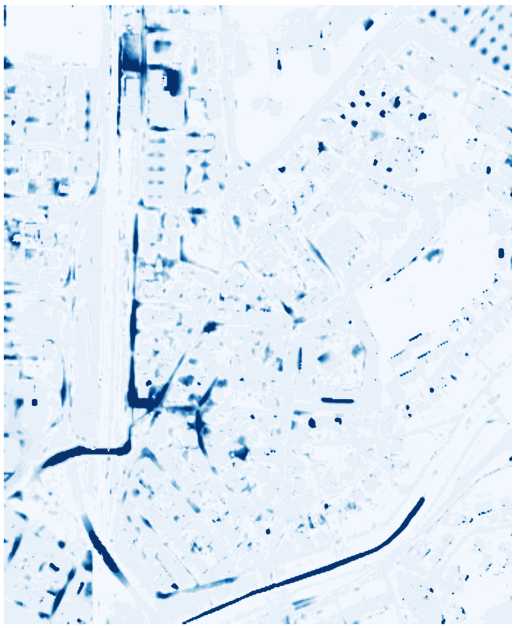
CHALMERS
UNIVERSITY OF TECHNOLOGY



Towards More Accurate and Adaptable Surrogate Models

Evaluating Autoencoders, LSTMs, and Data-Driven Techniques in a Surrogate Model framework for Hydrodynamic Simulations

Master's thesis in Complex Adaptive Systems

ELSA DELSHAMMAR
ADINA HELLSTRÖM

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2025

www.chalmers.se

Master's thesis 2025

Towards More Accurate and Adaptable Surrogate Models

Evaluating Autoencoders, LSTMs, and Data-Driven Techniques in a Surrogate Model framework for Hydrodynamic Simulations

ELSA DELSHAMMAR
ADINA HELLSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2025

Towards More Accurate and Adaptable Surrogate Models

Supervisors: Lars Jonasson, DHI
Freja Petersen, DHI
Johan Jonasson, Department of Mathematical Sciences
Examiner: Johan Jonasson, Department of Mathematical Sciences

Master's Thesis 2025
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualizations developed in Python. To the right: surface elevation in Øresund region, to the left: water depth in Kungsbacka.

Typeset in LaTeX
Gothenburg, Sweden 2025

Towards More Accurate and Adaptable Surrogate Models
Evaluating Autoencoders, LSTMs, and Data-Driven Techniques in a Surrogate Model framework for Hydrodynamic Simulations
Elsa Delshammar, Adina Hellström
Department of Mathematical Sciences, Chalmers University of Technology

Abstract

Physics-based models are essential for understanding and predicting environmental change, but their high computational cost limits their use in rapid forecasting and scenario analysis. This study explores data-driven surrogate models for approximating physics-based models to reduce computational demands without sacrificing accuracy. Building on an existing surrogate modeling framework, we aim to: (1) enhance model performance in a coastal case study through alternative dimensionality reduction and regression techniques, (2) improve realism by integrating observational data into latent space representations, and (3) evaluate the framework's transferability through application to an urban case study. The results from the coastal case indicate that surrogate models using autoencoders and LSTM networks improve predictive performance by approximately 20% compared to the baseline implementation of PCA and linear regression, as measured by RMSE. This improvement is primarily attributed to the LSTM model. However, the autoencoder also demonstrated potential, such as its ability to incorporate observational data into the latent space. The urban case demonstrated the framework's potential, but revealed new challenges. While predictions were acceptably accurate in most instances, model instability suggested that the dataset was insufficient and that alternative methods may be more suitable. Overall, the results highlight the strong potential of surrogate modeling as a computationally efficient complement to physics-based models. This capability enables rapid evaluation of climate scenarios and real-time forecasting. Future work should focus on incorporating extreme events into training objectives and tailoring model complexity to specific application needs.

Keywords: Autoencoder, Principal component analysis (PCA), LSTM, Surrogate model, Reduced order model, Scientific machine learning, Hydrodynamical model

Acknowledgements

We want to express our sincere thanks to Lars Jonasson and Freja Petersen at DHI for their support, guidance, and valuable input throughout this thesis work. Your feedback and expertise have played an important role in shaping this project and it has been a real pleasure working with you.

We also want to thank Johan Jonasson at Chalmers, who has served as both supervisor and examiner. Your advice have been a great help and are truly appreciated.

Adina Hellström and Elsa Delshammar, Gothenburg, June, 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

PCA	Principal component analysis
AE	Autoencoder
LR	Linear regression
LSTM	Long short-term memory
FMROM	Flexible mesh reduced order model
RMSE	Root mean square error
t2RMSE	Root mean square error of top 2% values
NRMSE	Normalized root mean square error

Contents

List of Figures

List of Tables

1

Introduction

A physics-based model is a way to describe a system through the fundamental principles of physics, which are expressed as mathematical equations. These equations are then solved with appropriate boundary conditions to predict the system's behavior. Physics-based modeling has been used for decades in many scientific and engineering applications, including design optimization and forecasts. In the context of climate change, such modeling is especially important as it enables us to simulate complex environmental systems, anticipate future impacts, and develop strategies to reduce damages [2]. These models are essential tools for informing policy decisions and supporting short- and long-term planning in the face of a rapidly changing climate.

Modeling a complex system often requires an extensive number of computations which can be highly computationally demanding, especially when fine resolution, multiphysics interactions, or large-scale domains are involved [3]. As a result, the computational cost limits the possibilities of exploring different scenarios or making real-time predictions. This challenge underscores the need for more efficient modeling approaches that can approximate system behavior with reduced computational cost.

Surrogate modeling is an established technique for addressing the problem with computationally intensive digital simulators. Purely data-driven surrogate models imitate physics-based models by learning the behaviors through input-output relationships from simulated data, often by using machine learning techniques. The application areas are many, and a wide range of techniques are used in the field. Concepts such as digital twins [4] and graph-based surrogate models [5] has shown to be successful in providing early warnings and better predictions for extreme weather events, such as tropical cyclones, atmospheric rivers, and extreme temperatures. With these types of models, results can be delivered in seconds rather than hours or days, while maintaining a high level of accuracy.

1.1 Previous work

Data-driven surrogate models have been implemented in various scientific fields [6], including design optimization [7] and environmental modeling [8]. A previous study has proposed a framework for a flexible-mesh reduced order model (FMROM) to serve as a surrogate [1]. The framework combines dimensionality reduction with a regression model to map input values to output values. It was evaluated on three test cases from the hydrodynamical MIKE 21 Flow Model FM [9], all achieving a speed-up factor of 1,000 while maintaining 90% accuracy. However, the study only investigated three specific implementations of the FMROM framework, combining one reduction method with three different regression models. Although the study yielded promising results, it is interesting to explore other implementations of the framework. As this study builds upon the FMROM framework, it is described in more detail in Section 2.5.

Another similar framework has been proposed by Jamlech Iram Gojo Cruz et al. [10]. In this work, a pipeline of PCA and different regression models is used to predict temperatures in urban areas, and the results generally show good performance. Some aspects are related to this work, both in terms of the application on time-dependent weather data, such as wind velocity and precipitation, and the techniques used, such as using PCA to project data into a latent space for regression. However, this is not a surrogate model framework as it relies on real data and does not aim to emulate a physics-based model. In addition, the pipeline is designed to predict a single variable output and does not generate a time series across a full spatial domain.

In the field of surrogate modeling for urban case studies, Lishu Xu and Liang Gao propose a hybrid surrogate model for real-time coastal urban flood prediction applied to a region in China [11]. In the work presented in the article, a long short-term memory (LSTM) neural network for predicting drainage outflows is combined with a one-dimensional convolutional neural network (1D CNN) for predicting water depths. Flood simulation results produced by the physics-based models MIKE21 and MIKE Urban framework [12] are used to train and assess the hybrid surrogate model. The study area was divided into 1.2 million cells, and to reduce training costs and improve training accuracy, cells with water depths below 0.05 m were excluded in the surrogate model. To further reduce computational cost, a suggestion to use a lower resolution (40 m x 40 m) is proposed but not implemented, and no

other types of dimensionality reduction techniques are mentioned. When working with even larger domains, other types of dimensionality reduction techniques, such as PCA or autoencoders, could instead be highly relevant techniques to decrease computational cost.

Comparisons between techniques in the field of dimensionality reduction has shown that autoencoders in many cases outperform linear models like POD (referred to as PCA in this report). Milano et al. [13] performed a study showing that for a relatively small additional computational cost nonlinear neural networks provide improved reconstruction and prediction capabilities for the near wall velocity fields. Similarly, [14] shows that in both test scenarios, autoencoder-based ROMs exhibit better prediction accuracy in emulating spatial distributions for physical variables of interest than POD-based ROMs.

The influence of number of hidden layers nodes on performance of autoencoders is investigated in [15]. The paper finds that when the size of the hidden layer in an autoencoder matches the true underlying structure (intrinsic dimensionality) of the data, it performs better. This suggests that autoencoders work best when they're neither too simple nor too complex but just enough to capture the essential features.

When it comes to surrogate model forecasting, LSTMs are widely used and often outperforms other models. Roy et al. [16] show that a surrogate model trained on high-fidelity physics-based model accurately forecasted street-scale flooding, with the sequence-to-sequence LSTM demonstrating superior performance, especially for long-term forecasts, by effectively modeling both input and output sequences jointly. A study by Laviola da Silva et al. [17] demonstrates the LSTM's capability to enhance the accuracy of significant wave height predictions, a testament to its ability to assimilate and learn from complex temporal patterns within the data. It also states that challenges arise when using results from numerical models for training, as the inherent biases of these models can also be transferred to the neural network. Ideally, observed data should be used to train the network to accurately learn the actual behavior of natural phenomena, thus avoiding the consistent underestimation or overestimation of variables. This further motivates the need of integrating observation data when working with physics-based model.

Another study combines POD and LSTM as a surrogate model for predicting counter jet behavior [18]. It demonstrates the effectiveness of combining POD for spatial di-

mensionality reduction with LSTM for capturing temporal dynamics and emphasizes the importance of sufficient training data, noting that model performance improves with increased training data. The identification of higher predictive errors in regions with complex vortex dynamics highlights the challenges in modeling turbulent or highly dynamic flow features. This suggests that special attention may be needed in areas with similar complexities, such as water current velocities, and that such variables may require more sophisticated modeling techniques or additional data.

In summary, current applications of data-driven surrogate models still face limitations in accurately predicting state variables across diverse spatial structures, such as coastal [19, 1] and urban environments [11]. Therefore, enhancing these models and implementing frameworks in various settings represents a valuable contribution to the field of surrogate modeling for water-related systems. This advancement could lead to more robust and generalizable solutions that better support decision-making in complex, real-world hydrological scenarios.

1.2 Aim

This research project aims to improve the performance of a surrogate model, following the FMROM framework. Specifically, it will investigate whether the prediction error of the surrogate model, when compared to physics-based model data, can be reduced. It also aims to examine whether data assimilation with observational data can make the surrogate potentially surpass the physics-based model in approximating real-world behavior. Furthermore, this project will assess the model's flexibility and adaptability across other scenarios.

1.3 Research questions

To achieve the aim, this project will address the following research questions.

1. Can the performance of the surrogate model be improved by:
   (a) using an alternative dimensionality reduction technique that yields more informative latent space representations?
   (b) using an alternative regression model that may better capture the relationships between the input and output values in the reduced-order data?

2. Can the physics-based model data align better with real-world behavior by incorporating observation data into the latent space representations?
3. Can the surrogate model framework be applied to an urban model and does it show the same potential?

1.4 Limitations

The scope of this report include analysis of two test cases produced by the hydrodynamical MIKE 21 Flow Model [9] and MIKE+ [20]. Drawing overall conclusions about MIKE models might not be possible, since their properties may differ significantly. Also, the surrogate model is tied to the setup of the physics-based model, and no structural modifications are possible. However, there are hybrid model types, such as physics-informed neural networks (PINNs), where physical constraints can be incorporated, but these are not explored in this project. Further, the two cases are limited to specific geographical domains and to datasets of limited size and diversity, which constrain the generalizability and applicability of the results to other contexts.

In surrogate modeling, a key advantage is the significant savings in computational resources enabled by faster model execution. This study primarily focuses on enhancing performance in terms of accuracy, while also exploring the framework's applicability and adaptability across different contexts. Consequently, model runtime is limited to be addressed in the discussion rather than reported as a primary result.

2

Theory

This chapter begins with an overview of the theoretical background of the data types used in the study. It then describes the statistical and machine learning models applied in the project. Finally, the FMROM framework, which serves as the foundation for this work, is described in detail.

2.1 Physics-based models

The behavior of a complex system can be simulated and predicted with the help of physics-based models. These models use physical laws and equations to create computational representations of reality. When numerically solving these equations on which the representations are built, we get physics-based models. One example is the numerical solutions of some partial differential equation (PDE) or systems of PDEs, often used to describe real-world dynamical systems where the components change in multiple directions or over time.

2.1.1 MIKE software

The MIKE software developed by DHI is a tool for water-related modeling. The MIKE 21 Flow Model FM [9], Hydrodynamic Module is used for applications of hydraulic phenomena in lakes, estuaries, bays, coastal areas and seas. It is based on the numerical solution of the two-dimensional shallow water equations, obtained by depth-integrating the incompressible Reynolds averaged Navier-Stokes equations. The model is based on a fixed set of parameters, such as bed roughness and turbulence coefficients. The model calculates the resulting flow and distributions of salt and temperature, subject to a variety of forcing and boundary conditions.

The 2D Overland module in MIKE+ [20] is used for modeling 2D free-surface flows. It allows simulation of hydraulic phenomena on land surfaces, overland water bodies, and coastal areas. The models for urban modeling include topography and

infrastructure layers, and simulates water level variations and flows in response to a variety of forcing functions.

2.1.2 Data assimilation

Data assimilation is a methodological framework that combines observations with numerical models to improve the estimation of a system's state [21]. In the context of physics-based models, such as models for weather forecasting, oceanography, or climate science, data assimilation integrates real-world measurements into simulations governed by physical laws. Because observations are often sparse, noisy, or incomplete, and models may have uncertainties or simplifications, data assimilation provides a way to optimally merge both sources of information. Techniques like the Kalman filter, variational methods, or ensemble-based approaches update the model state by adjusting it toward observations while respecting the underlying physical dynamics. This process helps maintain physical consistency, improves forecast accuracy, and supports better decision-making in complex systems where both observational data and theoretical models are essential but individually insufficient.

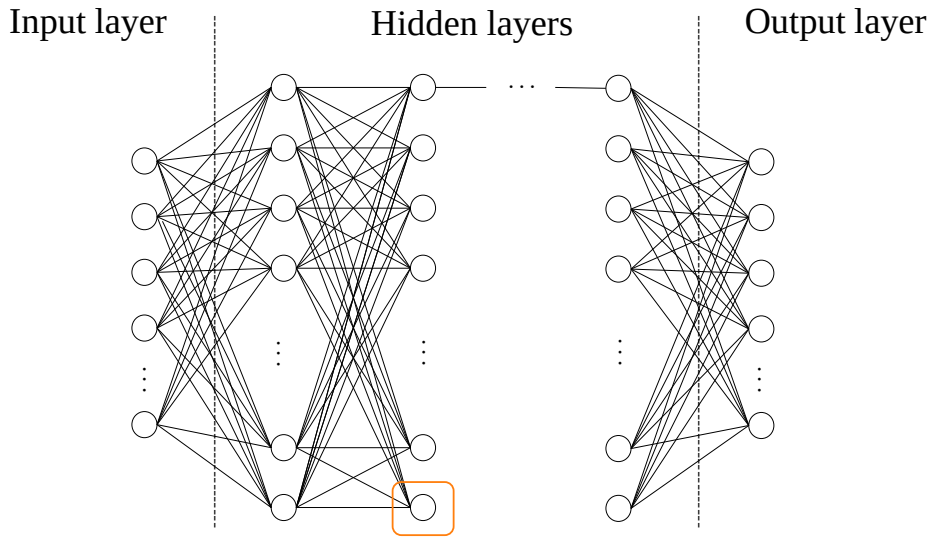
2.2 Neural Networks

Neural networks are a class of machine learning models that mimic how neurons in the human brain process information. They consist of interconnected units, typically organized in layers. Neural networks are designed to identify complex, nonlinear patterns in data and are used for tasks such as forecasting and classification. There are various types of neural networks. This section will introduce one type of feedforward neural network and one type of recurrent neural network, followed by a discussion about activation functions, which play a crucial role in all neural networks. Finally, we bring up the issue of overfitting and introduce some regularization techniques.

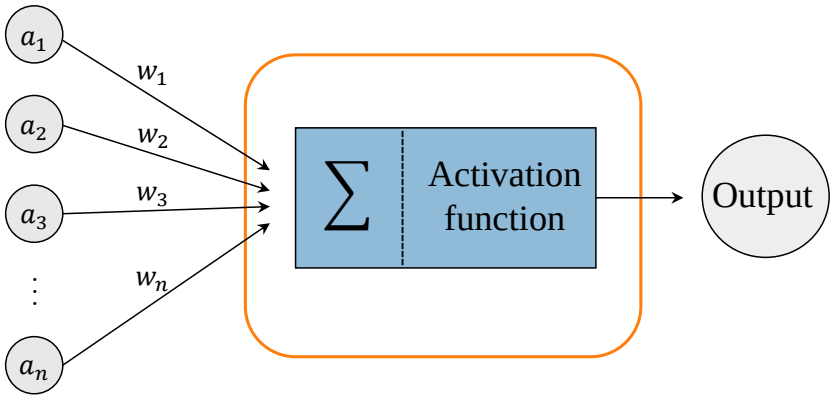
2.2.1 Fully connected Neural Networks

A fully connected neural network is a type of feedforward neural network, where information flows in a single direction from the input layer to the output layer, without any cycles or loops. In this architecture, each neuron in one layer is connected to every neuron in the next layer, forming a dense pattern of connections. This structure, illustrated in figure 2.1(a), enables the network to learn complex representations by allowing maximum interaction between layers. Looking at an individual neuron, as shown in figure 2.1(b), each unit receives inputs through weighted connections.

When data are passed through the network, each neuron computes a weighted sum of its inputs, adds a bias term, and applies an activation function to this sum. The activation function determines the neuron's output, which is then transmitted to subsequent layers. Performing this process across all neurons in the network generates an output, which is the model's prediction.



(a) A neural network



(b) A single neuron

Figure 2.1: Overview of a neural network with a closeup of a single neuron, marked with an orange box.

For a neural network with L layers, the activation vector of layer l, which contains the activations of all units in that layer, is given by:

$$a^l = g(z^l) \quad \text{with} \quad z^l = W^l a^{l-1} + b^l$$

where W^l is the weight matrix containing all the weights associated with the incoming connections to layer l [22]. The vector b^l holds the biases for each unit in layer l, and g denotes the activation function applied uniformly across all units in the layer. The activation vector of the input layer, a^0, correspond to the input data, and the activations of the output layer, a^L, correspond to the network's prediction. In an untrained neural network, the weights and biases are typically initialized in a defined manner. A feedforward neural network then learns by passing training data through the network, computing a loss based on the error between the predicted output and the target output, and then adjusting the weights and biases to minimize the loss. The loss function can be defined in various ways. The mean squared error (MSE) is a common choice:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

where n is the total number of data points, y_i is the actual value of the ith data point and $\hat{y}_i$ is the predicted value of the ith data point. The process of computing the gradient of the loss function with respect to the weights and biases, and then updating the parameters based on these gradients, is known as backpropagation. To compute the gradients, the process begins with the loss function $\mathcal{L}$ and propagates the error backward [23]. This allows us to determine how much each weight and bias contributes to the error, so they can be adjusted accordingly during training. The error in each layer is determined by:

$$\delta^L = \nabla_a \mathcal{L} \odot g'(z^L) \tag{2.2}$$

$$\delta^l = (W^{l+1})^T \delta^{l+1} \odot g'(z^l), \quad l = L-1, L-2, \ldots, 2 \tag{2.3}$$

Here, $\nabla_a \mathcal{L}$ is the gradient of the loss function with respect to the activations at the output layer, and $\odot$ denotes the Hadamard product (element-wise product). The gradients of the parameters of the network are computed as:

$$\frac{\partial \mathcal{L}}{\partial W^l} = \delta^l (a^{l-1})^T$$
$$\frac{\partial \mathcal{L}}{\partial b^l} = \delta^l$$

How the gradients are used is defined by the optimizer. The most basic is gradient descent which is defined as:

$$W^l \leftarrow W^l - \eta \frac{\partial \mathcal{L}}{\partial W^l}$$
$$b^l \leftarrow b^l - \eta \frac{\partial \mathcal{L}}{\partial b^l}$$

where η is the learning rate. A more advanced and widely used optimization method is adaptive moment estimation (Adam) [24], which adjusts the learning rate for each parameter individually during training, improving convergence and stability.

2.2.2 Long Short-Term Memory

A Recurrent Neural Network (RNN) is a type of neural network characterized by a looped network structure, allowing to keep past information and obtain new information by going backward and forward in the network structure. During training, data is passed forward through the network, incorporating not only the current time step but also information from previous time steps, as seen in Figure 2.2(a). The states of each unit are computed as follows:

$$h_t = g(z_t) \quad \text{with} \quad z_t = W_x x_t + W_h h_{t-1} + b_h$$

where x_t is the input at time step t, and h_{t-1} is the hidden state capturing information from previous time step. The hidden state vector is also known as the output vector of the RNN unit; hence, the notation a used in feedforward networks is replaced with h. As in feedforward neural networks, the error between the network's prediction and the target is calculated using a loss function, with mean squared error being a common choice (2.1). To update the network's weights, gradients are propagated back through both the layers and the time steps, following the equation:

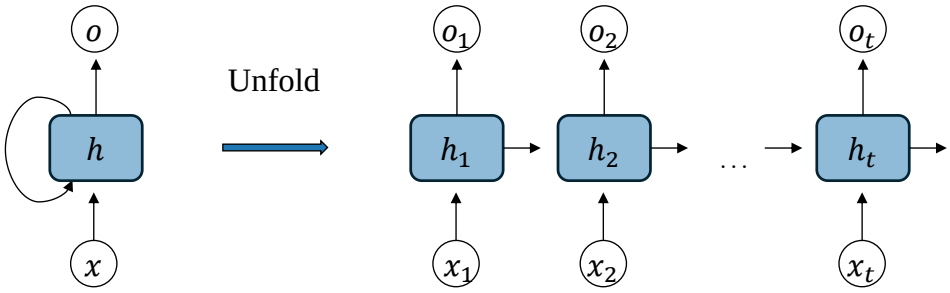
$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial W} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}_i}{\partial W} \propto \sum_{t=0}^{T} \left(\prod_{t=k+1}^{y} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_k}{\partial W}$$

If the term highlighted in red is less than one, the network has problems learning long-term dependencies. This problem is called the vanishing gradient problem. If the term is greater than one, the learning process becomes unstable, which is called the exploding gradient problem [25]. This issue has led to several extensions of RNNs and one of the most popular extensions is the Long Short-Term Memory (LSTM) [26].

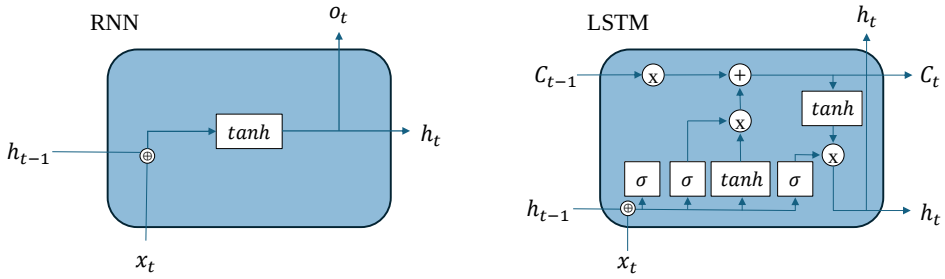
LSTM functions are similar to standard RNNs, they just have another way of computing the hidden states, following the equations:

$$
\begin{aligned}
i_t &= \sigma\left(x_t W_x^i + h_{t-1} W_h^i + b_f\right) \\
f_t &= \sigma\left(x_t W_x^f + h_{t-1} W_h^f + b_i\right) \\
o_t &= \sigma\left(x_t W_x^o + h_{t-1} W_h^o + b_o\right) \\
\tilde{C}_t &= \tanh\left(x_t W_x^g + h_{t-1} W_h^g + b_c\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= \tanh(C_t) \odot o_t
\end{aligned}
$$

The variables i, f and o are called the input, forget, and the output gates. The input gate regulates the amount of new information incorporated into the cell state, while the forget gate determines how much information from the previous state to let through. The output gate controls the extent to which the internal state is exposed to the external network, including higher layers and the next time step. The variable $\tilde{C}_t$ can be interpreted as a candidate for the hidden state and is computed in the same way as the hidden states in the standard RNN. Furthermore, C_t represents the internal memory of the unit, which is updated based on the previous memory C_{t-1} and the candidate $\tilde{C}_t$. The proportions of the two are determined by the forget gate f and the input gate i. Finally, the updated internal memory is used to compute the hidden state h_t weighted by the output gate o. The calculation of the hidden state candidate C_t uses the sigmoid activation function, while h_t uses the hyperbolic tangent activation function, and b_f, b_i, b_o and b_c represent the biases for each term. The design of this network allows for memory and temporal dependencies, making LSTMs appropriate for sequential data. A visualization of an RNN cell and an LSTM cell can be seen in 2.2(b) and how an LSTM network could be used for regression tasks will be further discussed in section 2.4.



(a) A recurrent neural network processes sequences by maintaining a hidden state, h, that evolves over time, shown here as an unrolled sequence of time steps. x is the input and o it the output of the network.



(b) Comparison of a basic RNN cell and an LSTM cell, highlighting the internal structure of each.

Figure 2.2: An overview of a recurrent neural network and a close-up of two cell types, RNN and LSTM. $\oplus$ represents vector concatenation, $+$ and x represent element-wise addition and multiplication and σ and $tanh$ are activation functions

2.2.3 Activation functions

The purpose of activation functions are to introduce nonlinearity to a neural network [27]. A widely used activation function is the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid activation function maps any input value to a range between 0 and 1, effectively normalizing the output. However, during backpropagation, the gradient of the activation function plays a significant role in updating the network's weights, see equation (2.2) and (2.3). As the activation of the neurons approaches the flatter regions of the sigmoid function, the gradient diminishes, leading to inefficient weight updates. This phenomenon, known as the vanishing gradient problem, can hinder the learning process and slow down convergence, particularly in deep neural networks. A second popular activation function is the hyperbolic tangent function, defined as:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

It has a shape similar to the sigmoid function but outputs values in the range of -1 to 1. Because the hyperbolic tangent function is centered around zero, it can handle negative values more effectively. However, like the sigmoid function, it also suffers from the vanishing gradient problem. A third example of an activation function is ReLU, which is an example of a piecewise linear function, defined as:

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

The ReLU activation function help mitigate the vanishing gradient problem in deep neural networks due to its piecewise linear structure. Additionally, it is computationally efficient, making it well-suited for larger models. However, ReLU has a drawback known as the 'dying neuron' problem, where neurons become inactive by outputting zero for all negative values. Leaky ReLU is a variation of ReLU that addresses the 'dying neuron problem' by allowing small negative values instead of zero. It is defined as:

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

Finally, there are also fully linear activation function which in practise is the same as using no activation function at all. Such activation function could be suitable for cases when nonlinearity is unwanted in shallow neural networks. It is also widely

used in the output layer of deep neural networks for regression problems. The linear activation function does not transform the output in any way, and the actual predicted value is therefore returned.

2.2.4 Overfitting and regularization

Underfitting occurs when a machine learning model fails to capture the underlying patterns within the training data, resulting in poor performance on both the training and test sets. If a model performs well on the training set but poorly on the test set, overfitting occurs. This indicates that the model has not only learned the underlying patterns in the training data but also memorized noise, which hinders its ability to generalize to unseen data. To improve the model's ability to generalize one can take on different regularization techniques. The idea behind regularization is to keep the network's weights small, thereby limiting the model's complexity and helping to prevent overfitting. Regularization applies penalties on the layers' parameters, which are added to the loss function the network seeks to minimize. The penalties can be computed in various ways, two common ways are L1- and L2 regularization, defined as follows:

$$\mathcal{L}_{\text{L1}} = \mathcal{L}_{\text{original}} + \lambda \sum_{i=1}^{n} |w_i|$$

$$\mathcal{L}_{\text{L2}} = \mathcal{L}_{\text{original}} + \lambda \sum_{i=1}^{n} w_i^2$$

where $\mathcal{L}_{\text{original}}$ is the original loss, commonly defined as the mean squared error, w_i are the model parameters, and λ is the regularization strength.

Two other techniques for regularization are early stopping and dropout. Early stopping halts the training process when the model's performance on a validation set ceases to improve - either after a set number of stagnant epochs or when improvements fall below a certain threshold. Dropout is when a random fraction of neurons is temporarily "dropped out" (i.e., set to zero) during training, with a given probability p. Each element is dropped independently using a Bernoulli distribution, and to keep the overall output scale consistent, the remaining active elements are scaled by $\frac{1}{1-p}$ during training. During evaluation, dropout is turned off, and the layer simply passes the input through unchanged. This encourages the network to develop redundant representations and reduces reliance on any single neuron, thereby improving generalization.

2.2.5 Transfer learning

Transfer learning involves using a pretrained machine learning model and applying it to a new, yet related, problem. The idea is to leverage the knowledge the model has acquired from a task with a lot of training data to improve performance on a different task where data is limited. To adapt the pretrained model to the new task it needs to be retrained, this process is often referred to as fine-tuning.

In transfer learning, the layers of the neural network are typically categorized as either frozen or trainable. Frozen layers retain the knowledge acquired during pretraining and remain unchanged during the fine-tuning process. In contrast, trainable layers are updated during fine-tuning to adapt the model to the new task. The trainable layers can consist of either selected layers from the original model or new layers added to the existing architecture for this purpose. What layers to freeze depends on the task. Generally, it is advisable to freeze the layers that capture more generic features, which are applicable to both the original and the new problem.

2.3 Dimensionality reduction

The goal of dimensionality reduction is to transform high-dimensional data into a lower-dimensional space while preserving the most important information. High-dimensional data can lead to overfitting, where a model becomes too complex and learns noise in the data rather than meaningful patterns. In many cases, high-dimensional data contains redundant or irrelevant features, which can obscure true relationships. By reducing the number of dimensions, noise is minimized, improving the model's ability to generalize. Additionally, lowering the number of features reduces computational complexity and storage requirements, making the model more efficient. There are numerous different techniques for dimensionality reduction, two of them are described in more detail in the following subsections.

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique used to transform data into a new coordinate system where the axes (principal components) are ordered by the amount of variance they capture in the data. The first principal component represents the direction of maximum variance, the second captures the second highest variance, and so on. Each principal component is orthogonal to the others. When the data matrix contains temporal data, where each column repre-

sents a feature (e.g., a spatial element) and each row corresponds to a time step, the technique is known as Proper Orthogonal Decomposition (POD). However, since Principal Component Analysis (PCA) is a more widely recognized term, we have opted to refer to this technique as PCA throughout the report, regardless of the structure of the data matrix.

The principal components can be obtained by performing Singular Value Decomposition (SVD) of the feature data matrix $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$$

Here, $\mathbf{W}$ are the right singular vectors and contain the principal directions, $\mathbf{\Sigma}$ holds the singular values σ_k, and $\mathbf{U}$ contains the left singular vectors. $\mathbf{W}$ is the spatial basis in this case, and $\mathbf{U}$ the temporal, and the projected data are given by $\mathbf{U\Sigma} = \mathbf{XW}$. Since the principal directions are ordered by the variance they explain, the projected data exhibits decreasing variance across dimensions.

Dimensionality reduction is achieved by selecting the first d principal directions, corresponding to the first d columns of $\mathbf{W}$, and projecting the data accordingly. The number of dimensions, d, to which the data is reduced can be chosen in various ways. One approach is to set a threshold for the amount of variance from the original high-dimensional data that the low-dimensional representation should retain. Another way to determine d is to retain the principal components whose eigenvalue exceed the average eigenvalue. This approach is called North's rule [28].

2.3.2 Autoencoder

An autoencoder is a nonlinear dimensionality reduction method that can be described as a type of feedforward neural network that learns to compress and reconstruct data. An autoencoder is able to extract relevant features from data to deliver a lower-dimensional representation, which is embedded into a nonlinear manifold. It consists of two components: an encoder and a decoder. The encoder learns a latent representation of the input data and reduces full-dimensional data to a lower-dimensional representation. The decoder uses the encoded vector to reconstruct the data back to its original dimensions. A visual representation of an autoencoder network can be seen in 2.3.

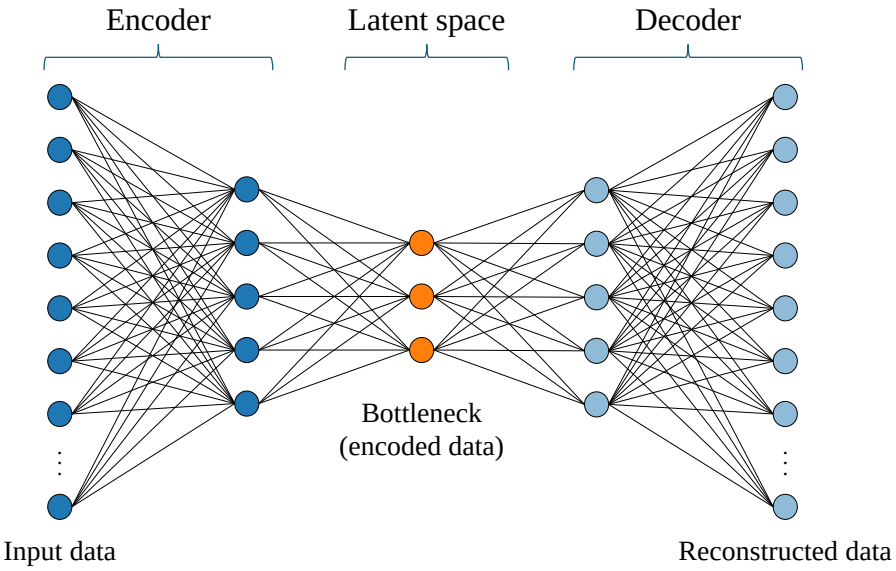


Figure 2.3: Schematic representation of an autoencoder.

2.4 Regression models

The idea of regression is to use statistical methods to find a relationship between independent variables (features) and a dependent variable (target). In machine learning and time series modeling, features refer to the input variables used by a model to make predictions. These can include past values of the target series, transformations of those values (such as time lags or rolling statistics), or external inputs such as weather or calendar data. These external features, also called covariates, are typically exogenous variables that are not part of the target series but may carry useful information for improving predictions. Unlike endogenous features (which are derived from the target itself), covariates come from external sources and are not predicted by the model. In the context of forecasting, the target is the series to be predicted, and covariates are incorporated at time t to help predict the target at that same time step [29].

The goal of the regression model when used for time series prediction is to find a function that can predict the value of the next time step based on past data. When making multiple predictions, there are different approaches to utilizing past data: one can either use actual historical values for each step, or recursively feed previously predicted values into the model to generate further forecasts. The latter approach is known as autoregressive forecasting.

In many practical applications, a single target series may be influenced by multiple factors simultaneously. Multivariate regression extends the basic regression framework to model the relationship between multiple independent variables and one or more dependent variables. In the context of time series forecasting, multivariate regression models can leverage both endogenous and exogenous features to make predictions. For example, instead of predicting the future value of a single time series using only its own past values, a multivariate model can incorporate information from other related time series or external covariates. This allows the model to capture interactions and dependencies between variables, which can be especially beneficial when the target variable is influenced by complex, interconnected factors. When applied to multiple targets simultaneously, this becomes a multivariate multi-output regression problem.

There are several types of regression models, the following sections will describe two in more detail: Linear Regression and LSTM Regression.

2.4.1 Linear Regression

Linear regression estimates the linear relationship between independent variable(s) and a dependent variable. A simple linear regression, using one dependent variable to predict one independent variable is defined as:

$$y_t = \beta_0 + x_t\beta_1 + \epsilon_t \tag{2.4}$$

where y_t is the dependent variable at time t. The coefficients β_0 and β_1 denote the intercept and the slope of the line respectively. The error term, ϵ_t, captures anything that may affect y_t other than x_t. When multiple independent variables are used to predict multiple dependent variables, the model is referred to as multivariate multiple linear regression and equation (2.4) transforms to:

$$\mathbf{Y}_t = \boldsymbol{\beta}_0 + \mathbf{X}_t\boldsymbol{\beta} + \boldsymbol{\epsilon}_t$$

A common approach when using linear regression for time series forecasting is to incorporate lagged values into the feature matrix. Lags refer to past values of a variable shifted in time, for example, y_{t-1}, y_{t-2} (referred to as one and two lag values). One can also incorporate lead values; future values of the variable shifted in time. They help the model learn from historical and/or future patterns to improve predictions for the next time step. This allows the model to capture temporal dynamics and autocorrelation structures present in the data.

Linear regression is a relatively simple model based on traditional statistical inference. It relies on several key assumptions to produce reliable and interpretable results. These include linearity and, in the context of time series data, stationarity, meaning that the relationship between independent and dependent variables is linear and stable over time. It also assumes that the residuals (errors) are uncorrelated. In time series settings, this implies that any autocorrelation (correlation of a time series with its own past values) in the data has been successfully modeled and is no longer present in the residuals.

2.4.2 LSTM Regression

A Long Short-Term Memory (LSTM) model is well-suited for sequential data because it effectively captures temporal dependencies, as detailed in Section 2.2.2. For regression tasks, the model must be trained to predict continuous values at the next time step based on previous values. Lags are not used in the same way as in linear regression. Instead, a sequence of historical data is provided to the model, which it uses to make a prediction. Through training, the model will weigh the historical data in a way that optimizes the results.

The LSTM model is a fully recurrent RNN, meaning that during prediction, each output is generated using several inputs [30]. One of these inputs is the previous target value, and for the first prediction, this is set to the last known target value. For all subsequent predictions, the previous target value will instead be the model's previous prediction, following an autoregressive approach. The model also uses the previous hidden state, which carries temporal information forward.

The autoregressive approach leverages the sequential nature of time series by using past outputs to inform future predictions, allowing the model to capture complex temporal dynamics. Also, the model can adapt and adjust its trajectory based on its own evolving forecast path. However, since each prediction is based on previous predictions, any small error can accumulate over time and lead to increasingly inaccurate forecasts, especially for long horizons.

2.5 Surrogate models

A surrogate model provides a simplified approximation of physics-based models and strives to emulate it by mapping inputs to outputs, often without explicitly un-

derstanding the relationship between them. This approach is valuable as many physics-based models are complex, high-order, and computationally expensive to run. Instead of solving the full physical equations, surrogate models typically rely on statistical methods or machine learning techniques. As a result, surrogate models can serve both as an alternative to, and as a complement for, computationally intensive models.

In order to speed up predictions of a physics-based model for coastal modeling, Freja Petersen et al. have proposed a surrogate modeling framework for a flexible-mesh reduced order model (FMROM) [1]. The framework uses a combination of reversible dimensionality reduction and a regression model to produce predictions of some state variables over a spatial domain. It suggests a flow map $\mathcal{M}$, based on a regression model trained on states and some external forcings. The forcings are assumed to be known in the next time step. The flow map is then described by

$$\mathbf{x}_{k+1} = \mathcal{M}(\mathbf{x}_k, \mathbf{u}_{k+1}), \qquad k = 0, ..., N_T.$$

Here, $\mathbf{x}_k$ is the state vector, N_T is the number of time steps and $\mathbf{u}_k$ the forcings. If the history of the state variables and forcings are known, lags can be included in the flow map according to

$$\mathbf{x}_{k+1} = \mathcal{M}(\mathbf{x}_k, \mathbf{x}_{k-1}, ..., \mathbf{x}_{k-l_x}, \mathbf{u}_{k+1}, \mathbf{u}_k, ..., \mathbf{u}_{k-l_u}), \qquad k = 0, ..., N_T.$$

where l_x and l_u is the number of lagged values for the state variables and forcings respectively. The critical steps of the flowmap are 1) the projection into a latent space using spatial dimensionality reduction, 2) the mapping of the states from one time step to the next (regression), and 3) the projection back to the original space. The results of the surrogate model is a full-domain prediction of the state values. In figure 2.4, originally from [1], the steps of the model framework are visualized in a schematic illustration.

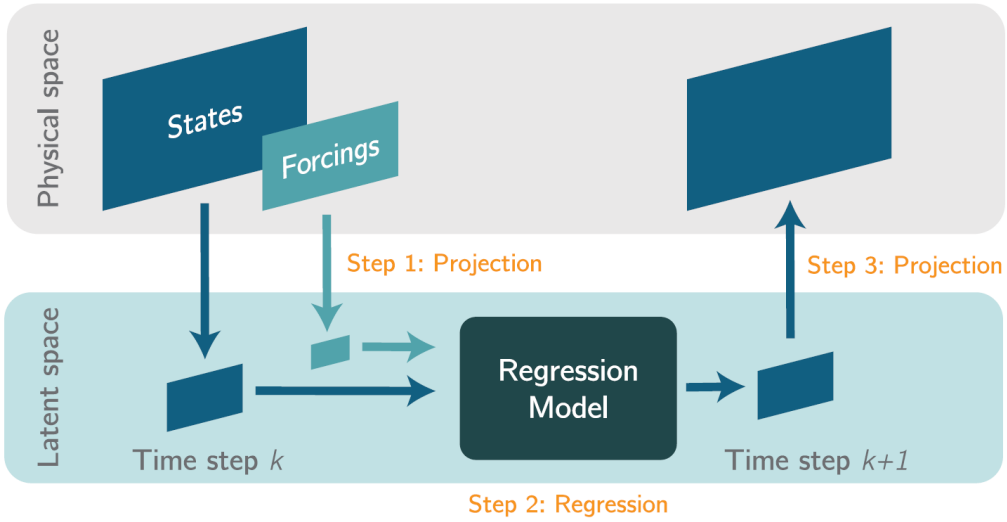


Figure 2.4: Visualization of FMROM framework suggested by Freja Petersen et al. [1]

3

Methodology

This chapter presents the methodology of two case studies separately: the coastal case in Section 3.2, followed by the urban case in Section 3.3. The coastal case forms the core of the project, investigating different implementations of the FMROM framework and examines which types of models yield the best performance. In contrast, the urban case serves as a secondary exploration, examining how well the framework adapts when applied to a different context.

3.1 Used Software

In this report, we use physics-based model results as input to the surrogate framework (from now on referred to as MIKE model data or just MIKE data). For the coastal case, the MIKE21 Hydrodynamic model is used to produce the data, and for the urban case, MIKE+ urban flooding model. Both datasets are preprocessed with the open source Python libraries ModelSkill [31] and MIKEIO [32].

Autoencoders are developed using the built-in functionality of the Python library TensorFlow [33], which enables layer-level customization of the architecture. Regression models are developed using Darts [34], a Python library built on PyTorch [35] and TensorFlow, designed for user-friendly time series forecasting, supporting both univariate and multivariate time series models.

3.2 Coastal case

As mentioned above, the FMROM framework was originally proposed for surrogate modeling in coastal scenarios. In this section, we investigate whether alternative implementations of the framework can improve the performance of the surrogate modeling. In Section 3.2.1, we describe the data used for the surrogate model and how it is preprocessed. In Section 3.2.2 and 3.2.3, we describe the details of the FMROM implementations, followed by a description of how the evaluation of the

results are conducted, in Section 3.2.4.

3.2.1 Data collection and preprocessing

The coastal case study focuses on the Øresund region, a strait located between Skåne (Sweden) and Zealand (Denmark), the geometry can be seen in 3.1. The data comes from the physics-based MIKE21 Hydrodynamic model of the Øresund region [36]. Both the input variables to the physics-based model, referred to as forcing variables, and the model outputs, referred to as state variables, are utilized in training the surrogate model. Details about the data are given in Tables 3.1 and 3.2.

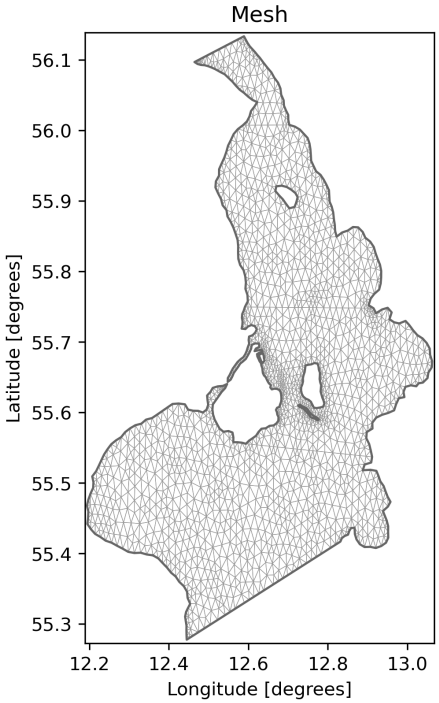


Figure 3.1: Geometry of the Øresund region used for the coastal case, containing mesh elements in an unstructured grid.

The available dataset spans ten years (2014–2023). Producing this amount of data is highly resource-intensive and time-consuming, which makes it uncommon to have access to such a large dataset. Relying on the full dataset would undermine the main advantage of using a surrogate model, namely, the ability to achieve significant time savings. To make our results more relevant in a real-world context, where data availability is often limited, we choose to make most of the development based on a subset of the full dataset, referred to as the regular dataset. However, in some cases it is relevant to see how the amount of data affects the results, which is when

States	Description	Unit
Surface elevation	Water height above or below a fixed reference point. 3,320 unstructured mesh elements.	m
U velocity (water current)	Vector component in u (x) direction. 3,320 unstructured mesh elements.	m/s
V velocity (water current)	Vector component in v (y) direction. 3,320 unstructured mesh elements.	m/s

Table 3.1: Description of state data variables in the coastal case

Forcings	Description	Unit
Surface elevation north boundary	Water height above or below a fixed reference point. 13 point elements.	m
Surface elevation south boundary	Water height above or below a fixed reference point. 27 point elements.	m
Air pressure	2D grid data file, 16 (4x4) elements.	hPa
U velocity (wind)	Vector component in u (x) direction. 2D grid data file, 16 elements.	m/s
V velocity (wind)	Vector component in v (x) direction. 2D grid data file, 16 elements.	m/s

Table 3.2: Description of forcing data variables in the coastal case

we use the full dataset, from now on referred to as the extended dataset. To ensure that seasonal variations are captured, we use one year of data for the regular training dataset, one month for validation, and just under a year for testing. Validation and test sets are the same regardless of which training set is used. For models that are not trained iteratively, it is not feasible to use a separate validation set; instead, the validation set is merged into the training set. The data contains variables with different temporal resolutions, some recorded at 30-minute intervals and others at hourly intervals. To ensure consistency, we extract evenly distributed time steps with an hourly resolution for all state and forcing data. The temporal distribution of the datasets is presented in Table 3.3.

Since the dataset is the output of a simulation with the MIKE software, there are no missing values or significantly incorrect outliers. Hence, no pre-processing in terms of gap-filling or outlier removal is performed. However, scaling data before applying machine learning algorithms is crucial for several reasons. We want to ensure equal weights for all variables and faster convergence in gradient-based algorithms. Hence,

Set	Period	Resolution
Extended training set	2014-01-15 00:00 to 2022-12-31 23:00	60 min
Regular training set	2022-01-01 00:00 to 2022-12-31 23:00	60 min
Validation set	2023-01-01 00:00 to 2023-01-31 23:00	60 min
Regular training set (when no validation)	2022-01-01 00:00 to 2023-01-31 23:00	60 min
Test set	2023-02-01 00:00 to 2023-12-31 23:00	60 min

Table 3.3: Train-, validation- and test-split of the dataset.

we standardize the features by removing the mean and scaling to unit variance.

3.2.1.1 Observation data

In addition to the simulated data, we also have access to observational data, consisting of real-world measurements of the three state variables. For surface elevation, observation data are available in 13 stations. Additionally, three of the stations (Helsingborg, Flinten7, and Drogden) also provide observations for U and V velocity but these will not be used due to low coverage. The location of observation stations can be seen in Figure 3.2, together with the temporal converge of surface elevation measurements.

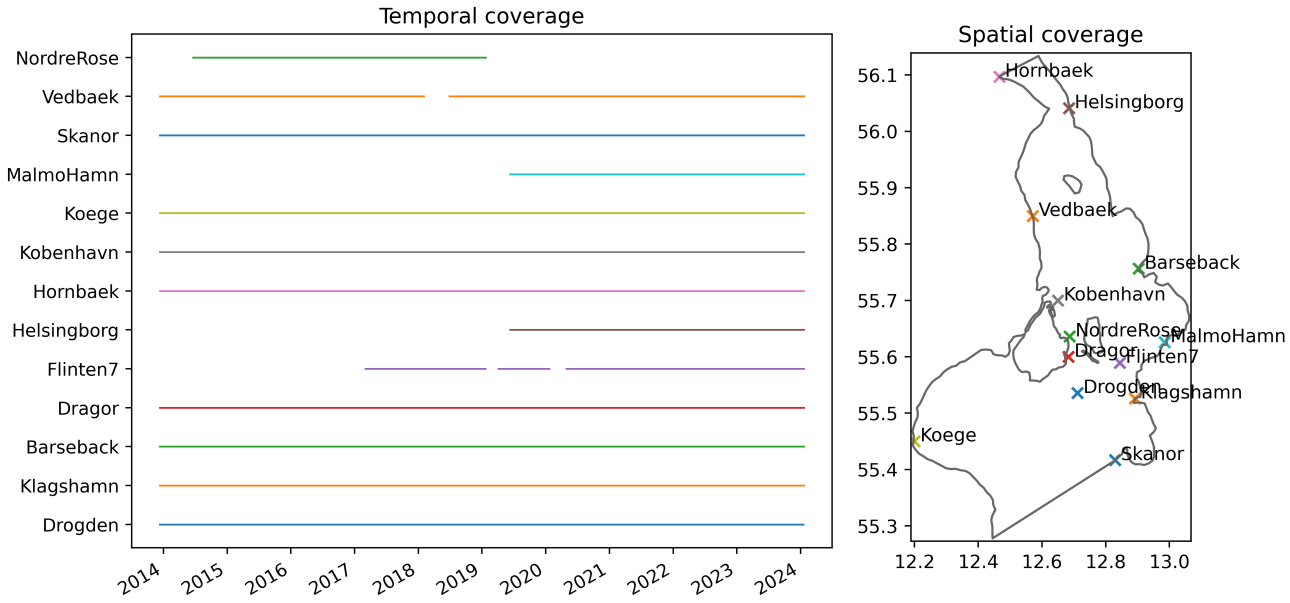


Figure 3.2: Temporal (to the left) and spatial (to the right) coverage of observation data for surface elevation.

The MIKE model simulation does not perfectly represent real-world conditions, so discrepancies between the MIKE data and observational data are to be expected. To enable meaningful comparison between the two datasets, it is necessary to correct for systematic bias. This is particularly important for variables like surface elevation, which are measured relative to a fixed reference point. In this study, we address this by applying a mean bias correction, adjusting the observational data by the difference in mean values between the observed and simulated datasets. Figure 3.3 illustrates the relationship between the MIKE simulation data and the observational data, in three selected observation stations. The blue line represents a perfect 1:1 correspondence, while the red line depicts the actual relationship between the datasets after applying the mean bias correction. Across all observation stations, the fitted slopes are less than one, suggesting that MIKE 21 exhibits a general tendency to overestimate observed values.

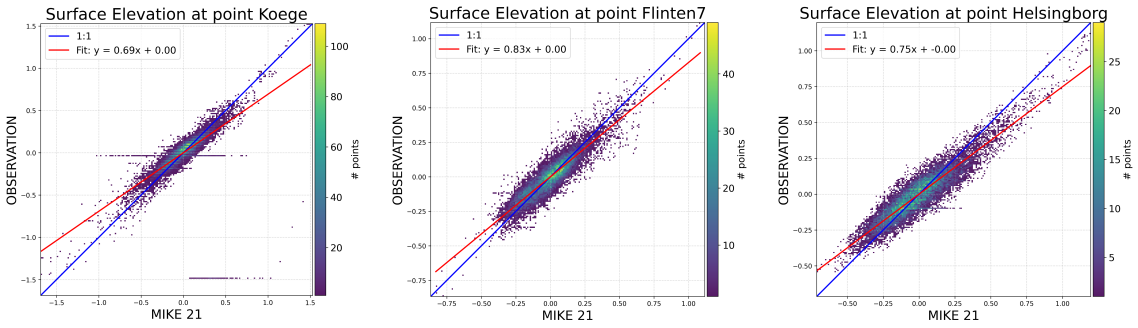


Figure 3.3: Observation data vs MIKE data in three evenly distributed points. The blue line shows a perfect relationship, i.e. the MIKE 21 simulation is perfectly aligned with the observed data. The red line shows the fitted line function $y = kx + m$. From left to right: Koege, Flinten7, and Helsingborg.

In Table 3.4 the difference between MIKE simulation data and observational data is pesented. The difference is computed as the Root Mean Squared Error (RMSE) across time, averaged across all stations.

State	Average RMSE
Surface elevation	7.41 cm
U velocity	8.17 cm/s
V velocity	9.27 cm/s

Table 3.4: RMSE between observed data and MIKE data across training (regular), validation, and test periods. Errors are computed per station against corresponding mesh elements and averaged across all stations.

3.2.2 Dimensionality Reduction

This section focuses on the details of the dimensionality reduction. Specifically, we concentrate on the state variables as they have a significantly higher dimensionality. The state variables have 3,320 dimensions to be compressed, whereas the forcings range from only 12 to 27 dimensions. As a result, forcings do not require as much compression. This suggest the state variables have greater potential for improvement with an alternative reduction technique. For this reason we focus solely on the state variables when exploring different reduction techniques.

As stated by Freja Petersen et al., there is a design choice regarding the use of a separate or combined projection of the variables [1]. A separate projection, i.e. projecting the physical space of each state variable into separate latent spaces, imply higher interpretability and flexibility. On the other hand, using a projection of a combined, i.e. shared physical space of the variables, will lead to a lower dimensional latent space since the variables are correlated. However, a combined projection of the variables may also lead to certain variables dominating the latent space. To maintain flexibility and enable separate regression models we reduce the state variables separately.

We do not impose a specific rule for selecting the latent space dimensionality in either method, as our primary focus is to compare PCA and autoencoders. Instead, we choose to compress the data to the same number of dimensions for both techniques; surface elevation is reduced to 50 dimensions and U and V velocities to 25 dimensions, respectively. The size remains consistent with the order of magnitude used in [1].

3.2.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is used as a baseline for comparison with the autoencoder. We apply PCA separately to all three state variables and the principal components are computed using the reduced training dataset, which consists of approximately one year of data, as described in Table 3.3. Additionally, we conduct an extended analysis to assess the impact of training data size by computing principal components from the full training dataset spanning approximately ten years. The performance of the PCA-based dimensionality reduction transformations is evaluated on the test set.

3.2.2.2 Autoencoder

The use of an autoencoder is explored for dimensionality reduction, as it represents a nonlinear alternative to traditional linear techniques. Building on the analysis in [1], which characterizes surface elevation data as highly linear while U and V velocity data exhibit greater nonlinearity, we design separate autoencoder architectures for surface elevation and velocity components to better capture their distinct characteristics. During the experimental phase, we focus exclusively on the V velocity to determine the optimal architecture for the velocity components, given that the velocities share similar characteristics. Once the final model architecture is found, we define two models with identical architecture: one is trained and evaluated on the U velocity data, and the other on the V velocity data. In this step of the process, we use the regular training set and the validation set to monitor when overfitting occurs. The temporal coverage and resolution of the datasets are presented in Table 3.3.

We start with the simplest architecture: a model without hidden layers and with only linear activation functions, which is functionally equivalent to a linear transformation. If the results suggest that incorporating nonlinearity could be beneficial, we explore more complex architectures. Beginning with this basic model, we incrementally increase the complexity until the network becomes complex enough to overfit the training data. We increase the complexity by adding hidden layers. We start with no hidden layers, then introduce a single hidden layer in both the encoder and decoder with 64 neurons. Next, we expand to two hidden layers with 64 and 256 neurons, in both autoencoder components. The tested architectures first utilize ReLU activation functions in the hidden layers to mitigate the vanishing gradient problem and enhance training stability. However, the network suffers from dying neurons, so ReLU is replaced with Leaky ReLU using a slope of 0.1. The activation function for the bottleneck layer is varied experimentally, and the one that results in the lowest reconstruction error is selected. Finally, a linear activation function is applied in the output layer to maintain the original scale of the data without introducing any transformations.

We have now identified sufficiently complex autoencoder architectures to overfit the training data, and the next step is to apply regularization techniques to prevent overfitting. This is a methodology performed to ensure the network is complex enough and able to learn, focus on improving its generalizability. When applying regularization, we begin with a weak L1 regularization strength and gradually increase it

until the desired effect is achieved. Interestingly, surface elevation does not show improvement with regularization, while the U and V velocity components benefit from L1 regularization strengths of 10^{-5} and 10^{-6}, respectively. The final choice of architecture for the surface elevation autoencoder consists of an input layer, no hidden layers, and an output layer, presented in Table 3.5. The autoencoders for U and V velocity consist of one input layer, two hidden layers, and an output layer, presented in 3.6.

Component	**Encoder**	**Decoder**	**Activation**
Input Layer	Input(n, N_Y)	Input(n, N_Z)	-
Hidden Layers	-	-	-
Output Layers	Dense(n, N_Z)	Dense(n, N_Y)	Linear

Table 3.5: Autoencoder architecture for the surface elevation variable where N_Y denotes the input dimension (3,320), N_Z is the dimensionality of the latent space (50), and n is the number of samples feed to the autoencoder.

Component	**Encoder**	**Decoder**	**Activation**
Input Layer	Input(n, N_Y)	Input(n, N_Z)	-
Hidden Layers	Dense$(n, 256)$	Dense$(n, 64)$	Leaky ReLU
	Dense$(n, 64)$	Dense$(n, 256)$	Leaky ReLU
Output Layers	Dense(n, N_Z)	Dense(n, N_Y)	Tanh / Linear

Table 3.6: Autoencoder architecture for the U velocity and V velocity variables where N_Y denotes the input dimension (3,320), N_Z is the dimensionality of the latent space (25), and n is the number of samples feed to the autoencoder.

For the final evaluation of the autoencoders, the unregularized networks are trained for 10,000 epochs with a batch size of 128, and ADAM optimizer with a learning rate of 0,0001 is employed to minimize the mean squared error (MSE) loss function. The regularized autoencoders follow the same training procedure, with the addition of early stopping, using a patience of 100 epochs.

As with the PCA, we conduct an extended analysis to evaluate the impact of training data size by training the autoencoders with the expanded dataset covering approximately ten years. Since the autoencoders for U and V velocity show improved performance after regularization, the regularized autoencoders are chosen for the extended dataset. For surface elevation, the autoencoder without L1 regularization is used. However, early stopping is applied for practical reasons in this extended

analysis. We evaluate the performance of three configurations on the test set: 1) unregularized autoencoders trained on the regular training set, 2) regularized autoencoders (applied only to the velocity components) trained on the regular training set, and 3) regularized autoencoders trained on the extended training set.

3.2.2.3 Autoencoder with observations

Until now, the autoencoder has been trained exclusively on MIKE model data. However, as this data is based on simulations, it does not provide a fully accurate representation of reality. Table 3.4 presents the RMSE between MIKE model outputs and observational data collected from 13 different monitoring stations. The RMSE for surface elevation reaches up to 7.41 cm (from Table 4.3), highlighting the difference between the MIKE data and actual observations. To reduce this gap and adjust the reconstructed data closer to reality, we introduce observational data into the training process. This step can be interpreted as a form of data assimilation. Due to limited availability of observation data of U and V velocity, we restrict this part of the study to surface elevation only.

We adopt a transfer learning strategy in which the autoencoder is first pretrained on MIKE model data and subsequently fine-tuned using observational data. A common practice in transfer learning is to freeze selected layers of the neural network during fine-tuning, meaning the parameters in those layers remain unchanged during the second training phase. Since the original surface elevation autoencoder lacks hidden layers (Table 3.5), we modify the architecture by adding a hidden layer on each side of the bottleneck. This extension, presented in Table 3.7, allows for effective layer freezing during fine-tuning.

Component	**Encoder**	**Decoder**	**Activation**
Input Layer	Input(n, N_Y)	Input(n, N_Z)	-
Hidden Layer	Dense$(n, 64)$	Dense$(n, 64)$	Linear
Output Layer	Dense(n, N_Z)	Dense(n, N_Y)	Linear

Table 3.7: Autoencoder Architecture for the surface elevation

Once the new autoencoder architecture is defined, the first step is to pretrain the model. This pretraining follows the procedure outlined in the previous section. The MIKE data covers the full spatial domain, compared to observational that which is limited to only 13 elements. Therefore, pretraining with MIKE data enables the

autoencoder to learn general spatial patterns across the entire domain.

The second phase of training aims to align the reconstructed data more closely with real-world observations. During fine-tuning, the autoencoder is trained using standard MIKE data as input, while the target data consists of a modified version of the MIKE data in which values at observation station locations have been replaced with observation data. The loss function combines two mean squared error (MSE) terms; one measuring the error between the observation data and the reconstructed data in the corresponding elements, and another measuring the error between the simulated MIKE data and the reconstructed data in all elements except those with observational data. The error terms are unequally weighted: the first term is multiplied with (1-α) and the second is multiplied with α. With an alpha value of 0.99 we put most emphasis on the MSE associated with observation data.

To ensure that the influence of the observation data extends beyond the elements in question, we freeze the outermost layers of both the encoder and decoder during this part. As a result, only the parameters near the bottleneck, representing the most compact latent representation, are updated. Because these parameters encode higher-level spatial and temporal patterns, modifying them is more likely to produce broader spatial effects across the reconstructed output.

For fine-tuning, the autoencoders are trained for 10,000 epochs with a batch size of 128, and ADAM optimizer with learning rate 10^{-5}. Early stopping is applied with a patience of 10 epochs. Since the observational data closely resembles the data the autoencoder has already been trained on, there is a high risk of overfitting. To mitigate this, both the learning rate and the early stopping patience are reduced, compared to the regular autoencoder.

To validate performance, we use a leave-one-out cross-validation strategy by iteratively excluding data from one observation station at a time during the second training. This allows us to assess whether the reconstructions are influenced by the observational data in a broader spatial domain. The goal is to ensure that the model does not merely adjust to specific elements for which it has been trained on observational data. By repeating this process for all stations, we can evaluate the model's generalization across the entire domain.

The autoencoder serves to demonstrate the feasibility of the approach, and we do

not perform any regression in the latent space found by this autoencoder. The goal of this fine-tuned autoencoder is not to achieve perfect reconstruction of the MIKE data, but to generate outputs that more accurately represent observed reality. Consequently, error metrics can only be computed for the spatial elements where observational data is available. This limitation makes it challenging to meaningfully evaluate a downstream regression model. Therefore, we assess the performance of the fine-tuned autoencoder solely based on its reconstruction metrics.

3.2.3 Regression

The three steps of the FMROM framework are, as mentioned, 1) dimensionality reduction into latent space, 2) regression in latent space and 3) transformation back to physical space. This section explains the methodology of putting together one technique for dimensionality reduction with one regression model, with focus on the regression part.

The state variables are reduced with either PCA or with autoencoders, as described in 3.2.2. The forcing variables are reduced with PCA across all implementations. The dimensions of the latent spaces of the forcings are 10, 19, 4, 4 and 4 for surface elevation's north boundary, surface elevation's south boundary, air pressure, U velocity (wind) and V velocity (wind), respectively. The numbers are found using North's Rule.

A two-step scaling process is applied to both PCA and autoencoder approaches. First, we standardize the data to ensure that the input features are on comparable scales before reduction. After projecting the data into a lower-dimensional latent space, a second standardization is performed to equalize the influence of each latent feature. The post-reduction scaling ensures that all components in the latent space contribute equally to the optimization of the regression model.

After projecting all variables, the multivariate regression is performed in the latent space. After obtaining the prediction in latent space, the pipeline of scaling and dimensionality reduction is performed backwards to get the transformation back into physical space. Therefore, the ability to reverse the transformation is essential to accurately reconstruct the predictions in the physical space.

Building on the analysis presented in [1], which concluded that the FMROM imple-

mentation using PCA for dimensionality reduction combined with linear regression showed the greatest potential in terms of speed and accuracy, we adopt this configuration as our baseline. Additionally, as in [1], no time lag values are used for the state variables while 3 lag values and one lead value are used for forcings. Using no lag values for state variables essentially means that the model represents a mapping from reduced forcings to reduced states.

The LSTM extension of a standard RNN for regression is known to be suitable for sequential data and is good for avoiding the vanishing gradient problem. Hence, it is chosen as an alternative to the models previously used in the FMROM framework implementations (Gated Recurrent Unit model and Multi-layer Perceptron model). The results of previous work show that the surface elevation variable exhibits more linear dynamics [1] and therefore the linear regression performs well on it. On the other hand, U and V velocity show more nonlinear dynamics which motivates the choice of using a regression model that captures more nonlinear dynamics. Due to these differing characteristics, we apply the LSTM model both to all three state variables collectively and to the two velocity components only.

All LSTM networks are trained on the training set for 200 epochs with early stopping, using a patience of 5 epochs. The validation set guides the stopping criterion. The training is performed using Adam optimizer and mean squared error (MSE) loss function.

The Darts hyperparameters can be seen in Table 3.8. All of them except *input chunk length* are found by performing a grid search that iterates through combinations of hyperparameters and evaluates the results on the validation set. The *input chunk length* parameter represents the number of past time steps fed to the forecasting model at prediction time. It is set to 24, and the number is decided based on domain knowledge; 24 hours provides sufficient historical context for the models to learn relevant temporal patterns without making the input so large that it slows down the computation. Other domains need longer history of the states, but Øresund can be considered very small relative to other areas. In addition, it is beneficial to exclude parameters from the grid search when possible, to reduce the number of combinations in the grid search. The parameter *output chunk length* is by default set to 1 without the option to change, resulting in an autoregressive prediction approach. Other neural network-based models are used in previous work [1], and we choose the hyperparameters from this research as a first try for

all LSTM implementations. Since the grid search is highly time-consuming, it is conducted only on the implementation that shows the best performance, and those hyperparameters are then used for all implementations. For evaluation, the metrics from Section 3.2.4 are used to compare the predicted values with the true values.

Hyper parameter	**Value**
input chunk length	24
training length	30
hidden dim	128
batch size	64
n rnn layers	2
dropout	0.05
n epochs	200
learning rate	0.0005

Table 3.8: Darts hyperparameter values for all implementations including regression with LSTM

A list of all surrogate model implementations is given in Table 3.9 below, where a dimensionality reduction technique is combined with a regression model, and used for a given set of variables. To compare the results with previous studies, we use the combination of principal component analysis (PCA) and linear regression (LR) as our baseline.

Notation	**Reduction**	**Regression**	**Variable**
PCA-LR (Baseline)	PCA	Linear regression	Surface el., U and V vel.
PCA-LSTM	PCA	LSTM	Surface el., U and V vel.
PCA-LR_{SE}	PCA	Linear regression	Surface el.
PCA-$LSTM_{U,V}$	PCA	LSTM	U and V vel.
AE-LR	AE	Linear regression	Surface el., U and V vel.
AE-LSTM	AE	LSTM	Surface el., U and V vel.
AE-$LSTM_{U,V}$	AE	LSTM	U and V vel.

Table 3.9: Table of surrogate models based on the FMROM framework, each with one dimensionality reduction and one regression model for a set of variables.

3.2.4 Evaluation

We use four primary metrics to evaluate the results of our FMROM framework implementations. The first three metrics are also applied to the dimensionality reduction process alone, allowing us to identify the source of any errors. The first metric, area-weighted root mean squared error (RMSE), is defined as:

$$\mathrm{RMSE}(\hat{\mathbf{x}}, \mathbf{x}) = \sqrt{\frac{1}{N_T N_x} \sum_{i=1}^{N_T} \sum_{i=1}^{N_x} w_i (x_{k,i} - \hat{x}_{k,i})^2}$$

where $x_{k,i}$ is the true value, $\hat{x}_{k,i}$ is the predicted value at time step k and mesh element i. N_x is the number of mesh elements, N_T is the number of simulated time steps, and w_i corresponds to the area of the ith mesh element. This metric provides an average of the difference between the predicted or reconstructed values and the actual values. The RMSE can also be averaged only in the time dimension (spatial error) or only in the spatial dimension (time error).

The second metric, t2RMSE, computes the same difference but only for the top 2% of values with the largest absolute magnitudes in the original data. Hence, it focuses specifically on the most extreme values in the dataset and serves as a stress test for the model's ability to handle edge cases, outliers, or extreme scenarios that could be masked by traditional RMSE. It is defined as:

$$\mathrm{t2RMSE} = (\hat{\mathbf{x}}, \mathbf{x}) = \sqrt{\frac{1}{N_T N_x} \sum_{i=1}^{N_T} \sum_{i=1}^{N_x} w_i (x_{k,i} - \hat{x}_{k,i})^2} \quad \text{for} \quad x_{k,i} > Q_{0.98}$$

The third metric is the Normalized Root Mean Squared Error (NRMSE), normalized by the standard deviation. This metric provides the prediction error relative to the natural variability of the true data, making it a scale-independent metric, defined as:

$$\mathrm{NRMSE}_{\mathrm{std}} = \frac{1}{\sigma_y} \sqrt{\frac{1}{N_T N_x} \sum_{i=1}^{N_T} \sum_{i=1}^{N_x} w_i (x_{k,i} - \hat{x}_{k,i})^2}$$

where σ_y is the standard deviation of the true values.

The last metric, the coefficient of determination (R^2), represents the proportion of the variance in the dependent variable that can be explained by the independent variables. It is defined as:

$$R^2 = 1 - \frac{\sum_{k=1}^{N_T} \sum_{i=1}^{N_x} w_i (x_{k,i} - \hat{x}_{k,i})^2}{\sum_{k=1}^{N_T} \sum_{i=1}^{N_x} (x_{k,i} - \bar{x})^2}$$

where $\bar{x}$ is the average of the actual data.

3.3 Urban Case

To investigate the application areas of the FMROM framework we explore an urban case as detailed in 3.3.1. We follow the methodology described in 3.2.2, 3.2.3 and 3.2.4, but adapt it to account for the differences in how water-related simulations are used in urban cases.

3.3.1 Data collection

The urban case study focuses on water depths after a cloudburst, in a selected region of Kungsbacka, Sweden. The simulation data comes from a MIKE+ flooding model and the dataset includes 13 rainfall events and their corresponding simulation results. These events vary in intensity, ranging from approximately 14 to 240 millimeters per hour, and in duration, from 5 minutes to 6 hours. The resolution for all data is 2 minutes. In this case, we use a single state variable, water depth, which has 147,350 dimensions in physical space. We also use one forcing, precipitation rate, which is one-dimensional. Details about the data are given in Tables 3.10 and 3.11 and a figure of the mesh structure can be seen in 3.4.

States	**Description**	**Unit**
Water depth	Water height above ground level 147,350 structured mesh elements.	m

Table 3.10: Description of state data variable in the urban case

Forcings	**Description**	**Unit**
Precipitation rate	Rate of rainfall in the domain, one value for the whole domain for each timestep	mm /timestep

Table 3.11: Description of forcing data variable in the urban case

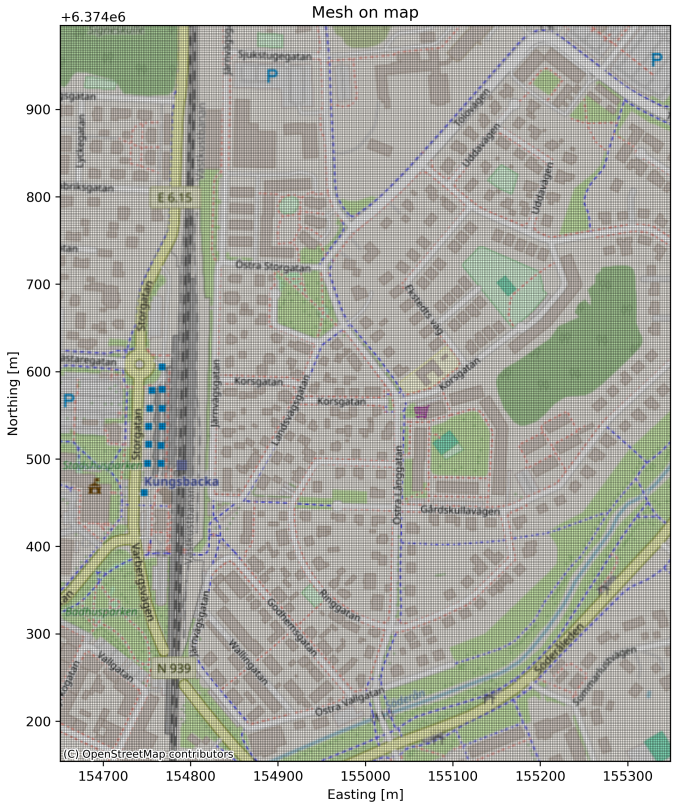


Figure 3.4: Geometry of the Kungsbacka region used for the urban case, containing mesh elements in a structured grid.

3.3.2 Dimensionality reduction and regression

As the dataset consists of individually simulated events rather than a continuous time series with varying dynamics, specific challenges for data partitioning arise. If we simply put some of the events in the training set and the rest in the test set, the results tend to show high variability in the results depending on the characteristics of the events in each subset. To address this issue and ensure a more robust evaluation, we implement 13-fold cross-validation to evaluate the performance of the framework across multiple event combinations. Since the number of rain events is relatively low, we use the leave-one-out cross-validation method. In leave-one-out cross-validation, the model is trained on all but one rainfall event and tested on the remaining one, and this process is repeated for each event. This approach allows every event to be used for both training and validation, providing a complete assessment of model performance across the entire dataset.

The results of this type of simulation are primarily used to identify risk areas where significant water accumulation may occur. Grid cells in which the water level never

exceeds a certain threshold during a rainfall event are not of practical interest. The threshold is set to 5 cm, a figure based on advice from stormwater experts and prior related work [11]. As an initial step in dimensionality reduction, feature selection is performed by setting these low-value cells to zero. The set of cells exceeding the threshold varies across different rainfall events, see Figure 3.5.

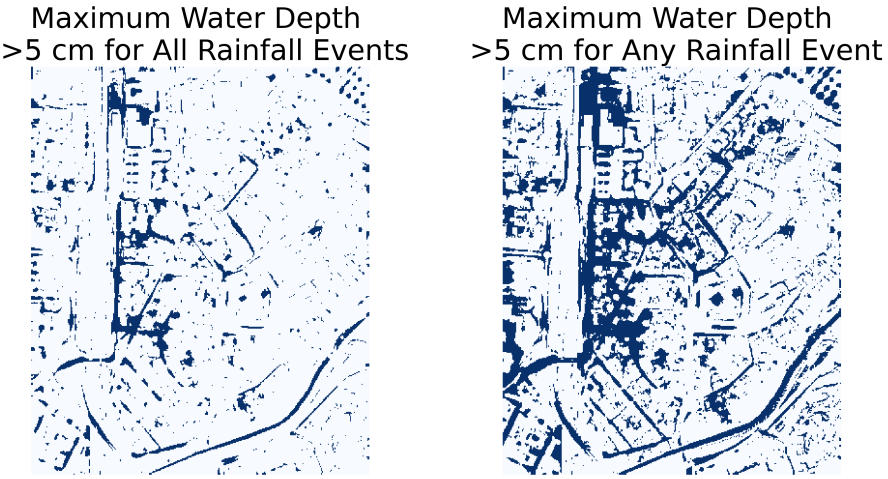


Figure 3.5: The intersection of cells exceeding the 5 cm maximum water depth threshold in all rain events (to the left) and the union of all elements exceeding 5 cm in any rainfall event, i.e. exceeding the threshold for any water event (to the right).

To further reduce the number of dimensions of the state variable, i.e. water depth, feature transformation with PCA is performed. A pipeline of a first standard scaling, PCA, and a second standard scaling is applied, and the number of dimensions is determined by the smallest number of principal components whose cumulative explained variance meets or exceeds a threshold of 99%. This threshold is chosen as a balance to retain nearly all of the original variance in the data while significantly reducing dimensionality. The forcing variable is one-dimensional and is therefore not reduced.

For the regression part, a linear regression model is implemented alongside a simple naive (dummy) model, which serves as a reference point to evaluate the performance of the linear regression. The dummy model predicts the mean value of the two most recent timesteps and operates in an autoregressive manner. This comparison helps assess whether the linear regression provides meaningful predictive improvements over a naive prediction strategy. A list of the implementations can be seen in 3.12.

Notation	Reduction	Regression	Variable
PCA-Dummy model	PCA	Naive model	Water depth
PCA-LR	PCA	Linear regression	Water depth

Table 3.12: Table of FMROM framework implementations of urban case, each with one dimensionality reduction, one regression model and a set of variables.

To fine-tune the linear regression model, a grid search is performed to find the optimal number of lags for state and forcing variables, and the combination showing the lowest RMSE is chosen. Plots for the first three splits can be seen in 3.6, and the rest can be found in Figures A.6, A.7, A.8 and A.9 in Appendix A. From the plots, it can be seen that the optimal lag value for state variables is 1 (2 minutes) and for forcing 25 (50 minutes). This is the case for all training sets except two, where it is instead 2 (4 minutes) and 30 lag values (60 minutes), respectively. The chosen lags are then tested with different forcing lead values, and based on the best results, the lead value is set to 1.

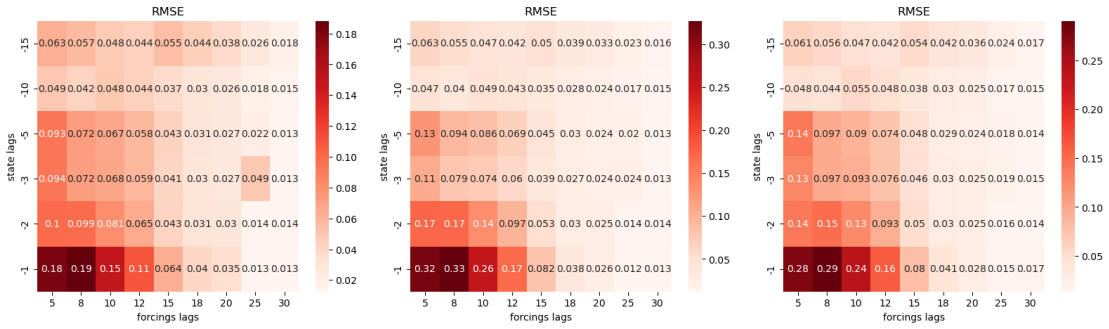


Figure 3.6: Results from grid search for the urban case using a surrogate model with PCA and linear regression, to find best combination of state and forcing lags based on RMSE. Note that the sign of the lag values is not meaningful here, both positive and negative lag values represent past observations.

3.3.3 Evaluation

For the urban case we use three of the metrics used for coastal case: RMSE, t2RMSE, R^2. NRMSE is excluded because its primary purpose is to enable comparisons across different variables, which is not necessary in the urban case, where only a single variable is predicted. As mentioned earlier, only cells exceeding a certain threshold are of practical interest. For this reason, the metrics are calculated on those elements only.

3.4 AI usage statement

The thesis is completed with the assistance of AI tools, particularly ChatGPT. It is partially used to support the development of code for the surrogate model implementation. The generated code is carefully reviewed, tested, and modified by us to ensure that the functionality aligns with the intended purpose. We employ AI tools to assist with tasks that we estimate we could complete independently, but which would have required significantly more time. In this context, AI serve as a supplement to code documentation. Importantly, this approach allowed us to maintain full understanding and control over the code.

Additionally, AI assistance is used to improve the clarity and language of the written report. No AI-generated content is used; the tool is strictly employed for refining existing formulations and enhancing the readability of the text. All final content was critically assessed and approved by us.

4

Results

4.1 Coastal case

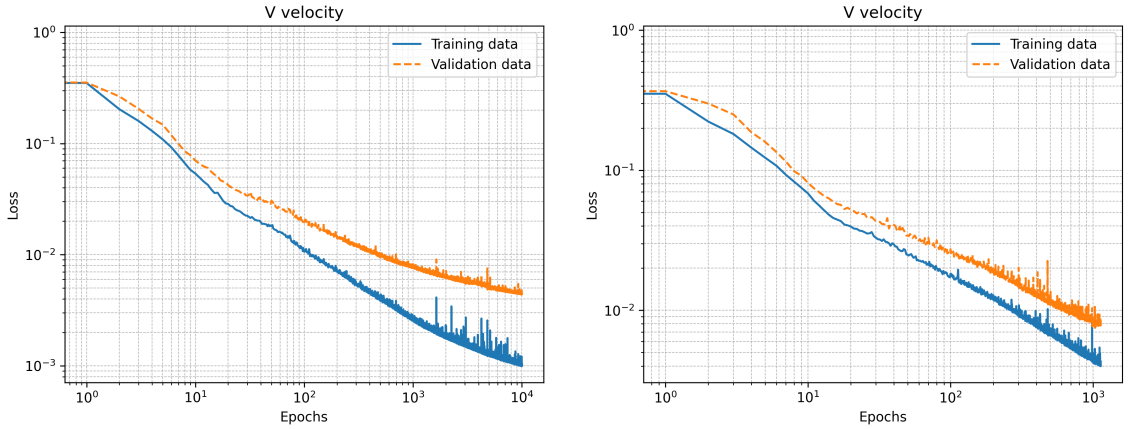
For the coastal case, we first present the results of the dimensionality reduction and reconstruction, comparing the autoencoder to PCA, which serves as the baseline dimensionality reduction method. This is followed by the results of the full surrogate model implementations, which are compared against the full baseline model (PCA-LR).

4.1.1 Dimensionality Reduction

In this section, the results of both dimensionality reduction techniques are presented side by side to enable direct comparison. The section concludes with a subsection showcasing the results of the autoencoder fine-tuned with observation data. These results are presented separately (see 4.1.1.1), as they are benchmarked against observation data rather than MIKE data.

In contrast to PCA, autoencoders require many design choices. To find a suitable autoencoder architecture, we started with a simple model and increased the complexity until the model was complex enough to overfit the data, and thereafter applied regularizing measures. The simplest autoencoder architecture performed relatively well on the surface elevation data. Combined with prior knowledge about the inherent linearity in this data, this justified not exploring more complex architectures. However, the procedure was carried out for the velocity components. Figure 4.1 shows the loss curves of the final autoencoder architecture trained on the V velocity data, both before and after applying regularization measures. The training loss curve in 4.1(a) for the unregularized network does not flatten out, which implies that the network has not fully converged. However, the gap between the training loss curve and the validation loss curve increases over the iterations as the slope of the validation loss curve seems to flatten out. This indicates that the network is

about to start to overfit. In contrast, the gap stays relatively consistent between the loss curves in 4.1(b) where the network had been regularized. This suggests that the selected regularization measures help ensure good generalization to unseen data. The U velocity showed similar, but not as clear, patterns in the training and validation loss of the networks before and after regularization (see appendix A).



(a) Unregularized (b) Regularized

Figure 4.1: Training and validation loss curves for V velocity for (a) the unregularized network and (b) the regularized network with L1-regularization and early stopping.

Table 4.1 present the reconstruction error metrics for PCA, as well as for both unregularized and regularized autoencoders applied to the standard training dataset. The performance of the surface elevation autoencoder did not improve with the applied regularization measures; therefore, the unregularized version was used for dimensionality reduction in the preceding steps. For this reason, surface elevation is also not included in the results for the regularized autoencoder (AE-reg).

A comparison of the error metrics of the velocity components from the unregularized autoencoder (AE-unreg) and the regularized autoencoder (AE-reg) reveals that the training error metrics worsen when regularization measures are taken, while the test error metrics improve. This outcome is expected as improving a model's generalization often requires sacrificing some training accuracy, ultimately leading to better performance on the test set. This also aligns with the analysis of the loss curves in earlier sections.

Focusing on the PCA results, we observe that the increase in RMSE between the test and training sets is relatively small, approximately 8%, for the velocity components.

Table 4.1: Time- and spatially averaged reconstruction error metrics. Units: Surface elevation [cm], U velocity [cm/s], and V velocity [cm/s].

		Train			Test		
Method	**State Variable**	**RMSE**	**NRMSE**	**t2RMSE**	**RMSE**	**NRMSE**	**t2RMSE**
PCA	Surface elevation	0.0190	0.0976	0.0570	0.0236	0.105	0.103
	U velocity	0.317	3.82	0.690	0.343	3.95	0.897
	V velocity	0.460	3.05	0.798	0.498	3.16	1.11
AE-unreg	Surface elevation	0.0366	0.188	0.110	0.0442	0.197	0.162
	U velocity	0.149	1.79	0.303	0.348	4.01	1.32
	V velocity	0.252	1.67	0.501	0.542	3.44	1.99
AE-reg	U velocity	0.177	2.13	0.385	0.329	3.79	1.01
	V velocity	0.347	2.30	0.600	0.493	3.13	1.19

This indicates that the PCA models generalize well to unseen data. In contrast, the PCA model for surface elevation exhibits a larger increase in RMSE of about 20% on the test set, suggesting reduced generalization performance for this variable.

Although surface elevation shows weaker generalization compared to its training performance, PCA performs better for surface elevation than for the velocity components. This is evident as all three metrics are notably lower for surface elevation than those of the velocity components. It is unlikely that this discrepancy is solely due to surface elevation being less compressed, as the later principal components contain less information, and the observed difference in performance is substantial. This supports the earlier observation that the surface elevation data exhibits greater linearity than the velocity data, making it more suitable for a linear transformation.

The autoencoder for the surface elevation did not approach the performance of the PCA, with an RMSE of 0.0442 cm compared to 0.0249 cm. It is somewhat surprising that the RMSE produced by the autoencoder is twice as large as the one from PCA, as both of them do a linear transformation of the data. PCA gives a closed-form solution, always yielding the same result to a certain dataset. In contrast, autoencoders rely on optimization during the training process to learn the optimal transformation that minimizes the reconstruction error. The fact that the autoencoder did not achieve the same reconstruction error as the PCA, suggests that the model may have converged to a suboptimal solution due to insufficient training.

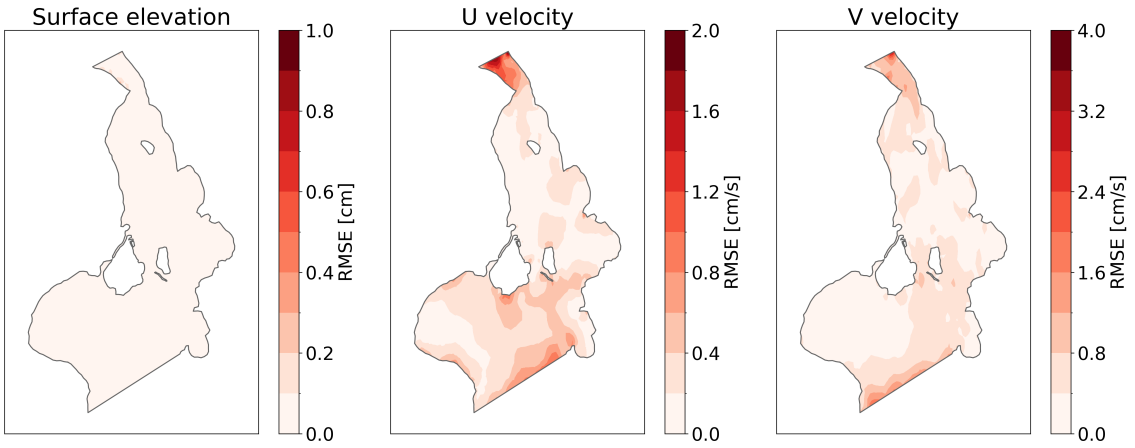
Focusing on the velocity components, the performance of the autoencoders and PCA is roughly equivalent on the test set, with the autoencoder achieving RMSEs of 0.329 cm/s for the U velocity and 0.493 cm/s for the V velocity with the autoencoder,

compared to 0.343 cm/s and 0.498 cm/s, respectively, for PCA. On the training set, however, the autoencoders perform significantly better than PCA, indicating a higher capacity for reconstruction. After applying regularization measures, the gap between the training and validation loss curves is reduced, and the training and test error metrics become more closely aligned. As previously mentioned, this indicates that the autoencoders for the velocity components achieve improved generalization. Nevertheless, the improvement is limited: the RMSE still increases substantially from training to test sets, by approximately 86% for U velocity and 42% for V velocity. These results indicate that overfitting occurs even after regularizing the networks, and that there might be even more possibilities to increase generalization.

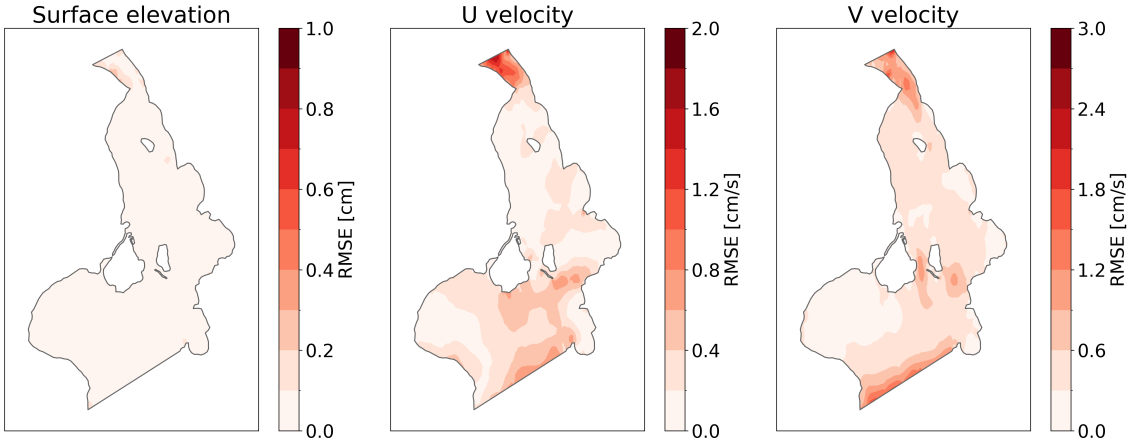
The reconstruction error increases for the top 2% of absolute values, as captured by the t2RMSE metric, across all state variables. This indicates that both PCA and autoencoders struggle with accurately reconstructing rare or high-magnitude events. PCA does not perform bad on high-magnitude values in general. If high-magnitude entries align with the directions of greatest variance, i.e. the principal components, PCA can reconstruct them accurately. However, if large values occur in regions of the original feature space that exhibit low variance in the training set, PCA will struggle to capture and reconstruct them. This effect is evident in the significantly lower t2RMSE observed on the training set, where these high-magnitude values influenced the learned components, compared to the test set, where such values may not align with the learned principal directions. Similarly, autoencoders underperform on these high-magnitude events because they are underrepresented in the training data. Since the models are trained to minimize overall mean squared error (MSE), they naturally place less emphasis on accurately reconstructing infrequent but high-magnitude cases. The lower t2RMSE observed on the training set supports this, as it reflects a better fit to the distribution seen during training.

The spatially distributed RMSE, for each state variable, are shown in Figure 4.2. The PCA and the autoencoder show similar patterns. The velocity components show the greatest reconstruction difficulty near the northern and southern boundaries. For the autoencoder, there is some additional error at the center of the domain. The spatial differences of the reconstruction error for surface elevation is so low that they cannot be discerned.

In addition to evaluating the reconstruction error metrics, one can understand the PCA by analyzing the coefficients of the principal components. The coefficients of



(a) Time-averaged reconstruction error for each state variable using PCA.



(b) Time-averaged reconstruction error for each state variable using autoencoder.

Figure 4.2: Comparison of spatial reconstruction errors across PCA (top) and autoencoder (bottom) methods.

the five first principal components, for all three state variables, are presented in Figure 4.3. The coefficients describe how each feature in the original space contributes to the principal components in the latent space. In other words, by how much weight each element in the spatial domain influences each latent dimension. Color intensity indicates the magnitude of a feature's contribution to a given principal component, while the color represents the direction of the relationship, positive or negative. The coefficients of 4.3(a) surface elevation, show some patterns in what dynamics they capture. The first component highlights areas with the highest variance, that is, the northern boundary and the center of the domain. The second principal component appears to capture north–south variability, while the third and fourth components

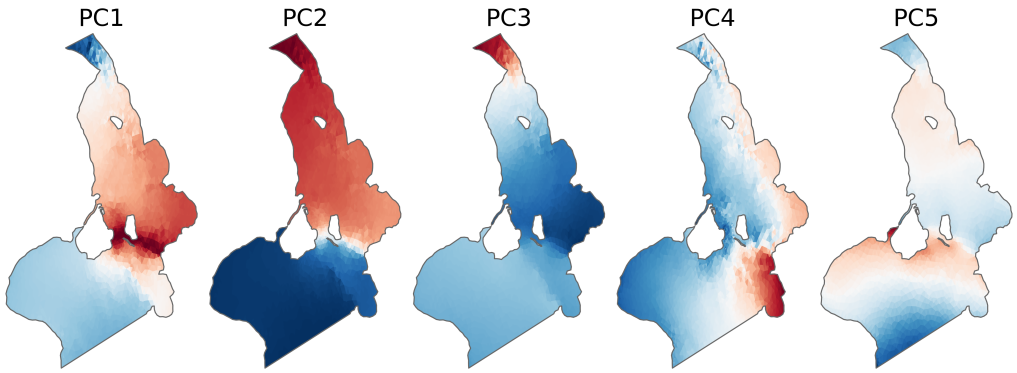
mainly reflect east–west dynamics. In contrast, the principal components of 4.3(b) U velocity and 4.3(c) V velocity, exhibit less clearly interpretable patterns. Nevertheless, the distribution of principal component coefficients for U and V velocities suggests that these state variables exhibit more nonlinear behavior, whereas surface elevation varies more smoothly across the domain.

To further understand the performance of the autoencoder, we can analyze the latent space representations. In Figure 4.4, the variance across all latent space dimensions is shown for each autoencoder. For the surface elevation latent space, a logarithmic scale was applied to the y-axis to ensure all bars were visible. The velocity components exhibit less extreme variance across latent space dimensions. This is expected, as their bottleneck layer uses a tanh activation function, which constrains latent space values to the range between -1 and 1, inherently limiting their variance. In contrast, the surface elevation autoencoder uses a linear activation function in the bottleneck, allowing for unbounded variance.

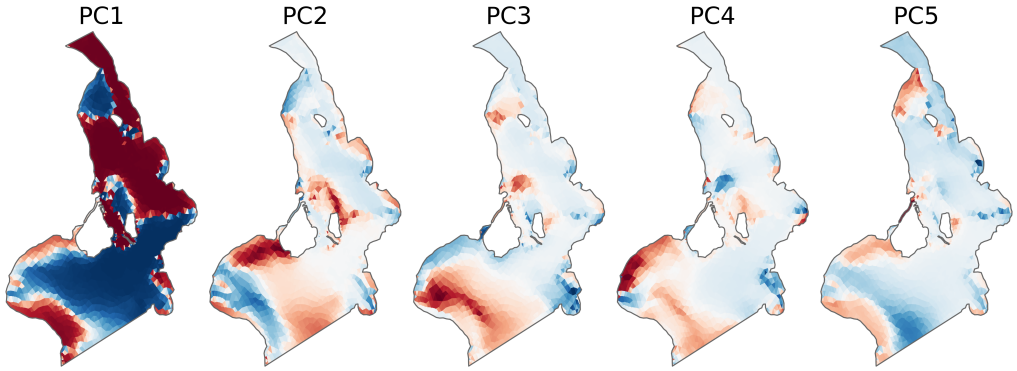
Visualizing the latent space of the autoencoder is not as intuitive as the PCA's latent space, and the components are not ordered as the components in PCA. Hence, to further analyze and visualize the latent dimensions, correlation matrices are compared to spatial perturbations. The idea of this analysis is to examine possible trends or structures of spatial influence for the latent space of the autoencoder.

In Figure 4.5, the correlation of the test data between all latent dimensions are visualized with a heatmap, for each state variable. As expected, the correlation of each latent dimension with itself is one, which appears as the red diagonal in all heatmaps. The correlation varies significantly across latent dimensions. For instance, the 10th dimension of the V velocity exhibits weak correlation with all other dimensions, whereas the first dimension shows stronger correlations with several others. Additionally, the overall correlation patterns differ markedly between state variables: the U velocity generally displays low correlation, while the surface elevation and V velocity exhibits higher correlation levels. It is somewhat surprising that the latent spaces of the autoencoders for the two velocity components exhibit such differences, despite being based on identical architectures and trained on datasets with similar characteristics.

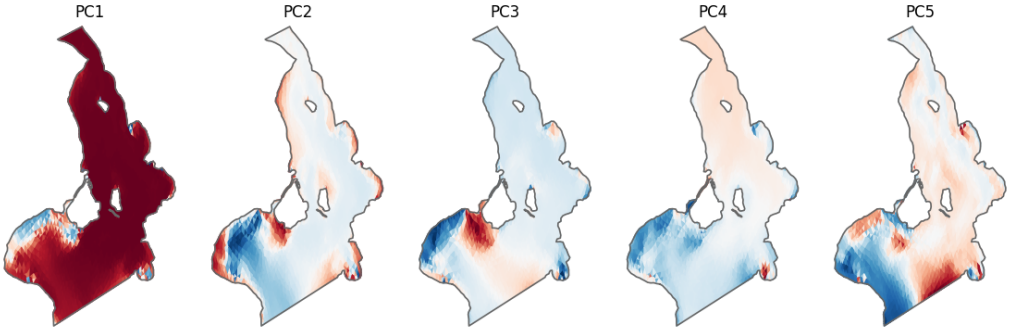
To understand the relationship between each latent space dimension and the original spatial domain, a randomly selected data sample was systematically perturbed



(a) Surface elevation



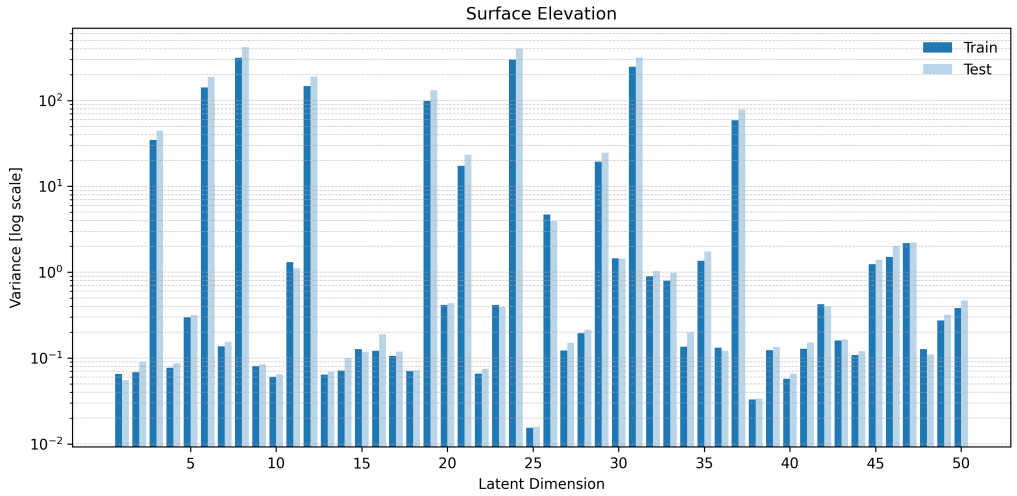
(b) U velocity



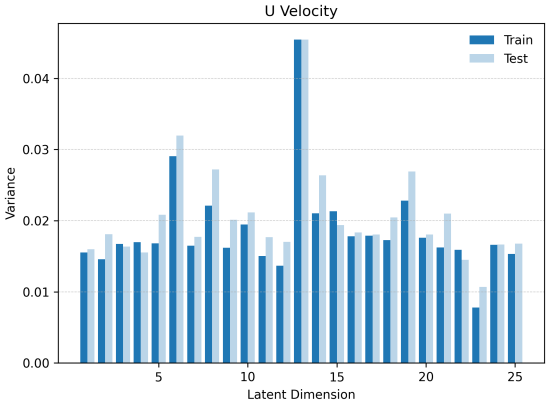
(c) V velocity

Figure 4.3: The coefficients of the five three principal components describing the state variables: (a) Surface elevation, (b) U velocity, and (c) V velocity. Red indicates a positive relationship between a cell and the principal component, while blue indicates a negative relationship. Color intensity reflects the strength of the association.

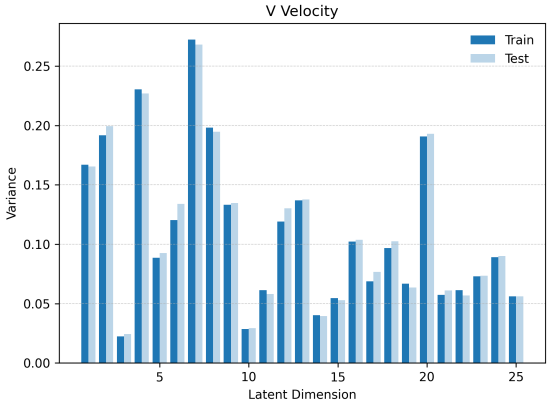
along one latent dimension at a time. The modified latent representations were then passed through the decoder, and the resulting increase in RMSE, caused by each perturbation, was calculated and visualized. These results are shown in Figure 4.6. We expected latent dimensions with stronger correlations to influence similar regions of the spatial domain, resulting in comparable spatial error patterns. However, no



(a) Surface elevation (log-scale)



(b) U velocity

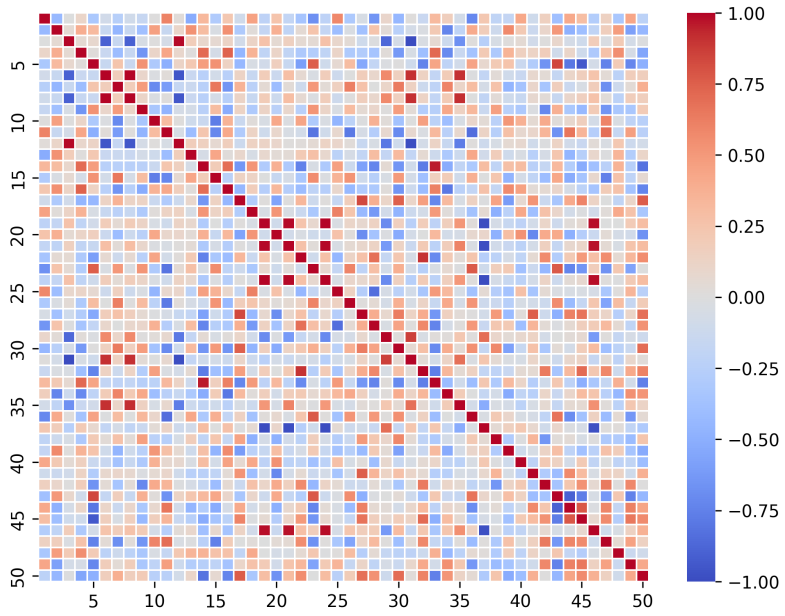


(c) V velocity

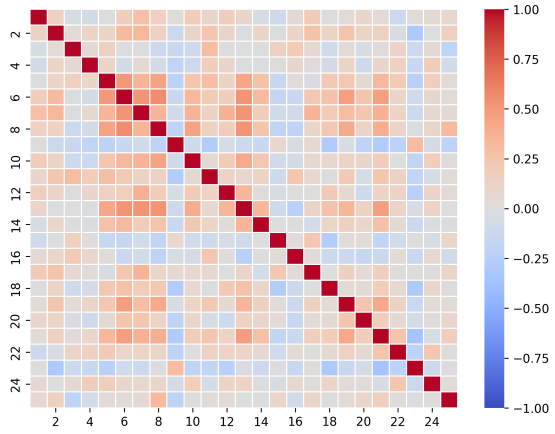
Figure 4.4: Variance of each latent dimension in the autoencoder's latent space for the state variables: (a) surface elevation, (b) U velocity, and (c) V velocity.

such relationship was observed. For example, in the case of the V velocity component, the first latent dimension shows a strong positive correlation with the second dimension and a weak negative correlation with the third. Despite this, the spatial error patterns between DIM1–DIM2 and DIM1–DIM3 are equally similar indicating no clear correspondence between correlation strength and spatial influence.

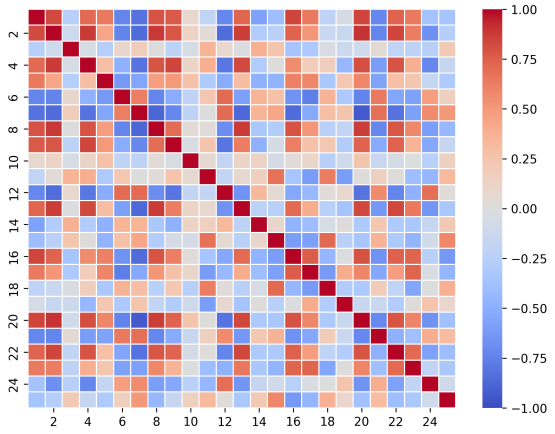
The spatial regions influenced by each latent dimension of the autoencoder are very different from the PCA components when comparing Figure 4.3 and 4.6. The visual representation of the autoencoder reveals that some regions tend to influence several dimensions, for example areas in the southwest region for the V velocity component. This analysis reveals a clear distinction between the two dimensionality reduction



(a) Correlation heatmap for surface elevation.



(b) Correlation heatmap for U velocity.

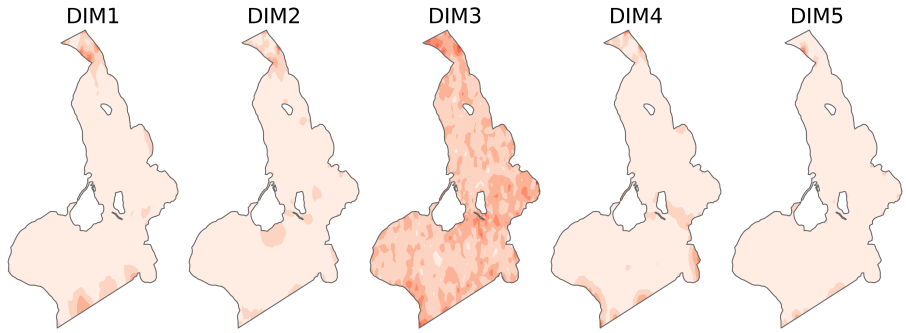


(c) Correlation heatmap for V velocity.

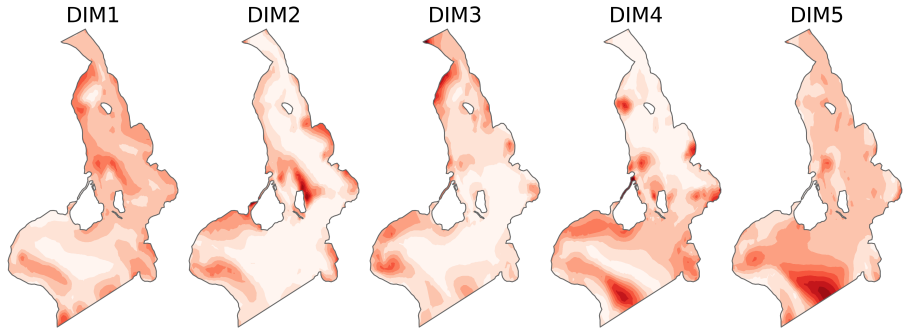
Figure 4.5: Correlation heatmaps between latent space dimensions of autoencoders: (a) surface elevation, (b) U velocity, and (c) V velocity.

techniques. The latent dimensions learned by the autoencoder are not linearly independent, whereas the latent dimensions obtained through PCA are, by definition, linearly independent.

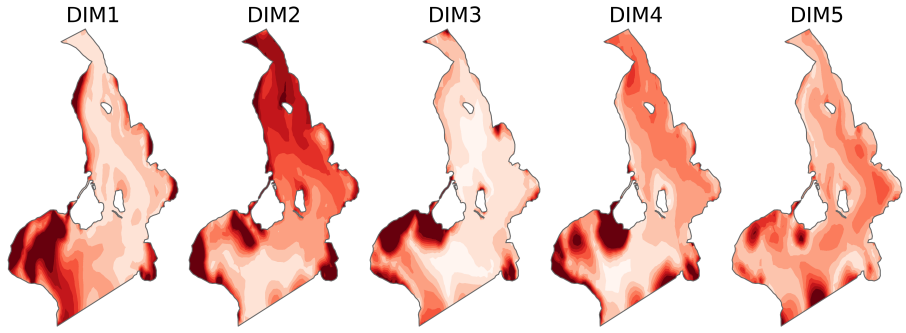
The results of the additional analysis evaluating the impact of training data size are presented in Table 4.2, which shows the reconstruction error metrics for the PCA model and the autoencoders trained on the expanded dataset. A comparison of the error metrics from the PCA fitted to the regular training set (Table 4.1) and the PCA



(a) Surface elevation.



(b) U velocity.



(c) V velocity.

Figure 4.6: RMSE between the original reconstruction of the test data and reconstructions with slight perturbations applied to individual latent space dimensions for the state variables: (a) Surface elevation, (b) U velocity, and (c) V velocity. This highlights the spatial regions influenced by each latent dimension.

fitted to the expanded training set (Table 4.2) reveals no consistent trend in how reconstruction accuracy changes when PCA models are fitted to a larger training dataset. For instance, the metrics for surface elevation improved with the expanded dataset, whereas the metrics for the V velocity component worsened. However, the difference is insignificant, implying that expanding the training dataset has little impact on the performance of the PCA models.

Table 4.2: Time- and spatially averaged reconstruction error metrics from PCA and AE (reg) trained on the expanded training set. Units: Surface elevation [cm], U velocity [cm/s], and V velocity [cm/s].

		Train			Test		
Method	**State Variable**	**RMSE**	**NRMSE**	**t2RMSE**	**RMSE**	**NRMSE**	**t2RMSE**
PCA	Surface elevation	0.0196	0.101	0.0606	0.0227	0.102	0.0977
	U velocity	0.325	3.90	0.708	0.341	3.93	0.911
	V velocity	0.484	3.21	0.861	0.503	3.20	1.17
AE-reg	Surface elevation	0.0363	0.187	0.109	0.0409	0.183	0.142
	U velocity	0.175	2.10	0.346	0.197	2.25	0.452
	V velocity	0.271	1.80	0.378	0.291	1.86	0.438

Unlike the PCA, the autoencoders benefited remarkably from the expanded training set. A comparison of the error metrics from the autoencoders fitted to the regular training set (Table 4.1) and the autoencoders fitted to the expanded training set (Table 4.2) reveals improved generalization for the velocity components. This is evidenced by an increase in training error alongside improved test performance. The error metrics for surface elevation were barely affected by the expanded training data. The performance gap between the training and test data is small across all three state variables, which may indicate that the autoencoder networks have learned to the full extent of their capacity.

4.1.1.1 Autoencoder with observations

The autoencoder fine-tuned with observational data undergoes a two-step training process. Figure 4.7 shows the loss curves for the first data split during fine-tuning, i.e. the second training step. After approximately 10 epochs, the network begins to overfit, as indicated by the upward trend in the validation loss curve. This behavior is expected, given that the fine-tuning data closely resembles the data used during pretraining. As a result, the model converges quickly but is also prone to early overfitting.

Table 4.3 presents the RMSE between the observational data and the reconstructed outputs from both the standard autoencoder and the autoencoder fine-tuned on observational data. The table is divided into two sections: Training stations and validation stations. The section for training stations reports RMSE values for elements where observational data is included during fine-tuning, while the section for validation stations shows RMSE values for elements where observational data is withheld during training. This separation enables us to assess whether the fine-

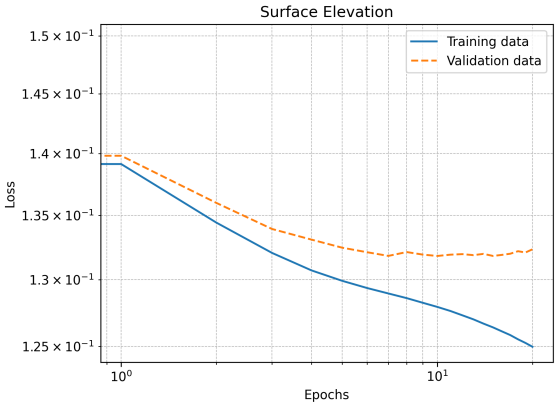


Figure 4.7: Training and validation loss curves for the first split, corresponding to using Drogden as validation station and the rest as training stations.

tuning process had a broader spatial impact.

Table 4.3: Time- and spatially averaged reconstruction error metrics from the autoencoders. Early stopping halted the training after 1189, 1950, and 1432 epochs for surface elevation, U and V velocity, respectively.

	Training stations		**Validation stations**	
Autoencoder	**RMSE**	**t2RMSE**	**RMSE**	**t2RMSE**
Standard AE	10.5	49.7	9.35	26.7
Observation-Tuned AE	9.40	47.1	9.05	26.1

All error metrics show improvement for the observation-tuned autoencoder. A similar trend is observed at the validation stations, although the improvements are minimal, only a few millimeters or less. This suggests that the fine-tuning primarily had a local effect. These patterns were consistent across all splits (see Appendix A).

Figure 4.8 presents a comparison of the MIKE data, the observation-tuned reconstruction, and the observational data in two elements: one corresponding to a training station and the other to a validation station. In elements where observational data was included during training, the observation-tuned reconstruction tends to deviate more from the MIKE data. The reconstruction tends to position itself between the MIKE and observational data, reflecting the influence of both the pretraining and the fine-tuning. However, this shift toward the observational data is not consistent across all time steps.

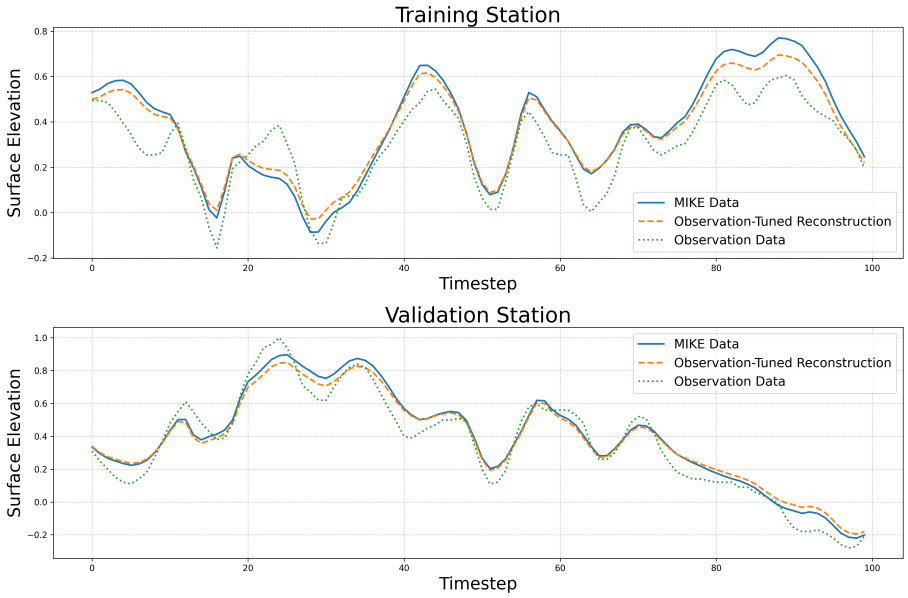


Figure 4.8: Surface elevation at a point included in Training Stations (top) and a point included in Validation Stations (bottom). The plot shows MIKE model data, observation-tuned reconstructed data, and observation data for 100 timesteps in the test set.

4.1.2 Surrogate model implementations

In this section, the results of all surrogate model implementations are presented. The table of the implementations from the methodology chapter is shown again in Table 4.4, to repeat the notations of the implementations, each combining one reduction technique with one regression model, for a given set of variables.

Notation	**Reduction**	**Regression**	**Variable**
PCA-LR (Baseline)	PCA	Linear regression	Surface el., U and V vel.
PCA-LSTM	PCA	LSTM	Surface el., U and V vel.
PCA-LR_{SE}	PCA	Linear regression	Surface el.
PCA-$LSTM_{U,V}$	PCA	LSTM	U and V vel.
AE-LR	AE	Linear regression	Surface el., U and V vel.
AE-LSTM	AE	LSTM	Surface el., U and V vel.
AE-$LSTM_{U,V}$	AE	LSTM	U and V vel.

Table 4.4: Table of surrogate models based on the FMROM framework, each with one dimensionality reduction and one regression model for a set of variables.

The results of the evaluation metrics for all implementations are found in Table 4.5. The baseline (PCA-LR) show strong performance across all variables. Focusing on the test error metrics it can be observed that R^2 is very high (> 0.96) and the RMSE and NRMSE values are low. It serves as a solid benchmark, and many models struggle to beat it.

The convergence behavior of the AE-LSTM implementation can be evaluated by examining the training and validation loss over each epoch, as shown in Figure 4.9, where the loss curves are plotted on a logarithmic scale. The training loss steadily decreases until it begins to converge, at which point early stopping halts the training process. This trend indicates that the model is effectively learning from the training data. The validation loss follows a similar downward trajectory but remains consistently higher than the training loss, suggesting a degree of overfitting. This is a common outcome, reflecting the model's closer fit to the training data relative to its generalization to unseen validation data. Nonetheless, the overall decrease in both losses reflects the model's learning capacity and performance. The loss curves for the remaining implementations follow a similar pattern (see Appendix A).

For the more complex models, PCA-LSTM and AE-LSTM, there are significant dif-

Table 4.5: Time- and spatially averaged error metrics from implementations listed in table 3.9. The implementation noted PCA-LR is referred to as the baseline. Units: Surface elevation [cm], U velocity and V velocity [cm/s]. The worst and best performing models are highlighted in red and green, respectively. The baseline is highlighted in gray.

		Train				Test			
Notation	**State variable**	**RMSE**	**NRMSE**	**t2RMSE**	**R^2**	**RMSE**	**NRMSE**	**t2RMSE**	**R^2**
PCA-LR	Surface elevation	1.15	5.90	2.21	0.9963	1.47	6.57	2.97	0.9954
(Baseline)	U velocity	1.35	16.20	4.08	0.9743	1.66	19.12	5.78	0.9645
	V velocity	2.14	14.18	6.85	0.9771	2.64	16.75	9.72	0.9684
PCA-LSTM	Surf. Elevation	3.38	17.36	7.81	0.9702	6.46	28.82	25.16	0.9214
	U velocity	1.51	18.11	3.28	0.9694	2.30	26.55	6.87	0.9352
	V velocity	2.41	15.95	5.24	0.9725	3.63	23.10	11.44	0.9425
$\text{PCA-LSTM}_{\text{U,V}}$	U velocity	0.87	10.44	1.71	0.9898	1.74	20.02	4.34	0.9637
	V velocity	1.38	9.16	12.53	0.9912	2.77	17.56	7.22	0.9680
AE-LR	Surf. Elevation	1.53	7.79	3.05	0.9935	1.76	7.88	3.58	0.9933
	U velocity	1.51	17.93	3.95	0.9701	1.59	18.27	4.15	0.9687
	V velocity	2.06	13.54	5.32	0.9803	2.25	14.32	5.78	0.9783
AE-LSTM	Surf. Elevation	0.10	5.10	1.57	0.9974	3.31	14.76	16.33	0.9796
	U velocity	0.73	8.84	1.02	0.9932	1.30	14.92	3.48	0.9799
	V velocity	1.15	7.61	1.52	0.9942	2.02	12.81	6.17	0.9830
$\text{PCA-LR}_{\text{SE}}$	Surf. Elevation	1.15	5.90	2.21	0.9963	1.47	6.57	2.97	0.9954
$\text{AE-LSTM}_{\text{U,V}}$	U velocity	0.57	6.85	1.05	0.9956	1.40	16.18	4.13	0.9757
	V velocity	0.93	6.18	1.61	0.9960	2.27	14.41	6.90	0.9779

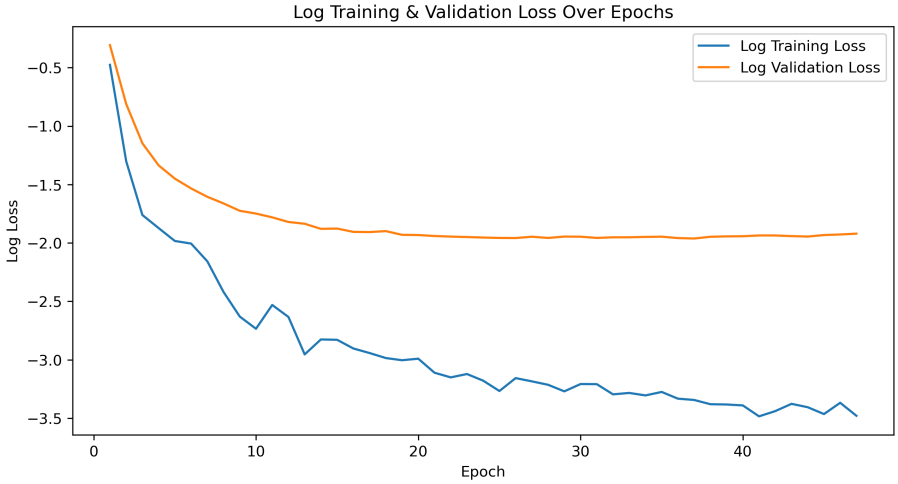


Figure 4.9: Log loss plot for FMROM implementation with autoencoder and LSTM regression, for surface elevation, U and V velocity.

ferences between train and test performance, with good train metrics and poorer test metrics. For example, the AE-LSTM implementation for surface elevation shows a notable increase in RMSE from 0.10 cm on the training set to 3.31 cm on the test set. This indicates that overfitting is likely, and that the models struggles to generalize. This is somewhat expected as the LSTM model has, due to its complexity, larger capacity to learn not only meaningful patterns in the data but also noise, which can hinder its ability to generalize.

Even though AE-LSTM show tendencies of overfitting, all test scores are better than the baseline model (PCA-LR) for both U and V velocities. It has test RMSE of 1.30 and 2.02 compared to 1.66 and 2.64 cm respectively. This indicates that AE-LSTM has strong modeling capacity and show potential to perform even better with improved regularization.

LSTMs use nonlinear activation functions, sigmoid and tanh, in their gates, allowing them to learn complex, nonlinear patterns in the data. Linear regression relies on a linear mapping between inputs and outputs, which can limit its ability to model intricate relationships. Hence, LSTMs often show better performance for more complex, nonlinear regression tasks. The improvement for U and V velocities aligns with the previous knowledge of the nonlinearity in the data which is better captured by the LSTM model. The decrease in performance of AE-LSTM for surface elevation could be explained by the linearity in the data, which likely reflects the simple dynamics of the Øresund region. However, the corresponding training performance is very high, indicating that it could be possible to achieve better test metrics with better regularization, but that the model is not optimized to generalize well for this variable.

Given the differing characteristics of the variables, the velocity components were also predicted separately from surface elevation, with the models denoted as PCA-LSTM$_{\mathrm{U,V}}$ and AE-LSTM$_{\mathrm{U,V}}$. This separation led to improved performance for PCA-LSTM$_{\mathrm{U,V}}$ relative to the joint PCA-LSTM model, which predicts all three variables. However, AE-LSTM$_{\mathrm{U,V}}$ exhibit reduced performance compared to AE-LSTM. This is somewhat surprising since the significant increases in errors for surface elevation in both cases indicate unoptimized learning. Potentially, the optimization problem is due to conflicting gradients, which can arise when changes in one predictor variable lead to unfavorable outcomes in another. Hence, simultaneous changes in predictor variables may drive the model's estimates in opposite directions.

We can see that the poorest performance is for PCA-LSTM on all three variables, highlighted in red. One explanation is that the grid search is performed only on AE-LSTM$_{\mathrm{U,V}}$, leading to a parameter setup that might not be not suitable for the PCA-LSTM implementation. These results further justify the design choice of separating the variables, but also emphasize the importance of proper parameter-tuning.

The AE-LR perform slightly poorer for surface elevation compared to the baseline. Although, it still shows the best balance of performance and generalization of all three variables; t2RMSE and NRMSE remain nearly identical between training and test, and R^2 is consistently high. This is expected as the linear regression model is not prone to overfitting.

Further, as shown in Section 4.1.1, the absolute improvement in reconstruction error when using an autoencoder instead of PCA for the velocity components is 0.014 cm for the U velocity and 0.005 cm for the V velocity. However, when combined with linear regression (in the implementation denoted AE-LR), the total improvement over PCA-LR increases to 0.07 cm and 0.39 cm, respectively. Although the only difference between these two implementations lies in the dimensionality reduction method, the resulting performance gap is significantly larger than the difference in reconstruction error alone. This suggests that the autoencoder not only achieves slightly better reconstruction, but also produces a latent space that is more amenable to learning by the linear regression model. This suggests that the autoencoder successfully creates a latent space that appears more linear by "untangling" or simplifying the underlying nonlinearities.

AE-$LSTM_{U,V}$ shows the best overall performance for velocities, highlighted in green. The test performance is better than the baseline for both U and V velocities and high R^2 and low NRMSE indicate the strongest modeling of the velocities. This suggests that even though the autoencoder creates a latent space representing the nonlinearities well, there are still dynamics that cannot be captured for the velocities when using a linear regression model, and hence, the performance improves when using a more complex model instead.

The LSTM model algorithm handles the historical influence in a less interpretable way compared to the linear regression, where lags and leads are used in a more straight forward way. The LSTM gates and candidates operate through complex, nonlinear transformations that conceal direct relationships between input variables and the final output. As a result, it becomes difficult to pinpoint which specific parts of the input history are most influential in determining the model's predictions, but to visualize the feature influence, one can plot the L2-norms of the input weights of each gate in the LSTM network. These are shown in Figure 4.10. The first 100 input features indices on the x-axis correspond to the state variables and the last 41 to the forcing variables, and it can be concluded that the influence from the

forcings is highest. While the L2 norm varies between 0 and 8 for the forcings, they stay consistent around 1 for all states. The clear distinction between the state and the forcings aligns with the analysis showing that no state history (i.e., state lags) is useful when using linear regression, and again confirms the knowledge that the Øresund domain is strongly driven by the forcings.

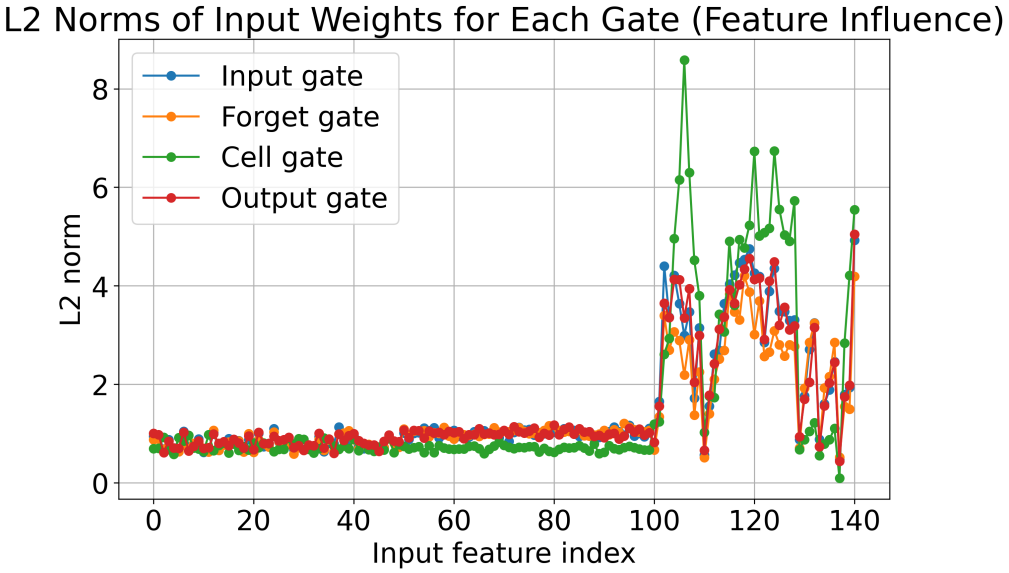


Figure 4.10: L2 norms of input weights for each gate in the LSTM network for AE-LSTM implementation.

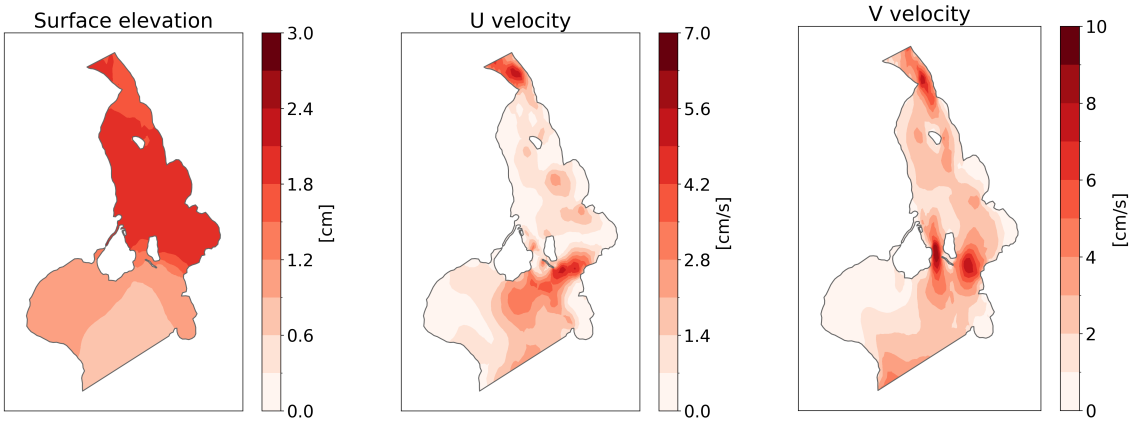
Looking at Table 4.5 again, it can be seen that the t2RMSE values are higher than the RMSE for all implementations. It is likely that the increased error is both inherited from the dimensionality reduction and arises in the regression part since these extreme events are (by definition) underrepresented in the training data. Hence, the increase in t2RMSE aligns with the analysis from the dimensionality reduction on its own, namely, that since the models are trained to minimize overall mean squared error (MSE), they naturally place less emphasis on accurately predicting infrequent but high-magnitude cases. The models with the lowest test t2RMSE are, except for the baseline, AE-LR and PCA-LR_{SE} for surface elevation with AE-$LSTM_{U,V}$ for the velocities (i.e., putting together two different model setups to predict the three variables). As mentioned above, these models also perform well according to other evaluation metrics, which confirm their overall robustness. The top 2% of the values in terms of magnitude often correspond to 'rare events', and accurately capturing these extremes is important for many real-world applications.

The normalized RMSE (NRMSE) metric evaluates model performance while accounting for the variance of each variable. Hence, it provides a scale-independent

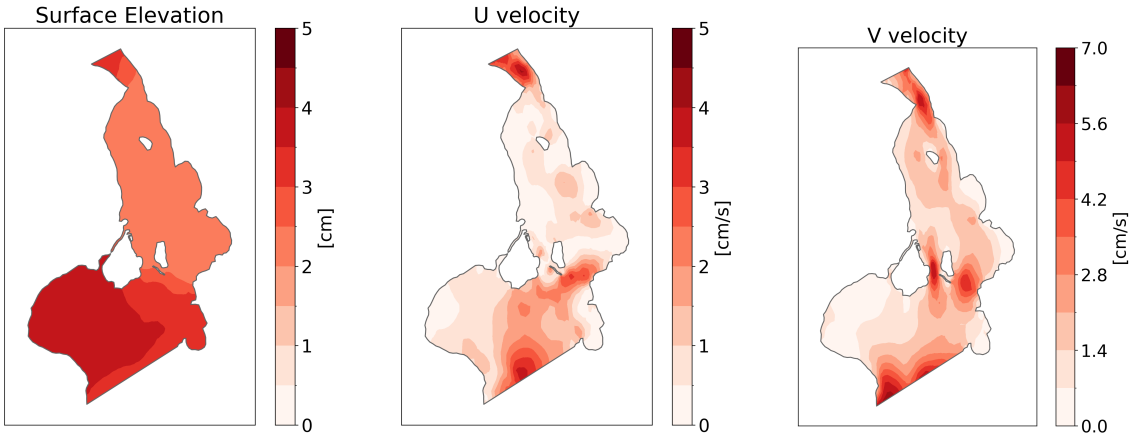
measure of prediction error. Across all implementations, the variable with highest RMSE and lowest R^2 is V velocity, indicating weaker predictive accuracy compared to other variables. However, despite poorer performance in absolute terms, the NRMSE is consistently lower for V velocity than for U velocity. This suggests that V velocity is more difficult to predict, as even small absolute errors represent a larger fraction of the variance in U velocity. Therefore, the models may be capturing a greater proportion of the explainable variance in the V velocity, despite its higher absolute error.

Next, we examine the spatial distribution of RMSE, as shown in Figure 4.11. Specifically, we compare the baseline implementation (PCA-LR) with two other selected models: AE-LSTM and AE-LR. The RMSE distribution for the velocity components follows similar patterns across all implementations. The largest errors are at the narrow parts located in the center of the domain, and close to the southern and northern boundaries. This is consistent with the spatially distributed RMSE obtained from the dimensionality reduction. These areas are elements with the highest variance and is expected that they are more difficult to predict. However, compared to the errors from the autoencoder dimensionality reduction (Figure 4.2), the error in the center of the domain appears to have increased more notably compared to the errors near the boundaries. One possible explanation is that surface elevation data near the boundaries are used as input (forcing) in the regression. Since surface elevation is at least partially correlated with velocity, it helps stabilize predictions near the boundaries more effectively than in the central region, potentially explaining the greater error increase observed in the center. For surface elevation, the spatial error is distributed similarly for AE-LR as for the baseline, but shifted in the AE-LSTM implementation where the error is larger in the southern parts. This suggests that the spatial distribution of the error varies more with the choice of regression model than with the dimensionality reduction technique.

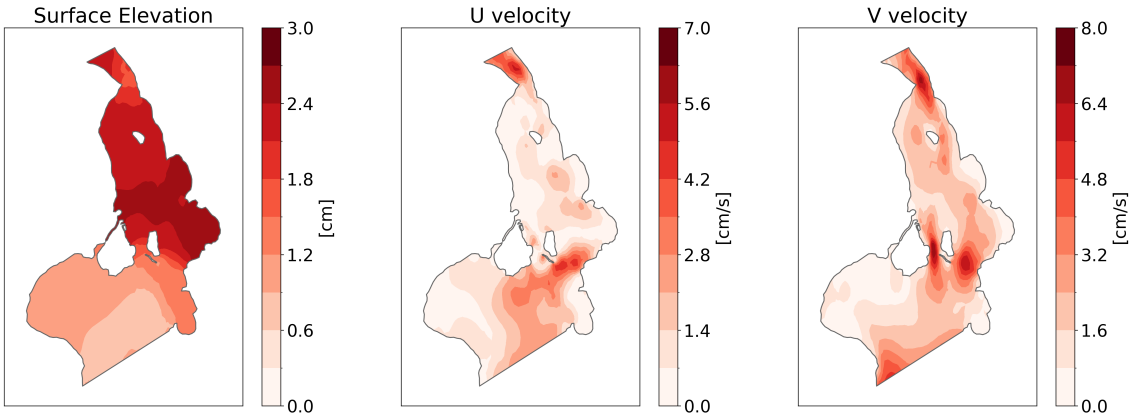
The AE-LSTM implementation aims to better capture the system's nonlinear dynamics. Figure 4.12 presents a scatter plot comparing the predicted values from three surrogate model implementations to the true outputs from the MIKE model at a selected domain point. For surface elevation using the PCA-LR implementation, the predictions align closely with the red line representing perfect agreement, indicating high accuracy. However, for the U and V velocity components, a subtle S-shaped deviation is observed, suggesting systematic nonlinear discrepancies between predicted and true values. This S-pattern is a common sign of model bias



(a) PCA-LR



(b) AE-LSTM



(c) AE-LR

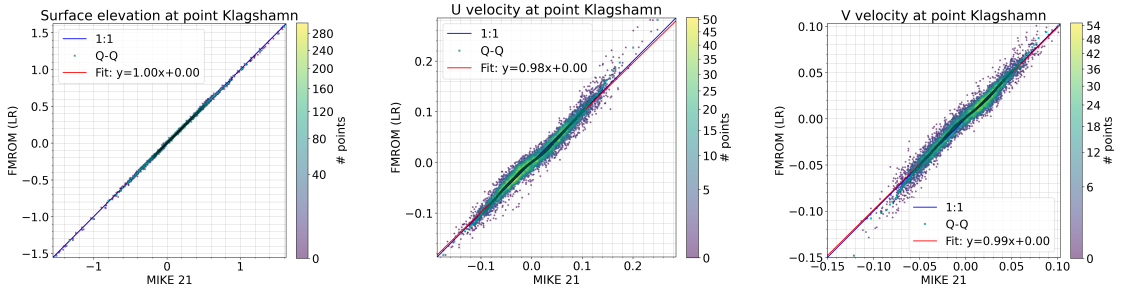
Figure 4.11: Spatial RMSE of all the state variables for three FMROM implementations: (a) PCA-LR, (b) AE-LSTM, and (c) AE-LR. Each row shows results for Surface elevation, U velocity, and V velocity, from left to right.

arising when a model is too simple to capture underlying nonlinear relationships. Typically, such patterns reflect systematic underprediction in some regions and overprediction in others, suggesting the model is missing key nonlinear behavior. This limitation is evident in PCA-LR. In contrast, the AE-LSTM implementation shows no S-shaped pattern, suggesting that the autoencoder and LSTM approach is more effective at modeling the system's nonlinear dynamics. Furthermore, the scatter plot for AE-LR neither shows a S-shape which again indicates that the latent space of the autoencoder represents the nonlinearities well and makes the regression more robust in terms of systematic under- and overprediction.

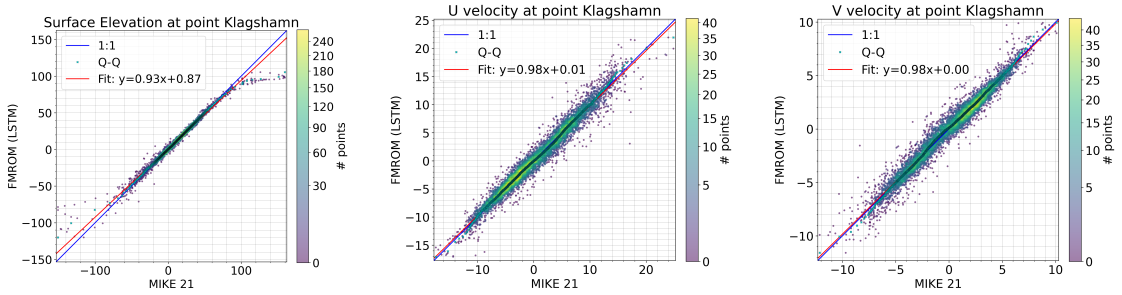
In summary, the two implementations showing the best performance in all three variables are the baseline (PCA-LR) and the PCA-LR_{SE} with AE-$LSTM_{U,V}$. PCA-LR outperforms all other models for surface elevation. For U and V velocities, AE-LSTM improves by 22% and 23% respectively, compared to the baseline. When separating surface elevation from the velocities, AE-$LSTM_{U,V}$, the improvement is instead 16% and 14% respectively. PCA-LR_{SE} shows no change in performance when separated from the velocities. This indicates that the correlation between the variables is utilized differently for the velocities than for surface elevation. The best result is obtained by either using PCA-LR_{SE} with AE-$LSTM_{U,V}$, or PCA-LR_{SE} with AE-LSTM (where surface elevation is included) but only consider the velocity components. It is not ideal from a computational cost perspective to perform several predictions simultaneously for a variable and not use the output, but from a accuracy perspective it could be the most beneficial.

In Figure 4.13(a) we can see the spatially averaged RMSE of the test set, distributed over time. It shows that the spread of outlier predictions is similar for the two implementations, with a few more outliers for V velocity in the PCA-LR_{SE} with AE-$LSTM_{U,V}$. Figure 4.13(b) contains the same data, but with the y-axis cut off to show the boxes more detailed. It shows that the box (the middle 50% of the predictions) is more compact for U and V velocities in PCA-LR_{SE} with AE-$LSTM_{U,V}$, in other words more tightly clustered around the median, indicating less variability or inconsistency in those predictions. It can also be seen that the spread of predictions (represented by the boxes) is slightly lower for U and V velocities.

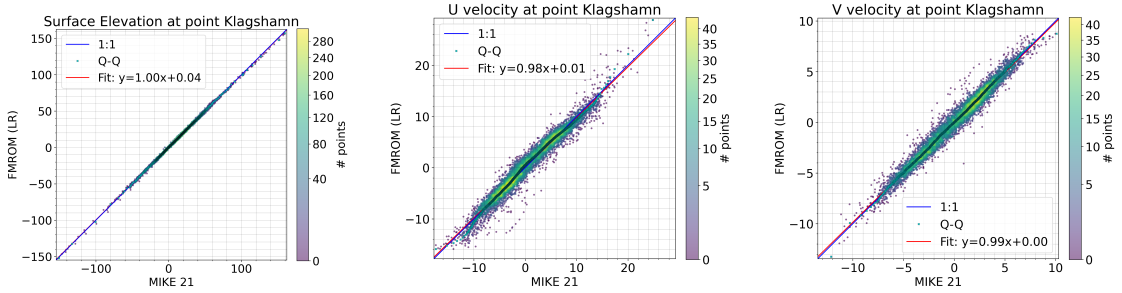
Figure 4.14 shows the spatially averaged RMSE over time, as a time series, illustrating how the error evolves for the best performing implementation of FMROM. Although the predictions are performed autoregressively, the errors remain within



(a) PCA-LR



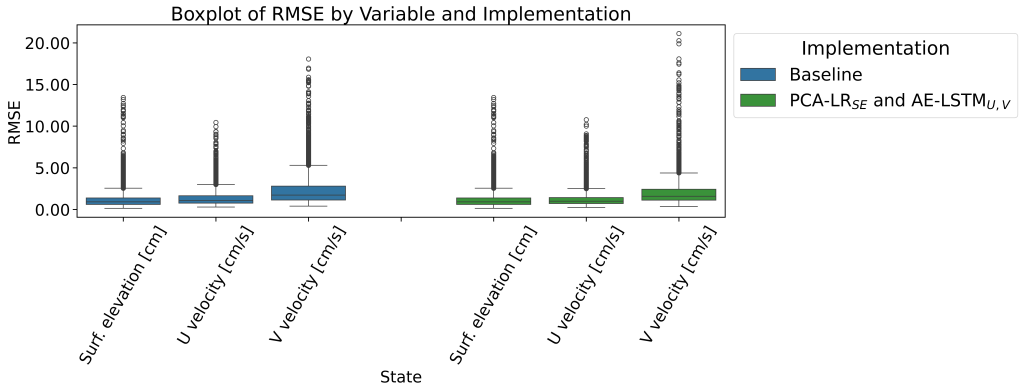
(b) AE-LSTM



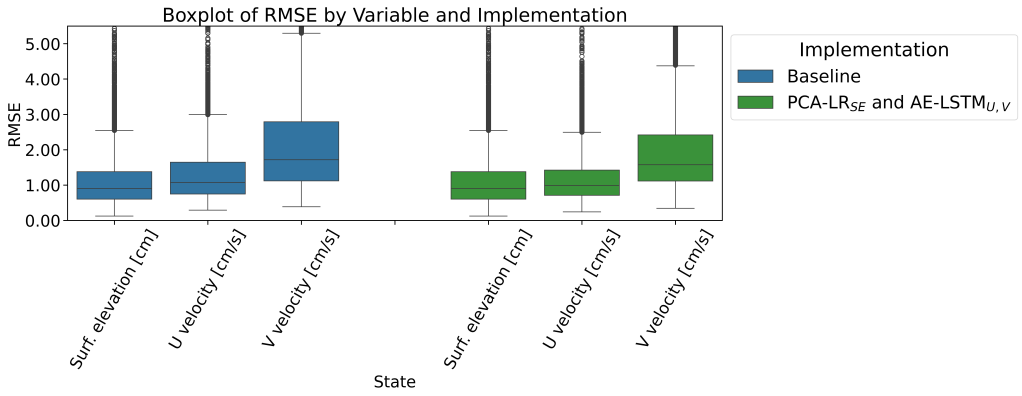
(c) AE-LR

Figure 4.12: FMROM result vs MIKE model for three FMROM implementations: (a) PCA-LR, (b) AE-LSTM, and (c) AE-LR. The red line shows a perfect one-to-one relationship. The blue line shows the fitted linear function $y = kx + m$. Each row shows results for Surface elevation, U velocity, and V velocity, from left to right.

a bounded range over time. This can be explained by the fact that the regression model incorporated boundary conditions as forcings, providing information that helps bounding the predictions to a realistic range.



(a) Boxplot with full y axis visible.



(b) Boxplot with y axis cut off.

Figure 4.13: Boxplot of spatially averaged RMSE of the test set, distribution over time. The box represents the middle 50% of data (Q1 to Q3), the middle line is the median (Q2) and the whiskers represent the spread of data excluding outliers. Lastly, the dots are the outliers (data points beyond 1.5×IQR from the box). In (b), the y-axis is cut off to 5.5 cm, leaving some of the outliers out to provide a better comparison between the models.

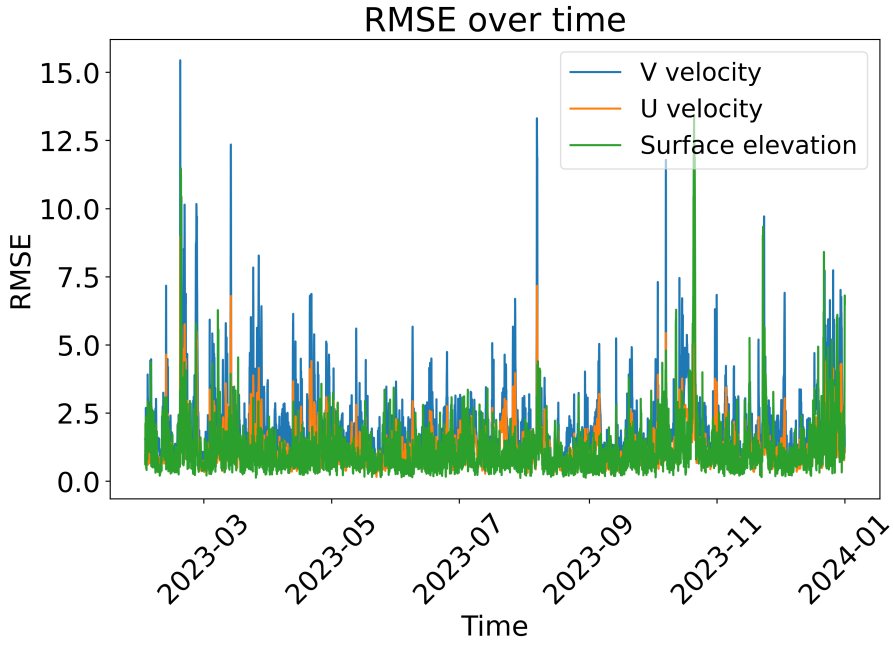


Figure 4.14: Temporal RMSE of all three state variables: surface elevation, U and V velocity for the best performing implementation of FMROM, i.e. PCA-LR_{SE} and AE-$LSTM_{U,V}$.

4.2 Urban Case

For the urban case, we begin by presenting the results of the dimensionality reduction and reconstruction of the data. Further, results from the surrogate model implementations are presented. Two implementations are compared, PCA with a naive dummy model (noted PCA-Dummy model) that serves as the baseline in this case, and PCA with linear regression (noted PCA-LR).

In Table 4.6, the reconstruction error metrics from PCA are presented. The table includes an average across all train and test splits, the best split, and the worst split. The metrics are calculated only for cells with a maximum water depth greater than 5 cm for a given rainfall event. The average test RMSE across all splits is 1.60 cm. Given that the average water depth is approximately 11.2 cm, the error may be considered relatively low. However, the PCA struggles to capture more high-magnitude events, as the average root mean squared error for the top 2% of values (t2RMSE) increases to 7.14 cm.

Table 4.6: Time- and spatially averaged reconstruction error metrics from the PCA. Unit: Water depth [cm].

	Train		**Test**	
Scope	**RMSE**	**t2RMSE**	**RMSE**	**t2RMSE**
All splits	0.626	1.95	1.60	7.14
Best split	0.638	2.01	0.830	2.80
Worst split	0.612	1.85	4.31	24.4

The test error metrics vary significantly between the best and worst splits, highlighting the sensitivity of our model to the specific train-test split used. Moreover, the standard deviation of RMSE values for all splits increases notably from 0.0159 cm in the training sets to 0.940 cm in the test sets. This further underscores the variability introduced by different data splits and emphasizes the importance of robust validation strategies.

The average error metrics increase notably from the training set to the test set. This trend is more pronounced in the worst-performing data split, while it appears only mildly in the best-performing split. One possible explanation is that some rain events, when excluded as test sets, exhibit high variability not present in the training data. Consequently, the PCA model struggles to reconstruct these test

sets accurately due to the absence of similar patterns during training. In contrast, splits where the test data variability is well represented in the training set, result in relatively strong performance across all error metrics for both training and test sets. These differences suggest that the model would benefit from a larger and more diverse training dataset.

In Figure 4.15, the coefficients of the first three principal components are visualized. Specifically, they describe the extent to which each original feature, in this case, each grid cell, influences the corresponding principal component. By comparing the coefficients with the masked-out cells, shown in white in Figure 3.5, it is evident that the masking achieved the intended effect. Setting these cells to zero eliminated their variance, which in turn prevent them from being captured by the principal components.

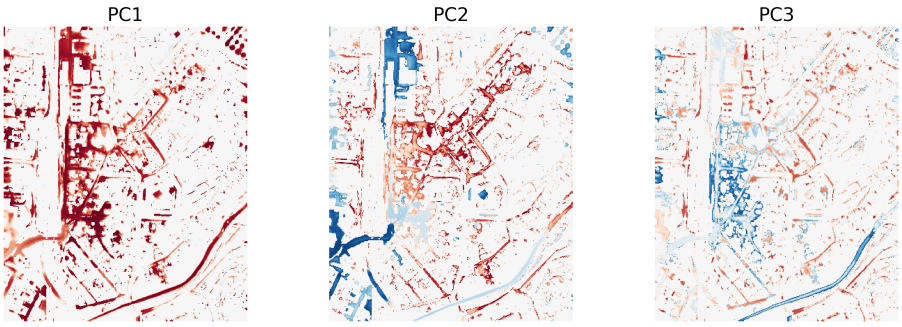


Figure 4.15: Coefficients of the principal components describing water depth for split 5 (the worst-performing split). Red indicates a positive relationship between a cell and the principal component, while blue indicates a negative relationship. Color intensity reflects the strength of the association.

The evaluation metrics for the surrogate model implementations for the urban case are presented in Table 4.7, where PCA-LR is compared to a naive dummy model. The metrics are presented as an average across all splits, the best, and the worst split. The PCA-LR model outperforms the PCA-Dummy model across all error metrics, indicating that the model has learned meaningful patterns from the data.

The differences in R^2 values between the best and worst splits obtained by the dummy model highlight the varying difficulty levels of the test sets. The test set in the best split appears to be simple, allowing even the dummy model to achieve reasonable performance with an R^2 value of 0.883. The simplicity of that specific test set is further underscored by the fact that its test metrics of PCA-LR surpass the model's performance on the training data.

Table 4.7: Time- and spatially averaged error metrics from FMROM implementations for the urban case. Unit: Water depth [cm].

Notation	Scope	Train			Test		
		RMSE	**t2RMSE**	$\mathbf{R^2}$	**RMSE**	**t2RMSE**	$\mathbf{R^2}$
PCA-Dummy model	All Splits	20.9	128	0.0796	18.7	115	0.431
	Best split	21.8	134	0.0613	8.34	50.5	0.883
	Worst split	20.1	123	0.138	30.4	187	-0.131
PCA-LR	All Splits	1.35	6.46	0.996	2.59	12.1	0.987
	Best split	1.39	6.55	0.996	1.02	4.11	0.998
	Worst split	1.41	6.82	0.995	7.10	38.3	0.938

Focusing on the average performance of the PCA-LR implementation, it demonstrates reasonable results with an RMSE of 2.59 cm. However, similar to the dimensionality reduction step, the model struggled to accurately predict the top 2% of values, with a t2RMSE value of 12.1 cm. A possible reason for this behavior is the absence of boundary conditions or external forcings to constrain the regression outputs. Without such constraints, the model's predictions may deviate significantly in certain regions, especially when encountering unfamiliar or extreme input conditions.

In Figure 4.16, the spatial RMSE is plotted onto the map of the selected region. In the worst-performing split, the largest errors are observed along the river, visible as a long line in the lower right corner, and in a viaduct, which appears as a shorter, thicker line in the left corner. Additionally, higher errors are evident in densely built-up areas with many houses. Notably, the regions with the highest errors correspond to areas with greater water depth.

Figure 4.17 shows the spatially averaged RMSE over time, illustrating how the error evolves for both the best and worst splits in the cross-validation. Although the error magnitudes differ significantly between the two cases, both show a clear increase over time. This can be explained by the use of autoregressive prediction combined with the absence of boundary conditions as forcings in the regression model. Without these constraints, the predicted values are not bounded, allowing the predictions to drift over time and causing the error rate to increase.

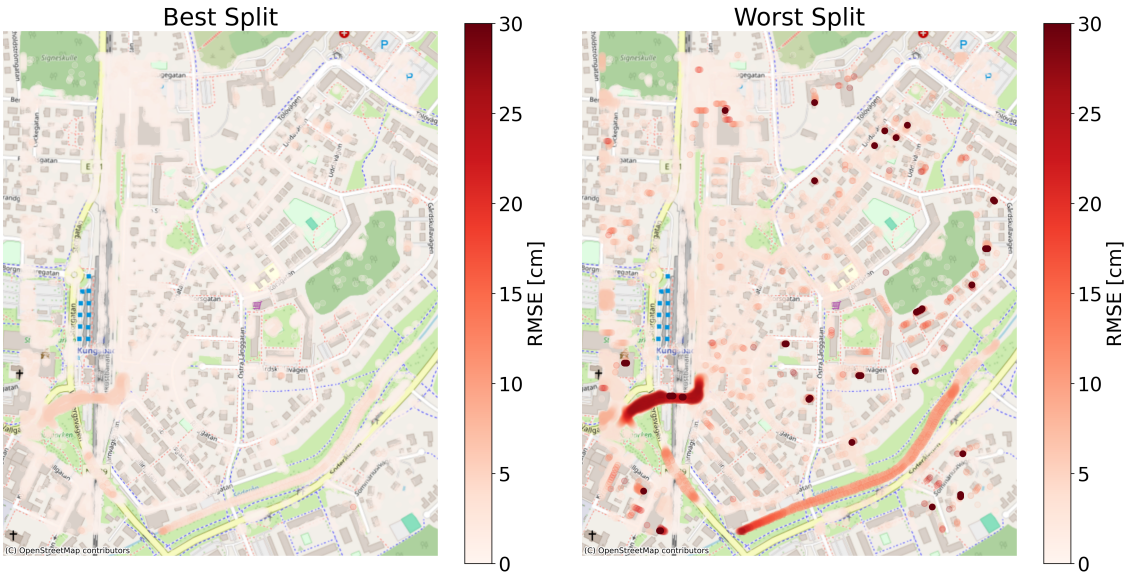


Figure 4.16: Spatial RMSE of the state variable water depth with an FMROM implementation using PCA and linear regression (PCA-LR).

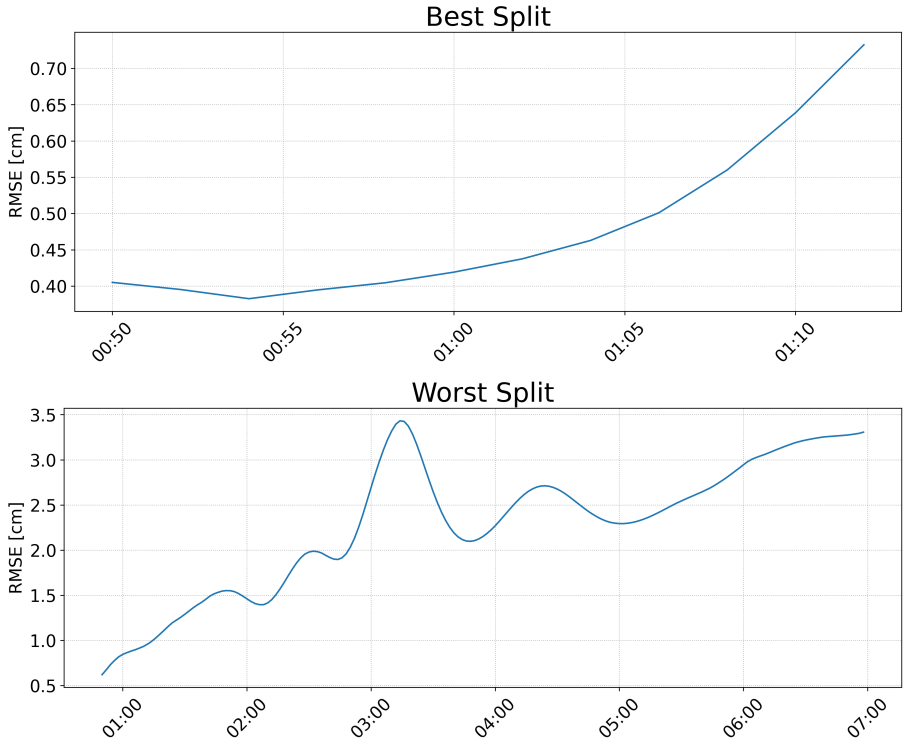


Figure 4.17: Temporal RMSE of the state variable water depth with an FMROM implementation using PCA and linear regression (PCA-LR).

5

Discussion

This research aims to enhance the surrogate modeling framework FMROM and extend its application to new settings, by using an urban domain in addition to a coastal one, and incorporating data assimilation into the dimensionality reduction process. The results show that it remains challenging to outperform PCA for dimensionality reduction, while new implementations of the surrogate model achieve higher performance than the baseline for two of the three state variables. Further, transfer learning was used as a method to integrate observational data into the autoencoder for surface elevation to better represent reality. The method showed that the observational data influence the reconstruction. Although the impact appears small, it extends beyond the specific elements where the observational data were directly available. Applying FMROM in an urban context demonstrates both the potential of the framework and the importance of having a diverse and sufficiently large dataset. This section will present reflections on the findings and suggestions for improvements and future work.

Model performance reflects the domain dynamics

The baseline for the coastal case was set using PCA-LR, as a previous study demonstrated strong potential for these methods in surrogate modeling of the Øresund region. This region is enclosed by land with only two openings, as a result, its behavior is strongly influenced by the boundary conditions at these openings, leading to relatively simple dynamics. The characteristics of the region makes it challenging to outperform the baseline, as the simple dynamics already allow linear methods to achieve strong performance. However, since both the autoencoder for dimensionality reduction and LSTM for regression outperformed the baseline for the velocities even in this simple case, it suggests that these more complex models may offer even greater advantages in more complex scenarios.

Variable separation enhances flexibility but increases complexity

Some implementations showed an improved performance for velocity components and a decrease in surface elevation, indicating that it could be beneficial to separate variables. If the goal is to achieve as high accuracy as possible, more complex models should be chosen according to this study. The flexibility of the framework is a strength in this sense, allowing for more design choices. However, the advantages of using the same regression model for all variables are 1) we make use of the correlation between the variables, 2) greater implementation and code simplicity, and 3) in some cases less computationally heavy. It is also important to note that while the flexibility allows for potential improvements, it also requires more extensive preprocessing. How much correlation should there be between variables to keep them together? Can the prediction still be improved, and when is it a worthwhile trade-off to try all different implementations, i.e. all possible combinations of one dimensionality reduction technique and one regression model? To know whether the result of the variable separation is case-specific or represents a more general trend, further investigation is needed. Moreover, since the framework offers the flexibility to be adapted to different state variables other than surface elevation and velocities, the analysis cannot simply be reused and needs to be reassessed for each new application.

Assessing model performance on high-impact events using t2RMSE

The results of a hydrodynamical simulation can serve various purposes, but in many cases, rare or extreme events are the most relevant. Average conditions are typically less interesting, as they are better understood and already accounted for in system design. In contrast, extreme events often pose greater risks and are therefore more important to analyze and understand through simulation. The t2RMSE metric used in this study evaluates how well the models reconstruct or predict data for the top 2% of high-magnitude values, providing a partial measure of model performance on extreme events. The autoencoder archived roughly similar performance as the PCA for the velocity components: the RMSE metric improved slightly, while the t2RMSE worsened. However, the autoencoder offers greater flexibility for fine-tuning to meet specific objectives. A potential improvement to this study would be to put more emphasis to the t2RMSE metric in the model's training objective. This could involve, for example, designing a custom loss function that places greater penalties on errors associated with high-magnitude values.

Data limitations and the potential of synthetic data generation

All error metrics improved when the autoencoders were trained on the extended training set, covering a period ten times longer than the original training data. The improvement was especially clear for the velocity components, and highlights the potential for the autoencoder to outperform PCA. However, generating training data is traditionally time consuming and resource-intensive, as it requires running the original physics-based model. If the goal of the surrogate model is to replicate the results of the physics-based model much faster, there is a trade-off to consider: improvements in surrogate model performance may come at the cost of increase of time demands, due to data generation. A possible way of handling this trade-off is to generate synthetic data. A possible way of generating data could be to use a variational autoencoder (VAE) for dimensionality reduction, which learns a probability distribution over the latent space. Unlike traditional autoencoders that map high-dimensional input data to fixed points in the latent space, VAEs map inputs to distributions in the latent space. This allows for the generation of multiple distinct samples from the same latent space, effectively expanding the dataset without requiring additional costly simulations.

On fairness in latent space comparisons between PCA and autoencoders

In this study, to compare different dimensionality reduction techniques in the coastal case, the three state variables were reduced to the same number of latent dimensions using each technique. The resulting representations were then evaluated and compared using a set of error metrics. However, it is debatable whether using the same number of latent dimensions provides a fair basis for comparison between the two methods. PCA is highly predictable, as the basis vectors of its latent space are orthogonal. This means the reconstruction error consistently decreases as more dimensions are added, although the rate of improvements gradually diminishes. In contrast, autoencoders are less predictable in this regard. A study showed that autoencoders tend to perform best when the size of the latent space matches the intrinsic dimensionality of the dataset [15]. Beyond that point, increasing the latent space does not necessarily lead to continued improvement in reconstruction quality. This could lead to an unfair comparison, likely disadvantaging the autoencoder, which may perform worse with too many latent dimensions. A possible improvement of the comparison could be to optimize each latent space dimensionality separately.

Reconstruction vs. prediction: decoupling may limit model performance

Although reconstruction error contributes to the total error, the prediction error from the regression is the dominant factor. This conclusion can be drawn since the regression error is several times larger than the reconstruction error across all variables. In this study, the dimensionality reduction methods were evaluated solely based on reconstruction error, without considering how well the regression model interprets the latent space. Our results suggest that the regression model's ability to understand the latent space can vary. The fact that the difference in prediction error between PCA-LR and AE-LR exceeds the difference in their reconstruction errors implies that LR performs better in the latent space found by AE, compared to the latent space determined by PCA. We made a design choice to develop and evaluate dimensionality reduction and regression separately, but the results raise the question of whether performance could be improved by coupling the two.

When to choose complex models: a trade-off between accuracy and efficiency

The comparison of models in this study shows that model selection requires a choice between simplicity and complexity. Both autoencoders and LSTMs are neural networks, which are known to be flexible with many design choices, for example regarding layer architectures. This flexibility can also be considered limiting. Once a model has been optimized for a specific problem or variable, it may not perform well on a different, but similar, task. Additionally, training these complex models is time-consuming, as shown in Table 5.1. This raises the question of whether the marginal improvement in performance justifies the added complexity and computational cost. To address this question, it is important to consider the intended application. Surrogate models that offer high accuracy at a lower computational cost open up new opportunities, particularly in contexts where large numbers of simulations are needed. It makes them well-suited for use in climate scenario modeling, ensemble-based forecasting, and sensitivity analysis [1]. If the most important factor is speed-up, a fast, low-complexity model might be preferable, even if it results in a trade-off with accuracy. On the other hand, if the most important goal is to emulate the physics-based model as closely as possible, more complex models should be used despite their longer training times. However, it is important to note that the training is performed only once, and the trained surrogate model can be used to predict multiple different scenarios. In contrast, the MIKE model must be re-run for each new scenario. This makes surrogate models generally preferable in multiple scenario prediction over physics-based models, as their runtime does not

scale in the same way with the number of scenarios.

Type of simulation	Computer	Approx. runtime
MIKE data generation	A	23 hours
Training PCA	B	10 seconds - 1 minute
Prediction PCA	B	<1 second
Training autoencoder	B	20 minutes - 1 hour
Prediction autoencoder	B	15 seconds
Training linear regression model	C	1 second
Prediction linear regression	C	<1 second
Training LSTM models	C	30-45 minutes
Prediction LSTM model	C	15 seconds

Table 5.1: Computer setup A: high-performance computer (details unknown). Computer setup B: Processor: AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz. Installed RAM: 8,00 GB (7,42 usable). Computer setup C: Processor: 13th Gen Intel(R) Core(TM) i7-1365U 1.80 GHz, Installed RAM: 32,0 GB (31,4 GB usable).

Advanced models open up for adaptations

More advanced models can be designed to better reflect real-world behavior by incorporating data assimilation, or tailored specifically to capture extreme events. This may be particularly relevant in cases where traditional physics-based models are insufficient, or when the goal is not to emulate MIKE as closely as possible but rather to learn and produce forecasts based on two different data sources (physics-based model data and observational data), in a hybrid model fashion. Moreover, models can be trained or calibrated using datasets that over-represent extreme events, e.g., synthetic extremes, or a dataset where extreme events are added manually. This could help the model improve predictive power in events that are underrepresented in typical simulated datasets. With simpler models, both for dimensionality reduction and regression parts, these types of adaptations may not be possible.

Applying surrogate model framework to urban environments: limitations and suggestions for future work

Applying FMROM to an urban case study demonstrates the potential to extend the framework to other water-related modeling tasks. However, adapting the model to

a different domain and state variable introduces new challenges. While the Øresund case models open water conditions in a relatively simple domain, the urban case involves modeling water depth in a highly complex environment. The results can be considered relatively promising, but with very high errors in some specific areas. We believe that the lack of performance is primarily due to lack of stabilizing boundary conditions and limited amount of data. It was concluded that the surrogate model for the urban case was sensitive to which rain events the model was trained on and which events it was tested on. This instability of train and test splits reveals that the model does not generalize well, specifically when extreme events are not a part of the training set. A possible continuation is therefore to generate a larger dataset. However, it is computationally expensive to run the physics-based MIKE model. Additionally, for a real world application the range of rain event types would need to be even more diverse, and for example include historical, real-world rain events and not only constant rains as in this project. By generating several MIKE simulations with varied rain events, the model will likely improve predictive power and generalization. Moreover, the areas of interest (AOI) should be considered specifically, and how to improve the performance in those areas. High errors were found in a viaduct, and in some parts of the densely built-up areas, which are part of the AOI's. Some of the AOI's match what the first principal components capture, and the performance could potentially improve by not performing the post-PCA scaling, i.e., letting those dimensions dominate in the latent space. However, for other AOI's, some type of element weighting could be a possible approach to force the regression model to prioritize some areas. This could be implemented by, for example, using another type of dimensionality reduction technique where a custom loss is applicable.

Another approach that we suggest as a more suitable method for the task of predicting urban flooding, is to forgo temporal prediction of the state and instead predict the maximum depth in each element of the domain. Then, each rain event (input) is mapped directly to a spatial snapshot of maximum depths (output). This is similar to the method used by Xu et al. [37], where a LightGBM model was employed to predict peak inundation depths at flood-prone locations. However, this approach falls outside the FMROM framework, where the temporal dimension is a central component.

6

Conclusion

This study explored alternative reduction methods and regression models for enhancing surrogate model performance within the FMROM framework. We found that autoencoders offer advantages over PCA when handling more nonlinear data, like the velocity components, while PCA remains more effective for linear features, like surface elevation. The flexibility of autoencoders also opens the door to more customized modeling objectives. Incorporating observational data into the training process showed promising results, pulling the reconstructed data closer to reality even beyond the areas with direct observation input, without degrading performance elsewhere.

The regression model's results mirrored the dimensionality reduction findings, but even more markedly. Simpler models like linear regression remained competitive for surface elevation, but nonlinear dynamics in velocity components benefited substantially from the use of the LSTM model. This highlights the benefit of FMROM's modularity, which allows each state variable to be reduced and predicted with the most suitable technique.

The framework proved applicable to an urban flooding context. However, the surrogate model lacked boundary conditions to constrain its predictions within a realistic range, causing the error to accumulate over time. This suggests that alternative surrogate modeling approaches, such as static input-output models, may be worth exploring.

Bibliography

[1] F. Petersen, J. Sandvig Mariegaard, and R. Palmitessa. A framework for building fast reduced order surrogates for flexible mesh coastal-ocean models. 2024, Unpublished.

[2] K. E. Taylor, R. J. Stouffer, and G. A. Meehl. "An Overview of CMIP5 and the Experiment Design". *Bulletin of the American Meteorological Society*, 93(4):485 – 498, 2012. [Online], Available: `https://journals.ametsoc.org/view/journals/bams/93/4/bams-d-11-00094.1.xml`.

[3] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan Read, Jacob Zwart, Michael Steinbach, and Vipin Kumar. Physics guided rnns for modeling dynamical systems: A case study in simulating lake temperature profiles, 2019. [Online], Available: `https://arxiv.org/abs/1810.13075`.

[4] A. Meray I. Mastilovic S. Praveen Z. Xu M. Memarzadeh A. Lavin H. Wainwright L. Wang, T. Kurihana. Multi-scale digital twin: Developing a fast and physics-informed surrogate model for groundwater contamination with uncertain climate models, 2022. [Online], Available: `https://arxiv.org/abs/2211.10884`.

[5] R. Lam et. al. "Learning skillful medium-range global weather forecasting". *Science*, 382(6677):1416–1421, 2023. [Online], Available: `https://www.science.org/doi/pdf/10.1126/science.adi2336`.

[6] D. Samadian and N. Dawood Muhit, B. Imrose. "Application of Data-Driven Surrogate Models in Structural Engineering: A Literature Review". *Archives of Computational Methods in Engineering*, 32(2):735–784, jul 2024.

[7] L. Liu et al. "Review of surrogate model assisted multi-objective design optimization of electrical machines: New opportunities and challenges". *Renewable and Sustainable Energy Reviews*, 215:115609, 2025. [Online], Available: `https://www.sciencedirect.com/science/article/pii/S1364032125002825`.

[8] P. Li Z. Xie J. Feng, Y. Tian and Hao Wang. "Short-term water quality prediction of reclaimed water plant effluent and key measurement sections based on a surrogate prediction model". *Journal of Environmental Manage-*

ment, 380:125147, 2025. [Online], Available: https://www.sciencedirect.com/science/article/pii/S0301479725011235.

[9] *MIKE 21 Flow Model FM*, DHI, Denmark, 2025, Available: https://manuals.mikepoweredbydhi.help/latest/Coast_and_Sea/MIKE_FM_HD_2D.pdf.

[10] J. M. L. de Vera J. Gojo Cruz and K. E. Pilario. "Machine learning-driven analysis of agro-climatic data for temperature modeling and forecasting in philippine urban areas". *Urban Climate*, 60:102339, 2025. [Online] Available: http://www.sciencedirect.com/science/article/pii/S0921452613008351.

[11] L. Xu and L. Gao. "A hybrid surrogate model for real-time coastal urban flood prediction: An application to Macao". *Journal of Hydrology*, 642:131863, 2024. [Online], Available: https://www.sciencedirect.com/science/article/pii/S0022169424012599.

[12] *MIKE FLOOD 1D–2D Modelling User Manual*, DHI, Denmark, 2017, Available: https://www.scribd.com/document/333681449/Mike-Flood-Usermanual.

[13] P. Koumoutsakos M. Milano. "Neural Network Modeling for Near Wall Turbulent Flow". *Journal of Computational Physics*, 182(1):1–26, 2002. https://www.sciencedirect.com/science/article/pii/S0021999102971469?ref=pdf_download&fr=RR-2&rr=93f996764bb6ebd2.

[14] X. Chu Q. Lu X. Wang S. Ding, C. Ni. "Reduced-order modeling via convolutional autoencoder for emulating combustion of hydrogen/methane fuel blends". *Combustion and Flame*, 274:113981, 2025. https://www.sciencedirect.com/science/article/pii/S0010218025000197.

[15] S. Zhao Y. Wang, H. Yao. "Auto-encoder based dimensionality reduction". *Neurocomputing*, 184:232–242, 2016. RoLoD: Robust Local Descriptors for Computer Vision 2014.

[16] D. McSpadden S. Goldenberg B. Roy, J. L. Goodall and M. Schram. "Forecasting Multi-Step-Ahead Street-Scale Nuisance Flooding using a seq2seq LSTM Surrogate Model for Real-Time Application in a Coastal-Urban City". *Journal of Hydrology*, 656:132697, 2025. https://www.sciencedirect.com/science/article/pii/S0022169425000356?ref=pdf_download&fr=RR-2&rr=93f99ab29d0aebd2.

[17] M. Carolina de Oliveira Costa C. Leandro da Silva Junior R. de Camargo M. Bonjour Laviola da Silva, F. Tulio Camilo Barreto. "Bias correction of significant wave height with LSTM neural networks". *Ocean Engineering*, 318:120015, 2025. https://www.sciencedirect.com/science/article/pii/S0029801824033535?via%3Dihub.

[18] P. Ghodratallah et al. A. Basem, M. Yasiri. "Prediction of the transient coolant jet released from the nose cone at supersonic flow via machine learning.". *Sci Rep*, 2025. 15, 3516, https://www.nature.com/articles/s41598-025-87926-4#Sec6.

[19] Spatio-temporal storm surge emulation using gaussian process techniques. *Coastal Engineering*, 180:104231, 2023. [Online], Available: https://www.sciencedirect.com/science/article/pii/S0378383922001442.

[20] *MIKE+*, DHI, Denmark, 2025, Available: https://manuals.mikepoweredbydhi.help/latest/Cities/MIKE_Plus_2DOverland.pdf.

[21] University of Reading. What is data assimilation? https://research.reading.ac.uk/met-darc/aboutus/what-is-data-assimilation/.

[22] Massachusetts Institute of Technology, "Neural Networks", Intro to Machine Learning lecture notes, fall 2022, Chapter 7, [Online] Available: https://introml.mit.edu/_static/fall22/LectureNotes/chapter_Neural_Networks.pdf.

[23] M. Nielsen. "Neural Networks and Deep Learning". Chapter 2: How the backpropagation algorithm works, Dec 2019, [Online], Available: http://neuralnetworksanddeeplearning.com/chap2.html#the_four_fundamental_equations_behind_backpropagation.

[24] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization", 2017. [Online] Available: https://arxiv.org/abs/1412.6980.

[25] B. Or. The Exploding and Vanishing Gradients Problem in Time Series. *Medium*, [Online], 2020. Available: https://medium.com/metaor-artificial-intelligence/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22.

[26] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural Computation*, 9:1735–1780, Nov. 1997. [Online], Available: https://www.researchgate.net/publication/13853244_Long_Short-Term_Memory.

[27] A. D. Rasamoelina, F. Adjailia, and P. Sinčák. "A Review of Activation Function for Artificial Neural Network". In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286, 2020. [Online], Available: https://ieeexplore.ieee.org/document/9108717.

[28] G. R. North, T. L. Bell, R. F. Cahalan, and F. J. Moeng. "Sampling errors in the estimation of empirical orthogonal functions". *Monthly Weather Review*, 110(7):699–706, 1982. [Online] Available: https://journals.ametsoc.org/view/journals/mwre/110/7/1520-0493_1982_110_0699_seiteo_2_0_co_2.xml.

[29] *Covariates - Darts User Guide*. Unit8, Switzerland, Accessed: Mar 2, 2025. [Online], Available: https://unit8co.github.io/darts/userguide/covariates.html.

[30] J. Herzen and F. Lässig et al. "Recurrent Neural Networks". Accessed: Feb. 1, 2025. [Online] Available: https://unit8co.github.io/darts/generated_api/darts.models.forecasting.rnn_model.html.

[31] J. H. Andersson et al. *ModelSkill*, mar 2025. Version 1.2.0, [Online] Available: https://github.com/DHI/modelskill.

[32] J. H. Andersson and J. S. Mariegaard. *MIKE IO*, jun 2013. Version 1.6.1, [Online] Available: https://dhi.github.io/mikeio/.

[33] M. Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems", 2015. Software available from https://www.tensorflow.org/.

[34] J. Herzen and F. Lässig et al. "Darts: User-friendly modern machine learning for time series". *Journal of Machine Learning Research*, 23(124):1–6, 2022. [Online] Available: http://jmlr.org/papers/v23/21-1177.html.

[35] The PyTorch Foundation. "pytorch". Accessed: Feb. 1, 2025, [Online], Available: https://pytorch.org/.

[36] DHI. "Hydrodynamic model of Øresund: Mike21 model setup, outputs and observation data", dec 2024. [Online], Available: https://doi.org/10.5281/zenodo.14160710.

[37] H. Xu L. Bin K. Xu, Z. Han. "Rapid Prediction Model for Urban Floods Based on a Light Gradient Boosting Machine Approach and Hydrological–Hydraulic Model". *International Journal of Disaster Risk Science*, 14(1):79–97, 2023. [Online], Available: https://link.springer.com/article/10.1007/s13753-023-00465-2.

A

Appendix 1

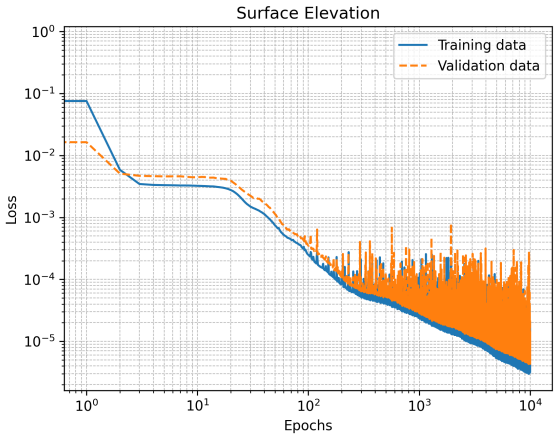


Figure A.1: Training and validation loss for the autoencoder trained on the regular dataset of surface elevation, for 10,000 epochs.

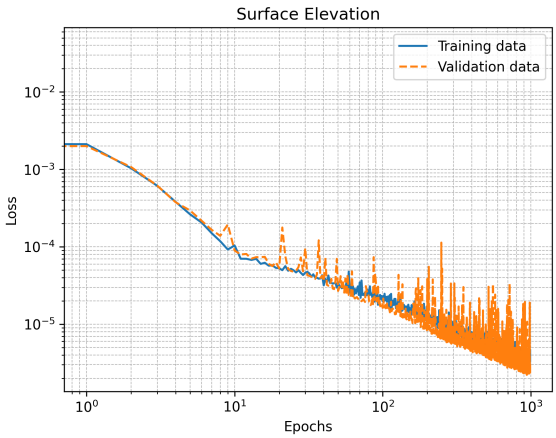


Figure A.2: Training and validation loss for the autoencoder trained on the expanded dataset of surface elevation, for 1,000 epochs until the early stopping halted the training.

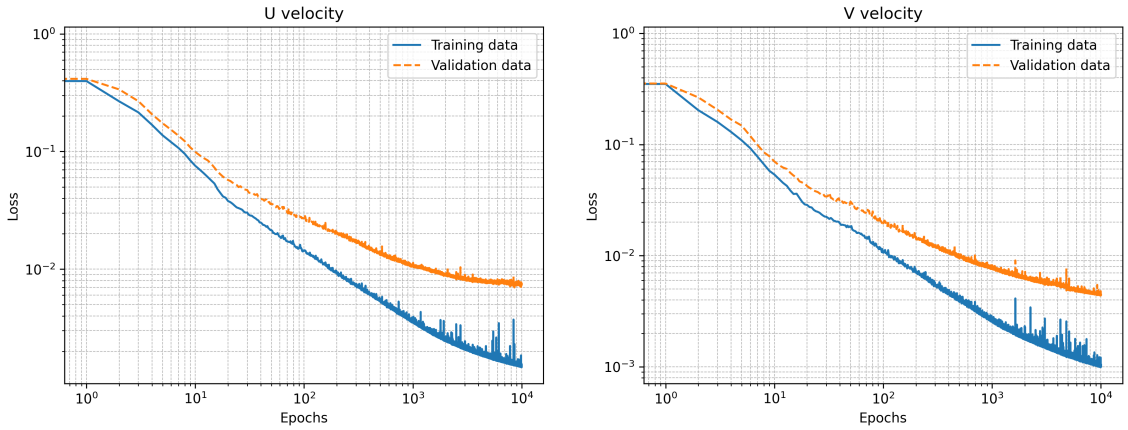


Figure A.3: Training and validation loss for autoencoders trained on the regular datasets of U and V velocities, for 10,000 epochs, respectively.

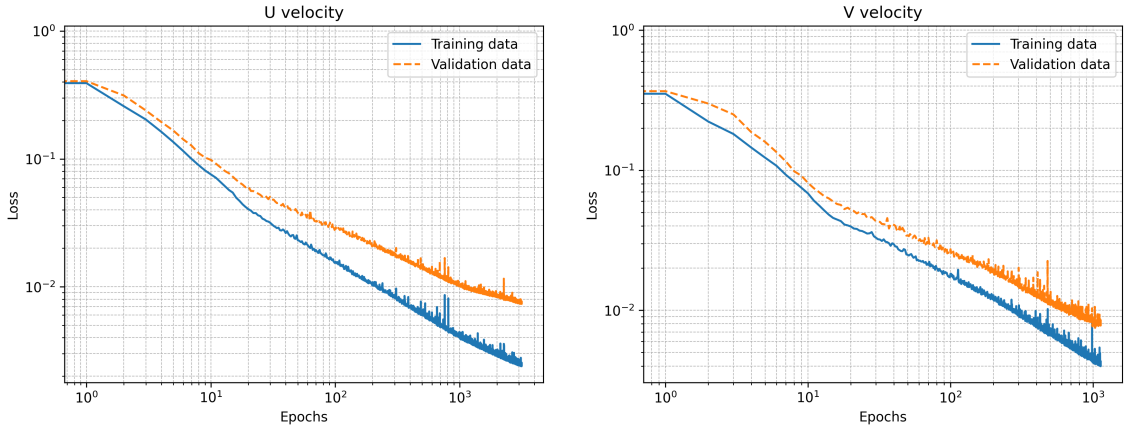


Figure A.4: Training and validation loss curves for the regularized autoencoders of the U and V velocity components, trained on the regular training dataset. The U-autoencoder was trained for 3,163 epochs until early stopping halted the training, while the V-autoencoder was trained for 1,027 epochs.

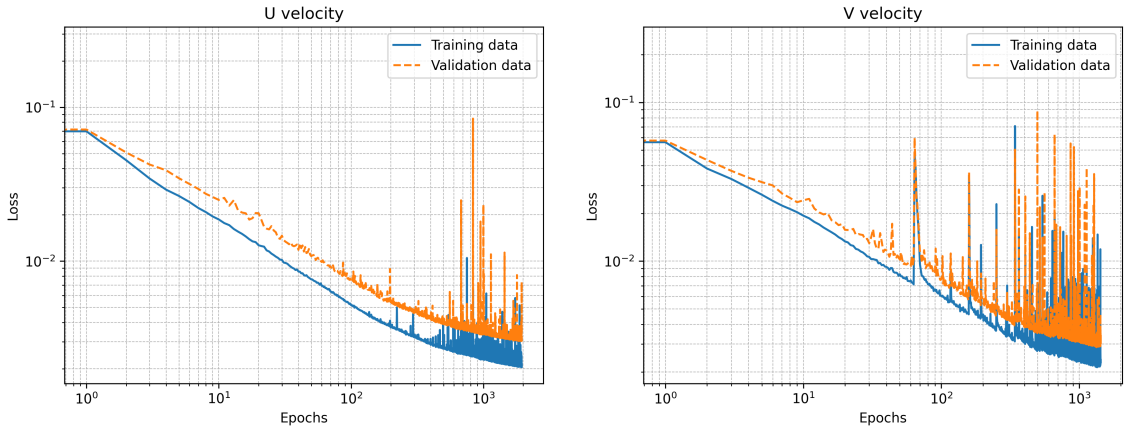


Figure A.5: Training and validation loss curves for the regularized autoencoders of the U and V velocity components, trained on the expanded training dataset. The U-autoencoder was trained for 1,950 epochs until early stopping halted the training, while the V-autoencoder was trained for 1,432 epochs.

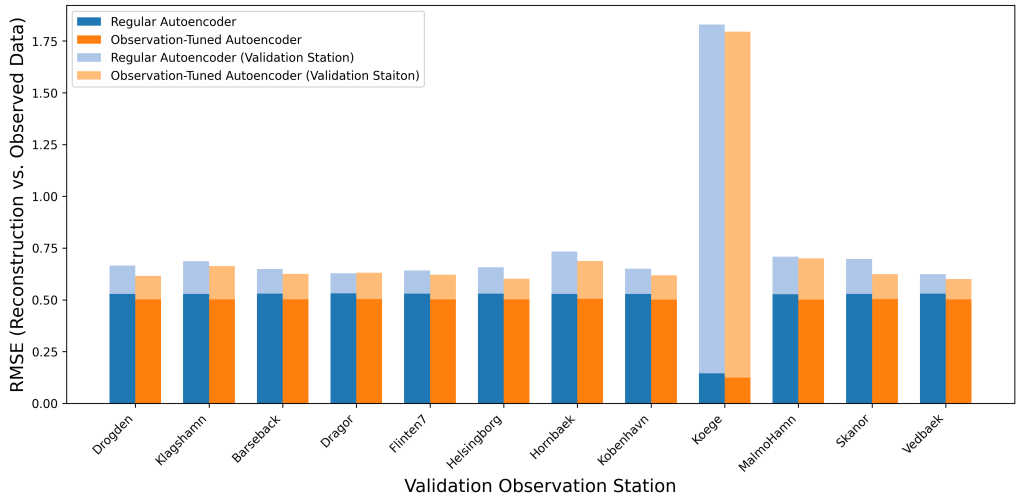


Figure A.6: RMSE between observed and reconstructed data for both the regular autoencoder (blue) and the observation-tuned autoencoder (orange), shown for locations used during fine-tuning (solid) and those excluded from fine-tuning (transparent).

A

Appendix 2 (Surrogate model implementations)

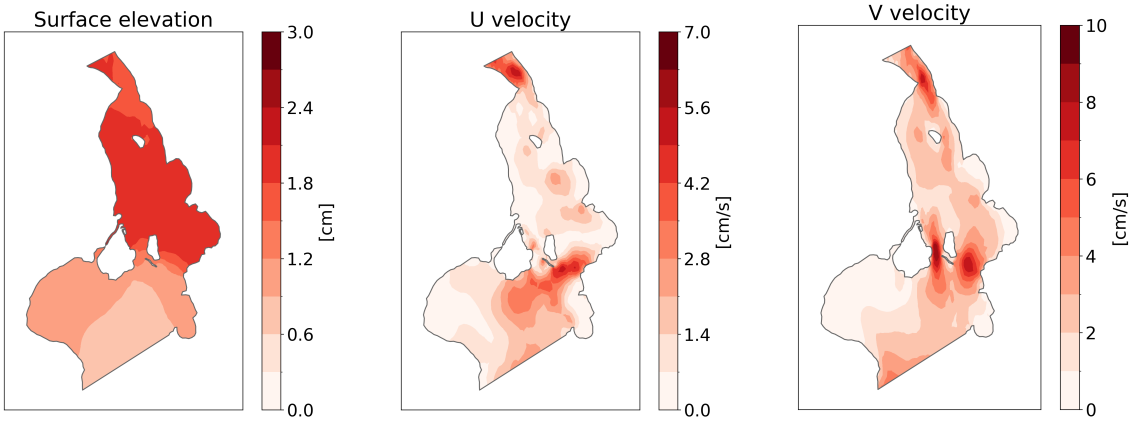


Figure A.1: Spatial RMSE of the three state variables with an FMROM implementation using principal component analysis and linear regression (PCA-LR), from left to right; surface elevation, U, velocity and V velocity.

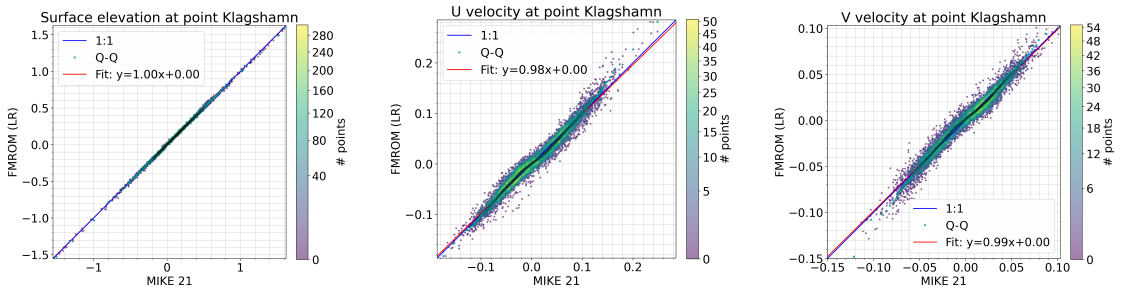


Figure A.2: FMROM result vs MIKE model for implementation with principal component analysis and linear regression (PCA-LR). The red line shows a perfect one to one relationship, i.e. the prediction from FMROM is exactly accurate. The blue line shows the fitted linear function $y = kx + m$.

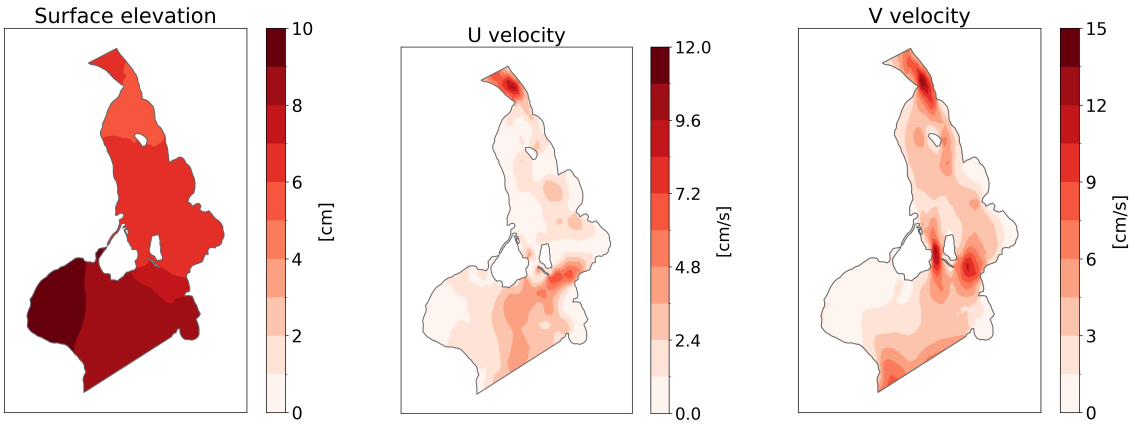


Figure A.3: Spatial RMSE of the three state variables with an FMROM implementation using principal component analysis and LSTM (PCA-LSTM), from left to right; surface elevation, U velocity and V velocity.

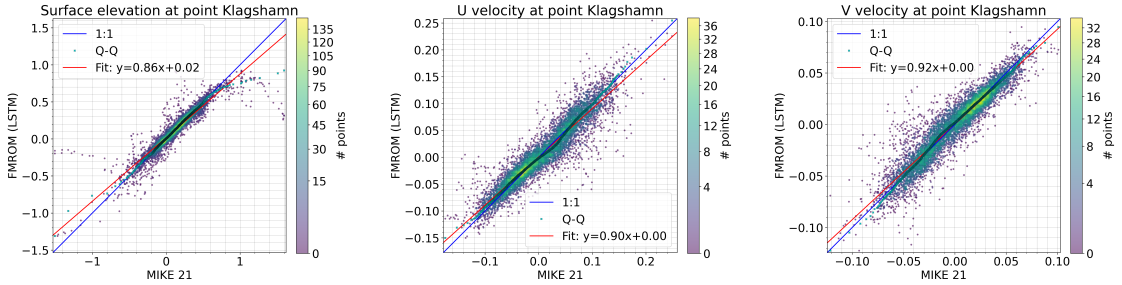


Figure A.4: FMROM result vs MIKE model for implementation with principal component analysis and LSTM (PCA-LSTM) in Klagshamn, from left to right; surface elevation, U velocity and V velocity. The red line shows a perfect one to one relationship, i.e. the prediction from FMROM is exactly accurate. The blue line shows the fitted linear function $y = kx + m$.

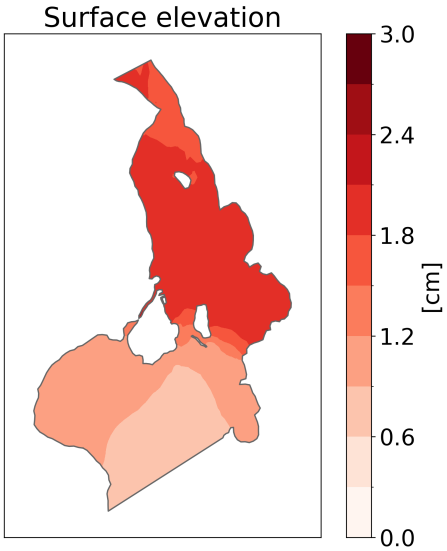


Figure A.5: Spatial RMSE of state variable surface elevation, FMROM implementation with principal component analysis and linear regression model (PCA-LR_{SE}).

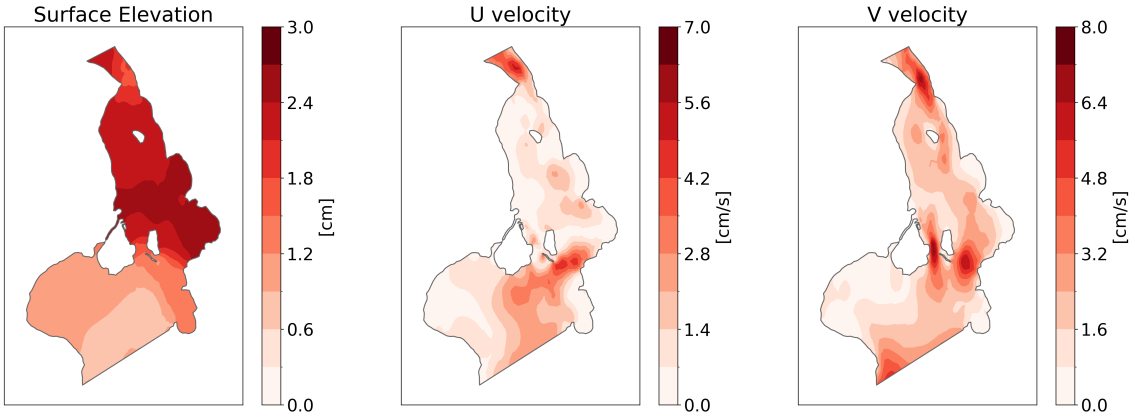


Figure A.6: Spatial RMSE of the three state variables with an FMROM implementation using AE and linear regression (AE-LR), from left to right; surface elevation, U velocity and V velocity.

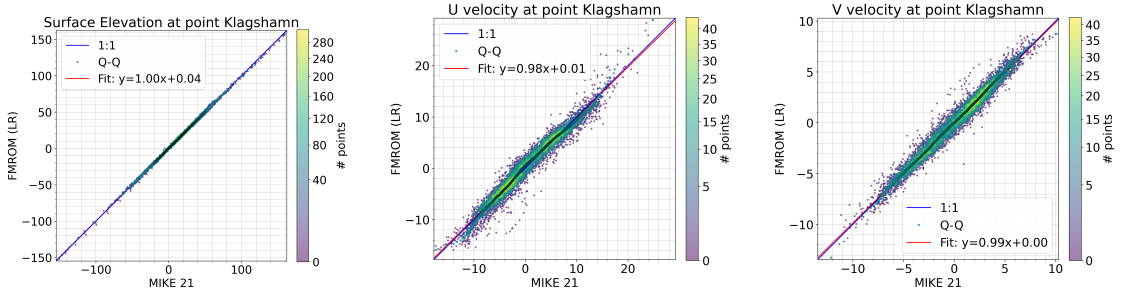


Figure A.7: FMROM result vs MIKE model for implementation with AE and linear regression (AE-LR) in Klagshamn, from left to right; surface elevation, U velocity and V velocity. The red line shows a perfect one to one relationship, i.e. the prediction from FMROM is exactly accurate. The blue line shows the fitted linear function $y = kx + m$.

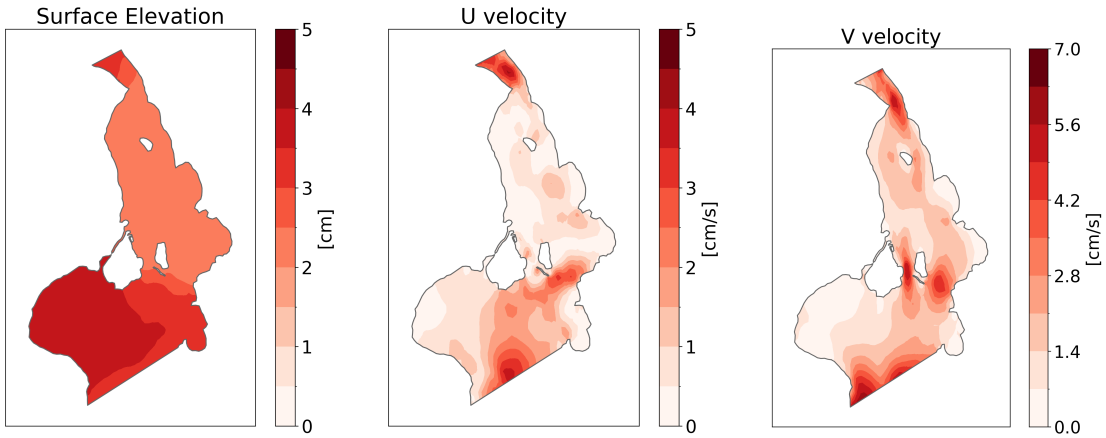


Figure A.8: Spatial RMSE of the three state variables with an FMROM implementation using autoencoder and LSTM (AE-LSTM), from left to right; surface elevation, U, velocity and V velocity.

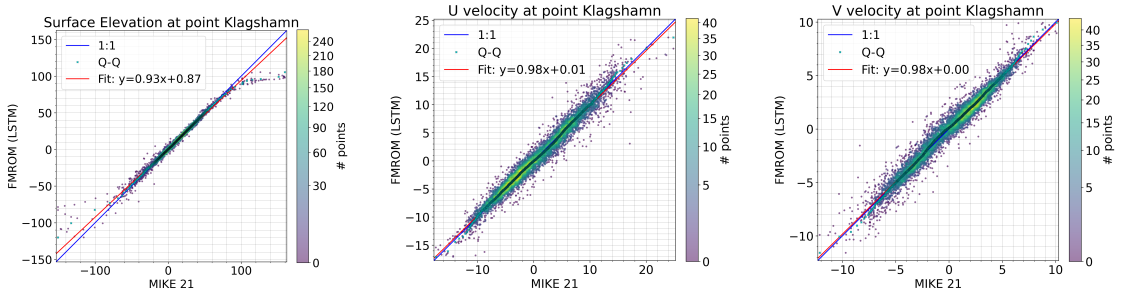


Figure A.9: FMROM result vs MIKE model for implementation with AE and linear regression (AE-LSTM) in Klagshamn, from left to right; surface elevation, U velocity and V velocity. The red line shows a perfect one to one relationship, i.e. the prediction from FMROM is exactly accurate. The blue line shows the fitted linear function $y = kx + m$.

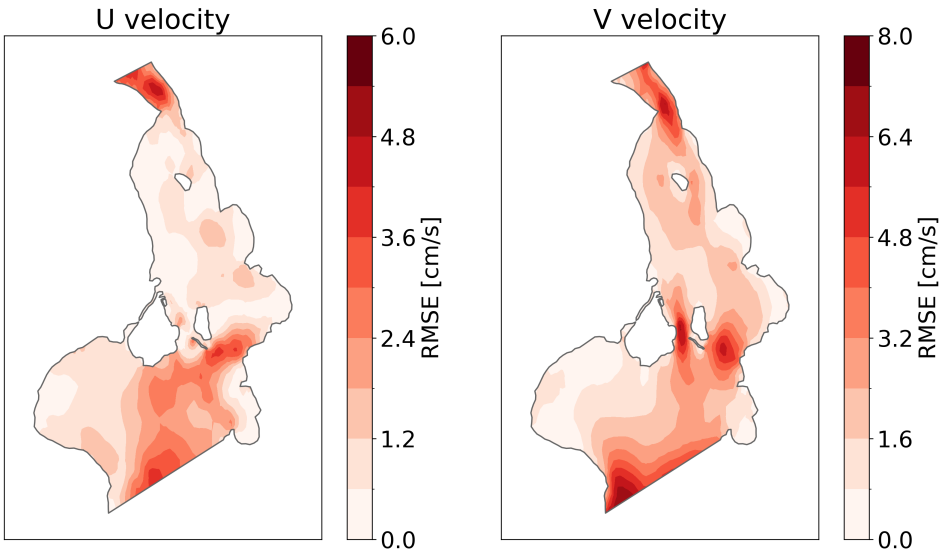


Figure A.10: Spatial RMSE of state variables U and V velocity, FMROM implementation with principal component analysis and LSTM regression model (PCA-$\text{LSTM}_{\text{U, V}}$).

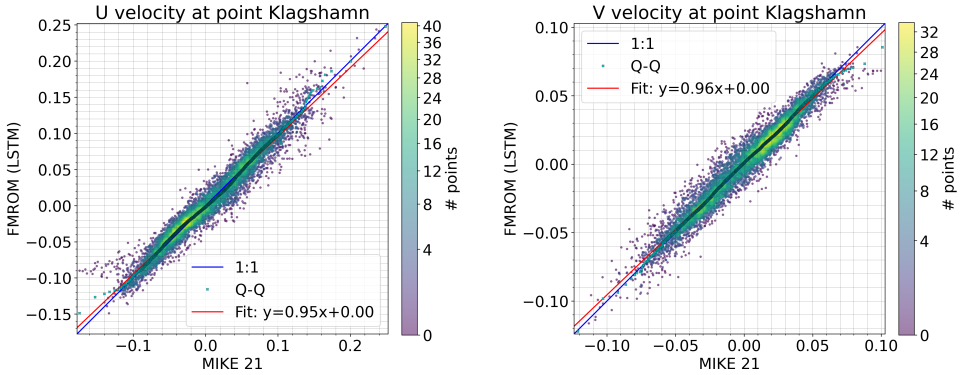


Figure A.11: FMROM result vs MIKE model for implementation with principal component analysis and LSTM (PCA-LSTM$_{U,\,V}$) in Klagshamn, left: U velocity, right: V velocity. The red line shows a perfect one to one relationship, i.e. the prediction from FMROM is exactly accurate. The blue line shows the fitted linear function $y = kx + m$.

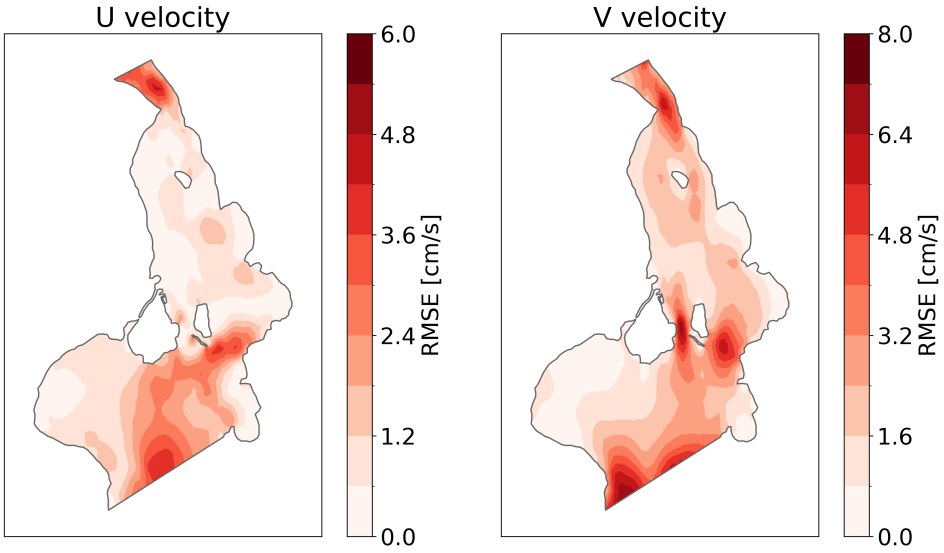


Figure A.12: Spatial RMSE of state variables U and V velocity, FMROM implementation with autoencoder and LSTM regression model (AE-LSTM$_{U,\,V}$).

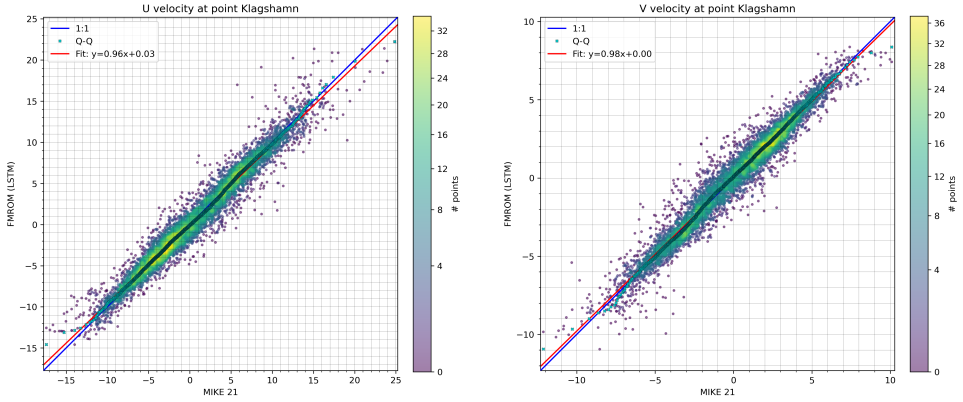


Figure A.13: FMROM result vs MIKE model for implementation with autoencoder and LSTM (AE-LSTM$_{\mathrm{U,\,V}}$) in Klagshamn, left: U velocity, right: V velocity. The red line shows a perfect one to one relationship, i.e. the prediction from FMROM is exactly accurate. The blue line shows the fitted linear function $y = kx + m$.

A

Appendix 3

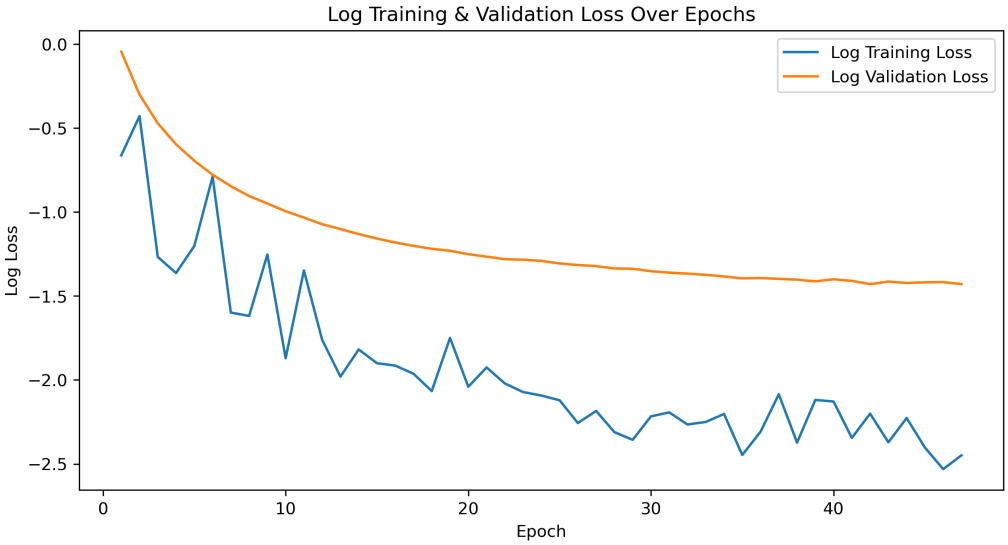


Figure A.1: Training and validation loss curves for FMROM implementation with principal component analysis and linear regression for surface elevation, U velocity and V velocity. The training halted after 47 epochs due to early stopping.

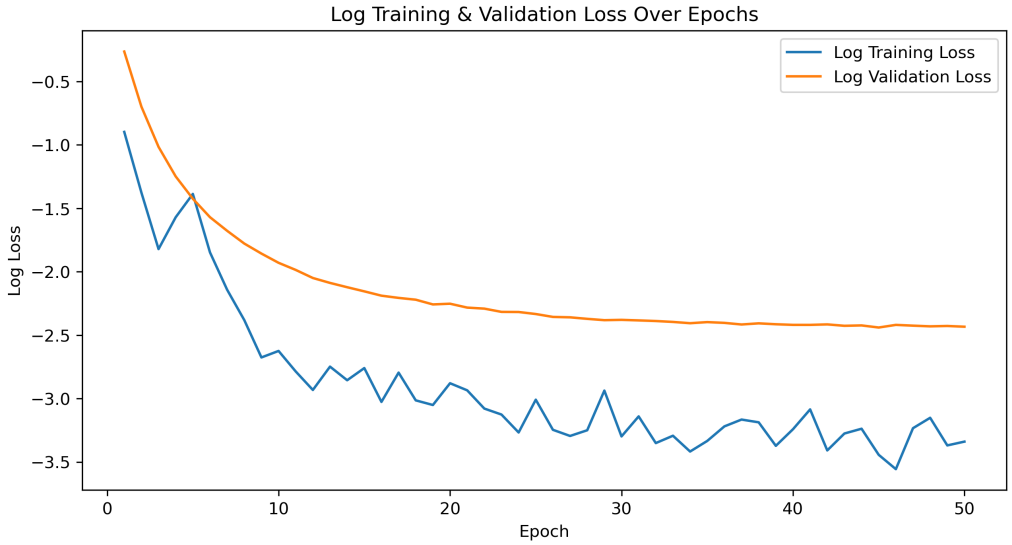


Figure A.2: Training and validation loss curves for FMROM implementation with principal component analysis and LSTM for surface elevation, U velocity and V velocity. The training halted after 50 epochs due to early stopping.

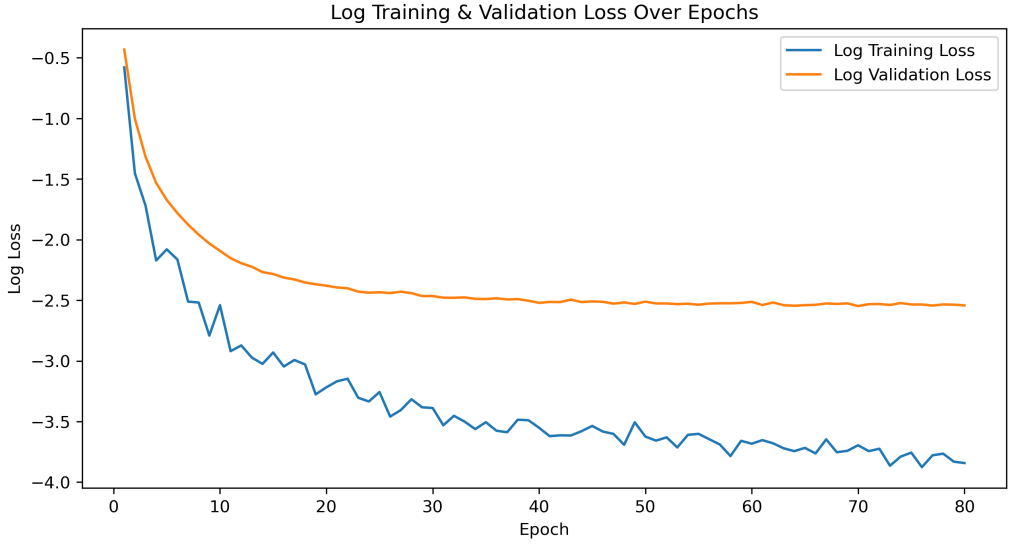


Figure A.3: Training and validation loss curves for FMROM implementation with principal component analysis and linear regression for surface elevation, U velocity and V velocity. The training halted after 80 epochs due to early stopping.

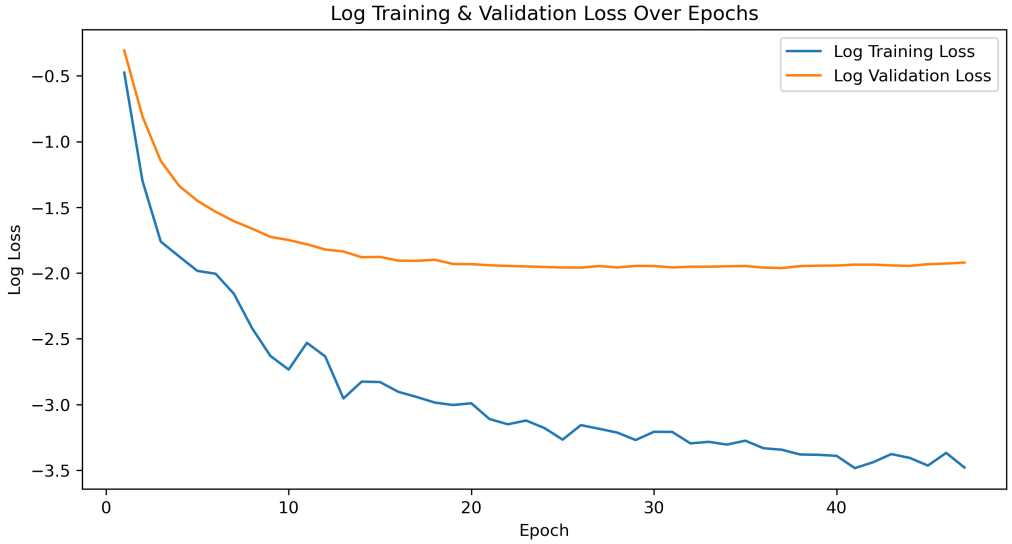


Figure A.4: Training and validation loss curves for FMROM implementation with autoencoder and LSTM for surface elevation, U velocity and V velocity. The training halted after 48 epochs due to early stopping.

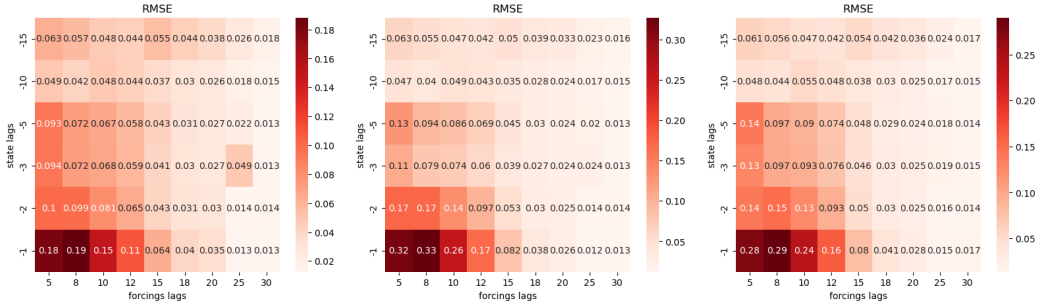


Figure A.5: RMSE values from grid search for forcing lags and state lags. Both the negative state lags and the positive forcing lags indicate past lag values. For train test split number 1 (left), 2 (middle) and 3 (right) out of 13 in urban case.

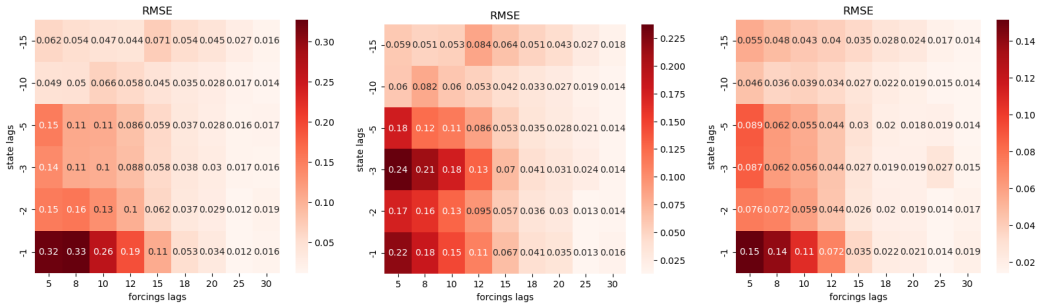


Figure A.6: RMSE values from grid search for forcing lags and state lags. Both the negative state lags and the positive forcing lags indicate past lag values. For train test split number 4 (left), 5 (middle) and 6 (right) out of 13 in urban case.

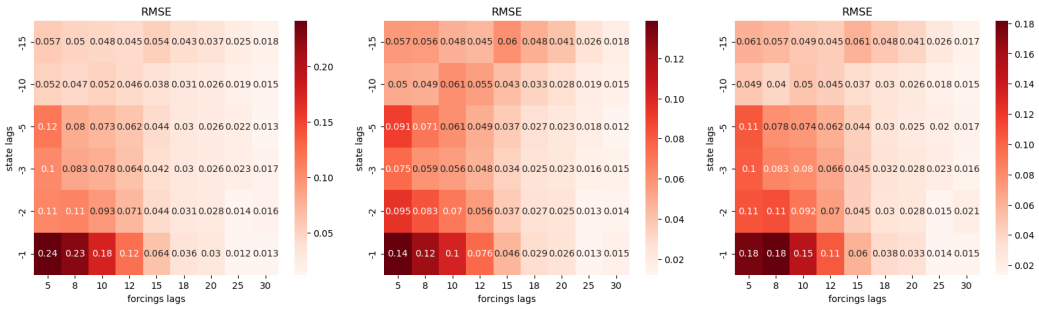


Figure A.7: RMSE values from grid search for forcing lags and state lags. Both the negative state lags and the positive forcing lags indicate past lag values. For train test split number 7 (left), 8 (middle) and 9 (right) in urban case.

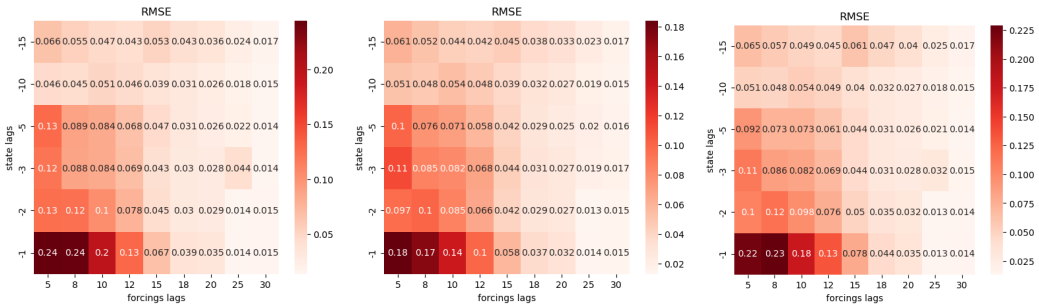


Figure A.8: RMSE values from grid search for forcing lags and state lags. Both the negative state lags and the positive forcing lags indicate past lag values. For train test split number 10 (left), 11 (middle) and 12 (right) out of 13 in urban case.

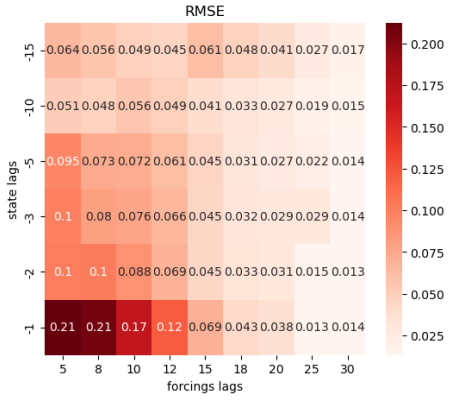


Figure A.9: RMSE values from grid search for forcing lags and state lags. Both the negative state lags and the positive forcing lags indicate past lag values. For the last train test split (number 13) in urban case.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY