasks, nu=2.5):
31          super(GPModel, self).__init__(None,None, likelihood)
```

```
32          self.mean_module = gpytorch.means.ZeroMean()
33          self.covar_module = gpytorch.kernels.MultitaskKernel(
34              gpytorch.kernels.SpectralMixtureKernel(num_mixtures=1, ard_num_dims=input_size),
35              num_tasks=num_tasks, rank=1)
36
37          self.input_size = input_size
38          self.output_size = output_size
39
40      def forward(self, x):
41          if x is None or not x.shape[0] > 0:
42              raise ValueError("The input `x` for the GP is invalid or empty.")
43
44          mean_x = self.mean_module(x)
45          if mean_x.dim() == 1:
46              mean_x = mean_x.unsqueeze(-1)
47          covar_x = self.covar_module(x)
48          return gpytorch.distributions.MultitaskMultivariateNormal(mean_x, covar_x)
49
50  class GP_LSTMModel(ModelContainer):
51      def __init__(self, input_dim, hidden_dim, layer_dim, gp_input_size, output_dim, device="
            cpu"):
52          super(GP_LSTMModel, self).__init__(input_dim, output_dim, device=device)
53          self.hidden_dim = hidden_dim
54          self.layer_dim = layer_dim
55          self.lstm = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)
56          self.fc = nn.Linear(hidden_dim, gp_input_size)
57
58          self.__gp_likelihood = gpytorch.likelihoods.MultitaskGaussianLikelihood(num_tasks=
                output_dim)
59          self.__gp_model = GPModel(gp_input_size, output_dim, self.__gp_likelihood, num_tasks=
                output_dim)
60
61      def forward(self, x, h0=None, c0=None):
62          # If hidden and cell states are not provided, initialize them as zeros
63          if h0 is None or c0 is None:
64              h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(x.device)
65              c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(x.device)
66
67          # Forward pass through LSTM
68          out, (hn, cn) = self.lstm(x, (h0, c0))
69          out = self.fc(out[:, -1, :])  # Selecting the last output
70          return out, hn, cn
71
72      def set_train_data(self, **kwargs):
73          self.__gp_model.set_train_data(strict=False, **kwargs)
74
75      @property
76      def gp_model(self):
77          return self.__gp_model
78
79      @property
80      def gp_input_dim(self):
81          return self.gp_model.input_size
82
83      @property
84      def gp_likelihood(self):
85          return self.__gp_likelihood
86
87      def to(self, device):
88          if device == "cpu":
89              self.lstm.to(device)
90              self.fc.to(device)
91              self.gp_model.to(device)
92          if device == "cuda":
93              self.lstm.to("cpu")
94              self.fc.to("cpu")
95              self.gp_model.to(device)
96
97  class GP_PCA_LSTMModel(GP_LSTMModel):
98      def __init__(self, input_dim:int, hidden_dim:int, layer_dim:int, gp_input_size:int,
            output_dim:int, device:str="cpu"):
```

```
        super().__init__(input_dim, hidden_dim, layer_dim, gp_input_size, output_dim, device=
            device)
        delattr(self, "fc")
        self.pca = PCA(n_components=gp_input_size, svd_solver="full")

    def forward(self, x, h0=None, c0=None):
        # If hidden and cell states are not provided, initialize them as zeros
        if h0 is None or c0 is None:
            h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(x.device)
            c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(x.device)

        # Forward pass through LSTM
        out, (hn, cn) = self.lstm(x, (h0, c0))
        out=out[:, -1, :]
        return out, hn, cn

    def to(self, device):
        if device == "cpu":
            self.lstm.to(device)
            try:
                self.pca.to(device)
            except ValueError as error:
                print(f"pca not fitted to device due to error: {error}")
            self.gp_model.to(device)
        if device == "cuda":
            self.lstm.to("cpu")
            try:
                self.pca.to(device)
            except ValueError as error:
                print(f"pca not fitted to device due to error: {error}")
            self.gp_model.to(device)
```

B.3. Model Executor

```
import copy
import warnings
from sklearn.preprocessing import MaxAbsScaler
import matplotlib.pyplot as plt
from tqdm import tqdm
import torch
from torch.optim import Adam
from torchdyn.numerics import odeint
from torch.nn.functional import mse_loss
import gpytorch
from AI_Models import LSTMModel, GP_LSTMModel, GP_PCA_LSTMModel, apply_scaler


class EarlyStopping:
    """Provided By Amalia Macali"""
    def __init__(self, patience=30, min_delta=1e-4):
        self.patience = patience
        self.min_delta = min_delta
        self.best_loss = float('inf')
        self.counter = 0
        self.best_state_dict = None

    def step(self, current_loss:float, model) -> bool:
        if self.best_loss - current_loss > self.min_delta:
            self.best_loss = current_loss
            self.counter = 0
            self.best_state_dict = {k: v.clone() for k, v in model.state_dict().items()}
        else:
            self.counter += 1

        return self.counter >= self.patience

    def restore_best_weights(self, model):
        if self.best_state_dict is not None:
            model.load_state_dict(self.best_state_dict)

```

```
class NNExecutorTemplate():
    def __init__(self, solver, model, device:str="cpu"):
        self.__solver = solver
        self.__model = model
        self.__device = device
        self.model.device = self.device
    @property
    def solver(self):
        return self.__solver

    @property
    def model(self):
        return self.__model

    @property
    def device(self):
        return self.__device

    @device.setter
    def device(self, device):
        assert device in ["cpu", "cuda"]
        self.__device = device
        self.model.device = self.__device

    def train_model(self, *args, **kwargs):
        return NotImplemented

    def run_model(self, *args, **kwargs):
        return NotImplemented

    def solve(self, *args, **kwargs):
        raise NotImplemented

    def save(self, path):
        torch.save(self, path)

class NNExecutorStateVarTemplate(NNExecutorTemplate):
    def __init__(self, solver, model, state_var_names: list[str], device:str="cpu"):
        super().__init__(solver, model, device=device)
        self.__state_variable_names = state_var_names

    @property
    def state_variable_names(self):
        return self.__state_variable_names



    @staticmethod
    def load(path):
        model = torch.load(path, weights_only=False)
        model.eval()
        return model


class LSTM_StateVar(NNExecutorStateVarTemplate):
    __model: LSTMModel
    def __init__(self, solver, model, state_var_names: list[str], device:str="cpu",):
        super().__init__(solver, model, state_var_names, device=device)


    def train_model(self,
                    data_train,
                    input_train,
                    xpoints_train,
                    state_variables_train,
                    epochs,
                    lr,
                    lr_scale,
                    lr_scale_step,
                    data_val=None,
```

```
                    input_val=None,
                    xpoints_val=None,
                    state_variables_val=None,
                    patience=None,
                    min_delta=None):

        if data_val is not None and input_val is not None and xpoints_val is not None and
            state_variables_val is not None:
            val = True
        else:
            val = False

        if patience is None and min_delta is None:
            # neither patience nor min delta defined
            early_stopper = None
        elif patience is None:
            # only min_delta defined
            early_stopper = EarlyStopping(min_delta=min_delta)
        elif min_delta is None:
            # only patience defined
            early_stopper = EarlyStopping(patience=patience)
        else:
            #both min_delta and patience defined
            early_stopper = EarlyStopping(patience=patience, min_delta=min_delta)

        model = self.model
        model.train()

        opt = Adam(model.parameters(), lr=lr)
        scheduler = torch.optim.lr_scheduler.StepLR(opt, step_size=lr_scale_step, gamma=
            lr_scale)

        losses = list()
        losses_val = list()

        input_train = model.apply_input_scaler(input_train, fit=True).to(self.device)
        data_train = model.apply_output_scaler(data_train, fit=True).to(self.device)

        xpoints_train = torch.tensor(xpoints_train, dtype=torch.float32).to(self.device)
        state_variables_train = torch.tensor(state_variables_train, dtype=torch.float32).to(
            self.device)

        if val:
            input_val = model.apply_input_scaler(input_val).to(self.device)
            data_val  =  model.apply_output_scaler(data_val).to(self.device)

            xpoints_val = torch.tensor(xpoints_val, dtype=torch.float32).to(self.device)
            state_variables_val = torch.tensor(state_variables_val, dtype=torch.float32).to(
                self.device)

        if not val and early_stopper is not None:
            warnings.warn("Early␣stopper␣is␣defined␣but␣validation␣data␣not␣provided.␣Early␣
                stopping␣will␣be␣ignored.")

        model.train()
        pbar = tqdm(range(epochs))

        i_epoch = 0
        for _ in pbar:
            opt.zero_grad()
            idx_train = 0

            x_pred_train = self.solve(input_train, xpoints_train, state_variables_train)
            loss = mse_loss(x_pred_train[:,:,0], data_train)
            loss.backward()
            losses.append(loss.item())

            if val:
                model.eval()
                x_pred_val = self.solve(input_val, xpoints_val, state_variables_val)
                loss_val = mse_loss(x_pred_val[:,:,0], data_val)
```

```
174                 losses_val.append(loss_val.item())
175
176                 if early_stopper is not None:
177                     if early_stopper.step(loss.item(), self.model):
178                         print(f"Early␣stopping␣at␣epoch␣{i_epoch}␣with␣loss␣{loss_val.item()}
                            ")
179                         break
180
181
182                 model.train()
183
184             opt.step()
185             scheduler.step()
186             if not val:
187                 pbar.set_postfix(loss = loss.item(),
188                                  lr = round(scheduler.get_last_lr()[0],6))
189             else:
190                 pbar.set_postfix(loss = loss.item(),
191                                  val_loss = loss_val.item(),
192                                  lr = round(scheduler.get_last_lr()[0],6))
193
194             i_epoch += 1
195
196         if early_stopper is not None:
197             early_stopper.restore_best_weights(self.model)
198
199         model.eval()
200         x_pred_train = self.solve(input_train, xpoints_train, state_variables_train)
201         features = x_pred_train.reshape((x_pred_train.shape[0]*x_pred_train.shape[1],
                x_pred_train.shape[2]))
202
203
204         if val:
205             return losses, losses_val
206         else:
207             return losses, losses_val
208
209     def run_model(self, input_data, xpoints, state_variables):
210         model = self.model
211         model.eval()
212
213         input_data = model.apply_input_scaler(input_data).to(self.device)
214
215         xpoints = torch.tensor(xpoints, dtype=torch.float32).to(self.device)
216         state_variables = torch.tensor(state_variables, dtype=torch.float32).to(self.device)
217
218         x_pred = self.solve(input_data, xpoints, state_variables)
219
220         return model.apply_output_scaler(x_pred, inverse=True)
221
222     def solve(self, data, xpoints, state_variables):
223         """runs time domain, requires scaled inputs, returns scaled outputs"""
224         if len(data.shape) == 2:
225             data = data.reshape(data.shape[0], data.shape[1], 1)
226         assert data.shape[:2] == xpoints.shape[:2], "data␣and␣xpoints␣must␣have␣the␣same␣
                shape"
227         assert state_variables.shape[0] == data.shape[0], "state␣variables␣must␣be␣defined␣
                for␣each␣time-domain-sample"
228         assert state_variables.shape[1] == len(self.state_variable_names), "every␣state␣
                variable␣must␣be␣defined"
229
230         assert data.dtype == torch.float32, "data␣type␣must␣be␣torch␣float32"
231         assert xpoints.dtype == torch.float32, "xpoints␣type␣must␣be␣torch␣float32"
232         assert state_variables.dtype == torch.float32, "state_variables␣type␣must␣be␣torch␣
                float32"
233
234         assert data.shape[2] + state_variables.shape[1] == self.model.input_dim, f"dimension␣
                of␣inputs␣must␣equal␣input␣dimension.␣data␣dimension␣is␣{data.shape[3]}␣state␣
                dimension␣is␣{state_variables.shape[1]}␣input␣dimension␣is␣{self.model.input_dim}
                "
235
```

```
236         _, x_pred = odeint(self.f, torch.zeros((data.shape[0], self.model.output_dim), dtype=
                torch.float32), xpoints[0,:], solver=self.solver, args={"h0": None, "c0": None, "
                data":data, "xpoints":xpoints, "state":state_variables})
237         x_pred = x_pred.permute(1, 0, 2)
238         return x_pred
239 
240     def f(self, t, x, h0, c0, input_data, xpoints, state_variables):
241         # set tpoints (xpoints must be identical for all inputs)
242         tpoints = xpoints[0,:]
243         global inpoints
244         dt = tpoints[1]-tpoints[0]
245         # find the index of the input
246         default_index = len(tpoints) - 1
247         idxt = default_index  # set the index of the tpoint in case t is outside the range of
                tpoints
248         for i, tpoint in enumerate(tpoints):
249             if tpoint >= t:  # check if te tpoint is higher than t
250                 idxt = i  # save the index of tpoint
251                 break
252 
253         # get the value of the input acceleration
254         if idxt == default_index:  # in case the index is the default index
255             inpoints = input_data[:, idxt]
256         else:  # linear interpolate between two indexes
257             deltat = t - tpoints[idxt]
258             inpoints = input_data[:, idxt] + (input_data[:, idxt + 1] - input_data[:, idxt])
                    * deltat /dt
259 
260         #reshape the input data
261         inpoints_reshaped = inpoints.reshape(inpoints.shape[0], 1, inpoints.shape[1])
262         #concatenate inputs with state variables to get the network input
263         inputs = torch.cat((inpoints_reshaped, state_variables.reshape(state_variables.shape
                [0], 1, state_variables.shape[1])), 2)
264 
265         # run the model
266         x_new, h0, c0 = self.model.forward(inputs, h0, c0)
267 
268         return x_new, h0, c0
269 
270 
271 class GP_LSTM_StateVar(NNExecutorStateVarTemplate):
272     __model: GP_LSTMModel
273     def __init__(self, solver, model, state_var_names: list[str], device:str="cpu"):
274         super().__init__(solver, model, state_var_names, device=device)
275         self.state_scalers = list()
276 
277     def solve(self, data, xpoints, state_variables, idx_start_gp=0, gp_data_stepsize=1,
            output_dim=None):
278         """runs time domain, requires scaled inputs, returns scaled outputs"""
279         if len(data.shape) == 2:
280             data = data.reshape(data.shape[0], data.shape[1], 1)
281         assert data.shape[:2] == xpoints.shape[:2], "data␣and␣xpoints␣must␣have␣the␣same␣
                shape"
282         assert state_variables.shape[0] == data.shape[0], "state␣variables␣must␣be␣defined␣
                for␣each␣time-domain-sample"
283         assert state_variables.shape[1] == len(self.state_variable_names), "every␣state␣
                variable␣must␣be␣defined"
284 
285         assert data.dtype == torch.float32, "data␣type␣must␣be␣torch␣float32"
286         assert xpoints.dtype == torch.float32, "xpoints␣type␣must␣be␣torch␣float32"
287         assert state_variables.dtype == torch.float32, "state_variables␣type␣must␣be␣torch␣
                float32"
288 
289         assert data.shape[2] + state_variables.shape[
290             1] == self.model.input_dim, f"dimension␣of␣inputs␣must␣equal␣input␣dimension.␣
                    data␣dimension␣is␣{data.shape[3]}␣state␣dimension␣is␣{state_variables.shape
                    [1]}␣input␣dimension␣is␣{self.model.input_dim}"
291 
292         if output_dim is None:
293             output_dim = self.model.gp_input_dim
294 
```

```
295         _, x_pred = odeint(self.f, torch.zeros((data.shape[0], output_dim), dtype=torch.
                float32),
296                            xpoints[0, :], solver=self.solver,
297                            args={"h0": None, "c0": None, "data": data, "xpoints": xpoints, "
                                state": state_variables})
298         x_pred = x_pred.permute(1,0,2)
299 
300         features = x_pred.reshape((x_pred.shape[0] * x_pred.shape[1], x_pred.shape[2]))
301 
302         if self.device == "cuda":
303             features = features[idx_start_gp::gp_data_stepsize].to(device="cuda")
304         else:
305             features = features[idx_start_gp::gp_data_stepsize].to(device="cpu")
306 
307         return features
308 
309     def train_model(self,
310                     data_train,
311                     input_train,
312                     xpoints_train,
313                     state_variables_train,
314                     epochs,
315                     lr,
316                     lr_scale,
317                     lr_scale_step,
318                     gp_data_stepsize=1,
319                     data_val=None,
320                     input_val=None,
321                     xpoints_val=None,
322                     state_variables_val=None,
323                     patience=None,
324                     min_delta=None):
325 
326         if data_val is not None and input_val is not None and xpoints_val is not None and
                state_variables_val is not None:
327             val = True
328         else:
329             val = False
330 
331         if patience is None and min_delta is None:
332             # neither patience nor min delta defined
333             early_stopper = None
334         elif patience is None:
335             # only min_delta defined
336             early_stopper = EarlyStopping(min_delta=min_delta)
337         elif min_delta is None:
338             # only patience defined
339             early_stopper = EarlyStopping(patience=patience)
340         else:
341             #both min_delta and patience defined
342             early_stopper = EarlyStopping(patience=patience, min_delta=min_delta)
343 
344         state_variables_train = copy.deepcopy(state_variables_train)
345 
346         if val:
347             assert state_variables_train.shape[1] == state_variables_val.shape[1], f"training
                    ␣and␣validation␣data␣must␣have␣the␣same␣number␣of␣state␣variables␣currently␣{
                    state_variables_train.shape[1]}␣and␣{state_variables_val.shape[1]}"
348             state_variables_val = copy.deepcopy(state_variables_val)
349         self.model.train()
350 
351         opt = Adam(self.model.parameters(), lr=lr)
352         scheduler = torch.optim.lr_scheduler.StepLR(opt, step_size=lr_scale_step, gamma=
                lr_scale)
353 
354         losses = list()
355         losses_val = list()
356         input_train = self.model.apply_input_scaler(input_train.cpu(), fit=True).to("cpu")
357         data_train = self.model.apply_output_scaler(data_train.cpu(), fit=True).to(self.
                device)
358 
```

```
359         data_train = data_train.reshape((data_train.shape[0]*data_train.shape[1], 1))
360 
361         xpoints_train = torch.tensor(xpoints_train, dtype=torch.float32).to("cpu")
362 
363         for i in range(state_variables_train.shape[1]):
364             self.state_scalers.append(MaxAbsScaler())
365             scaler = self.state_scalers[i]
366             state_variables_train[:,i] = apply_scaler(scaler, state_variables_train[:,i], fit
                =True, dtype=torch.float32)
367 
368         state_variables_train = torch.tensor(state_variables_train, dtype=torch.float32).to("
            cpu")
369 
370         if val:
371             input_val = self.model.apply_input_scaler(input_val).to("cpu")
372             data_val  =  self.model.apply_output_scaler(data_val).to(self.device)
373 
374             data_val = data_val.reshape((data_val.shape[0]*data_val.shape[1], 1))
375 
376             xpoints_val = torch.tensor(xpoints_val, dtype=torch.float32).to("cpu")
377 
378             for i in range(state_variables_val.shape[1]):
379                 scaler = self.state_scalers[i]
380                 state_variables_val[:,i] = apply_scaler(scaler, state_variables_val[:,i],
                    dtype=torch.float32)
381 
382             state_variables_val = torch.tensor(state_variables_val, dtype=torch.float32).to("
                cpu")
383 
384         mll = gpytorch.mlls.ExactMarginalLogLikelihood(self.model.gp_model.likelihood, self.
            model.gp_model)
385 
386         self.model.train()
387         pbar = tqdm(range(epochs))
388 
389         if not val and early_stopper is not None:
390             warnings.warn("Early␣stopper␣is␣defined␣but␣validation␣data␣not␣provided.␣Early␣
                stopping␣will␣be␣ignored.")
391 
392         i_epoch = 0
393         for _ in pbar:
394             opt.zero_grad()
395             idx_train = 0
396 
397             idx_start_gp = i_epoch % gp_data_stepsize
398 
399             features = self.solve(input_train, xpoints_train, state_variables_train,
                idx_start_gp=idx_start_gp, gp_data_stepsize=gp_data_stepsize)
400 
401             self.model.set_train_data(inputs = features, targets = data_train[idx_start_gp::
                gp_data_stepsize])
402 
403             output = self.model.gp_model(features)
404 
405             loss = -mll(output, data_train[idx_start_gp::gp_data_stepsize])
406             loss.backward()
407             losses.append(loss.item())
408 
409             if val:
410                 self.model.eval()
411                 features = self.solve(input_val, xpoints_val, state_variables_val,
                    idx_start_gp=idx_start_gp, gp_data_stepsize=gp_data_stepsize)
412 
413                 output = self.model.gp_model(features)
414 
415                 loss_val = -mll(output, data_val[idx_start_gp::gp_data_stepsize])
416                 losses_val.append(loss_val.item())
417 
418                 if early_stopper is not None:
419                     if early_stopper.step(loss_val.item(), self.model):
```

```
                        print(f"Early stopping at epoch {i_epoch} with loss {loss_val.item()}
                            ")
                        break

                self.model.train()


            opt.step()
            scheduler.step()
            if not val:
                pbar.set_postfix(loss = loss.item(),
                                 lr = round(scheduler.get_last_lr()[0],6))
            else:
                pbar.set_postfix(loss = loss.item(),
                                 val_loss = loss_val.item(),
                                 lr = round(scheduler.get_last_lr()[0],6))
            i_epoch += 1
        if early_stopper is not None:
            early_stopper.restore_best_weights(self.model)

        self.model.eval()
        features = self.solve(input_train, xpoints_train, state_variables_train,
            gp_data_stepsize=gp_data_stepsize)

        self.model.set_train_data(inputs=features, targets=data_train[::gp_data_stepsize])

        return losses, losses_val

    def run_model(self, input_data, xpoints, state_variables):
        state_variables = copy.deepcopy(state_variables)
        model = self.model
        model.eval()
        with torch.no_grad():
            input_data = model.apply_input_scaler(input_data.cpu()).to("cpu")

            xpoints = torch.tensor(xpoints, dtype=torch.float32).to("cpu")

            for i in range(state_variables.shape[1]):
                scaler = self.state_scalers[i]
                state_variables[:,i] = apply_scaler(scaler, state_variables[:,i], dtype=torch
                    .float32)

            state_variables = torch.tensor(state_variables, dtype=torch.float32).to("cpu")

            features = self.solve(input_data, xpoints, state_variables)

            out_gp = model.gp_model(features)
            out_gp = model.gp_likelihood(out_gp)

        out_shape = (input_data.shape[0], input_data.shape[1], model.output_dim)

        pred = out_gp.mean
        pred = model.apply_output_scaler(pred.cpu(), inverse=True)

        stddev = out_gp.stddev.cpu() * model.scale_output

        lower_bound = pred - stddev*2
        upper_bound = pred + stddev*2

        pred = pred.reshape(out_shape)
        lower_bound = lower_bound.reshape(out_shape)
        upper_bound = upper_bound.reshape(out_shape)

        pred = pred.cpu().detach().numpy()
        lower_bound = lower_bound.cpu().detach().numpy()
        upper_bound = upper_bound.cpu().detach().numpy()

        return pred, lower_bound, upper_bound

    def f(self, t, x, h0, c0, input_data, xpoints, state_variables):
        # set tpoints (xpoints must be identical for all inputs)
```

```
488         tpoints = xpoints[0,:]
489         global inpoints
490         dt = tpoints[1]-tpoints[0]
491         # find the index of the input
492         default_index = len(tpoints) - 1
493         idxt = default_index  # set the index of the tpoint in case t is outside the range of
                tpoints
494         for i, tpoint in enumerate(tpoints):
495             if tpoint >= t:  # check if te tpoint is higher than t
496                 idxt = i  # save the index of tpoint
497                 break
498 
499         # get the value of the input acceleration
500         if idxt == default_index:  # in case the index is the default index
501             inpoints = input_data[:, idxt]
502         else:  # linear interpolate between two indexes
503             deltat = t - tpoints[idxt]
504             inpoints = input_data[:, idxt] + (input_data[:, idxt + 1] - input_data[:, idxt])
                    * deltat /dt
505 
506         #reshape the input data
507         inpoints_reshaped = inpoints.reshape(inpoints.shape[0], 1, inpoints.shape[1])
508         #concatenate inputs with state variables to get the network input
509         inputs = torch.cat((inpoints_reshaped, state_variables.reshape(state_variables.shape
                [0], 1, state_variables.shape[1])), 2)
510 
511         # run the model
512         x_new, h0, c0 = self.model.forward(inputs, h0, c0)
513 
514         return x_new, h0, c0
515 
516 class GP_PCA_LSTM_StateVar(GP_LSTM_StateVar):
517     __model: GP_PCA_LSTMModel
518     def __init__(self, solver, model, state_var_names: list[str], device:str="cpu"):
519         super().__init__(solver, model, state_var_names, device=device)
520 
521     def solve(self, data, xpoints, state_variables, idx_start_gp=0, gp_data_stepsize=1):
522         features = super().solve(data, xpoints, state_variables, idx_start_gp=idx_start_gp,
                gp_data_stepsize=gp_data_stepsize, output_dim=self.model.hidden_dim)
523 
524         if self.model.training:
525             features = self.model.pca.fit_transform(features, determinist=False)
526         else:
527             features = self.model.pca.transform(features)
528 
529         return features
530 
531 
532 def load(path):
533     executor = torch.load(path, weights_only=False)
534     executor.model.eval()
535     return executor
```

C

PSD Results

C.1. Fold 1

C.1.1. 1 Foam sheet

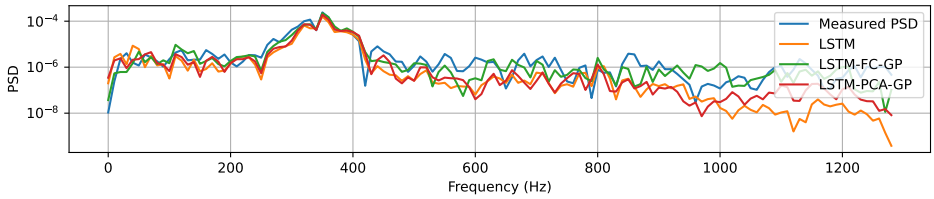


Figure C.1: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 1 of fold 1

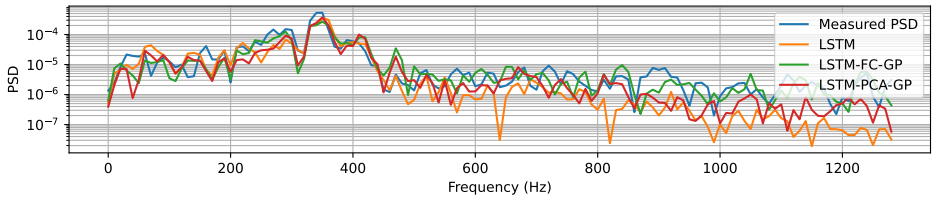


Figure C.2: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 2 of fold 1

C.1.2. 2 Foam sheets

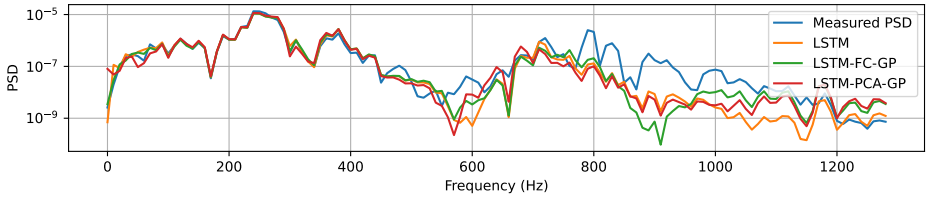


Figure C.3: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 3 of fold 1

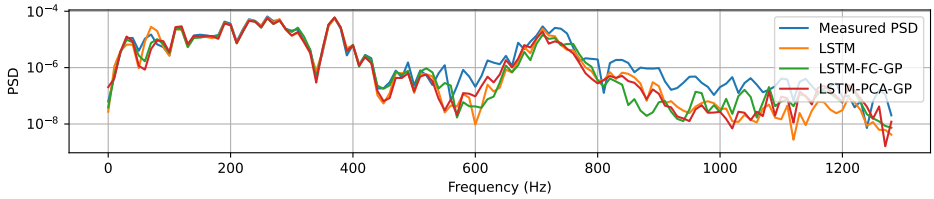


Figure C.4: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 4 of fold 1

C.1.3. 3 Foam sheets

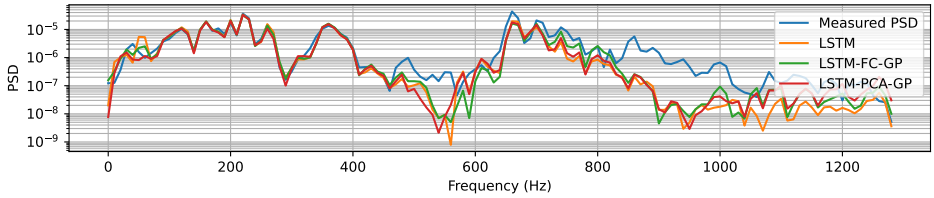


Figure C.5: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 5 of fold 1

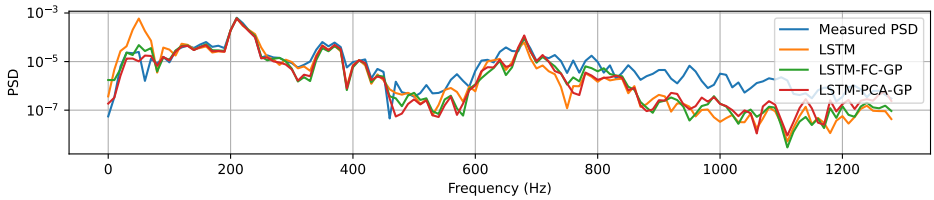


Figure C.6: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 6 of fold 1

C.2. Fold 2

C.2.1. 1 Foam sheet

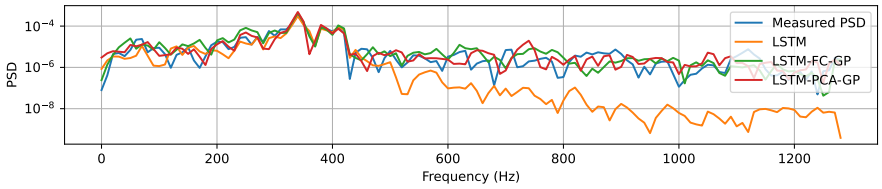


Figure C.7: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 1 of fold 2

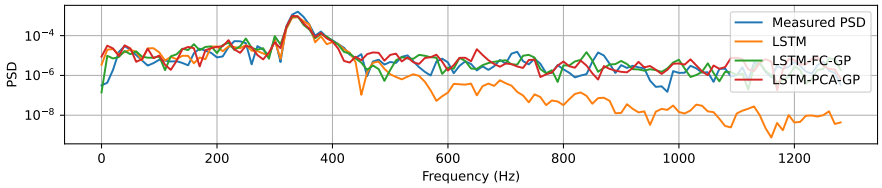


Figure C.8: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 2 of fold 2

C.2.2. 2 Foam sheets

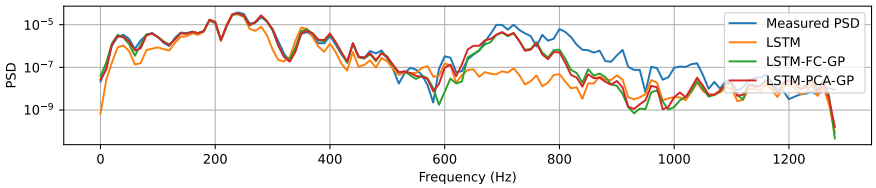


Figure C.9: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 3 of fold 2

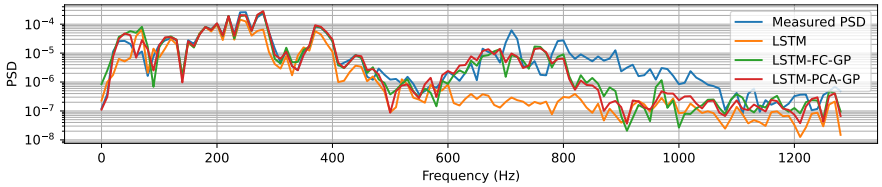


Figure C.10: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 4 of fold 2

C.2.3. 3 Foam sheets

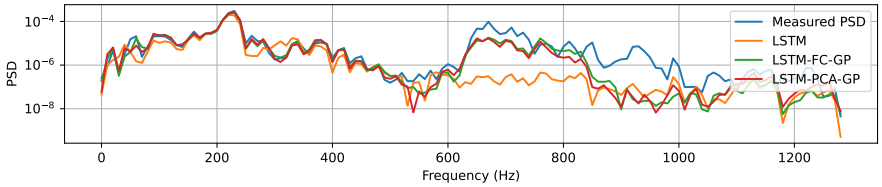


Figure C.11: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 5 of fold 2

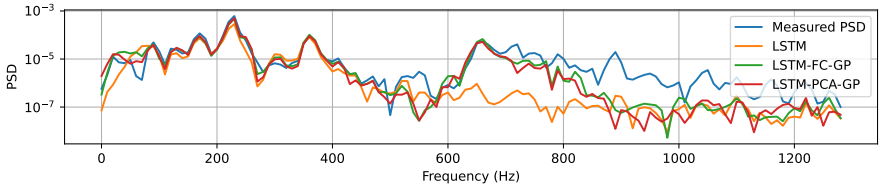


Figure C.12: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 6 of fold 2

C.3. Fold 3

C.3.1. 1 Foam sheet

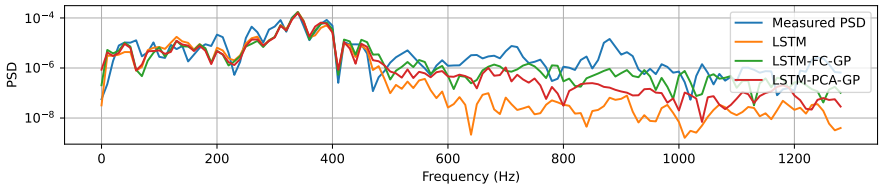


Figure C.13: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 1 of fold 3

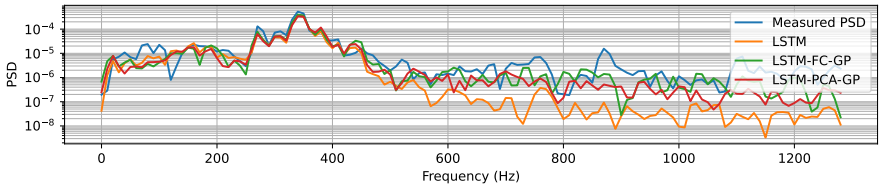


Figure C.14: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 2 of fold 3

C.3.2. 2 Foam sheets

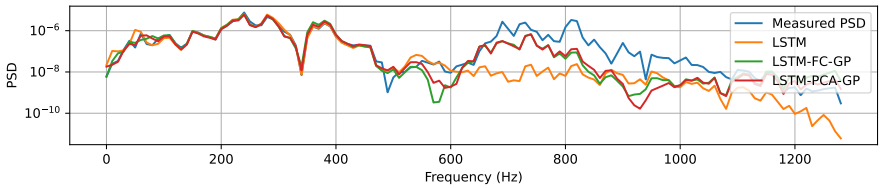


Figure C.15: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 3 of fold 3

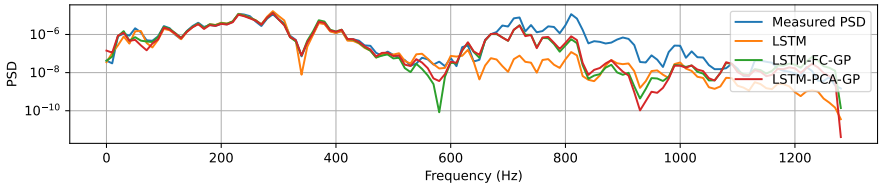


Figure C.16: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 4 of fold 3

C.3.3. 3 Foam sheets

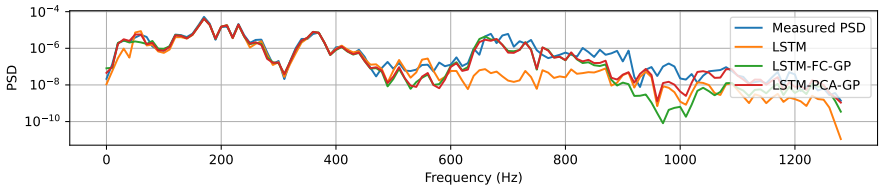


Figure C.17: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 5 of fold 3

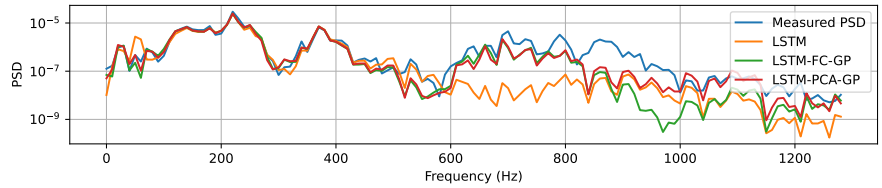


Figure C.18: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 6 of fold 3

C.4. Fold 4

C.4.1. 1 Foam sheet

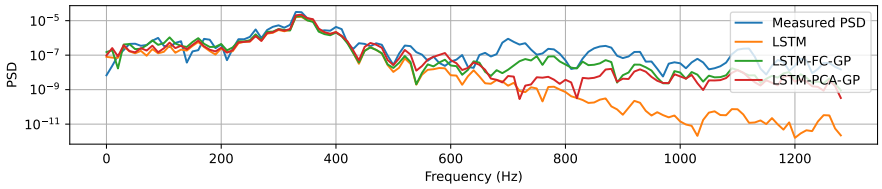


Figure C.19: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 1 of fold 4

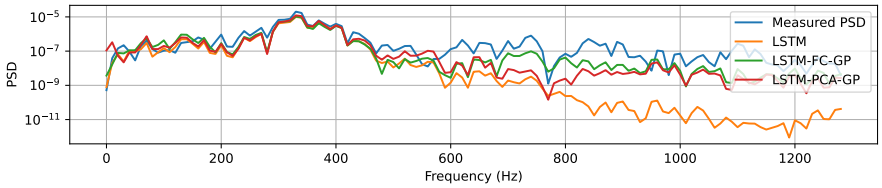


Figure C.20: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 2 of fold 4

C.4.2. 2 Foam sheets

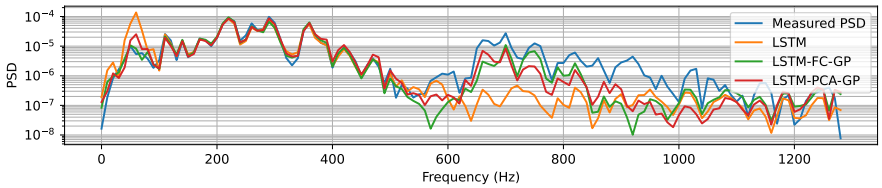


Figure C.21: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 3 of fold 4

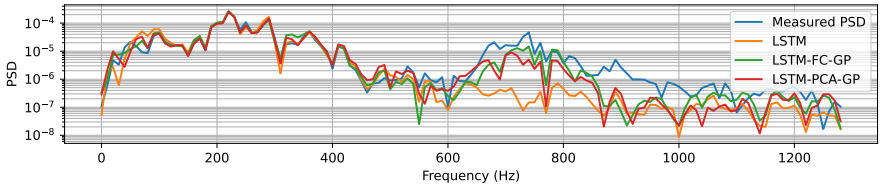


Figure C.22: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 4 of fold 4

C.4.3. 3 Foam sheets

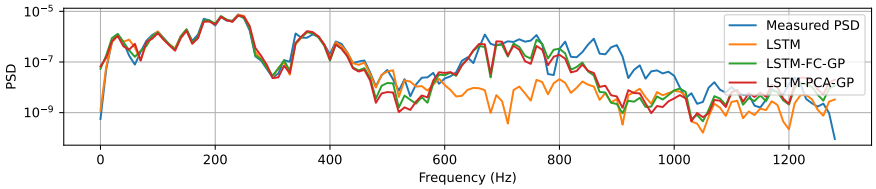


Figure C.23: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 5 of fold 4

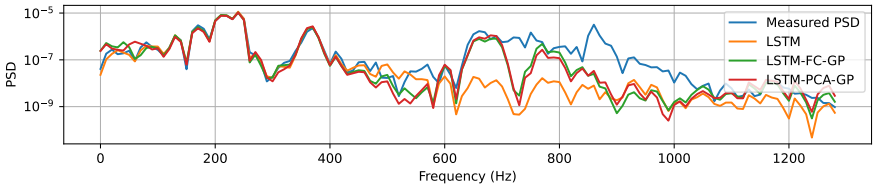


Figure C.24: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 6 of fold 4

C.5. Fold 5

C.5.1. 1 Foam sheet

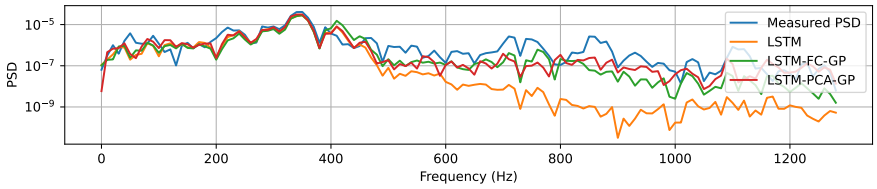


Figure C.25: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 1 of fold 5

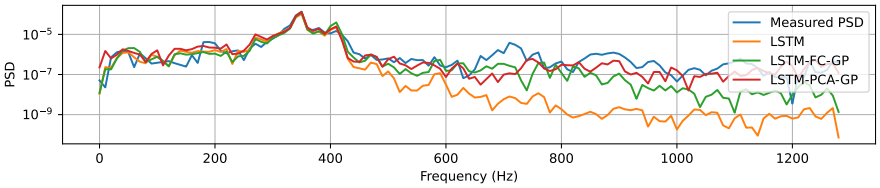


Figure C.26: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 2 of fold 5

C.5.2. 2 Foam sheets

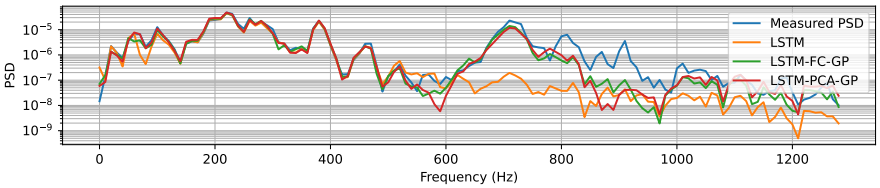


Figure C.27: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 3 of fold 5

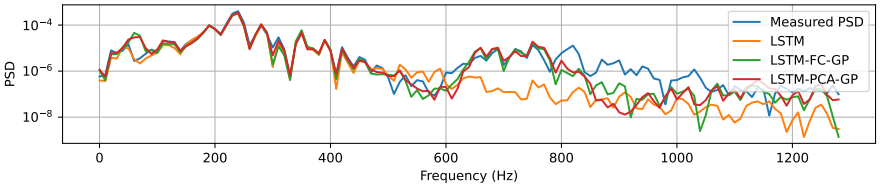


Figure C.28: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 4 of fold 5

C.5.3. 3 Foam sheets

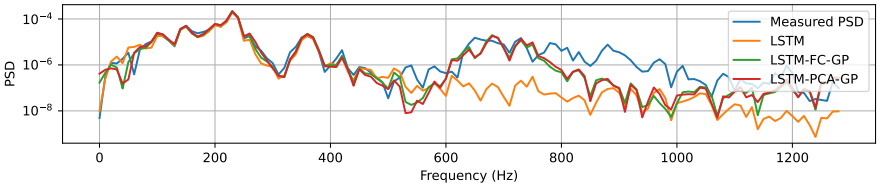


Figure C.29: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 5 of fold 5

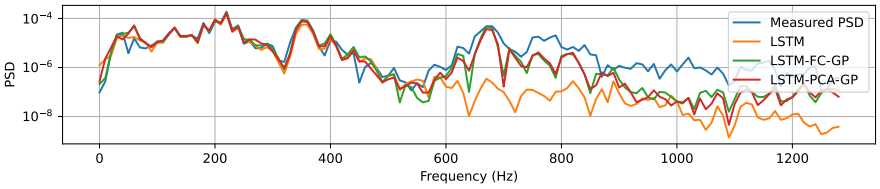


Figure C.30: Comparison between the PSD of the real measurement and the results of the LSTM, LSTM-FC-GP, and LSTM-PCA-GP models for validation result 6 of fold 5