Incorporating sensor measurements using data assimilation and machine learning to improve the accuracy of thermal finite element models.

by

Karl Hellberg

Submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Mechanical Engineering)

in the

Department of Mechanical and Aeronautical Engineering
Faculty of Engineering, Built Environment and Information Technology
University of Pretoria

Supervised by

Prof. PS Heyns

Prof. J Wannenburg

2022

Abstract

Complex systems are commonly encountered in engineering. Such systems have a degree of unpredictability, which can lead to undesirable results. The digital twin concept has been proposed as a method to obtain more information about the system so that its behaviour can be better predicted. To construct a digital twin of a system, a high-fidelity model that predicts the evolution of the system over time is required. Classically, such high fidelity models would be either physics-based or data-driven, though both of these approaches have disadvantages that may make them unsuitable for application in a digital twin. A hybrid model is a combination of physics-based and data-driven models that seeks to exploit the advantages of both approaches, and is a promising candidate for producing a model that is suitable for application in a digital twin.

This work investigates the training of hybrid models of real engineering systems. It considers systems that are dynamic and that are partially observed. Physics-based models of these systems are available in the form of partial differential equations that are solved numerically using the finite element method. To reduce the computational cost associated with such models, surrogate models are constructed. The construction of surrogate models involves a dimensionality reduction step in the form of proper orthogonal decomposition (POD), as well as a prediction step that involves the training of a data-driven model to predict the evolution of the POD coefficients over time. Hybrid models are then trained using the surrogate models as their physics-based component. The training of hybrid models takes place using a combination of data assimilation and machine learning. The machine learning-data assimilation (ML-DA) algorithm that is documented in the literature is used, together with two algorithms proposed in this work, called the data assimilation-observation (DA-O) and per-step DA-O algorithms.

It is the nature of the application of this methodology of training hybrid models that allows this work to make a contribution relative to the published literature. Previous uses of data assimilation in the training of hybrid models perform investigations using simplified problems that allow direct use of the physics-based model in the hybrid model. Meanwhile, the increased computational demands of the real engineering problem considered in this work necessitates the use of a surrogate model of the physics-based component of the hybrid model. Surrogate models have been previously applied in hybrid models constructed to solve engineering problems. However, these approaches do not apply naturally to applications such as digital twins where observations are continuously available. The use of data assimilation in this work allows it to address this shortcoming.

The proposed methodology of training hybrid models is evaluated using two case studies. The first case study involves a thermal simulation model of a small section of the freeboard of a process converter. Surrogate models of the simulation model are constructed using different data-driven function approximation techniques, such as Gaussian process regression, Support Vector Machines (SVMs) and neural networks. These surrogate models are then used to train hybrid models using simulated observations and the DA-O, per-step DA-O and ML-DA algorithms. When 30 of the 29077 nodes of the simulation model are observed, the per-step DA-O algorithm produces the best hybrid model in terms of a root mean square error (RMSE) metric calculated using the analysis states estimated during data assimilation. The ML-DA algorithm which performed next best is, however, easier to apply to different numbers of observed nodes and is potentially less sensitive to observation noise. The ML-DA algorithm is subsequently used to investigate the effect of using different numbers of observed nodes. While there is a benefit to using a greater number of observed nodes, the trained hybrid models still outperform the physics-based model when as few as two of the 29077 nodes of the simulation model are observed. These results indicate that the training of hybrid models for sparsely observed systems is feasible.

The second case study considered in this work involves a thermal half model of the process converter freeboard for which actual sensor observations are available. Surrogate models are again constructed using different data-driven function approximation techniques, and these are used in the training of hybrid models. Only the ML-DA algorithm is now used for the training of hybrid models. Simulated sensor observations are used at first to understand whether improvements in predictive performance that hybrid models make relative to physics-based models on the observed nodes extend to larger subsets of the nodes of the system. It is found that when the performance of the hybrid models is evaluated in terms of the RMSE calculated using analysis states estimated during data assimilation, it is possible that improvements in performance are made on the observed nodes but not on larger subsets of the nodes. When the RMSE is instead calculated using predictions of the evolution of the system over 30 time steps, the performance of the hybrid models on the observed nodes correlates with their performance on larger subsets of the nodes. When real observations are used to train hybrid models, the trained hybrid models improve on the performance of physics-based models on the observed nodes in terms of the RMSE calculated using analysis states and in terms of the RMSE calculated using 30 time step predictions. The improvement in performance on the latter could indicate that this improvement on the observed nodes extends to larger subsets of nodes of the system. There are, however, other possible explanations for this improvement.

Acknowledgements

I would like to thank Prof. Stephan Heyns and Prof. Johann Wannenburg, who, in their role as supervisors of this work, provided valuable insight, guidance and feedback without which this work would not have been possible. I would also like to thank Mr. De Wet Strydom for valuable discussions and for providing access to the sensor measurements that are a result of his work. My thanks also to Mr. Roelof Minnaar for providing the simulation models that were crucial to this work, and for making himself available for discussions in this regard. Additionally, I extend my thanks to Dr. Jannie Pretorius for his considerable effort in assisting me with the computational infrastructure that was used in this work. My gratitude also goes to the National Research Foundation who funded my studies in 2021. Lastly, I would like to thank my family and friends for their unwavering support during my studies.

Contents

Abstract i

Acknowledgements iii

List of symbols vii

List of acronyms ix

1 Introduction 1

1.1 Background 1

1.2 The process converter 2

1.3 Literature survey 4

1.3.1 Hybrid models 4

1.3.2 Data assimilation 10

1.3.3 Model order reduction and surrogate models 12

1.3.4 Data driven function approximation 14

1.3.5 Design of experiments 19

1.3.6 Applications in the engineering literature 20

1.4 Scope of research 23

1.5 Overview of study 28

2 Methodology 29

2.1 Surrogate models 29

2.1.1 Dimensionality reduction 30

2.1.2 Function approximation 33
2.2 Data assimilation 35
2.3 Hybrid models 37
2.3.1 Structure 37
2.3.2 Learning algorithms 38

3 Case study I: a small section of the process converter freeboard 44
3.1 Simulation model 44
3.2 Dimensionality reduction 47
3.3 Surrogate models 49
3.4 Data assimilation 54
3.5 Hybrid models 59
3.6 Discussion and conclusions 69

4 Case study II: half model of process converter freeboard 74
4.1 Simulation model 74
4.2 Sensor measurements 75
4.3 Dimensionality reduction 76
4.4 Surrogate models 78
4.5 Data assimilation with simulated measurements 81
4.6 Hybrid models with simulated measurements 87
4.7 Data assimilation with real measurements 95
4.8 Hybrid models with real measurements 96
4.9 Discussion and conclusion 101

5 Conclusion **105**

References **110**

A Additional data assimilation results for case study I **A**

List of symbols

Scalars

D	Number of inputs for design of experiments
K	Number of retained POD modes
L	Hidden layer size
M	Number of snapshots or ensemble size
n	Number of resampling steps of MPF
N	Number of training samples
o	Number of output units
O	Original system dimensionality
P	Number of likelihood tempering steps
r	Observation noise scale parameter
w	Particle weight
Greek symbols	
γ	System noise scale parameter
ζ	Likelihood tempering exponent
θ	MPF parameter
ι	POD mode selection threshold
κ	Data assimilation initialisation noise scale parameter
λ	Eigenvalue
τ	Neural network size parameter

Vectors

Roman symbols	
$\mathbf{k}$	Eigenvector
$\mathbf{m}$	Gaussian process mean
$\mathbf{p}$	Model parameter vector
$\mathbf{u}$	Snapshot or full system state
$\mathbf{v}$	Mean-centred snapshot
$\mathbf{y}$	Observation
$\mathbf{z}$	System state
Greek symbols	
$\boldsymbol{\alpha}$	POD coefficients
$\boldsymbol{\epsilon}$	System noise
$\boldsymbol{\mu}$	Observation noise

Matrices

Roman symbols	
$\mathbf{A}$	Matrix of POD coefficients
$\mathbf{C}$	Gaussian process covariance
$\mathbf{H}$	Observation matrix
$\mathbf{I}$	Identity matrix
$\mathbf{K}$	Matrix of eigenvectors
$\mathbf{Q}$	System noise covariance matrix
$\mathbf{R}$	Observation noise covariance matrix
$\mathbf{U}$	Snapshot matrix
$\mathbf{V}$	Mean-centred snapshot matrix
Greek symbols	
$\mathbf{\Lambda}$	Matrix of eigenvalues
$\mathbf{\Phi}$	POD basis

Functions

Roman symbols	
$\mathbf{f}_d$	Difference between successive states
$\mathbf{f}_f$	Flow rate
$\mathbf{f}_k$	System equation at time k
$\mathbf{f}_r$	Resolvent
$\mathbf{h}_k$	Observation function at time k
Calligraphic symbols	
$\mathcal{D}$	Data-driven model
$\mathcal{H}$	Hybrid model
$\mathcal{L}$	Loss function
$\mathcal{P}$	Physics-based model
Greek symbols	
δ	Delta function

List of acronyms

ARD	Automatic relevance determination
ASIR	Auxiliary SIR filter
DA-O	Data assimilation-observation
ETPF	Ensemble transform particle filter
EWPF	Equivalent weights particle filter
FE	Finite element
FFNN	Feed-forward neural network
LSTM	Long-short term memory
MAE	Mean absolute error
ML-DA	Machine learning-data assimilation
MPF	Merging particle filter
MSE	Mean square error
NLML	Negative log-marginal likelihood
POD	Proper orthogonal decomposition
RMSE	Root mean square error
RVM	Relevance vector machine
SIR	Sampling importance resampling
SVM	Support vector machine
SVR	Support vector regression

1 Introduction

1.1 Background

In engineering, examples of complex systems are common. Though the precise definition of a complex system is unclear, a notable feature of such systems is their unpredictability, which can lead to undesirable outcomes [1]. Access to more information about a system can be used to avoid undesirable outcomes and to operate the system more efficiently [1].

A proposed framework that can provide access to more information about a system is the digital twin. A digital twin consists of a physical entity - the physical system - and a virtual entity, which is a digital representation of the physical entity [2]. Ideally, the virtual entity should reflect the physical entity to the extent that any information that an inspection of the physical entity could reveal should also be available from the virtual entity [1]. This information can be more dense than the information provided by sensors on the physical system and can include properties of the system that cannot be directly measured during operation [3]. It can be used to predict the degradation of a system [4] or to determine the risk associated with a set of operating conditions [5], ideally without having to inspect the physical entity. The additional information about the system that is made available by the digital twin can therefore be used to better plan operations, including maintenance tasks [6].

To achieve the required correspondence with the physical entity, the virtual entity must be a high-fidelity model of the physical entity [2]. One possibility to obtain such a model is to use understanding of the physics that govern the behaviour of the system. Such physics-based models provide interpretable results and generalise well to new situations governed by similar physics [5]. For simple systems, when the physical processes involved are well understood, physics-based modelling works well. For complex systems, however, understanding of the governing physics may be limited. Additionally, choices must be made on how to model the different physical processes involved and possibly on which processes to neglect, contributing to model uncertainty [7]. A further source of error in many applications is the discretisation that is required to obtain numerical solutions. These factors result in errors when the physics-based modelling approach is applied to complex systems. In addition to having problems with accuracy, physics-based models are commonly computationally expensive, which may prevent real-time updates when applied in the digital twin framework.

Alternatively, models can be constructed by inferring the relationships between inputs and out-

puts from data. An advantage of such data-driven models is that the data from which the input-output relationships are inferred are a manifestation of both known and unknown physics [5]. Data-driven models therefore have the potential to be more accurate than physics-based models. They are also commonly less computationally expensive than physics-based models. However, data-driven models do not derive relationships based on insight, and neither do they provide insight when making predictions. This means that unlike physics-based models, they are not interpretable [5]. Further problems include the need for a large amount of training data [8] and poor generalisation to novel situations [9].

A third possibility is a hybrid model that incorporates both physics-based and data-driven features. The idea behind hybrid models is that they can provide the advantages of both physics-based and data-driven models without suffering from the drawbacks associated with either of the two approaches. In this work, the possibilities for constructing hybrid models and their application to real systems will be explored.

The systems that will be considered have some characteristics that narrow the focus of this work. The systems are dynamic with important transient effects. Furthermore, physics-based models of the systems are available, and these models are partial differential equations that are solved by numerical methods in which the domain of the system is discretised. The mesh that results from this discretisation provides high-dimensional information about the system. The observations of the system that are used in the training of hybrid models are sparse, meaning that the number of locations where the system is observed is much smaller than the number of nodes in the mesh that is used by the physics-based model. Finally, these observations are made at high frequency, so that the number of observations that are available for the training of the hybrid models is large. The application of hybrid models is studied in this work using two case studies involving the freeboard of a process converter. This system is discussed in section 1.2.

1.2 The process converter

This section serves as an introduction to the process converter that will be used in this work to investigate the application of hybrid models. A schematic representation of the process converter is shown in Figure 1. It is a top-submerged-lance furnace that processes granular matte received from smelters upstream in the production process. This granular matte is fed into the furnace through the lance together with oxygen and coal, so that iron and other metals in the matte are oxidised, leading to reactions like those described in [10]. Silica flux and additional coal are fed into the furnace via a roof port. Off-gas rich in sulphur dioxide leaves the fur-

nace at the top of the freeboard to be treated in an off-gas treatment plant. Matte and slag are periodically tapped from the furnace.

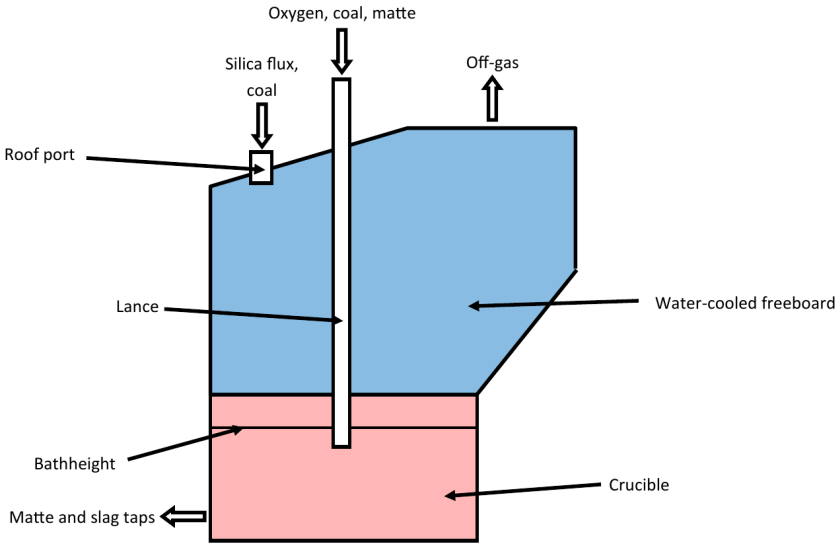

Figure 1: A schematic representation of the process converter.

The molten matte and slag are contained in a crucible that is cooled by copper coolers. Above the crucible is a water-cooled freeboard that is the focus of this study. It consists of water cooler tubes that are connected to each other by webs, as shown in Figure 2. The thermal boundary conditions on the fireside of the freeboard are highly uncertain. The freeboard is by design covered by a refractory coating, though this coating is gradually lost during operation. The condition of the refractory coating is therefore unknown. In addition, molten slag sporadically splashes onto the fireside and solidifies, first causing a spike in heat flux, then forming a thermally insulating coating. In equally sporadic fashion, the solidified slag coating is lost leaving the fireside or refractory coating directly exposed to the gases inside the furnace.

Changes in the thermal boundary conditions on the fireside of the freeboard cause the temperature field in the material of the freeboard to change. This causes changes in the thermally-induced stresses in the freeboard, which in turn result in the accumulation of fatigue damage. Additionally, by virtue of its exposure to high temperatures, the freeboard is susceptible to the actions of damage mechanisms such as corrosion and creep. Eventually, the accumulation of damage can lead to the appearance of through-thickness cracks in the water cooler tubes of the freeboard. The resulting ingress of water into the converter can cause steam explosions, making the integrity of the freeboard critical to the safe operation of the process converter.

Because of the importance of the integrity of the freeboard to the safe operation of the converter,

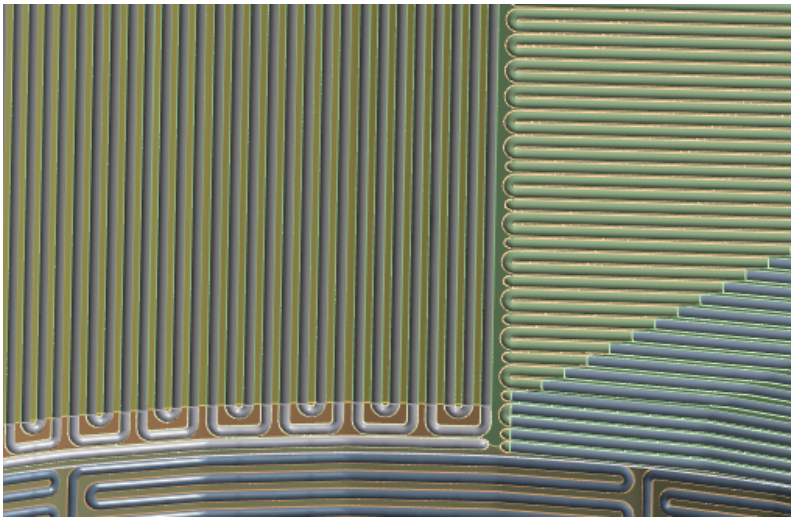

Figure 2: A close-up view of the fireside of a 3D model of the freeboard of the process converter.

real-time information about the health of the process converter freeboard that could be provided by a digital twin would be of great value. A problem that hinders the creation of a digital twin of the freeboard is that the uncertainty in thermal boundary conditions makes the modelling of the thermal behaviour of the freeboard by physics-based models difficult. As discussed in section 1.1, hybrid models that combine physics-based and data-driven models are potential enablers of digital twins, especially when the physics involved are not fully understood. This work focuses on the construction of hybrid models that predict the thermal behaviour of the process converter freeboard.

Once the temperature field in the freeboard is available, it can be used in the process of modelling the stresses in the freeboard. The stresses in the freeboard together with its temperature can then be used in damage models to track the health of the freeboard. These additional steps fall outside the scope of this work and are left to future work.

1.3 Literature survey

1.3.1 Hybrid models

As mentioned in section 1.1, hybrid models consist of a physics-driven component and a data-driven component. There are different possible objectives for constructing hybrid models, depending on the application. For example, Willard *et al.* [9] identify nine possible objectives, of which two are of special interest for this work. These are:

1. Using observations to improve on the predictions made by a physics-based model.
2. Retaining access to the predictions made by a physics-based model at lower computational cost.

While interest of this work in the first point should be apparent from section 1.1, interest in the second point may require some explanation. The investigation of hybrid models has been motivated using the digital twin concept. Models that are used in digital twins should be capable of evolving the state of the virtual entity of the digital twin in real time [2]. High fidelity numerical models such as finite element (FE) models are usually too computationally expensive to achieve this, so some steps must be taken to reduce their computational cost.

The ability of models to run in real time is not considered essential for this work. Nevertheless, a reduction in computational cost is still expected to be beneficial when attempting to achieve an improvement in the predictive performance of the physics-based model. The improvement in the predictions made by the physics-based model suggested by the first point can be achieved by correcting the error made by the physics-based model based on observations. To make this correction, it is necessary to compare the predictions made by the physics-based model to the observations in some way. When a large number of observations resulting from high-frequency observation of the system is available for training, the physics-based model will need to be evaluated for each time step between observations, possibly multiple times for each time step, so that the predictions of the physics-based model can be compared to the observations. As a result, a large number of evaluations of the physics-based model is likely to be necessary, which would make a reduced computational cost of the physics-based model desirable if not necessary.

The remainder of this section is dedicated to the first point: improving on the predictions made by a physics-based model. The reduction of the computational cost of physics-based models is considered separately in section 1.3.3.

1.3.1.1 Output of hybrid models

The issue that will be considered in this section is the question of what the final output of the hybrid model should predict. For this question, there is little distinction between hybrid and data-driven models, and approaches from the literature that have been applied to either modelling approach will be considered. To model a transient system, a mathematical description of its evolution over time is required. In the literature, this mathematical description usually

takes one of two forms. The first form describes the evolution of the system between two discrete time steps:

$$\mathbf{z}_{k+1} = \mathbf{f}_r(\mathbf{z}_k). \tag{1}$$

Here $\mathbf{f}_r(\cdot)$ is an unknown function in the case of a purely data-driven approach and partially known in a hybrid approach. In the literature, $\mathbf{f}_r(\cdot)$ is sometimes called the resolvent [11–13]. Therefore, this first form will from now on be referred to as the resolvent approach.

The second form considers the rate of change of the state of the system $\mathbf{z}$ with respect to time t as follows:

$$\frac{d\mathbf{z}}{dt} = \mathbf{f}_f(\mathbf{z}). \tag{2}$$

As before, $\mathbf{f}_f(\cdot)$ is a function that is unknown in the case of a purely data-driven approach or partially known in the case of a hybrid approach. While the first form is discrete in time, the second form is continuous in time. Note the assumption that the system is autonomous, which may be a poor assumption if the system is non-stationary. In the literature, $\mathbf{f}_f(\mathbf{z})$ is sometimes referred to as the flow rate [11, 12, 14] or the tendency [15, 16]. Here, the former terminology is adopted, partly because the use of the tendency terminology in the literature is not consistent. Some authors [15,16] use it to describe $\mathbf{f}_f(\cdot)$ in equation 2, while others [17,18] use it to describe the difference between successive states, considering the modification

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \mathbf{f}_d(\mathbf{z}_k) \tag{3}$$

of equation 1, where $\mathbf{f}_d(\cdot)$ is again partially known in a hybrid approach and unknown in a purely data-driven approach.

The use of the resolvent and flow rate approaches in the literature is summarised in Table 1. Strictly speaking, the approaches in [8,19,20] do not conform to equation 1. Because a reservoir computer is used in these approaches, the evolved state $\mathbf{z}_{k+1}$ depends on all previous states and not just on $\mathbf{z}_k$. Nevertheless, for the purposes of this work, these approaches are classified as resolvent approaches because they predict states at discrete time steps rather than a rate of change.

A comparison between the two approaches is interesting. The flow rate approach allows the model to be evaluated continuously in time, whereas the resolvent approach is limited to discrete time steps. This makes the flow rate approach preferable for assimilating data from sources with

Table 1: A summary of the use of the flow rate and resolvent approaches in a number of references.

Approach	**References**
Resolvent	[8], [19], [20], [21], [15], [16]
Flow rate	[14], [11], [12], [22], [16]

different sampling frequencies, as pointed out by Ayed *et al.* [22]. The same authors further argue that the flow rate approach is better suited to the modelling of time varying processes, which results in simpler training, but present no empirical evidence of its superiority to the resolvent approach. A disadvantage of the flow rate approach is that it requires integration over time. In terms of computational requirements, the resolvent approach may therefore be better. An interesting difference between the two approaches in the hybrid modelling context is pointed out by Farchi *et al.* [16], namely that in the flow rate approach, the outputs of the corrected model interact with the physics model during time integration. It is argued that this can make the model more expressive. For partial and noisy data, a comparison between the resolvent and flow rate approaches in [16] showed that the flow rate approach yielded better results over one model integration interval, but for predictions over longer time intervals, the two approaches were comparable.

A desirable property of the resolvent approach is described by Scher and Messori [17]. They studied the generalization performance of feed-forward neural networks when trained to model dynamic systems. One finding was that the resolvent approach resulted in a model that tended to return to regions of the state space that were explored during training, when initialised in a region that was not explored during training. This property results in stability of predictions. They also found that the approach described by equation 3 resulted in unstable predictions when initialised outside regions of the state space that were explored during training. It is unclear how problem-dependent the stability property of the resolvent approach is, and how models constructed using the flow rate approach respond when initialised in novel regions.

1.3.1.2 Application of model error correction

The next question relating to hybrid models is how the data-driven model should apply a correction to the physics-based model. A simple method for applying the correction is residual modelling [9]. In this method, the model inputs are processed by both the physics-based and by the data-driven model. The task of the data-driven model is to learn the error made by the

physics-based model for a given set of model inputs and then to apply an additive correction term to the output of the physics-based model. Some examples are [23], where a neural network is used to learn the model error, and [24] where a Gaussian process model is used for the same purpose. More recently, the approach has been used to construct hybrid models of partially observed dynamic systems [15, 16].

An alternative approach is to use the concatenation of the model inputs and the output of the physics-based model as the input of the data-driven model. The data-driven model can then again predict and correct the error made by the physics-based model, or it can predict the corrected output directly. The results obtained by Daw *et al.* [25] suggest that using the output of the physics-based component as one of the inputs of the data-driven component improves performance, but whether the prediction of model error or of the corrected output is better appears to be application specific.

1.3.1.3 Training hybrid and data-driven models of physical systems

Once the structure of a hybrid model has been defined, it must be trained using available data. The problem that is under consideration in this work is transient in nature, and involves a system that is only partially observed. Both of these factors complicate the training process. The main purpose of this section is to discuss how hybrid models of partially observed transient systems can be trained. Literature on the training of purely data-driven models of partially observed transient systems will be included in this discussion, because the training is similar.

Some approaches in the literature assume that all state variables will be observed and be available during training and to make predictions from. Examples of such approaches can be found in [8, 17, 20, 21]. A subtly different assumption is made by Lu *et al.* [26], namely that all state variables are observed during training, but that only a subset of state variables is available to make predictions from. The availability of the full state simplifies training, because the model output can be compared directly with observations.

By contrast, when the system state is only partially observed during training, a subset of the model outputs will not have corresponding observations to be compared to. Bocquet *et al.* [14] develop a general approach that can be applied to both fully and partially observed systems. To do this, they derive a loss function that can be optimised jointly for the system state trajectory and the model parameters. In a different approach, Brajard *et al.* [13] use an ensemble Kalman filter to estimate the state trajectory and then use a simple least squares loss function to

optimise the model parameters. Because a change in model parameters affects the state trajectory estimate produced by the ensemble Kalman filter, iteration over the data assimilation and parameter optimisation steps is performed. As pointed out by Bocquet *et al.* [11], the work in [13] can be interpreted as a co-ordinate descent version of the joint optimisation done in [14]. In [11], the problem of training a model of a partially observed dynamic system is considered in the expectation-maximization framework, resulting in an approach that can be used to estimate system noise parameters in addition to state trajectories and model parameters. Bocquet *et al.* [12] consider online estimation of both the system state and of the model parameters. To do this, ensemble Kalman filters are used to update both the state and the model parameters when sensor measurements become available. This approach may become impractical for models with large numbers of trainable parameters, as recognised in [12].

Gottwald and Reich [27] train random feature map models of partially observed systems using a procedure that incorporates data assimilation in the form of ensemble Kalman filters. Unlike the other approaches discussed thus far, their models are capable of predicting the observed variables only, with the sole purpose of the data assimilation being the control of measurement noise. Like in [12], the model parameters are estimated online using the ensemble Kalman filter.

The approaches to training of models of partially observed dynamic systems that have so far been discussed are limited to purely data-driven models. These can be extended to the training of hybrid models, an example being the approach discussed in Wikner *et al.* [19]. They use an iterative procedure that is similar to the procedure in [13] to train hybrid models consisting of an imperfect physics model, and a reservoir computer as data-driven component. Farchi *et al.* [15] also train hybrid models using a similar procedure. They propose that a reduction in the sensitivity of the training to the quality of the data assimilation may be achieved by training the model using time steps longer than the time step being used in the data assimilation. Their results do not show a clear benefit of this strategy. Farchi *et al.* [16] pursued an online approach similar to the approach in [12], which they applied to a hybrid model that uses a convolutional neural network with 113 parameters as the data-driven component. In terms of model performance, the online approach outperformed an offline approach iterating over successive data-assimilation and parameter optimisation steps after sufficient time had passed, though the need to control the number of trainable model parameters when using realistic physics models was recognised.

The approaches in [8, 11–17, 19] make use of simplified models such as the Lorenz63 [28], Lorenz96 [29] and Lorenz05III [7] models for numerical experiments. Other references [20, 21] consider application to real systems. None of the hybrid modelling approaches [8, 15, 16, 19] included in this literature review are applied to real systems. Therefore, there exists an

opportunity to investigate the application of hybrid models to a realistic system.

1.3.2 Data assimilation

As touched on in section 1.3.1.3, a complicating factor in the process of training hybrid models is when sensor measurements are available of only a fraction of the state variables necessary to initialise a transient simulation of the system. Additionally, sensor measurements are noisy and therefore never give a perfect representation of the quantity that they measure. Data assimilation is a framework that can be used to estimate the full state of a system from such partial and noisy sensor measurements.

Consider a system that has a state vector $\mathbf{z}_k$ at time k. The evolution of the state vector is given by the function $\mathbf{f}_k$, so that the evolved state vector at time $k+1$ is given by

$$\mathbf{z}_{k+1} = \mathbf{f}_k(\mathbf{z}_k, \boldsymbol{\epsilon}_k), \tag{4}$$

where $\boldsymbol{\epsilon}_k$ is a random noise vector known as the system noise. At time k, the system is observed, resulting in an observation vector $\mathbf{y}_k$, which is related to the system state through the observation function $\mathbf{h}_k$ as follows:

$$\mathbf{y}_k = \mathbf{h}_k(\mathbf{z}_k, \boldsymbol{\mu}_k). \tag{5}$$

Here $\boldsymbol{\mu}_k$ is a vector of random noise known as the measurement noise. The goal of data assimilation is to recursively update the posterior distribution over the system state using Bayes' rule as follows [30]:

$$p(\mathbf{z}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{z}_k)p(\mathbf{z}_k|\mathbf{y}_{1:k-1})}{p(\mathbf{y}_k|\mathbf{y}_{1:k-1})}. \tag{6}$$

The notation $\mathbf{y}_{1:k}$ refers to all measurements made from the initial time to time k. The prior distribution $p(\mathbf{z}_k|\mathbf{y}_{1:k-1})$ can be obtained using the state update equation given in equation 4, and the likelihood $p(\mathbf{y}_k|\mathbf{z}_k)$ can be obtained using the measurement equation given by equation 5.

If the noise vectors in equations 4 and 5 are normally distributed and additive, if $\mathbf{f}_k$ and $\mathbf{h}_k$ are linear functions, and if the prior distribution is Gaussian, then the posterior distribution will remain Gaussian, and the problem can be solved analytically using the Kalman filter [30]. These conditions are very restrictive and are seldom met in practice.

One method to cope with nonlinear state update and measurement equations as well as with arbitrary forms of system and measurement noise is to use particle filters. Many different types of particle filters exist. They are examples of sequential Monte Carlo methods, and were first proposed by Gordon *et al.* [31]. The particle filter proposed in [31] is commonly referred to as the sampling importance resampling (SIR) filter [30]. Particle filters represent the probability distributions in equation 6 using samples from those probability distributions. These samples are commonly referred to as particles. Consider as an example the posterior distribution, which is approximated as [32]

$$p(\mathbf{z}_k|\mathbf{y}_{1:k-1}) \approx \sum_{i=1}^{N} w_i \delta(\mathbf{z}_k - \mathbf{z}_{k,i}), \tag{7}$$

where w_i and $\mathbf{z}_{k,i}$ are the weight and location in state space of the i^{th} particle respectively and $\delta(\cdot)$ is the delta function.

Unlike Kalman filters, particle filters are capable of dealing with nonlinear state update and measurement equations, though they have problems of their own. Two common problems are weight collapse and sample impoverishment [30]. The former occurs when only one particle has a weight close to 1 and all other particles have a weight close to 0. The latter occurs when all particles collapse onto the same location. In both cases, the result is a poor representation of the probability distributions involved. Particle filters suffer from the curse of dimensionality, so that these problems become worse when the dimensionality of the problem increases. For instance, the number of particles needed to prevent weight collapse from occurring within a single time step grows exponentially with the dimensionality of the observation vector $\mathbf{y}$ [32].

A number of potential solutions to the problems of particle filters have been proposed. One of these proposed solutions is the auxiliary particle filter. A disadvantage of ordinary SIR filters is that samples are propagated to the next time step irrespective of whether they are close to the observation. Auxiliary particle filters aim to make an improvement relative to ordinary SIR filters by propagating only those samples forward to the next time step that are likely to be close to the observation [30]. Another proposal is the equivalent weights particle filter (EWPF) [33,34], which aims to avoid weight collapse by forcing all particle weights to a similar value. The ensemble transform particle filter (ETPF) [35] uses linear programming to minimise the expected distance between samples of the prior and samples of the posterior subject to the constraint that each sample of the posterior is still drawn with the correct probability. Another potential solution is the merging particle filter (MPF) [36]. This approach generates new samples by considering linear combinations of samples of the posterior distributions. The weightings in the

linear combinations are chosen to preserve the mean and covariance of the posterior distribution. The posterior distribution is thus not preserved exactly. Still another possibility is to make use of a tempered likelihood [32]. The idea of using a tempered likelihood is to more gradually move particles from the prior distribution to regions of high probability in the posterior distribution. When the tempered likelihood approach is used, a method of generating new particles in a way that preserves the posterior distribution is required [32].

Equation 6 forms the basis of filtering approaches. In this equation, only measurements up to time k are taken into account to estimate the state $\mathbf{z}_k$. Smoothing approaches, on the other hand, take future measurements into account as well to estimate $\mathbf{z}_k$. This could result in better state estimates. One approach to smoothing using Monte Carlo methods is the backward simulation approach proposed in [37]. In this approach, a time series of sample representations of the posterior distribution $p(\mathbf{z}_k|\mathbf{y}_{1:k})$ is generated using a particle filtering approach, after which smoothed samples are generated using a backward pass through time to take future measurements into account.

1.3.3 Model order reduction and surrogate models

As discussed in section 1.3.1, achieving a reduction in the computational cost of high-fidelity physics-based models serves both to make the construction of hybrid models more tractable and to better align this work with the goals of digital twins. High-fidelity physics-based models such as FE models generally require the manipulation of high-dimensional objects, such as vectors of nodal solutions. The first step in model order reduction is usually to reduce the dimensionality of these objects while still being able to reconstruct them as well as possible. Usually the governing equations are projected onto the reduced space that has been found using a dimensionality reduction technique [38]. The implementation of such an approach requires access to the source code of the numerical method that is being used, which is often not possible. As a result, the focus here is on non-intrusive model order reduction methods. This section is organised by two important tasks that must be performed to achieve model order reduction. Section 1.3.3.1 discusses dimensionality reduction, while section 1.3.3.2 discusses how the response of the model can be predicted after dimensionality reduction has taken place. The focus will be on approaches that are applicable to transient models. The models that are obtained through model order reduction are referred to in this work as surrogate models.

1.3.3.1 Dimensionality reduction

Proper orthogonal decomposition (POD) is a method that can be used to reduce the number of degrees of freedom (the variables required to describe the state of the model) of models while maintaining their accuracy [39]. The principle of the method is to project the high-dimensional model state onto an optimal co-ordinate system of lower dimensionality [40]. The co-ordinate system of lower dimensionality is optimised to retain as much information of the full, high-dimensional model state as possible. This optimisation is based on snapshots of the model state [41]. These snapshots are usually obtained from sample solutions of the model [38, 40, 42]. Dimensionality reduction by POD has been applied in the construction of non-intrusive reduced-order models for fluid flow problems [38,40,42]. An application to the construction of a reduced-order model of a thermal FE model is described in [39]. Further discussion of POD is provided in section 2.1.1.

One of the problems with POD is that the projection is linear, which may limit its ability to utilise the structure that is present in data. Some attempts have been made at extending POD to make it capable of nonlinear dimensionality reduction. An example is the Isomap algorithm [43]. A problem with this approach is that reconstruction to the full dimensionality is challenging, and reconstruction is critical for model order reduction since all insight from the model is derived from the high-dimensional space.

A possible alternative for nonlinear dimensionality reduction is to make use of autoencoders, in which the hidden layers of neural networks learn lower-dimensional representations of the input units [44]. Another possibility is the use of variational autoencoders [45], which attempt to learn the distribution of the lower-dimensional representation, as well as the distribution of the reconstruction given a lower dimensional representation. This makes uncertainty quantification possible. Autoencoders usually use densely connected neural networks, which may lead to a prohibitively large number of trainable parameters. A possible solution is to use convolutional layers in the dimensionality reduction and expansion [46, 47]. Convolutional layers extract localized features from data [48], so careful consideration of how to arrange objects such as nodal solution vectors for presentation to convolutional layers is necessary. This is potentially a challenging problem.

1.3.3.2 Prediction

This section discusses how predictions of model response can be made non-intrusively. A possible approach is presented by Walton *et al.* [40]. They deal with transient problems by performing POD twice, once on the normal snapshot matrix, and then again on the time-varying POD coefficients of each POD mode. This method is able to approximate the model response for new parameter values by interpolating between cases with known response. The approximation of model response is obtained for the same discrete time steps as those present in the known cases. This may hinder the application of the method to situations where continuous predictions are necessary, although it may be possible to view initial conditions as a parameter.

A more natural way to deal with situations requiring continuous predictions is to model the evolution of the POD coefficients using a data-driven function approximation technique. This has been done using Taylor series and the Smolyak sparse grid method [38], as well as using radial basis functions [42]. In principle, it appears to be possible to use any data-driven function approximation technique to model the evolution of POD coefficients.

1.3.4 Data driven function approximation

In two of the sections so far, the need to approximate a function based on observations of the input-output relationship of the function has been encountered. The discussion of model error correction in section 1.3.1.2 implies the need for a method to approximate either the error made by a physics-based model or the corrected output of a physics-based model for some inputs based on observations of the correct output. Additionally, some of the techniques mentioned in section 1.3.3.2 require the approximation of the input-output relationships of the physics-based model itself. This section contains a brief discussion of three function approximation techniques, namely neural networks, Gaussian processes and support vector machines (SVMs).

1.3.4.1 Artificial neural networks

Artificial neural networks are data-driven models that consist of a layered structure. The input layer receives external inputs to the model, and the prediction made by the model is provided at the output layer. Between the input and output layers is an arbitrary number of hidden layers. Each layer consists of a number of units. The number of input and output units is constrained by the dimensionality of the data, but each hidden layer can consist of an arbitrary number of

hidden units. Signals are sent between units by means of weighted connections, and the inputs arriving at each hidden unit are usually passed through a nonlinear activation function to give the output of the hidden unit. If the input layer is seen as the bottom of the network and the output layer is seen as the top of the network, then a simple type of neural network is the feed-forward neural network (FFNN) in which each layer receives inputs only from layers below it and sends outputs only to layers above it [49].

The weights of the connections between the units of neural networks are parameters that must be adjusted so that the output of the neural network matches the target output as closely as possible. An error function must be defined to compare the output of the network to the desired output. If the error function and the activation functions of the network are differentiable, then it is possible to find the derivatives of the error function with respect to the weights of the network by using backpropagation of errors [49].

With the derivatives available, it is possible to use gradient-based optimisation to optimise the weights. The error functions used in neural networks can be complex and contain multiple local minima, and as a result, many specialised gradient-based optimisation algorithms have been developed for the optimisation of neural networks. Examples are stochastic gradient descent [50], momentum [51], RPROP [52], Adagrad [53] and Adam [54].

Neural networks are very flexible function approximators. An FFNN with a single hidden layer, and hidden units that use a bounded, continuous and nonconstant activation function can approximate any function with arbitrary accuracy provided the number of hidden units is sufficient, and FFNNs are therefore referred to as universal approximators [55]. Unfortunately, it is usually not possible to determine *a priori* the architecture of the neural network that will be able to learn the underlying function for a particular application. In addition to the architecture of the neural network, additional hyperparameters such as the type of activation function used and parameters related to the optimiser may need to be chosen. Some optimisation methods such as genetic algorithms have been proposed for the selection of neural network architectures [56] as well as other hyperparameters [57]. These approaches are computationally expensive, because a separate neural network model must be trained and evaluated for each hyperparameter combination.

Alternatively, analytical insight may be used. For instance, Huang [58] constructs an FFNN with two hidden layers that is able to learn every training sample with arbitrary accuracy. The number of hidden units in the first hidden layer L_1, and the number of hidden units in the second

hidden layer L_2 of this FFNN are given respectively by

$$L_1 = \sqrt{N(o+2)} + 2\sqrt{\frac{N}{o+2}} \tag{8}$$

and

$$L_2 = o\sqrt{\frac{N}{o+2}}. \tag{9}$$

Here o is the number of output units and N is the number of training samples. Huang [58] also provides a method of determining the parameters of the neural network so that all training samples are approximated arbitrarily well. It is recognised, however, that such a network is overfit to the training samples and is unlikely to generalise well. Nevertheless, the architecture may be a useful starting point before further tuning takes place. A further insight that may be useful to architecture selection is that increasing the number of hidden units by adding more hidden layers increases the expressiveness of FFNNs more than increasing the number of hidden units per layer [59]. Neural networks with multiple hidden layers are commonly referred to as deep neural networks.

Neural networks can deal with multivariate regression problems through the use of multiple output units, though it is not necessarily the best approach to consider all output variables using a single network. For example, Du and Xu [60] propose the hierarchical deep neural network for multivariate regression problems. This approach splits the target vector into multiple subvectors, and trains a separate network for each subvector. They argue that this makes the learning problem easier and present results suggesting an improvement over deep neural networks that predict the entire target vector.

1.3.4.2 Gaussian processes

The artificial neural networks discussed in section 1.3.4.1 make predictions based on parameters learned from training data. The training data itself is not required to make predictions. Other methods use linear combinations of the same kernel function evaluated at all training samples to make predictions [61]. One such kernel method is Gaussian process regression.

In Gaussian process regression, function values are regarded as random variables. A Gaussian process is a finite number of such random variables that have a joint Gaussian distribution [62]. A collection of function values $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_M)]^T$ evaluated at the collection of

points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ is therefore distributed according to

$$\mathbf{f} \sim \mathcal{N}(\mathbf{m}, \mathbf{C}),$$

where $\mathbf{m} = [m(\mathbf{x}_1), m(\mathbf{x}_2), \ldots, m(\mathbf{x}_M)]^T$ is found from the mean function $m(\cdot)$, and the covariance matrix $\mathbf{C}$ has entries $C_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ found from the covariance function $k(\cdot, \cdot)$ [61,62]. Gaussian process regression involves inferring the mean and covariance functions from data [62].

It is common to set the mean function to zero and to perform regression based solely on the covariance function [62]. Different forms of the covariance function are possible, some of which are provided in Table 2. The Gaussian process regression problem consists of finding appropriate values for the hyperparameters of the covariance function rather than of adjusting large numbers of model parameters as is the case for artificial neural networks. An advantage of Gaussian process regression is that analytical expressions for the marginal likelihood are available, and that these expressions are differentiable with respect to the model hyperparameters [62]. Gradient-based optimisation can therefore be used to find the optimal hyperparameters given the available training data. Unfortunately, local optima of the loss function are not necessarily a global optimum [62], so it is possible for gradient-based optimisation of the hyperparameters to converge to poor solutions.

Table 2: Different covariance functions and their hyperparameters. Functional forms are taken from the documentation of the GPML Matlab Code version 4.2 [63], which is maintained by the authors of [62]. Functional forms are given in terms of $\rho = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{P}^{-1} (\mathbf{x}_i - \mathbf{x}_j)$.

Name	**Functional form**	**P**	**Hyperparameters**
Square exponential	$\sigma_f^2 \exp(-\frac{1}{2}\rho)$	$l^2\mathbf{I}$	σ_f, l
Matérn, $\nu = 1/2$	$\sigma_f^2 \exp(-\rho)$	$l^2\mathbf{I}$	σ_f, l
Matérn, $\nu = 3/2$	$\sigma_f^2(1 + \sqrt{3}\rho)\exp(-\sqrt{3}\rho)$	$l^2\mathbf{I}$	σ_f, l
Matérn, $\nu = 5/2$	$\sigma_f^2(1 + \sqrt{5}\rho + (\sqrt{5}\rho)^{2/3})\exp(-\sqrt{5}\rho)$	$l^2\mathbf{I}$	σ_f, l
Square exponential, ARD	$\sigma_f^2 \exp(-\frac{1}{2}\rho)$	$\mathrm{diag}(l_1^2, l_2^2, \ldots, l_D^2)$	$\sigma_f, l_1, l_2, \ldots, l_D$

1.3.4.3 Support vector machines

As discussed in section 1.3.4.2, Gaussian process regression models are examples of kernel methods that make predictions for new points based on all of the training samples. This can become computationally expensive for large training sets. Like Gaussian process regression

models, SVMs are kernel methods, but they use only a subset of the training samples to make predictions [61].

SVMs can be applied to both classification and regression tasks, the latter capability being relevant for the function approximation tasks considered in this work. In a two-dimensional plane, a support vector regression (SVR) model can be thought of as a narrow tube. The model is fit to the training data by softly penalising points that lie outside the tube. The only training samples that contribute to predictions for new data points are those that lie on the boundary of the tube or outside the tube [61].

The optimisation problem that must be solved during SVR training is a quadratic programming problem with linear constraints, resulting in the desirable property that a local optimum is also a global optimum [61]. This means that it is not possible for the optimisation to converge to a poor local optimum as is possible in neural network optimisation. Although techniques exist to solve general quadratic programming problems [64], specialised algorithms such as sequential minimal optimisation [65] have been developed for SVR.

When compared to Gaussian process regression models, SVR models usually have the advantage of lower computational cost for predictions due to the fact that predictions rely on only a subset of the training samples rather than on the entire training set. A disadvantage relative to Gaussian process models is that no analytical expression relating model hyperparameters to the marginal likelihood or to the model error exists. As a result, hyperparameter selection is usually done using k-fold cross-validation [66], which is computationally expensive. A further disadvantage is that the subset of training samples on which predictions depend can become large when the training set is large, resulting in high computational cost [67].

The relevance vector machine (RVM) [67] is a technique that makes use of Bayesian statistics to address some of the limitations of SVR. The hyperparameters of RVMs can be estimated by optimisation of a loss function [67], and it is even possible to find the optimal values of many of the hyperparameters analytically given the values of all other hyperparameters [68]. This makes the constructive approach for training RVMs proposed by Tipping and Faul [69] possible. An additional advantage of RVMs is that they typically result in much sparser models than SVR, which reduces their computational cost when making predictions [61].

1.3.5 Design of experiments

The training of data-driven models requires samples of the input-output relationship of the function which the model should approximate. Which training samples are available when training hybrid models is constrained by the states that are visited by the physical system and by the sensor measurements that are collected. For the construction of surrogate models, however, the choice of which inputs and outputs of the physics-based model to sample is free. This section is dedicated to a discussion on how to choose which training samples are generated, a process commonly known as design of experiments.

Latin hypercube designs are popular solutions to the design of experiments problem [70]. Latin hypercube designs divide each dimension of the input space into a number of levels, and are arranged so that each level of each dimension is sampled only once. An advantage of Latin hypercube designs is that the number of training samples generated is equal to the number of levels, so the number of experiments that must be run can be finely controlled according to the availability of computational resources. The number of training samples is thus independent of the number of input dimensions. By contrast, for a grid-based sampling approach, the number of training samples is N^D, where N is the number of training samples and D is the number of input dimensions. It has also been argued that because no two points of a Latin hypercube design share a value of any input dimension, the amount of information contributed by each new point is higher when compared to designs in which points do share input dimension values [71].

For a given number of dimensions and levels, many different Latin hypercube designs are possible. Usually, the designs are optimised to promote good exploration of the design space. The optimisation of Latin hypercubes is a challenging combinatorial optimisation task with a large design space, and it is unclear which loss function best characterises the design space exploration [70]. An example of an optimisation technique that has been used to optimise Latin hypercube designs was proposed by Chen *et al.* [71] and is a specialised version of the particle swarm optimisation technique.

Notably, the non-intrusive model order reduction techniques presented in [38, 42] do not use special design of experiments techniques to obtain training samples for their surrogate models. Rather, they use the same snapshots that were used for dimensionality reduction. This could limit the performance of the resulting surrogate models because they are limited to the trajectories represented in the original snapshots. An improvement may therefore be possible if the application of design of experiments techniques allow better exploration of different model initialisations, especially if the system reaches steady state and only a small portion of possible

states are therefore visited in the snapshot generation process.

1.3.6 Applications in the engineering literature

The purpose of this section is to document some applications in the engineering literature of concepts discussed so far in the literature review. This will aid in understanding how the application of these concepts in an engineering context can make a contribution to the literature. The focus will be on the use of observations to improve model performance, on the use of sequential data assimilation, on the use of surrogate models and on the use of hybrid models in applications similar to the one considered in this work, preferably making use of FE models.

Yue *et al.* [72] train long-short term memory (LSTM) networks to predict the thermally induced deflection of a cable-stayed bridge based on partial information about the temperature field in the bridge. Though they make use of physics-based knowledge in the training process, this extends only to feature selection, rather than to using physics-based knowledge in the process of making predictions. As will be discussed in section 1.4, this is not the understanding of a hybrid model used in this work.

Monte Carlo simulations are used for the calibration of fatigue crack growth models by Jiang *et al.* [73] in an application of the digital twin framework to predict the residual life of bridges. While high-fidelity FE models are used in the work, only the computationally inexpensive fatigue crack growth models that are used require multiple evaluations. Therefore, no surrogate models are considered.

In another model calibration approach, Gomez *et al.* [74] use Bayesian inference to estimate the parameters of structural FE models. The structural response is observed using computer vision, and the parameters relating to the load on the structure are inferred from these measurements. Sparse Bayesian learning is used by Huang *et al.* [75] to infer changes in the stiffness parameters of structures, thus detecting damage in the structures. Neither in [74] nor in [75] are surrogate models used to reduce the computational cost of the parameter estimation.

Surrogate models are considered by Nikolopoulos *et al.* [76], to accelerate Monte Carlo simulations performed using nonlinear transient structural models. They use a convolutional autoencoder for dimensionality reduction and a neural network to predict the latent variables from the parameters of the model. The surrogate models are capable of predicting the entire spatial solution field, but cannot be used for different initialisations, and the output of the model is constrained to the same discrete time steps as in the original training set. This makes the approach

similar to the one presented by Walton *et al.* [40].

Both Ramancha *et al.* [77] and Lin *et al.* [78] use surrogate models to reduce the computational cost of parameter estimation. In [77] the parameters of an FE model of a miter gate are estimated using Bayesian inference. The parameter estimation problem is approached differently in [78], where the multi-verse optimiser is used to minimise the least square error between model and observations.

An example of the use of data assimilation in the engineering literature is given by Azam and Mariani [79]. They propose using data assimilation to estimate model parameters as well as to track the reduced state of a transient model obtained by POD. A reduced order model is obtained by projection of the governing differential equations onto the reduced subspace obtained by POD. An interesting aspect is that the POD modes are updated over time, which could result in improvements in the dimensionality reduction.

The application of hybrid models as understood in the current work has been documented in the engineering literature, commonly as part of approaches to the model calibration problem. An example is given by Yucesan *et al.* [80], who estimate the parameters of a physics-based model and train a neural network model to correct the error of the physics-based model in a joint optimisation. Their hybrid model is applied to the prediction of a steady-state process and therefore no sequential estimation of system states is required like for the transient process considered in this work.

Another example of the use of hybrid models as part of the model calibration problem is given by Ramancha *et al.* [81]. They account for the model error of linear dynamic systems using a Gaussian process, thereby creating a hybrid model. They use physics-based knowledge in the form of a physics-based model and in the form of a covariance function of the Gaussian process that is designed for the physics involved in the problem. The hybrid model is applied to improve the identification of model parameters during model calibration by Bayesian inference. The error correcting term depends only on time, and a known initial system state is used with no method of updating the state based on measurements. While this is sufficient for the model calibration that is performed, it does restrict the applicability of the method to cases where the initial state is uncertain, where long measurement time series exist and where the model must generalise to new situations with different initial conditions. It is also assumed that all properties of the system to which an error correction term is applied are observed.

Yet another use of hybrid models in model calibration is documented by Higdon *et al.* [82], who use Gaussian process models to account for model discrepancy in implosion tests. They reduce

the dimensionality of simulation outputs and experimental results using POD, and construct a surrogate model of the simulation model using a Gaussian process. A disadvantage of the approach used is that it cannot easily be extended to applications that require the ability to make continuous predictions, since it relies on the time grid that was used in the simulation and experiments. An interesting feature of the approach is that different dimensionality reduction is used for the simulation model and for the discrepancy model. The approach from [82] is applied by Higdon *et al.* [83] to a thermal problem and by Kumar *et al.* [84] to transient aerodynamic and heat transfer problems.

While the previously discussed applications of hybrid models have been as components of solutions to model calibration problems, Pawar *et al.* [85] focus specifically on hybrid models. They construct reduced-order models using POD and Galerkin projection and use an LSTM network to correct the POD coefficients based on observations. A form similar to equation 2 is used, making the approach potentially suitable for continuous assimilation of observations. However, the method is applied to cases with known initial conditions and to full observations that are generated from simulations or analytical solutions. This is a shortcoming that is recognised in [85]. The method can therefore not be applied in the same way when observations are partial and the initial conditions are uncertain, like in this work.

Table 3: A summary of methods applied in the engineering literature and how they are studied. The columns of the table correspond to the following: I - use of observations to improve the model, II - use of sequential data assimilation, III - use of surrogate models or model order reduction, IV - a hybrid model is constructed, V - experiments conducted using simulated data, and VI - experiments conducted using real data.

Reference	**I**	**II**	**III**	**IV**	**V**	**VI**
[72]	●					●
[73]	●					●
[74]	●					●
[75]	●				●	●
[76]			●		●	
[77]	●		●		●	
[78]	●		●			●
[79]	●	●	●		●	
[80]	●			●	●	
[81]	●			●	●	
[82]	●		●	●	●	●
[83]	●		●	●		●
[84]	●		●	●	●	
[85]	●		●	●	●	

A summary of the application of different methods in the literature, that are of interest for this work is provided in Table 3. Also summarised in Table 3 is whether the methods are studied using simulated or real data. As the summary shows, none of the literature references documented in this section combine all methods considered in Table 3. Of the surrogate models used in the literature documented in this section, those used in [77] are for steady-state problems and those used in [76,78,82–84] deal with transient problems in manner that restricts the surrogate models to the same time grid used by the simulation models. Only in [79] is a reduced-order modelling approach used that does not suffer from this restriction, though unlike the other approaches it is not a data-driven surrogate modelling approach.

Another important point that is not clear from the summary, is that the hybrid modelling approaches in [80–84] are either applied to steady-state problems, or to transient problems in a way that predictions can be made only on the same time grid that was used in the computer experiments used to construct surrogate models or on which experimental observations are available. Furthermore, uncertainty in initial conditions must be dealt with by considering the initial conditions as parameters of the model. This makes these approaches poorly suited to the problem considered in this work, where observations are continuously available, and initial conditions are likely to be high-dimensional. The approach in [79] can deal with uncertainties in initial conditions through the use of data assimilation, but it does not involve hybrid models. Only the approaches in [79,80] can deal with partial observations, and none of the hybrid model approaches considered in this section make use of data assimilation during training in the way discussed in section 1.3.1.3.

1.4 Scope of research

Before defining the scope of the research that will be conducted in this work, it is useful to consider its context, which is illustrated in Figure 3. In the solution of the overall problem to which this research seeks to contribute, there exists some physical system, the behaviour of which is governed by physics which is partially known. The known physics is used to construct a physics-based model capable of approximating the response of the system. Sensors are placed on the system to observe the actual response of the system, which is a manifestation of both the known and unknown physics. The sensor measurements are used in the training of a data-driven model, which is to be combined with a physics-based model to construct a hybrid model. Like the physics-based model, the purpose of the hybrid model is to predict the response of the system, though ideally it does so with greater accuracy than the physics-based

model. These predictions of the system response can be used in conjunction with the original sensor measurements in a damage model. The damage model provides information about the condition of the system, which in conjunction with the predicted response of the system can be used for the purposes of control or decision making.

In this work, the research is restricted to the aspects highlighted in green broken lines in Figure 3. The research excludes the processes of accumulating knowledge of the physics involved and of constructing physics-based models of systems based on such knowledge. It likewise does not consider aspects related to observation by sensors, such as sensor placement. The use of the predicted response of the system in damage models and for control and decision making is also excluded from the research. Rather, the focus is on using the information that is available from sensors to improve on the predictions of the response of a system that are made by a physics-based model that is already available.

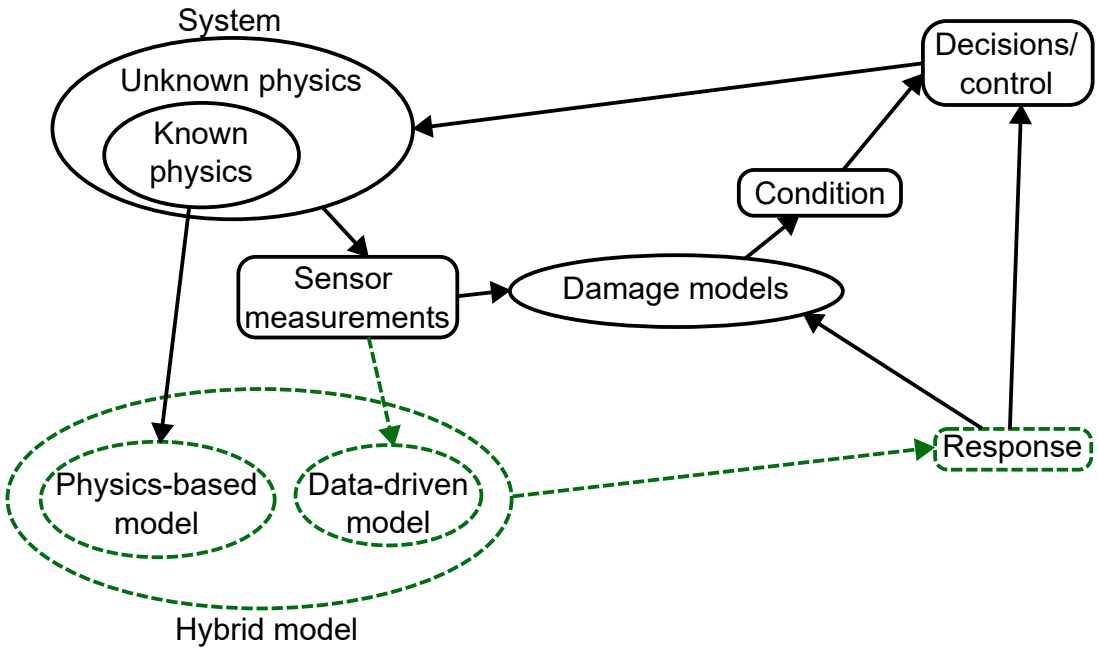

Figure 3: The context of the research conducted in this work, with the aspects on which the research focuses highlighted in green broken lines.

The systems that will be studied in this research are transient systems that are described by physics-based models in the form of partial differential equations that are solved numerically. Specifically, physics-based models in the form of FE models will be considered. Typically, FE models of real engineering systems are not sufficiently accurate for applications such as digital twins in which the aim is to faithfully represent changes in the system so that meaningful decisions regarding the operation of the system can be made. The cause for this lack of

accuracy is that, as discussed previously, such models are usually constructed using only a partial understanding of the physics involved. This makes FE models of real engineering systems an interesting potential application area of the idea of improving physics-based models using observations made by sensors.

As has already been alluded to in Figure 3 and the discussion surrounding it, this work investigates the use of hybrid models as a means of leveraging observations made by sensors to improve on the predictions made by physics-based models. In section 1.3.1, it was found that an interesting application of hybrid modelling is to correct the error made by a physics-based model by employing a data-driven model to learn this error. This is what is understood as a hybrid model in this work.

The training of an error-correcting hybrid model requires some form of comparison between the predictions made by the physics-based model and observations made by sensors. The number of such comparisons is likely to be large, as discussed in section 1.3.1, which motivates the need for reducing the computational cost associated with evaluation of the physics-based model. This is especially true for large, computationally expensive FE models. A further motivation is that an error-correcting hybrid model requires the prediction made by the physics-based model to make its corrected prediction. Therefore, the computational cost of the physics-based model has direct implications for the suitability of hybrid models for applications such as digital twins in which real-time performance is required. A reduction in computational cost can be achieved by constructing a surrogate model of the physics-based model. This is a data-driven model that replicates the predictions made by the physics-based model at reduced computational cost. Such surrogate models can themselves be interpreted as hybrid models, as discussed in section 1.3.1, though this is not the understanding of a hybrid model in this work.

Because a reduction in the computational cost of physics-based models is potentially beneficial for both the training and application of hybrid models, the construction of surrogate models of physics-based models will be considered in this work. Models of transient systems require knowledge of the initial system state to predict the evolution of the system state, as suggested by equations 1 and 2. In this work, the system state is considered to be the nodal solution vector of a FE model, which is a high-dimensional object. A first step in the construction of a surrogate model is therefore to pursue methods of reducing the dimensionality of the system state. In this work, dimensionality reduction will be accomplished by POD, which avoids the potential difficulties of the nonlinear alternatives to POD discussed in section 1.3.3.1.

In addition to dimensionality reduction, a second problem that must be addressed in the con-

struction of surrogate models is that of learning the input-output relationship of the physics-based model. This work considers only non-intrusive surrogate modelling techniques which entail the use of one of the data-driven function approximation techniques discussed in section 1.3.4. It is unclear which of these techniques is best suited for the surrogate modelling task. Therefore, surrogate models will be trained using each of these approaches. Their suitability as surrogate models will be evaluated in terms of how accurately they replicate the outputs of the physics-based model and in terms of their computational cost.

Another aspect of the training of a hybrid model is deciding on the structure of the hybrid model in terms of how physics-based and data-driven models are integrated. As discussed in section 1.3.1.2, it has been found that using both the inputs and outputs of the physics-based model as inputs of the data-driven component is beneficial for the performance of hybrid models. Such an arrangement also makes it possible to predict the corrected output directly instead of predicting the error made by the model, though this does not necessarily impact performance. While the structure of hybrid models may be important for the performance of hybrid models, it is not a focus of the research conducted in this work. Only a single structure will therefore be considered in this work, and this structure is discussed further in section 2.3.1.

Before a hybrid model can be trained, the type of model constituting its data-driven component must be decided. An important feature of the hybrid model training problem that is considered in this work is that sensor measurements are expected to be continuously available, and so the number of training samples that can possibly be considered is expected to increase over time. Therefore, approaches that make predictions based on training samples are not suitable, as the set of points required to make predictions will increase over time, leading eventually to intractable computational cost. This is likely true even of the sparse SVM and RVM approaches discussed in section 1.3.4.3, and is certainly true of the Gaussian process approach discussed in section 1.3.4.2. In contrast, the neural network approach discussed in section 1.3.4.1 makes predictions based on parameters learned from the training set, and the number of parameters can be kept constant if the number of training samples is increased. Neural networks are therefore the most suitable of the data-driven models discussed in section 1.3.4. Only hybrid models with neural network data-driven components will be used in this work.

Another question that must be addressed when training hybrid models is the nature of the output of the hybrid model. The discussion in section 1.3.1.1 found that there are two possibilities that are commonly used in the literature. The first is referred to in this work as the resolvent approach and involves predicting the updated system state after a discrete time step has taken place. The second approach is referred to in this work as the flow rate approach, which involves predicting

the rate of change of the system state. This second approach requires time integration to update the system state. The question of the output of the hybrid model is not considered to be a primary focus of this work, and although the flow rate approach will be used in the construction of some of the surrogate models, the resolvent approach will mostly be used.

Another tool used in this work is data assimilation. It can be used to estimate the full state of a system using a model of how the system evolves together with observations of the system. Interestingly for this work, the observations do not need to be of the full system. While data assimilation can itself be seen as a method of using the information from observations to make predictions using physics-based models, it is also useful for the training of hybrid models. Section 1.3.1.3 discusses different methods of using data assimilation in the training of hybrid models. While online learning has shown promise, there are some challenges to its application, notably the dimensionality of the parameter space of the data-driven component of the hybrid model. For this reason, only offline approaches will be considered in this work. Furthermore, this work will consider only the co-ordinate descent version of combining data assimilation and machine learning, rather than a joint optimisation over the system state trajectory and the parameters of the data-driven component.

When the co-ordinate descent version of combining data assimilation and machine learning is used, a data assimilation algorithm is required. Particle filters will be used in this work for this purpose. The performance of various of the particle filter approaches discussed in section 1.3.2 will be evaluated and compared. Based on this comparison, a decision will be made regarding which approach is most suitable for use in the training of hybrid models.

There are differences between the research conducted in this work and the work that is presented in the literature surveyed in section 1.3. These differences arise mainly as a result of the application of the hybrid modelling approach discussed in this section to a real engineering problem involving a complex system. The approach has previously been applied to simplified simulated problems for which computationally inexpensive simulation models are available. Limitations in computational resources make it necessary to construct surrogate models to apply the approach to the process converter freeboard described in section 1.2. This is an important difference to the hybrid model training approaches for transient systems that have been documented in section 1.3.1.3 which use the physics-based model directly. The application of the methodology to an engineering problem does itself result in a useful contribution to the engineering literature. The hybrid modelling approaches that were reviewed in section 3 are poorly suited to transient applications where observations are available continuously and where simulations cannot be performed on a well-defined time grid and with well-known initial conditions.

Data assimilation is used in [79] which overcomes these problems, but it is applied for model calibration and not for the training of hybrid models. Relative to the literature considered in section 1.3, this work therefore extends the use of data assimilation in the training of hybrid models to situations in which the use of surrogate models is required for computational reasons, while introducing the use of data assimilation in the training of hybrid models to the engineering literature.

1.5 Overview of study

The remainder of this work is divided into four sections. In section 2, a methodology is presented that describes how the training of hybrid models is approached for the remainder of the work. This methodology is then applied in two case studies. Case study 1 is presented in section 3. It is intended as a preliminary investigation into the application of the methodology, and uses a model of a small region of the process converter freeboard. Case study 2 in section 4 then investigates the application of the methodology using a much larger model in the form of a half model of the process converter freeboard. Whereas the investigations that are performed in case study 1 use only simulated data, both simulated and real data are used in case study 2. The conclusions that are drawn from the investigations performed in the two case studies are presented in section 5.

2 Methodology

The broad methodology that is proposed for training hybrid models is illustrated in Figure 4. Computer experiments are conducted to obtain samples of the predictions made by the physics-based model of the system for different inputs. These samples are then used to train a data-driven model to act as a surrogate model of the physics-based model. The purpose of this surrogate model is to reduce the computational cost of the physics-based model. The surrogate model is combined with a data-driven model to form a hybrid model, which is to replace the physics-based model as the model that is used to predict the response of the system. The system is partially observed, and for this reason data assimilation is used to obtain a sequence of estimated system states with which training can be conducted. This training set is used to train the data-driven component of the hybrid system model. The process of training the data-driven component of the hybrid model changes the system model, which in turn means that the sequence of states that will be estimated using data assimilation will change. As a result, it may be necessary to iterate over the steps of training set generation and hybrid model training. This possibility is indicated in Figure 4.

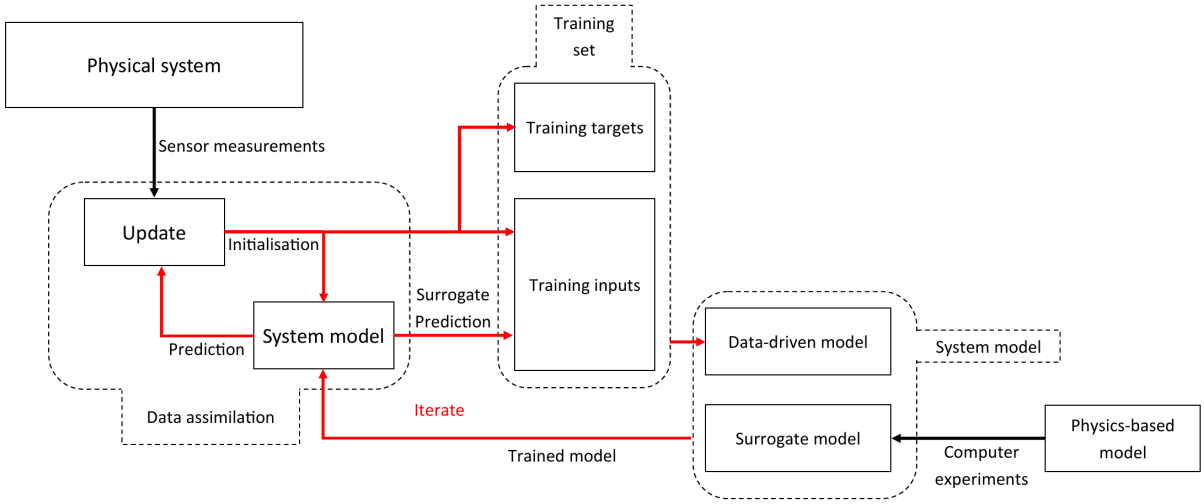

Figure 4: A schematic of the hybrid model training process.

2.1 Surrogate models

A schematic of the surrogate modelling approach that is used to reduce the computational cost of the physics-based model is given in Figure 5. The full system state is reduced by means of a dimensionality reduction technique. This reduced system state is then used together with

other variables such as different boundary condition parameters to predict the evolution of the reduced state over a time interval. The evolved reduced state is predicted using a data-driven model that is trained using samples of the predictions made by the physics-based model. Finally, the evolved reduced state is reconstructed to the full dimensionality to provide a prediction of the evolved full system state.

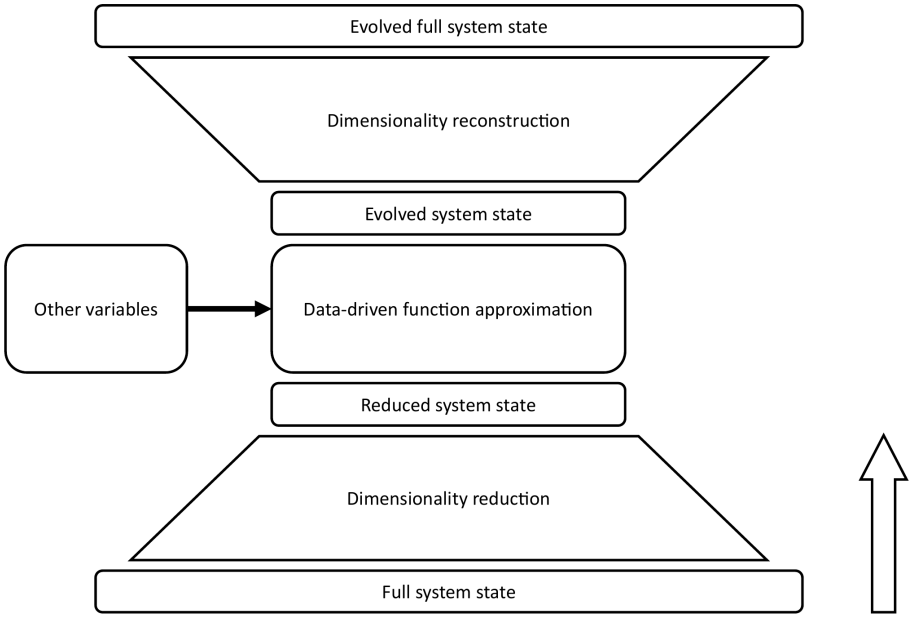

Figure 5: A schematic of the surrogate modelling approach.

The remainder of this section discusses two of the components shown in Figure 5. Section 2.1.1 discusses the dimensionality reduction used in this work, and data-driven function approximation in the context of the surrogate models is discussed in section 2.1.2.

2.1.1 Dimensionality reduction

In this work, POD will be used to perform dimensionality reduction. The choice of POD for this task is based on its simplicity, the ease with which it allows reconstruction, and the existence of fewer design considerations when compared to approaches that allow nonlinear dimensionality reduction. For a brief discussion of some nonlinear approaches, see section 1.3.3.1. The remainder of this section aims to describe the implementation of POD used in this work.

Suppose M snapshots $\mathbf{u}$ of a model solution are available. Each $\mathbf{u}$ is an O-dimensional column vector. In this work, the snapshots $\mathbf{u}$ are of the nodal solution vector at discrete time steps. The

ensemble of M snapshots is collected in an $O \times M$ snapshot matrix $\mathbf{U}$:

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_M \end{bmatrix}.$$

From this snapshot matrix, a mean centred snapshot matrix $\mathbf{V}$ is obtained by subtracting the ensemble mean of snapshots, so that

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_M \end{bmatrix},$$

where

$$\mathbf{v}_i = \mathbf{u}_i - \frac{1}{M} \sum_{j=1}^{M} \mathbf{u}_j$$

Following the method in [39,41], the following eigenvalue problem is then posed:

$$\mathbf{B}\mathbf{k} = \lambda \mathbf{k}, \tag{10}$$

where $\mathbf{B} = \mathbf{V}^T\mathbf{V}$, $\mathbf{k}$ is an eigenvector of $\mathbf{B}$, and λ is an eigenvalue of $\mathbf{B}$. The M eigenvectors of $\mathbf{B}$ can be collected as columns of the matrix $\mathbf{K}$, and the M eigenvalues can be used to construct the diagonal matrix

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_M \end{bmatrix}.$$

These can be used to find a matrix $\mathbf{\Phi}$, which is referred to in [39] as the POD basis:

$$\mathbf{\Phi} = \mathbf{V}\mathbf{K}\mathbf{\Lambda}^{-1/2},$$

where the notation $\mathbf{\Lambda}^{-1/2}$ denotes that $\mathbf{\Lambda}^{-1/2}\mathbf{\Lambda}^{-1/2} = \mathbf{\Lambda}^{-1}$. The mean-centred snapshots are then projected into M-dimensional space:

$$\mathbf{A} = \mathbf{\Phi}^T\mathbf{V}.$$

The entries of the matrix $\mathbf{A}$ are sometimes referred to as POD coefficients [38, 39]. The POD coefficients can be reconstructed to the O-dimensional space by premultiplying $\mathbf{A}$ with the POD

basis $\mathbf{\Phi}$:

$$\mathbf{\Phi A} = \mathbf{\Phi \Phi}^T\mathbf{V} = \mathbf{VK\Lambda}^{-1/2}\mathbf{\Lambda}^{-1/2}\mathbf{K}^T\mathbf{V}^T\mathbf{V} = \mathbf{VK\Lambda}^{-1}\mathbf{K}^T\mathbf{V}^T\mathbf{V} = \mathbf{VK\Lambda}^{-1}\mathbf{K}^T\mathbf{B}^T,$$

where the fact that $\mathbf{B}$ is symmetric has been used. Substituting in $\mathbf{BK} = \mathbf{K\Lambda}$

$$\mathbf{\Phi A} = \mathbf{VK\Lambda}^{-1}\mathbf{\Lambda K}^T = \mathbf{VKK}^T.$$

Because $\mathbf{B}$ is a symmetric matrix, $\mathbf{KK}^T = \mathbf{I}$. Hence,

$$\mathbf{\Phi A} = \mathbf{V}.$$

Therefore, if the full POD basis is used, the mean-centred snapshot matrix is perfectly reconstructed. If the columns of $\mathbf{A}$ are considered,

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \cdots & \boldsymbol{\alpha}_M \end{bmatrix},$$

then each $\boldsymbol{\alpha}_i$ can be seen as an M-dimensional representation of the mean-centred snapshot $\mathbf{v}_i$. Thus, if $M < N$, dimensionality reduction has been achieved. This condition is typically met, especially for models with large meshes. Nevertheless, additional dimensionality reduction is possible by truncation of the POD basis. This truncation can be accomplished by using only the eigenvectors corresponding to the K largest eigenvalues. The truncated POD basis can be found from

$$\mathbf{\Phi}_K = \mathbf{VK}_K\mathbf{\Lambda}_K^{-1/2},$$

where $\mathbf{\Lambda}_K$ is a diagonal matrix containing the K largest eigenvalues on its diagonal, and $\mathbf{K}_K$ is an $M \times K$ matrix containing the corresponding eigenvectors as columns. Again, a matrix of POD coefficients can be found

$$\mathbf{A}_K = \mathbf{\Phi}_K^T\mathbf{V}.$$

Using the same method as before, the reconstruction is now

$$\mathbf{\Phi}_K\mathbf{A}_K = \mathbf{VK}_K\mathbf{K}_K^T.$$

The reconstruction is no longer perfect, because now $\mathbf{K}_K\mathbf{K}_K^T \neq \mathbf{I}$. The accuracy of the reconstruction depends on how accurately $\mathbf{K}_K\mathbf{K}_K^T$ approximates the identity matrix. Now consider

$\mathbf{A}_K = \begin{bmatrix} \boldsymbol{\alpha}_{K,1} & \boldsymbol{\alpha}_{K,2} & \cdots & \boldsymbol{\alpha}_{K,M} \end{bmatrix}$. Each $\boldsymbol{\alpha}_{K,i}$ can be seen as a K-dimensional representation of $\mathbf{v}_i$. The use of a truncated basis allows reduction to an arbitrary dimensionality subject to $K \leq M$.

When truncating the POD basis, one must decide on the number K of eigenvectors to consider. A method to do this is to keep adding eigenvectors in the order of decreasing eigenvalues until

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{j=1}^{M} \lambda_j} \geq \iota, \tag{11}$$

where ι is a threshold. For fluid flow problems, this criterion can be interpreted as the fraction of the energy of the system that use of K eigenvectors is able to capture [41, 42].

2.1.2 Function approximation

This section primarily discusses specifics of the implementation of different function approximation techniques for the purpose of constructing surrogate models. For a discussion of what these function approximation techniques are, see section 1.3.4.

2.1.2.1 Gaussian process

Gaussian process regression models can deal with multidimensional inputs, but predict scalar values only. As a result, a separate surrogate model must be trained for each output dimension. Three different kernel functions were used, namely the square exponential kernel, the Matérn kernel with $\nu = 5/2$ and the automatic relevance determination (ARD) kernel. The values of the hyperparameters of these kernel functions (see Table 2) as well as the observation noise hyperparameter were determined separately for each output dimension by minimising the negative log-marginal likelihood (NLML) using the BFGS algorithm [64]. The optimisation was terminated when the NLML had converged to within 10^{-1} or when the number of iterations reached 50.

2.1.2.2 Neural network

In this work, neural network surrogate models were trained using either the MLPRegressor implementation in scikit-learn [86], or using TensorFlow 2.8 [87]. All output dimensions were

predicted using the same neural network regression model. Feed-forward neural networks with two hidden layers were used. The number of hidden units in each hidden layer was determined using equations 8 and 9, and the tanh activation function was used by each of these hidden units. The Adam optimisation algorithm was used for training, and early stopping was used with a portion of the training data set aside as a validation set.

2.1.2.3 Support vector regression

As was the case with the Gaussian process regression models, a separate SVR model must be trained for each output dimension of the surrogate model. The LIBSVM implementation [88] for Python was used to train the SVR models. A square exponential kernel function was used. For each output dimension of the surrogate model, the values of three hyperparameters must be determined. These are the width of the insensitive region of the error function used in SVR training, a regularisation parameter, and the scale parameter of the kernel function. The values of the hyperparameters were determined by evaluating the performance of the SVR models for different combinations of hyperparameters on a $10 \times 10 \times 10$ logarithmically spaced grid. The performance of the SVR models was evaluated in terms of the root mean square error (RMSE) which was estimated from the training set using 5-fold cross validation.

2.1.2.4 Relevance vector machines

When an RVM regression model is used for the function approximation task of the surrogate model, a separate model must be trained for each output dimension. The RVM implementation used in this work makes use of a square exponential kernel function. The hyperparameters of the RVM can be found by minimising the negative log-marginal likelihood, the form of which can be found in [67]. The scale parameter of the kernel function is found using a gradient-only line search, which is a line search that terminates when the point at which the current search direction is no longer a descent direction is found. The gradient of the negative log-marginal likelihood with respect to the scale parameter of the kernel function can be found in [67]. The other hyperparameters that must be found are the precision of the prior distribution over the weight of each basis function, and the variance of the observation noise. These can be found using the re-estimation equations in [67] or the constructive approach in [69]. Both options are explored in this work. The first option prunes basis functions that have large precision from the model, while the second option adds basis functions to the model until all basis functions

that are still left out of the model have a precision hyperparameter that is infinite. An infinite precision hyperparameter indicates that the basis function with which it is associated has zero weight and thus correctly contributes nothing to the model. Hyperparameter estimation consists of iteration over two steps. In the first step, the precision and noise variance hyperparameters are estimated using an initial estimate of the kernel function scale parameter. In the second step, the kernel function scale parameter is optimised given the estimates of the other hyperparameters. Hyperparameter estimation terminates when some convergence criterion is met, in this case when the difference between the natural logarithm of the kernel function scale parameter changes by less than some threshold between successive iterations.

2.1.2.5 Neural network flow rate

The data driven models discussed in sections 2.1.2.1 - 2.1.2.4 all predict the evolution of the system state over a discrete time step and are thus examples of the resolvent approach given by equation 1. It is also possible to use the flow rate approach given by equation 2 to construct the surrogate model, although parameter optimisation then becomes more challenging. Though it is in principle possible to find the derivative of the least squares error function with respect to the model parameters [14], in this work the automatic differentiation capabilities of TensorFlow 2.8 [87] are instead exploited. The same architectures as for the discrete time neural networks discussed in section 2.1.2.2 were used. The flow rate was integrated using the fourth-order Runge-Kutta method, and early stopping was used with a portion of the training data set aside as a validation set.

2.2 Data assimilation

In this work, data assimilation is done using particle filters. It is assumed that the system noise in equation 4 and the measurement noise in equation 5 both are additive and Gaussian. Four different particle filter algorithms were implemented, making extensive use of the functionality provided by `scipy.stats` 1.7.1 [89]. The first of these algorithms is the SIR algorithm, which was implemented based on the discussion in [31]. Another of the algorithms is the auxiliary particle filter (ASIR), which is stated in [30]. Implementation of the ASIR filter requires making a choice between using the expectation $\mathbb{E}[\mathbf{z}_k|\mathbf{z}_{k-1}]$ or using a sample from $p(\mathbf{z}_k|\mathbf{z}_{k-1})$ as the point estimate of $p(\mathbf{z}_k|\mathbf{z}_{k-1})$ for resampling. In this work, the expectation was used. A third algorithm that was implemented is the equivalent weights particle filter (EWPF), based on

its documentation in [32, 34].

One of the reasons for weight collapse and sample impoverishment is that the likelihood is significant at only a single sample. Tempering of the likelihood can mitigate this problem by factorising the likelihood as follows [32]:

$$p(\mathbf{y}|\mathbf{z}) = \prod_{i=1}^{P} p(\mathbf{y}|\mathbf{z})^{\zeta_i},$$

where $0 < \zeta_i < 1$, $\sum_i \zeta_i = 1$ and P is the number of tempering steps. The posterior distribution given by equation 6 can be obtained by following the recursion

$$p_i(\mathbf{z}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{z}_k)^{\zeta_i} p_{i-1}(\mathbf{z}_k|\mathbf{y}_{1:k})}{p(\mathbf{y}_k|\mathbf{y}_{1:k-1})^{\zeta_i}} \tag{12}$$

for $i = 1, 2, \ldots, P$, and where $p_0(\mathbf{z}_k|\mathbf{y}_{1:k}) = p_P(\mathbf{z}_k|\mathbf{y}_{1:k-1}) = p(\mathbf{z}_k|\mathbf{y}_{1:k-1})$. The number of tempering steps P is an algorithm parameter. The benefit of tempering can be realised when reweighting and resampling of particles, together with the generation of new samples from the tempered posterior is performed after each step of the recursion given by equation 12. It can be challenging to obtain new samples from the posterior distribution, because the samples themselves are the only available representation of the posterior distribution. As discussed in section 1.3.2, the merging particle filter (MPF) proposed in [36] generates new samples in a manner that preserves the mean and covariance of the posterior distribution. A tempered likelihood was therefore used in conjunction with the MPF as the fourth particle filtering algorithm implemented in this work. The MPF works by performing resampling n times, instead of only once as in the standard SIR filter. This results in n sets of M samples, where M is the original ensemble size. Here, each particle is denoted by $\mathbf{z}_k^{i,j}$ as the i^{th} sample of the j^{th} set at the k^{th} time step. The i^{th} sample of the final ensemble at time step k, denoted by $\mathbf{z}_k^i$, is then obtained by using a linear combination of samples from each of the n sets as follows:

$$\mathbf{z}_k^i = \sum_{j=1}^{n} \theta_j \mathbf{z}_k^{i,j},$$

where θ_j are parameters such that $\sum_j \theta_j = 1$ and $\sum_j \theta_j^2 = 1$. The algorithm that results from the use of a tempered likelihood together with the MPF algorithm is stated in Algorithm 1. A detailed description of aspects such as the assignment of particle weights w, and a resampling algorithm can be found in [30]. To ensure that diverse new samples are generated, the samples should be shuffled after resampling. In this work the parameters $n = 3$, $\theta_1 = 0.75$, $\theta_2 = (\sqrt{13}+$

$1)/8$ and $\theta_3 = -(\sqrt{13}-1)/8$ were used for the MPF. These parameters were obtained from [36]. For likelihood tempering, the parameters $P = 5$, $\zeta_1 = \zeta_2 = 0.125$ and $\zeta_3 = \zeta_4 = \zeta_5 = 0.25$ were used.

Algorithm 1 The MPF algorithm with a tempered likelihood for updating the sample ensemble at time step k.

Parameters: $P, M, n, \{\zeta_i\}_{i=1}^{P}, \{\theta_i\}_{i=1}^{n}$
Input: $\{\mathbf{x}_{k-1}^i\}_{i=1}^{M}, \{w_{k-1}^i\}_{i=1}^{M}$
Output: $\{\mathbf{x}_k^i\}_{i=1}^{M}, \{w_k^i\}_{i=1}^{M}$
for $i \in \{1, 2, \ldots, M\}$ **do** ▷ SIR filter proposal distribution
  $\mathbf{x}_k^i \sim p(\mathbf{x}_k|\mathbf{x}_{k-1}^i)$
end for
for $j \in \{1, 2, \ldots, P\}$ **do**
  for $i \in \{1, 2, \ldots, M\}$ **do** ▷ Assign weights using tempered likelihood
    $w_k^i \leftarrow p(\mathbf{y}_k|\mathbf{x}_k^i)^{\zeta_j}$
  end for
  for $i \in \{1, 2, \ldots, M\}$ **do** ▷ Normalise weights
    $w_k^i \leftarrow w_k^i / \sum_i w_k^i$
  end for
  for $l \in \{1, 2, \ldots, n\}$ **do** ▷ Resample n times
    $\{\mathbf{x}_k^{i,l}\}_{i=1}^{M} \leftarrow \text{Resample}(\{\mathbf{x}_k^i\}_{i=1}^{M}, \{w_k^i\}_{i=1}^{M})$
  end for
  for $i \in \{1, 2, \ldots, M\}$ **do** ▷ Linear combination of samples
    $\mathbf{x}_k^i \leftarrow \sum_{l=1}^{n} \theta_l \mathbf{x}_k^{i,l}$
    $w_k^i \leftarrow 1/M$
  end for
end for

2.3 Hybrid models

2.3.1 Structure

In this work, a hybrid model structure is used in which a data-driven model takes as input the same inputs as the physics-based model in addition to the outputs of the physics-based model. The data-driven model predicts the corrected model output, rather than a correction to be added to the model output. This structure is illustrated in Figure 6. The hybrid model operates in the same space as the surrogate models, so its inputs and outputs are in the reduced space obtained by POD. The system state is therefore $\mathbf{z} = \boldsymbol{\alpha}_K$. If $\mathcal{P}$ is the function representing the physics-based model, which in this work is the surrogate model, and $\mathcal{H}$ is the function representing the

hybrid model, then

$$\mathcal{H}(\mathbf{z}, \mathbf{p}) = \mathcal{D}(\mathbf{z}, \mathbf{p}, \mathcal{P}(\mathbf{z}, \mathbf{p})),$$

where $\mathcal{D}$ is the function representing the data-driven model and $\mathbf{p}$ are parameter inputs of the physics-based model additional to the system state $\mathbf{z}$.

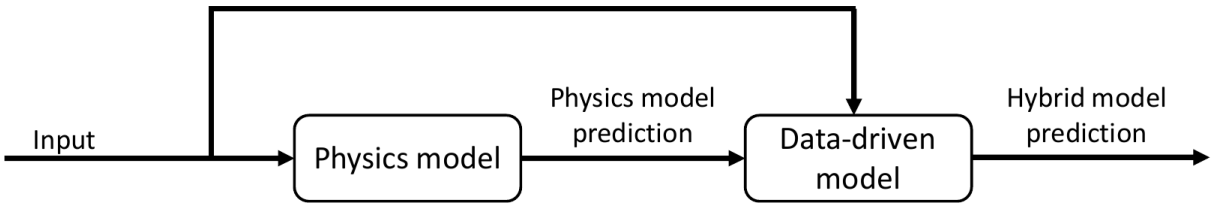

Figure 6: A schematic illustration of the structure of the hybrid models.

2.3.2 Learning algorithms

2.3.2.1 DA-O

The first algorithm discussed here is called the data assimilation-observation (DA-O) algorithm because it uses data assimilation to find initial conditions for the physics-based and hybrid models during training, and uses observations directly in the loss function of the data-driven component of the hybrid model. At initialisation, the algorithm starts with an initial distribution over system states and with the first available observation. It then uses an approximation of the initial distribution, such as a sample representation or an expected value, as the input to the hybrid model, and compares the output of the hybrid model to the observation. The hybrid model is then optimised to produce an output that matches the observation as closely as possible. This optimised hybrid model is then used in a data assimilation procedure to find the distribution over system states at the next time step. This distribution together with the initial distribution and the first two observations is used as the training set to re-optimise the hybrid model. This re-optimised hybrid model is then used in a data assimilation procedure to find the two system state distributions after the initial distribution, which are then used together with the initial distribution and the first three observations to obtain a new training set. The algorithm continues in this fashion until all available observations have been utilised. The DA-O algorithm is stated in Algorithm 2, and is illustrated in Figure 7. Note that the distributions over system states at each time step are re-estimated every time the model is updated.

Algorithm 2 The DA-O algorithm. The notation $\mathbf{y}_{1:0}$ denotes the absence of observations. $F\left[p(\cdot)\right]$ denotes some approximation of the distribution $p(\cdot)$, perhaps a sample representation or an expected value. The function $\mathcal{L}(\cdot,\cdot,\cdot)$ denotes the loss function used in the training of the hybrid model, and $\text{DA}(\cdot,\cdot,\cdot)$ denotes a data assimilation step. T is the number of sequential observations that are available. $\mathcal{P}$ is the physics-based model and $\mathcal{H}$ is the hybrid model.

Input: Initial distribution over states $\mathbf{z} \sim p(\mathbf{z}|\mathbf{y}_{1:0})$, observations $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$, $\mathcal{P}$
Output: $\mathcal{H}$
Training inputs $\leftarrow F\left[p(\mathbf{z}|\mathbf{y}_{1:0})\right]$
Training targets $\leftarrow \mathbf{y}_1$
$\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)
for $k \in \{2, 3, \ldots T\}$ **do**
  for $l \in \{1, 2, \ldots k-1\}$ **do**
    $p(\mathbf{z}|\mathbf{y}_{1:l}) \leftarrow \text{DA}(\mathcal{H}, p(\mathbf{z}|\mathbf{y}_{1:l-1}), \mathbf{y}_l)$
  end for
  Training inputs $\leftarrow \left(F\left[p(\mathbf{z}|\mathbf{y}_{1:0})\right], F\left[p(\mathbf{z}|\mathbf{y}_{1:1})\right], \ldots, F\left[p(\mathbf{z}|\mathbf{y}_{1:k-1})\right]\right)$
  Training targets $\leftarrow (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k)$
  $\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)
end for

An important component of the DA-O algorithm is the loss function $\mathcal{L}$ that it uses. Two loss functions were used in this work. The first is a simple modification of the standard least squares loss function:

$$\mathcal{L} = (\mathbf{y} - \mathbf{H}\mathbf{z})^T(\mathbf{y} - \mathbf{H}\mathbf{z}), \tag{13}$$

where $\mathbf{H}$ is an observation matrix that maps the system state onto the observations. It is possible to do this using linear algebra because the dimensionality reduction by POD is linear. The second loss function can be derived by noticing that equation 13 has the same form as the loss function used in standard linear regression. It is minimised if the system state is

$$\mathbf{z}^* = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}.$$

Then the standard least squares loss function can be used:

$$\mathcal{L} = (\mathbf{z} - \mathbf{z}^*)^T(\mathbf{z} - \mathbf{z}^*). \tag{14}$$

Note that equation 14 can be used only if the dimensionality of the reduced state $\mathbf{z}$ is smaller than the dimensionality of the observation $\mathbf{y}$.

Another important consideration is the approximation F of the probability distributions that is

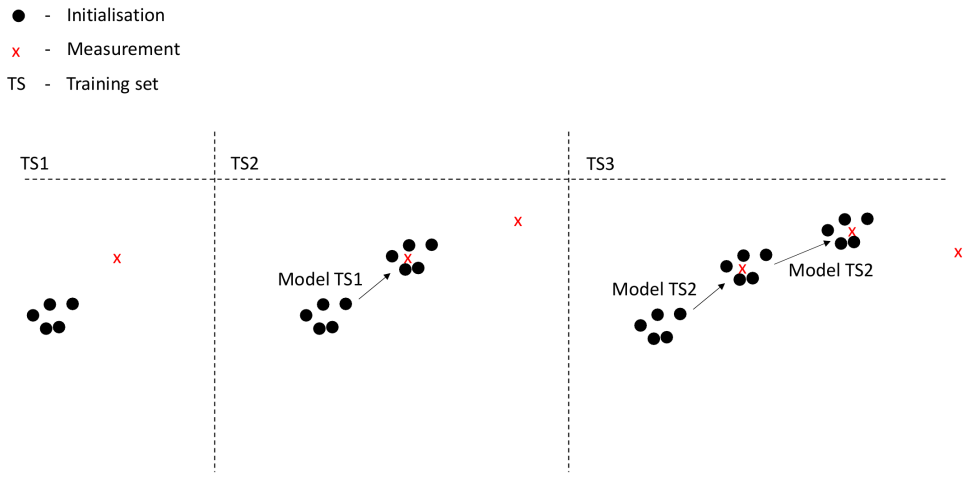

Figure 7: An illustration of the DA-O algorithm. Each training set (TS) consists of the initialisation as inputs and the observations as target outputs. Each arrow indicates a data assimilation time step, and the text accompanying each arrow indicates which model has been used as the system model to perform this time step. The training set is used to identify which model is used, so for example *Model TS1* denotes that the model that has been trained on training set TS1 is being used.

used in Algorithm 2. When the expected state $\mathbf{z}^a = \mathbb{E}[\mathbf{z}]$ is used, equations 13 and 14 can be used as is. This expected state is referred to as the analysis state. When a sample representation from a particle filter is used, the expected loss can be used instead. For either of the loss functions, this expected loss can be found as

$$\mathcal{L} = \sum_i w_i \mathcal{L}_i,$$

where w_i and $\mathcal{L}_i$ are the weight and loss function value associated with the i^{th} particle respectively. If a single time series of sensor measurements is used, only the expected loss makes sense for use in the first steps of the algorithm. If analysis states were to be used, only a single training point would exist for the first time step. This would likely not lead to a model that generalises well enough to be useful in the data assimilation steps that are subsequently carried out by the algorithm. An alternative approach that may make the use of analysis states practical, would be to consider multiple different time series of sensor measurements concurrently. This would lead to a training set size in the first step of the algorithm that is equal to the number of concurrent time series that are being considered.

2.3.2.2 Per-step DA-O

The per-step DA-O algorithm is a modification of the DA-O algorithm discussed in section 2.3.2.1. The modification is based on the idea that it may not be beneficial to always increase the number of time steps, and hence the complexity of the function that must be learnt by the model. This may lead to the model becoming worse for previous time steps for which a more specialised model had previously existed. This is a problem because the system state distributions are re-estimated after each model update in the DA-O algorithm. The modification made to arrive at the per-step DA-O algorithm is to dispense with this re-estimation, and to use models trained on one time step at a time to arrive at a time series of system state distributions. This time series can then be used in the training of a global model. The per-step DA-O algorithm is stated in Algorithm 3, and is illustrated in Figure 8.

Algorithm 3 The per-step DA-O algorithm. The notation $\mathbf{y}_{1:0}$ denotes the absence of observations. $F\left[p(\cdot)\right]$ denotes some approximation of the distribution $p(\cdot)$, perhaps a sample representation or an expected value. The function $\mathcal{L}(\cdot,\cdot,\cdot)$ denotes the loss function used in the training of the hybrid model, and $\text{DA}(\cdot,\cdot,\cdot)$ denotes a data assimilation step. T is the number of sequential observations that are available. $\mathcal{P}$ is the physics-based model and $\mathcal{H}$ is the hybrid model.

Input: Initial distribution over states $\mathbf{z} \sim p(\mathbf{z}|\mathbf{y}_{1:0})$, observations $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$, $\mathcal{P}$
Output: $\mathcal{H}$
Training inputs $\leftarrow F\left[p(\mathbf{z}|\mathbf{y}_{1:0})\right]$
Training targets $\leftarrow \mathbf{y}_1$
$\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)
for $k \in \{1, 2, \ldots T-1\}$ **do**
  $p(\mathbf{z}|\mathbf{y}_{1:k}) \leftarrow \text{DA}(\mathcal{H}, p(\mathbf{z}|\mathbf{y}_{1:k-1}), \mathbf{y}_k)$
  Training inputs $\leftarrow F\left[p(\mathbf{z}|\mathbf{y}_{1:k})\right]$
  Training targets $\leftarrow \mathbf{y}_{k+1}$
  $\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)
end for
Training inputs $\leftarrow \left(F\left[p(\mathbf{z}|\mathbf{y}_{1:0})\right], F\left[p(\mathbf{z}|\mathbf{y}_{1:1})\right], \ldots, F\left[p(\mathbf{z}|\mathbf{y}_{1:T-1})\right]\right)$
Training targets $\leftarrow (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T)$
$\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)

The same loss functions and the same system state distribution approximations that can be used for the DA-O algorithm can be used for the per-step DA-O algorithm as well. The argument made in section 2.3.2.1 that only the use of the expected loss or the concurrent consideration of multiple time series make sense, applies even more to the per-step algorithm because only single time steps are considered for the majority of training.

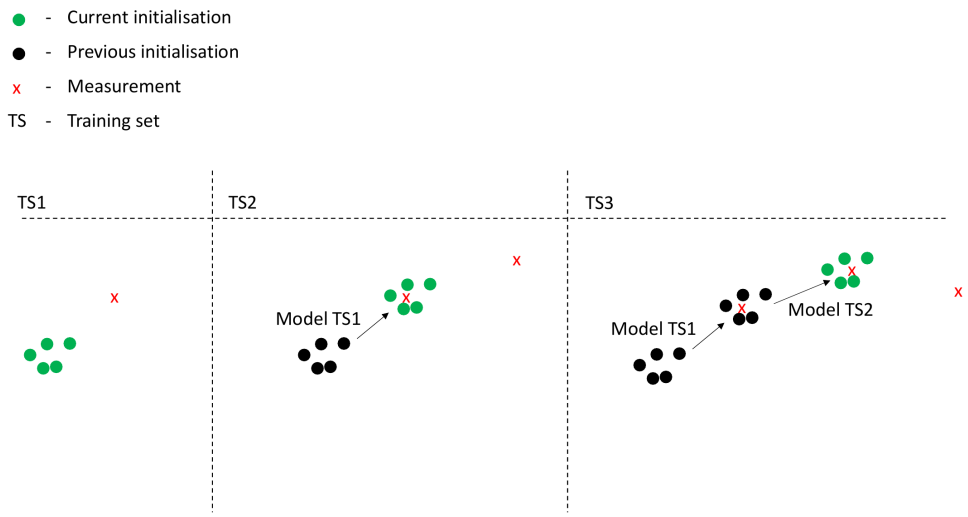

Figure 8: An illustration of the per-step DA-O algorithm. Each training set (TS) consists only of the current initialisation as inputs and the subsequent observation as target output. Each arrow indicates a data assimilation time step, and the text accompanying each arrow indicates which model has been used as the system model to perform this time step. The training set is used to identify which model is used, so for example *Model TS1* denotes that the model that has been trained on training set TS1 is being used.

2.3.2.3 ML-DA

The machine learning-data assimilation (ML-DA) algorithm is documented in the literature in [13, 16, 19]. It essentially consists of training a hybrid model using a sequence of states estimated using a data assimilation procedure. In the first iteration, the physics-based model is used in the data assimilation procedure [16, 19]. Subsequent iterations of the algorithm may take place in which the trained hybrid model is used to re-estimate the sequence of system state estimates. Unlike the DA-O and per-step DA-O algorithms, training takes place directly in the state space, and no mapping to the observation space is required for constructing a loss function. The standard least squares loss function can therefore be used. It is in principle possible to use an expected loss based on the initial and final particle ensembles. However this is complicated, and it is much simpler to use analysis states as in [13,16,19]. Therefore, only the use of analysis states will be considered in this work. The ML-DA algorithm is stated in Algorithm 4, and is illustrated schematically in Figure 9. While Algorithm 4 shows that $\mathbf{z}_0^a$ is used to construct the training set, in practice it may be necessary to discard this analysis state as well as the first few analysis states estimated by data assimilation to account for burn-in of the data assimilation algorithm.

Algorithm 4 The ML-DA algorithm. The notation $\mathbf{y}_{1:0}$ denotes the absence of observations. The function $\mathcal{L}(\cdot,\cdot,\cdot)$ denotes the loss function used in the training of the hybrid model, and $\text{DA}(\cdot,\cdot,\cdot)$ denotes a data assimilation step. T is the number of sequential observations that are available. $\mathcal{P}$ is the physics-based model and $\mathcal{H}$ is the hybrid model. The algorithm runs for N_{iter} iterations.

Input: Initial distribution over states $\mathbf{z} \sim p(\mathbf{z}|\mathbf{y}_{1:0})$, observations $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$, $\mathcal{P}$
Output: $\mathcal{H}$
$\mathbf{z}_0^a \leftarrow \mathbb{E}_{p(\mathbf{z}|\mathbf{y}_{1:0})}[\mathbf{z}]$
for $k \in \{1, 2, \ldots, T\}$ **do**
  $p(\mathbf{z}|\mathbf{y}_{1:k}) \leftarrow \text{DA}(\mathcal{P}, p(\mathbf{z}|\mathbf{y}_{1:k-1}), \mathbf{y}_k)$
  $\mathbf{z}_k^a \leftarrow \mathbb{E}_{p(\mathbf{z}|\mathbf{y}_{1:k})}[\mathbf{z}]$
end for
Training inputs $\leftarrow (\mathbf{z}_0^a, \mathbf{z}_1^a, \ldots, \mathbf{z}_{T-1}^a)$
Training targets $\leftarrow (\mathbf{z}_1^a, \mathbf{z}_2^a, \ldots, \mathbf{z}_T^a)$
$\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)
for $i \in \{2, 3, \ldots, N_{iter}\}$ **do**
  for $k \in \{1, 2, \ldots, T\}$ **do**
    $p(\mathbf{z}|\mathbf{y}_{1:k}) \leftarrow \text{DA}(\mathcal{H}, p(\mathbf{z}|\mathbf{y}_{1:k-1}), \mathbf{y}_k)$
    $\mathbf{z}_k^a \leftarrow \mathbb{E}_{p(\mathbf{z}|\mathbf{y}_{1:k})}[\mathbf{z}]$
  end for
  Training inputs $\leftarrow (\mathbf{z}_0^a, \mathbf{z}_1^a, \ldots, \mathbf{z}_{T-1}^a)$
  Training targets $\leftarrow (\mathbf{z}_1^a, \mathbf{z}_2^a, \ldots, \mathbf{z}_T^a)$
  $\mathcal{H} \leftarrow \arg\min_{\mathcal{H}} \mathcal{L}$ (Training inputs, Training targets, $\mathcal{H}$)
end for

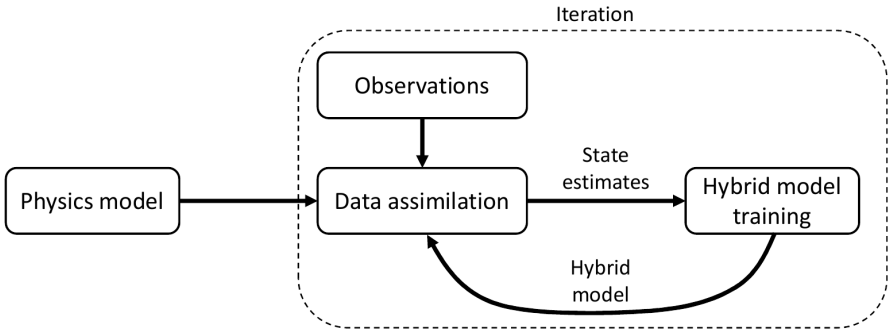

Figure 9: A schematic representation of the ML-DA algorithm.

3 Case study I: a small section of the process converter freeboard

A case study involving a small section of the process converter freeboard described in section 1.2 was performed to explore different aspects of the methodology of training hybrid models discussed in section 2. Section 3.1 describes the simulation model that was used for the case study. The dimensionality reduction and training of data-driven surrogate models is discussed in sections 3.2 and 3.3 respectively. Computational experiments that were conducted with data assimilation techniques are discussed in section 3.4. Finally, section 3.5 presents results that were obtained on the training of hybrid models.

3.1 Simulation model

This case study uses the geometry, boundary conditions and model parameters that were used for the analyses performed by Mr. Roelof Minnaar in [90]. The model was implemented and solved using Ansys Student 2021 R1 [91]. The geometry and mesh of the model are shown in Figure 10. As indicated in Figure 10, two different meshes were generated, a fine mesh with 104882 nodes, and another coarser mesh with 29077 nodes. In this case study, the coarse model is considered an imperfect model of the fine model, with a model error that is introduced due to the difference in the discretisation. The Ansys Mechanical solver was used for the models shown in Figure 10.

This case study considers only thermal response. The simulation model has two sets of surfaces that are subjected to thermal boundary conditions. These are the fireside and waterside surfaces, shown in Figure 11 (a) and (b) respectively. A convection boundary condition is applied to the waterside surfaces to model the flow of cooling water. Two different situations were considered for the fireside surfaces, resulting in different boundary conditions. In the first situation, the fireside surfaces are exposed directly to the gases inside the process converter, which was modelled with the application of a convection boundary condition. In the second situation, the fireside is covered by slag. The geometry of the slag covering was modelled as shown in Figure 12, and the thermal response of this slag covering was solved separately using the Ansys Fluent solver. For the slag-covered fireside case, the Ansys Mechanical and Ansys Fluent solvers were coupled as shown in Figure 13 and an iterative solution process was carried out.

The values of the model parameters required to specify the thermal boundary conditions are

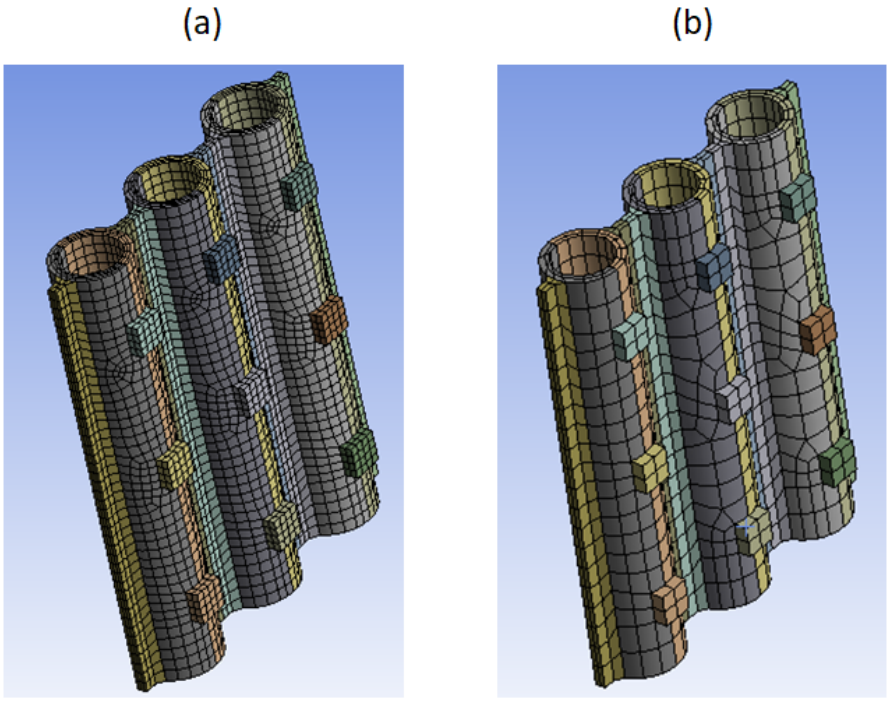

Figure 10: The geometry used in the simulation model, discretised using a mesh with (a) 104882 and (b) 29077 nodes.

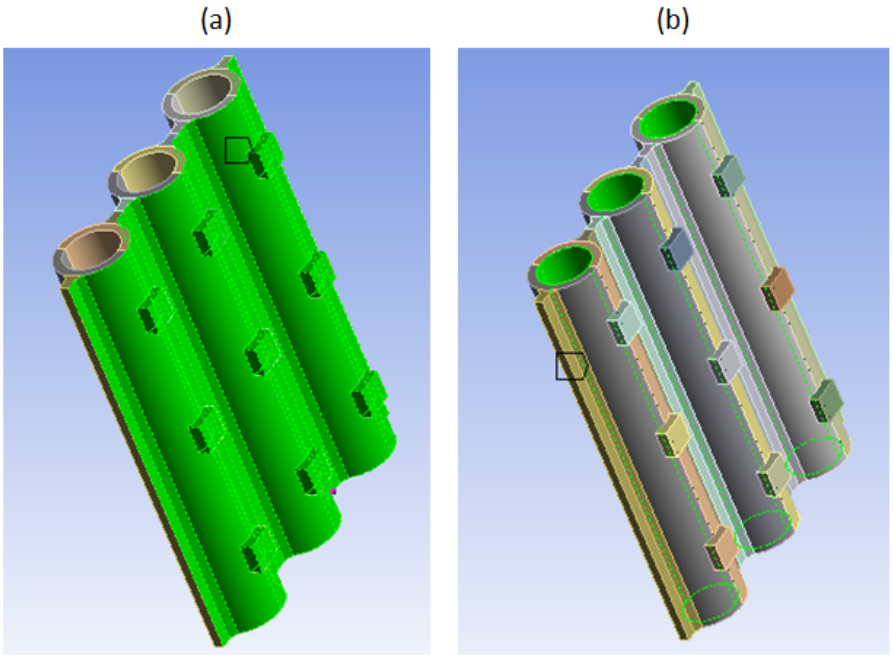

Figure 11: Thermal boundary conditions are applied to the (a) fireside and (b) waterside surfaces, which are highlighted in green.

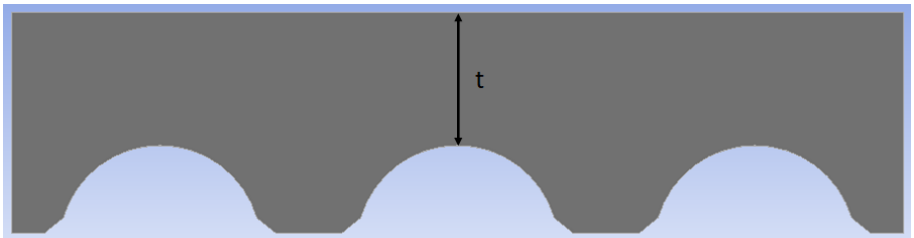

Figure 12: The geometry of the slag covering, which has thickness t.

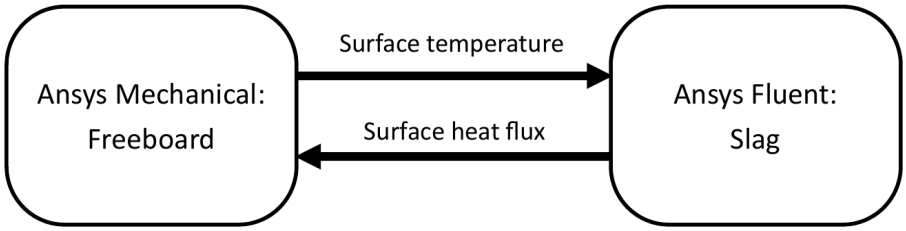

Figure 13: A schematic representation of the coupling between the Ansys Mechanical and Ansys Fluent solvers.

shown in Table 4. These parameter values were mostly obtained directly from [90], with the exception of the slag heat of solidification. In [90], the heat capacity of the slag is given as a function of temperature, with the heat of solidification already taken into account. This allows the assumption that the slag is always solid which can simplify the model setup. However, it was found that the solution process in the Ansys environment is more robust when the melting and solidification model in Ansys Fluent is used. The melting and solidification model assumes a constant heat of solidification, and the value of this parameter was obtained by comparison of results with those presented in [90].

Table 4: The values of various model parameters, obtained from [90].

Parameter	**Value**	**Units**
Fireside temperature	1350	$^{\circ}C$
Fireside convection coefficient	170	$W/m^2 \cdot^{\circ} C$
Waterside temperature	250	$^{\circ}C$
Waterside convection coefficient	13000	$W/m^2 \cdot^{\circ} C$
Slag thickness	25	mm
Slag solidus temperature	976	$^{\circ}C$
Slag liquidus temperature	1386.2	$^{\circ}C$
Slag heat of solidification	158.25	kJ/kg

The model described in [90] is a realistic model that was used for gaining insight into the oper-

ation of the process converter. Therefore, some efforts were made so that the simulation model used in this case study provides comparable results to those presented in [90]. However, because of differences in the modelling environments and model setups, some differences between the results provided by the two models are to be expected. This is not problematic for this case study, because the model was used only for a simulation-based study and the conclusions drawn from this case study do not depend on a comparison with any of the results presented in [90].

3.2 Dimensionality reduction

The snapshots that are needed for the extraction of a POD basis were generated by performing transient simulations with the coarse model using a time step of $0.1\,s$. Snapshots were generated for three different situations:

1. The freeboard is initially at a uniform temperature of $270\,°C$, and the fireside is bare.
2. The freeboard is initially at a uniform temperature of $270\,°C$, and the fireside is covered by slag that is initially at solidus temperature.
3. The freeboard is initially at the steady state condition reached with a bare fireside, and the fireside is covered by slag that is initially at liquidus temperature.

Separate POD bases were extracted from the freeboard shown in Figure 10 and from the slag covering shown in Figure 12. This means that potentially useful correlations between degrees of freedom on the slag and degrees of freedom on the freeboard are ignored when finding a reduced system state. However, because situation 1 does not involve a slag covering, this arrangement means that the reduced system state can be reused when the boundary conditions change from bare fireside to slag-covered fireside.

For the extraction of a POD basis from the freeboard, 200 snapshots of each of the three situations were used. Similarly, 1800 snapshots of both situation 2 and 3 were used for the extraction of a POD basis from the slag. In this case study, a metric based on the RMSE of the reconstruction was used to decide how many eigenvectors K to use in the POD bases. This is different from the metric based on eigenvalues that was proposed in section 2.1.1 and used in case study 2. For a proposed POD basis $\mathbf{\Phi}_K$, the RMSE was calculated as follows:

$$\text{RMSE} = \sqrt{\frac{\sum(\mathbf{\Phi}_K\mathbf{\Phi}_K^T\mathbf{V} - \mathbf{V})^2}{OM}},$$

where, as in section 2.1.1, $\mathbf{V}$ is the mean centred snapshot matrix, and O and M are the dimensionality of the snapshots and the number of snapshots respectively. The square is elementwise and the summation is over all elements of the matrix. A percentage RMSE was calculated using

$$\%\text{RMSE} = \frac{\text{RMSE}}{\max_j\{\max_i U_{ij} - \min_i U_{ij}\}} \times 100,$$

where U_{ij} is the element in row i and column j of the snapshot matrix before mean centring. Eigenvectors were added to the proposed POD basis until the $\%$RMSE metric was below 0.01%.

A further deviation from the approach presented in section 2.1.1 is that mean centring of the snapshots was done by subtracting the mean value across all nodes rather than the ensemble mean of the snapshots. The mean is therefore a scalar value that is subtracted elementwise from each snapshot and is different for each snapshot. As a result, it is a quantity that must be predicted by the surrogate model.

After performing this dimensionality reduction procedure, the POD bases for the freeboard and for the slag consisted of 14 and 56 modes respectively. Because the dimensionality reduction performed in this section is different to how it is performed in the rest of this work, it was repeated with the methodology from section 2.1.1 to understand what differences could arise. The results are summarised in Table 5. Note that for the purposes of the comparison in Table 5, the snapshot average temperature is counted as an additional mode for the method used in this section, because it is an additional variable that is unique for each snapshot. For the method described in section 2.1.1, the mean is common for all snapshots and only the POD coefficients are unique to each snapshot.

Table 5: POD bases obtained using the method described in this section (Method I) compared to POD bases obtained using the method described in section 2.1.1 (Method II). For both methods, performance is quantified in terms of the $\%$RMSE metric used in this section. The performance of each method is evaluated both for the number of modes obtained using Method I as well as for the number of modes obtained using Method II with a threshold of $\iota = 0.999$.

POD basis	**Number of modes according to**	**Method I**	**Method II**
Freeboard, 4 modes	Method II	0.159	0.116
Freeboard, 15 modes	Method I	0.00862	0.00591
Slag, 5 modes	Method II	0.494	0.343
Slag, 57 modes	Method I	0.00982	0.00972

The comparison in Table 5 shows that fewer modes are selected using the method from section 2.1.1, though this depends on the thresholds that are chosen for each of the two methods. More importantly, the method from section 2.1.1 results in a reconstruction error that is approximately

30% lower than the method used in this section, except for the slag POD basis with 57 modes, where it is less than 2% lower. This indicates that the method described in section 2.1.1 is more efficient, motivating its use for the remainder of this work.

3.3 Surrogate models

In this case study, two different surrogate models must be constructed. The reason is that in the bare fireside case, only the reduced state variables associated with the freeboard must be considered, while in the covered fireside case the state variables associated with the slag covering must be considered as well. As indicated in Figure 5, the surrogate models take as inputs the reduced system state and other relevant variables, and provide as outputs an evolved system state. The mean centring method used in this case study requires that the snapshot mean is present both as an input and as an output of the surrogate models. The snapshot mean can be interpreted as an additional dimension of the reduced system state. Four additional input variables were considered in this case study, namely the fireside and waterside temperatures and convection coefficients. The resulting input and output dimensionalities of the two surrogate models are summarised in Table 6.

Table 6: The surrogate model input and output dimensionalities for the bare and covered fireside cases.

Situation	**Input dimensionality**	**Output dimensionality**
Bare fireside	19	15
Slag-covered fireside	76	72

As discussed in section 1.3.5, training data for the surrogate models may be obtained either from the snapshots used in the dimensionality reduction or by using a design of experiments technique such as Latin hypercube sampling. The model used in this case study reaches a steady state condition smoothly when the boundary conditions are constant. The result is that only a small portion of system states that are likely to occur during operation are explored in the snapshot generation process, and use of a Latin hypercube sampling strategy for generating training samples is therefore likely to lead to better exploration of the state space.

For this case study, a Latin hypercube design with 76 dimensions and 256 levels was used. The design was optimised by minimising the Φ_p criterion using the LaPSO-L algorithm [71] with 8 swarms of population 32. The algorithm was run for 50 iterations. From the optimised

76 dimensional hypercube, a 19 dimensional hypercube was obtained by dropping the excess dimensions. This is still a valid Latin hypercube design, though its optimality is uncertain [70].

The Latin hypercube design specifies the relative values of the parameters for which model runs are to be conducted (initial conditions can be viewed as parameters), but to specify their concrete values, a range of values for each parameter must be specified. For the reduced state variables, the range of values was chosen to include the values reached in:

1. the snapshots generated for dimensionality reduction,
2. a constant temperature initialisation, and
3. the steady state condition reached with a slag covered fireside and the lower bound of the convection parameters in Table 7.

Of these situations, the first is expected to include a maximum condition that is reached when slag at liquidus temperature covers an initially bare fireside. Similarly, the third is expected to include a minimum condition because the fireside is then maximally insulated from the fireside gases. The second situation is included to ensure that constant temperature initialisations are included in the range of values, though the snapshots should also accomplish this as some of them are initialised from constant temperature. The range of values of the average temperature parameters in Table 7 were found in a similar manner, though there was a less strict adherence to the maximum and minimum values encountered in the above situations. The ranges of the convection parameters are given in Table 7.

Table 7: The upper and lower bounds of convection and average temperature parameters used in the design of experiments.

Parameter	**Lower bound**	**Upper bound**	**Unit**
Fireside convection coefficient	140	200	$W/m^2 \cdot^\circ C$
Fireside temperature	1250	1450	$^\circ C$
Waterside convection coefficient	10000	16000	$W/m^2 \cdot^\circ C$
Waterside temperature	240	260	$^\circ C$
Average temperature of the slag	686.2	1386.2	$^\circ C$
Average temperature of the freeboard	240	360	$^\circ C$

For each of the bare and slag-covered fireside cases, 256 transient simulations with initial and boundary conditions determined by the Latin hypercube design were performed. The simulations were run over $2\,s$ with a time step of $0.1\,s$. The system states predicted by the simulation

after $2\,s$ were used as the target outputs of the surrogate models. Of the 256 simulation runs, 240 were used as the training set for the surrogate models, and the remaining 16 were used as a test set for evaluating the performance of the surrogate models. The RMSE was used as the performance measure, and was calculated separately on the test set for each output dimension.

In this case study, Gaussian process, neural network and support vector regression surrogate models, which are discussed in sections 2.1.2.1 - 2.1.2.3, were constructed. In addition, surrogate models using neural networks to learn the flow rate, as discussed in section 2.1.2.5 were also constructed. In the construction of neural network surrogate models, 10% of the training data was used as a validation set for early stopping, whereas for the flow rate surrogate models, this fraction was 20%. The scikit-learn implementation of neural network regression was used to construct the neural network surrogate models. In the construction of the flow rate surrogate models, 8 time steps were used in the Runge-Kutta integration.

Two different training sets were used for the neural network surrogate models. The first is the same training set that was previously described and that was used in the training of the Gaussian process surrogate models. The second is an artificial training set consisting of 4096 sample points arranged in a Latin hypercube design and produced using the Gaussian process surrogate model with the square exponential covariance function. The architecture of the neural networks was adjusted for each training set size. The increased training set size of the artificial training set may be beneficial to the performance of the neural network, although because the training set is obtained from another surrogate model, it is equally possible that performance is reduced. The artificial training set strategy only makes sense if the Gaussian process model used to generate it outperforms the neural network model on the original training set.

The performance of each surrogate model was evaluated in terms of a $\%$RMSE metric. For each output dimension k of the surrogate model, the RMSE was calculated according to

$$\text{RMSE} = \sqrt{\frac{\sum_n (y_{kn} - y_{kn,true})^2}{N}},$$

where the subscript n denotes the n^{th} of N samples. Here, y is used to denote the outputs and desired outputs of the surrogate models. The $\%$RMSE metric was then obtained by normalising the RMSE by the range of values the output dimension takes on across the 256 simulation runs that make up the training and test sets.

Tables 8 and 9 show the test set results of various surrogate models of the bare and slag-covered fireside models respectively. An important note is that the performance of the neural network

surrogate models that were trained with synthetic data were evaluated using the same test set as the remaining models. Of the surrogate models, the Gaussian process surrogate models performed best. The results presented in Tables 8 and 9 indicate that there is not one kernel function that results in the best performing Gaussian process surrogate across all outputs. It is interesting that the Gaussian process surrogate models sometimes perform worse when the ARD kernel is used rather than the square exponential kernel. By inspection of the equations in Table 2, if the same values for σ_f are used by both kernel functions and $l_1 = l_2 = \ldots = l_D = l$, both surrogate models will make the same predictions. This indicates that the optimisation for the ARD kernel has terminated in a local minimum that is not optimal.

Table 8: The minimum, average and maximum % RMSE across 15 output parameters for a surrogate model of the bare fireside transient simulation. The results of the best performing model are printed in italics and underlined.

Surrogate model	**Minimum**	**Average**	**Maximum**
Gaussian process, square exponential	0.139	*0.540*	*1.07*
Gaussian process, Matérn, $\nu = 5/2$	0.174	1.71	3.95
Gaussian process, ARD	*0.105*	1.19	4.26
Neural network, simulation data	1.86	3.24	5.00
Neural network, synthetic data	0.66	2.11	4.06
Support vector regression	0.194	0.642	1.09
Neural network flow rate	0.419	0.773	1.28

The surrogate models of the bare fireside simulation performed better than the surrogate models of the slag-covered fireside simulation, to the extent that the results in Table 9 for the slag-covered simulation are worse than the results in Table 8 for the bare fireside simulation without exception. This indicates that it is more difficult for data-driven models to approximate the relationship between successive states in the slag-covered fireside case. There are multiple possible reasons for this. It is possible that the interaction between the freeboard and the slag covering is more difficult to model than the bare freeboard. It could also be that the increased dimensionality of the input space of the data driven models has reduced their efficiency, or that this dimensionality increase has reduced the effectiveness with which the design of experiments is able to explore the input space. It is unclear what precisely is responsible for the difference between the bare and slag-covered fireside cases.

Another interesting comparison that can be made is between the performances of the neural network models, two of which predict the resolvent (simulation and synthetic data), and one of which predicts the flow rate. As the results in Table 8 show, prediction of the flow rate results in the best neural network surrogate model of the bare fireside simulation. By contrast, the

Table 9: The minimum, average and maximum % RMSE across 72 output parameters for a surrogate model of the slag-covered fireside transient simulation. The results of the best performing model are printed in italics and underlined.

Surrogate model	**Minimum**	**Average**	**Maximum**
Gaussian process, square exponential	0.767	8.66	*16.4*
Gaussian process, Matérn, $\nu = 5/2$	0.786	*8.34*	*16.4*
Gaussian process, ARD	*0.236*	9.19	20.7
Neural network, simulation data	5.33	10.7	18.8
Neural network, synthetic data	1.17	8.41	16.7
Support vector regression	0.978	9.18	18.8
Neural network flow rate	3.22	10.5	20.5

synthetic data neural network surrogate model, which predicts the resolvent, is better in the slag-covered fireside case as shown in Table 9. In this case, the prediction of the flow rate is comparable in terms of average performance to the simulation data neural network, and has a larger difference between best and worst dimensions than the simulation data neural network. In the bare fireside case, prediction of the flow rate resulted in the lowest difference between the best and worst dimensions among the neural network surrogate models. This case study therefore indicates that the relative performance of surrogate models predicting flow rate and the resolvent is problem-dependent.

The surrogate models that are constructed in this work exist to reduce the computational cost associated with high-fidelity finite element models. As a result it is interesting to investigate how well this is achieved. Tables 10 and 11 show the wall-clock time required to perform a $2\,s$ simulation with a bare and a slag-covered fireside respectively for different models, including the original simulation model. They also show the factor by which the wall-clock time is compressed by each model relative to the original simulation. All wall-clock times of the surrogate models were measured using the `timeit` [92] module, and all models including the original simulation model were executed using a dual-core Intel i5-5200U CPU.

The results in Tables 10 and 11 show that all surrogate models achieve a time compression factor of at least 10^3. The surrogate model with the quickest execution time is the simulation data neural network model, which achieves a time compression factor over 10^7. Interestingly, the time compression factors for the kernel methods (Gaussian process and SVR models) are lower in the slag-covered case, whereas the time compression factors for the neural network models are higher in the slag-covered case. This is likely a result of the fact that for the kernel method models, 72 different models must be evaluated in the slag-covered case, compared to 15

Table 10: The wall-clock time required to perform a 2 s bare fireside simulation with the original simulation model and with different surrogate models. A time compression factor relative to the original simulation is shown in each case.

Model	**Wall-clock time (s)**	**Time compression factor**
Ansys simulation	300	1
Gaussian process, square exponential	3.23×10^{-2}	9.3×10^{3}
Gaussian process, Matérn, $\nu = 5/2$	4.24×10^{-2}	7.1×10^{3}
Gaussian process, ARD	4.71×10^{-2}	6.4×10^{3}
Neural network, simulation data	1.05×10^{-5}	2.9×10^{7}
Neural network, synthetic data	3.04×10^{-5}	9.9×10^{6}
Support vector regression	5.5×10^{-3}	5.4×10^{4}
Neural network flow rate	2.72×10^{-3}	1.1×10^{5}

in the bare fireside case. For the neural network models, only a single model must be evaluated in both cases, and this model evaluation can be efficiently parallelised.

Table 11: The wall-clock time required to perform a 2 s slag-covered fireside simulation with the original simulation model and with different surrogate models. A time compression factor relative to the original simulation is shown in each case.

Model	**Wall-clock time (s)**	**Time compression factor**
Ansys simulation	750	1
Gaussian process, square exponential	3.43×10^{-1}	2.2×10^{3}
Gaussian process, Matérn, $\nu = 5/2$	3.48×10^{-1}	2.2×10^{3}
Gaussian process, ARD	6.14×10^{-1}	1.2×10^{3}
Neural network, simulation data	2.38×10^{-5}	3.1×10^{7}
Neural network, synthetic data	8.19×10^{-5}	9.2×10^{6}
Support vector regression	7.56×10^{-2}	9.9×10^{3}
Neural network flow rate	2.78×10^{-3}	2.7×10^{5}

3.4 Data assimilation

This section considers an investigation into the application of different data assimilation techniques in the context of this case study. For the purposes of this investigation, the coarse simulation model was used to simulate sensor measurements, and the synthetic data neural network surrogate model was used as the physics-based system model in the data assimilation methods. All four of the data assimilation algorithms discussed in section 2.2 were considered in this section.

The system represented by the simulation model was partially observed by recording at each time step the nodal temperature of every 1000^{th} node. This results in 30 observed nodes at each time step. Note that these observed nodes are uniformly spaced in terms of how the nodes are numbered, and that this does not necessarily imply any particular distribution of the observed nodes in the physical space. Measurement noise was simulated by adding a noise sample $\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ at every time step, where the entries of the measurement noise covariance matrix are given by $R_{ij} = r^2\delta_{ij}$, and where δ_{ij} is the Kronecker delta. Different values of the parameter r were used to simulate the measurement noise. In the data assimilation algorithms, it was assumed that the value of r is known exactly. Simulated measurements were generated using $180\,s$ long simulations with a time step of $0.1\,s$, and of situations 1, 2 and 3 from section 3.2. The partial observations described above were made every $2\,s$.

The nodal temperature vector after dimensionality reduction by POD was used as the system state $\mathbf{z}$. System noise was assumed to be distributed according to $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. The entries of the covariance matrix of the system noise were taken as $Q_{ij} = \delta_{ij}\left(\frac{q_i}{\gamma}\right)^2$, where q_i is the difference between the minimum and maximum values of the i^{th} reduced state parameter in the design of experiments, which is discussed in section 3.3. Different values of the parameter γ were considered.

At time $t = 0$, the particle filters were initialised by drawing samples from the distribution $\mathcal{N}(\mathbf{z}_{0,true} + \boldsymbol{\psi}, \hat{\mathbf{Q}})$, where $\boldsymbol{\psi} \sim \mathcal{N}(\mathbf{0}, \hat{\mathbf{Q}})$, and the entries of the covariance matrix $\hat{\mathbf{Q}}$ are given by $\hat{Q}_{ij} = \delta_{ij}\left(\frac{q_i}{\kappa}\right)^2$. Here κ is a further parameter of which multiple different values were used. The ensemble size, or number of particles, that was used was varied as well.

Data assimilation runs were conducted for different combinations of the filter type and of the parameters r, γ, κ and ensemble size. For each of these combinations, 60 data assimilation runs with different initialisations were conducted to account for the stochastic nature of particle filters. The performance of each data assimilation run was evaluated in terms of a RMSE metric, defined as

$$\text{RMSE} = \sqrt{\frac{\sum(\mathbf{u} - \boldsymbol{\Phi}_K\mathbf{z}^a)^2}{O}}, \tag{15}$$

where $\mathbf{u}$ is the true nodal temperature vector and $\mathbf{z}^a$ is the analysis state. The same meanings as in section 2.1.1 apply for $\boldsymbol{\Phi}_K$ and O, and the summation is over all elements.

Figure 14 shows the average RMSE over the first 10 time steps for situation 1 for different values of the system noise parameter γ. For each of the filters, there is an optimal value of the system noise parameter. This is expected, because a system noise parameter should exist that

is most appropriate for the model error and for the actual stochastic behaviour in the system. It is notable that the optimal value of the system noise parameter for the EWPF is different to the optimal value shared by all other filters. The behaviour of the RMSE appears to be consistent for different ensemble sizes. Figure 41 in Appendix A shows that results are similar when averaged over all 90 of the time steps that were simulated.

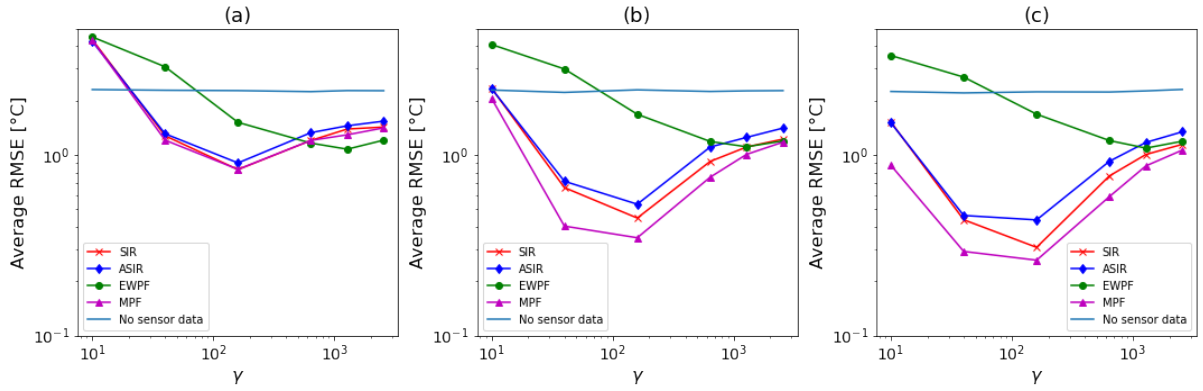

Figure 14: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 1 as a function of the system noise parameter γ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. The initialisation noise parameter was $\kappa = 10$ and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

The average RMSE over the first 10 time steps for situation 1 is shown as a function of the initialisation noise parameter in Figure 15. The performance of all filters appears to improve as the initialisation noise parameter is increased, which corresponds to a decrease in the initialisation noise. This is again the expected result, as increased initial certainty about the system state leads to better filter performance. When averaged over all 90 time steps, the dependence of the filter performance on the initialisation noise parameter is reduced, as shown in Figure 42 in Appendix A. This indicates that filter performance becomes more independent of initial knowledge as time passes, which is to be expected.

Figure 16 shows the dependence of the average RMSE over the first 10 time steps for situation 1 on the measurement noise parameter. The performance of all particle filters is reduced when the measurement noise is increased. This reflects a reduction in the quality of the information from which the system state must be estimated when the measurement noise is increased. Because the measurement noise parameter is the same as the parameter used in the generation of measurement noise, there is not expected to be an optimal point where the measurement noise parameter matches actual conditions. This is consistent with the lack of an optimal point in Figure 16. The results averaged across all 90 time steps are similar, as shown in Figure 43 in Appendix A.

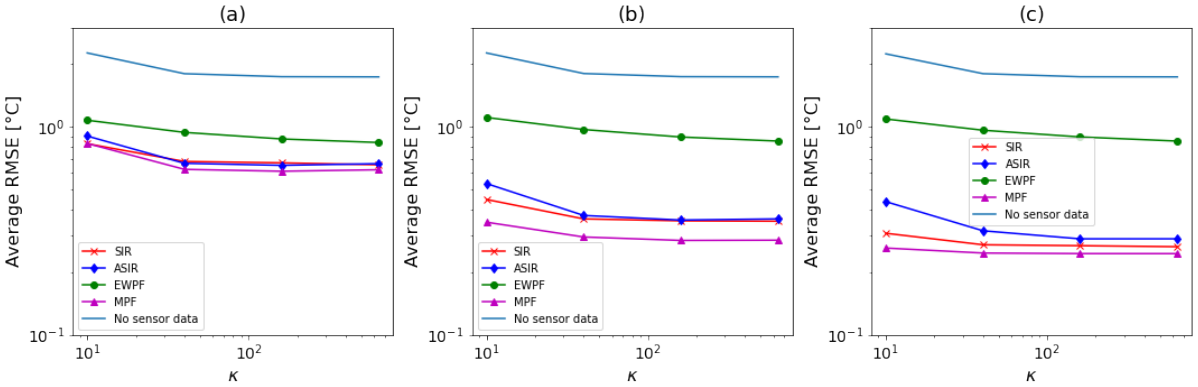

Figure 15: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 1 as a function of the initialisation noise parameter κ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The measurement noise parameter was set to $r = 0.5\ ^\circ C$.

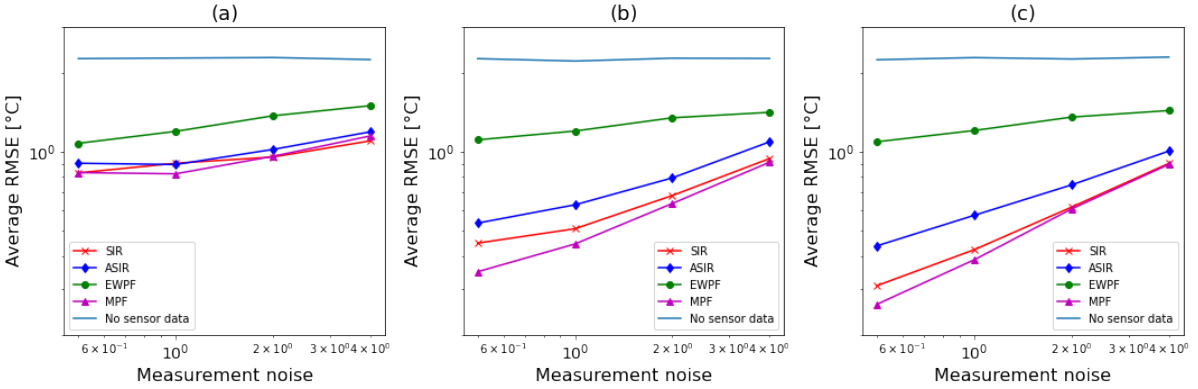

Figure 16: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 1 as a function of the measurement noise parameter r. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$.

The dependence of the average RMSE over the first 10 time steps for situation 1 on the ensemble size is shown in Figure 17. For the SIR, ASIR and MPF filters, performance is improved as ensemble size is increased. This is expected because the approximation given by equation 7 approaches the true distribution involved as the ensemble size becomes infinite. The performance of the EWPF does not appear to improve as ensemble size is increased, which is possibly a reflection of the bias inherent in the EWPF [32]. Similar results are obtained when the RMSE is averaged over all 90 time steps, as shown in Figure 44 in Appendix A.

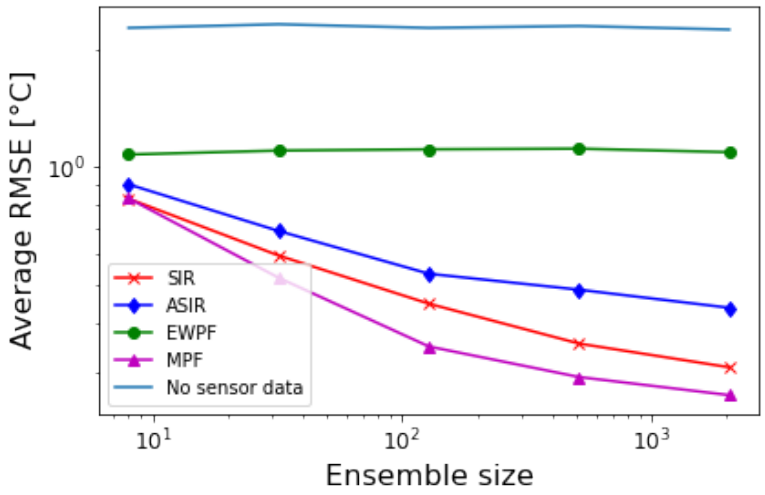

Figure 17: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 1 as a function of the ensemble size. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$, and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

The results presented in Figures 14 - 17 and in Figures 41 - 44 in Appendix A for situation 1 are also available for situations 2 and 3 in Figures 45 - 60 in Appendix A. A notable difference is that the dependence of the results on the initialisation and measurement noise parameters is reduced when compared to situation 1. This is possibly a result of the model error, and hence the system noise becoming dominant. That the model error is more significant for situations 2 and 3 can be seen by comparing the performance of the synthetic data neural network surrogate model in Table 8 for situation 1 with the performance of the same surrogate model surrogate model in Table 9 for situations 2 and 3.

An important result is that for each situation, a combination of filter parameters exists so that the average RMSE after filtering is lower than the average RMSE obtained when the surrogate models are used without sensor data. This demonstrates that the filters that have been implemented

successfully carry out the task of data assimilation. Of the filters that have been implemented, it appears that the MPF gives the best performance in most situations. The behaviour of the EWPF appears to be distinct from the behaviour of the other filters, which often behave similarly.

3.5 Hybrid models

In this section, the training of hybrid models for this case study is considered. For the purposes of this section, the real system from which measurements are collected is the fine simulation model. The physics-based model that is available is the synthetic data neural network model discussed in section 3.3.

Measurements were generated from simulations conducted with 20 different initial conditions, with each of these simulations being conducted over $180\,s$. To obtain 20 different initial conditions, a transient simulation of a slag-covered fireside was conducted, with the slag initially at liquidus temperature and the freeboard initially at the steady-state reached with a bare fireside. Every nine seconds, the system state reached in this simulation was saved as one of the 20 initial conditions. This simulation therefore represents a situation in which the slag covering falls off at different times after it had splashed onto the fireside at liquidus temperature. An illustration of this method of obtaining different initial conditions is represented in Figure 18.

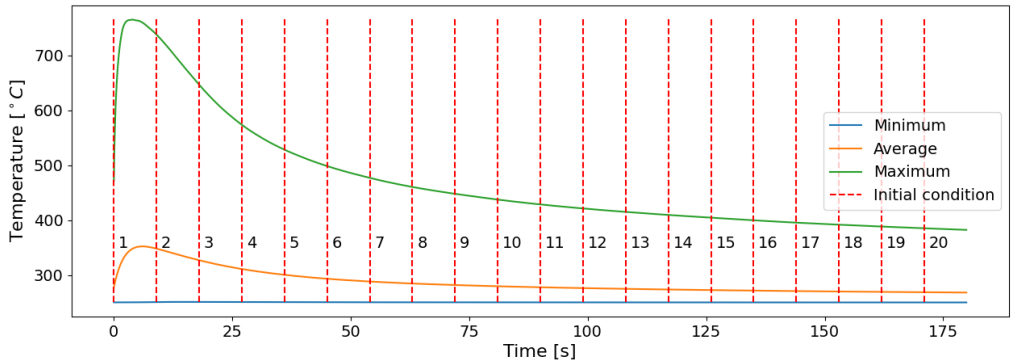

Figure 18: An illustration of how initial conditions were generated for simulating measurements for hybrid model training. The minimum, average and maximum temperatures of the slag-covered fireside system are shown to illustrate the evolution of its state over time. At each of the red dashed lines, the system state was extracted and used as an initial condition for a bare fireside simulation. Each line is labelled with a number for reference.

From the 20 different simulations that were performed, 20 simulated measurement time series were generated. This was done by finding the location of every 1000^{th} node of the coarse

simulation model, and finding the nearest node on the fine simulation model. The temperature at each of these nodes, of which there are 30, was recorded every $2\,s$. Noise was added to these measurements in the manner described in section 3.4, with a measurement noise parameter of $r = 0.5\ ^{\circ}C$. Of the 20 simulated measurement time series generated in this way, 17 were used in the hybrid model training process, while the remaining three were used as part of a test set to verify the performance of the trained models. The simulations with initial conditions numbered 2, 4 and 19 were those set aside for the test set.

As discussed in section 1.4, neural networks are used in this work as the data-driven component of the hybrid models. Before testing the performance of the training algorithms, some experiments were conducted to find an appropriate architecture for these neural networks. The training set inputs for these experiments were obtained by running a data assimilation procedure on all 90 time steps of each of the 17 measurement time series that are part of the training set. The training set targets were obtained from the measurements themselves, as per the DA-O and per-step DA-O algorithms. This training set does not allow a valid hybrid model to be trained, as the mapping between uncorrected inputs and measurements is learned. Rather, the mapping between corrected inputs and measurements should be learned, and this is what the DA-O and per-step DA-O algorithms attempt to do. This method of generating the training set was used to save computation relative to the DA-O and per-step DA-O algorithms while presenting a learning problem that is similar to the learning problem encountered in those algorithms. There is no computational saving relative to the ML-DA algorithm, however.

The data assimilation parameters were chosen based on the results in section 3.4. This is a flawed choice, because in section 3.4, simulated measurements were generated from the coarse simulation model, while here they are generated from the fine simulation model. The data assimilation problem is therefore different. However, given the coarse exploration of the parameter space in section 3.4, repetition of the experiments in this section would likely not lead to significantly different results. Furthermore, the results in section 3.4 are obtained by comparison of the estimated system state with the full system state, whereas in reality only the observations of this full system state would be available. The selection of parameters based on the full true system state would therefore represent the use of information that is not available in reality. A different method of dealing with this problem is proposed in case study 2.

The measurement noise parameter was set to $r = 0.5\ ^{\circ}C$. This means that the measurement noise parameter is assumed to be known exactly. Given that this is the lowest of the noise parameters used in section 3.4, it represents a best-case scenario for the training of hybrid models. The system noise parameter was set to $\gamma = 160$, which was found to work well for the

bare fireside case, as shown in Figure 14. The initialisation noise parameter $\kappa = 40$ was used, because as Figure 15 shows, the data assimilation performance does not depend strongly on κ if $\kappa \geq 40$, especially if the SIR and MPF filters are used. The SIR filter was used for these computational experiments, because while it mostly performs slightly worse than the MPF, it requires less computational steps, especially since the MPF implementation used here makes use of a tempered likelihood, as discussed in section 2.2.

Table 12 shows the different neural network architectures that were evaluated for the data-driven component of the hybrid model. In addition to varying the architecture, different activation and error functions were used, and pre-training was also used for some architectures. The number of training epochs was varied in some cases. The architecture used for C1-C4 is sized according to equations 8 and 9. For C5, C6 and C9-C12, the architecture was chosen to evaluate the performance of the hierarchical neural networks proposed in [60] relative to other approaches. The architecture used in C7 and C8 is designed so that a hierarchical neural network with 5 subdivisions of the output vector has the same number of hidden units as the hierarchical neural network used in C6. The architecture that is used in C13-C15 is designed to evaluate the effect of the depth of the neural network and has as many hidden units as the architecture used in C5.

As activation functions, either the hyperbolic tangent (tanh) or the rectified linear unit (ReLu) functions were used. The direct error function refers to the loss function given by equation 13, while the least squares error function refers to the loss function given by equation 14. Pre-training here refers to a layerwise pre-training. In the first step of pre-training, the first hidden layer is used to train an autoencoder. The input weights of the first hidden layer are then fixed, and a second hidden layer is added and used to train an autoencoder. This process is repeated until all hidden layers have been added, and the input weights to these hidden layers are then used as the weight initialisation before final training with the entire network takes place. As a final variation across the 15 configurations, the number of training epochs was varied to account for the effects of additional training epochs that are performed with pre-training and with hierarchical neural networks.

Each of the 15 configurations in Table 12 was evaluated in terms of the mean square error (MSE) on the training set by averaging the result of equation 13 over all training samples and by normalising by the number of output dimensions. When evaluating the performance of neural networks, the generalisation performance of the neural network is normally of greater interest than its performance on the training set. In this case, the training set performance was considered as a proxy for the flexibility of the neural network, which was considered to be more important. The reasoning behind this was that very large training sets would be accumulated

Table 12: The different combinations of architecture, activation function, error function and pre-training that were evaluated, together with their performance on the test problem. Each of these 15 combinations is given a label. Neural network architectures are given in the form [input layer size-hidden layer sizes-output layer size]. The difference between the number of input and output dimensions is a result of the use of the physics-based predictions as an input of the neural networks.

Label	**Architecture**	**Trainable parameters**	**Activation**	**Error function**	**Pre-training**	**Epochs**	**MSE**
C1	[30, 901, 711, 15]	679 933	ReLu	Direct	No	100	0.53
C2	[30, 901, 711, 15]	679 933	tanh	Direct	No	100	0.86
C3	[30, 901, 711, 15]	679 933	ReLu	Least squares	No	100	0.69
C4	[30, 901, 711, 15]	679 933	tanh	Least squares	No	100	2.35
C5	[30, 200, 200, 15]	49 415	tanh	Least squares	No	100	1.05
C6	[30, 200, 200, 5], Hierarchical, K=3	142 215	tanh	Least squares	No	200	1.27
C7	[30, 120, 120, 15]	20 015	tanh	Least squares	No	200	0.93
C8	[30, 120, 120, 3], Hierarchical, K=5	93 015	tanh	Least squares	No	200	0.99
C9	[30, 200, 200, 15]	49 415	tanh	Least squares	No	400	1.08
C10	[30, 200, 200, 15]	49 415	tanh	Least squares	Yes	200	1.23
C11	[30, 200, 200, 15]	49 415	ReLu	Least squares	No	400	0.48
C12	[30, 200, 200, 15]	49 415	ReLu	Least squares	Yes	200	0.60
C13	[30, 80, 80, 80, 80, 80, 15]	29 615	ReLu	Least squares	No	400	0.45
C14	[30, 80, 80, 80, 80, 80, 15]	29 615	ReLu	Least squares	Yes	200	0.63
C15	[30, 80, 80, 80, 80, 80, 15]	29 615	ReLu	Direct	No	400	0.47

over time, which would permit the use of flexible neural network models without the risk of severe overfitting. For each of the 15 configurations, 30 training runs were conducted. For each of these training runs, a data assimilation run was conducted, resulting in the use of a different training set for each run.

As Table 12 shows, configuration C13 results in the best performance on the training set. Neither hierarchical neural network was able to outperform a standard neural network with the same architecture. Similarly, the use of pre-training degraded performance. The best performing neural network was a deep neural network with fewer hidden nodes compared to the shallow neural networks used in configurations C1-C4, and with the same number of hidden nodes as the architecture used in C5. A comparison of the results for configurations C13 and C15 shows that use of the least squares error function results in better performance than use of the direct error function. Interestingly, the best-performing architecture also has the second-least trainable parameters of all architectures that were evaluated. This is probably related to the deep nature of the architecture, allowing it to outperform architectures with more trainable parameters even in a context in which overfitting is not penalised. Based on these results, the architecture from configuration C13 was used for the remainder of this section, and the least squares error function was used for the DA-O and per-step DA-O algorithms.

To test the performance of the different learning algorithms tested in section 2.3.2, the 20 simulated measurement time series were used to train and test hybrid models using each learning algorithm. The same data assimilation setup that was used to investigate different architectures of the data-driven component of the hybrid models was used for this purpose. The DA-O algorithms evaluated in this section make use of the full ensemble of samples from the posterior distributions estimated by data assimilation during training, whereas the ML-DA algorithm makes use of analysis states. Two different versions of the DA-O algorithm were used - one that uses all 90 simulated measurements in training and one that uses only the first 20 measurements in training. The former will be referred to here as the full DA-O algorithm and the latter as the shortened DA-O algorithm. The ML-DA algorithm was run for 16 iterations. The 17 time series that were set aside for training were used simultaneously in generating the training set. This means, for instance, that in the case of the ML-DA algorithm, data assimilation is performed using the current version of the model on each of the 17 time series, and the resulting analysis states are then assembled into the training set.

After the completion of training, the performance of the trained hybrid models was evaluated using the three remaining simulated measurement time series. To do this, a data assimilation run was conducted on each time series using the trained hybrid model, and the performance

was evaluated using equation 15. The RMSE was calculated across all nodes of the model, and additionally across only the observed nodes of the model using a suitable modification of equation 15.

To account for the stochastic nature of training, multiple training runs were conducted for each training algorithm. For the shortened DA-O algorithm, 30 different models were trained, and for the remaining algorithms 16 different models were trained. This means that there are 90 pairs of model and test time series for the shortened DA-O algorithm, and there are 48 such pairs for the remaining algorithms. For each of these pairs, 30 data assimilation runs were conducted to evaluate the performance of the trained hybrid models. On each test time series, 30 data assimilation runs were also conducted using the physics-based model against which the performance of the hybrid model can be compared. In this section, hybrid and physics-based models are compared in terms of RMSE averaged over the 90 test time steps, and this is what is meant when the RMSE is referred to.

One of the comparisons that was made between the hybrid and physics based models is whether for each model-test time series pair the data assimilation run is more accurate when the hybrid model is used compared to when the physics-based model is used. When the hybrid model outperforms the physics-based model it is termed a win, otherwise it is termed a loss. To make this comparison, the RMSE was averaged over the 30 data assimilation runs conducted for each model-test time series pair. A t-test was conducted with a p-value of 0.05 to determine whether the difference between the hybrid and physics-based models is statistically significant. The results are shown graphically in Figures 19 and 20 as fractions of the total number of model-test time series pairs that are classified as wins and losses.

As shown in Figures 19 and 20, hybrid models trained using the per-step DA-O and the ML-DA algorithms outperform the physics-based model for every model-test time series pair, and this is always statistically significant. This is not the case for the shortened and full DA-O algorithms. For these algorithms the number of wins is larger when the comparison between hybrid and physics-based models is based on only the observed nodes rather than all nodes of the model. This difference in results between all nodes and observed nodes is especially pronounced for the full DA-O algorithm. Importantly, the difference in results between all nodes and observed nodes shows that it is possible for a hybrid model to be an improvement relative to the physics-based model in a manner that is local to the observed nodes, but not simultaneously be an improvement globally over all nodes of the model. It is also notable that the full DA-O algorithm outperforms the shortened version of the algorithm. This is to be expected when considering that the shortened DA-O algorithm trains hybrid models based on

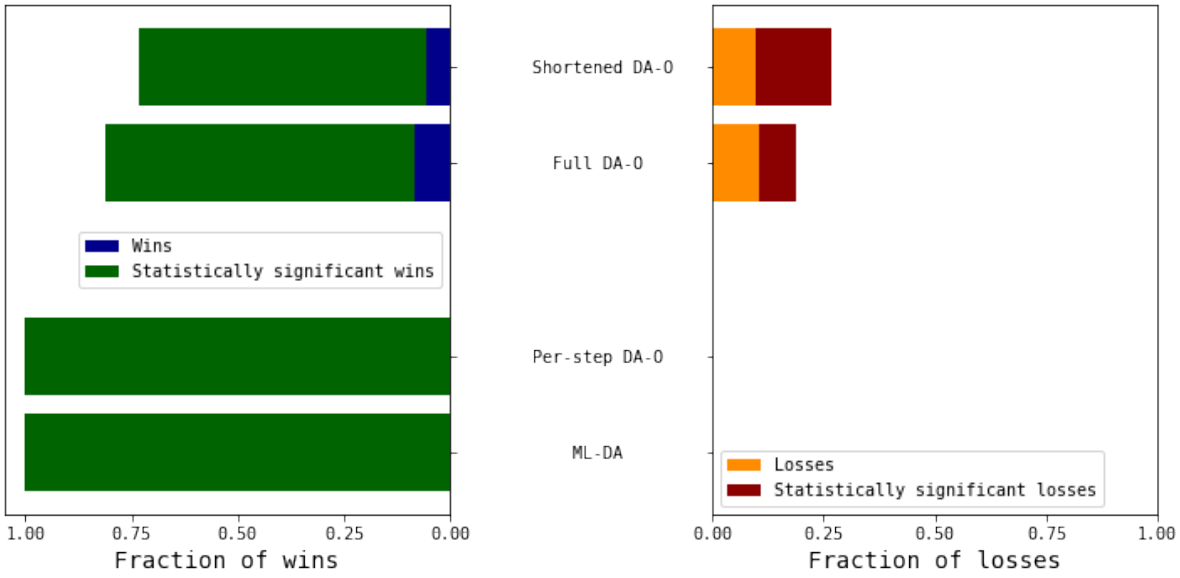

Figure 19: The fraction of wins and losses in terms of RMSE over all 29077 nodes, of the hybrid model compared to the physics-based model. The portion of this fraction that is statistically significant is shown.

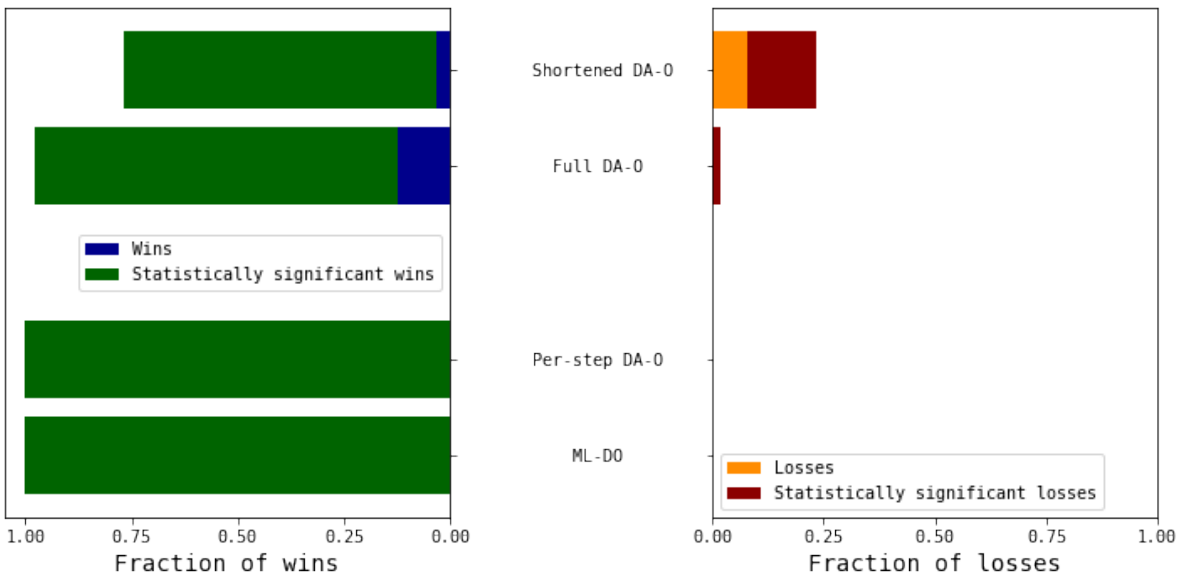

Figure 20: The fraction of wins and losses in terms of RMSE over the 30 observed nodes, of the hybrid model compared to the physics-based model. The portion of this fraction that is statistically significant is shown.

only a fraction of the information that is available to the full DA-O algorithm.

To quantify the performance of each training algorithm, the RMSE of each model was averaged over the three test time series and the 30 data assimilation runs conducted for each model-test time series pair. The median result of the 30 models trained using the shortened DA-O algorithm and of the 16 models trained using each of the other algorithms was then recorded. The median was used in place of the mean so that outlier models with poor performance do not dominate the results. Table 13 shows the median results for hybrid models trained using each training algorithm, as well as the median results for the physics-based model. Results are shown for when the RMSE is calculated across all nodes and for when it is calculated across the observed nodes. Also shown is the ratio of the RMSE over all nodes to the RMSE over observed nodes.

Table 13: The median RMSE (in $^{\circ}C$) results of models trained using each of the training algorithms. The RMSE is shown for two cases, one in which it is calculated over all nodes and another in which it is calculated over observed nodes. The ratio of the RMSE on all nodes to the RMSE on the observed nodes is also shown.

Training algorithm	**Median RMSE all**	**Median RMSE observed**	**Error ratio**
None (physics-based model)	5.13	2.25	2.28
Shortened DA-O	4.93	1.38	3.57
Full DA-O	4.87	1.09	4.47
Per-step DA-O	4.60	0.63	7.30
ML-DA	4.70	0.90	5.22

As the results in Table 13 show, the per-step DA-O algorithm performed best, while the shortened DA-O algorithm performed worst. All training algorithms resulted in hybrid models that outperform the physics-based model. As the median RMSE of the hybrid models on all nodes is reduced, so is the median RMSE on the observed nodes. Notably, the ratio of the median RMSE on all nodes to the median RMSE on the observed nodes increases as the performance of the hybrid models increases. This is expected, as the information that is available for training hybrid models is available directly at the observed nodes, but must be inferred using data assimilation for the unobserved nodes, which constitute the majority of the nodes. Importantly, the improvement in performance on the observed nodes is accompanied by an improvement in the performance on all nodes.

All computational experiments that have been conducted so far with data assimilation and hybrid model training have made use of the same number of observed nodes, namely 30. It is expected that the number of observed nodes has a large impact on the performance of hybrid models, being related to how much information is available about the system. Therefore, further

computational experiments were conducted to investigate the relationship between the number of observed nodes and the performance of trained hybrid models. To do this, the ML-DA training algorithm was used since it is most easily adaptable to different observation vector sizes. Nine different numbers of observed nodes logarithmically spaced between 2 and 455 were investigated. The same data assimilation parameters as before were used, but both the SIR and MPF filters were used.

For each number of observed nodes, 30 model training runs were conducted using the ML-DA algorithm. The ML-DA algorithm was used in spite of the better performance of the per-step DA-O algorithm seen in Table 13, because very little modification is needed to use the ML-DA algorithm for different numbers of observed nodes. Furthermore, when the number of observed nodes is less than the number of state variables (i.e. the dimensionality of the reduced space found by POD), the pseudo-inverse used to find $\mathbf{z}^*$ in equation 14 is no longer full rank. The ML-DA algorithm has no need for the pseudo-inverse in order to be able to apply the least squares loss function. For each number of observed nodes, the median RMSE was found in the manner used to obtain the results in Table 13.

Figure 21 shows the median RMSE calculated across all nodes of the model for different numbers of observed nodes. Interestingly, the median RMSE appears to be relatively constant on two sides of a step in the RMSE that occurs between 8 and 15 observed nodes, especially if the SIR filter is used. This step occurs at the dimensionality of the state space, which is indicated in Figure 21. This likely indicates that when the number of observations is greater than the state space dimensionality, there is an excess of information to constrain the system, while when the number of observations is less than the state space dimensionality, the observations cannot constrain the system. It is unclear whether the location of the step is a function of the dimensionality of the state space, or whether it would remain in the same location even if no dimensionality reduction were to be used. The latter is entirely possible, given that the number of dimensions used in the POD dimensionality reduction is based on the fraction of the variance accounted for by those dimensions, and that POD dimensionality reduction therefore gives an insight into the true number of degrees of freedom in the system.

Another interesting feature of the results in Figure 21 is that the performance of the physics-based models appears to improve as the number of observed nodes is decreased when the MPF is used and the number of observed nodes is greater than 15. This counter-intuitive result could perhaps be attributable to a hyperparameter such as the system noise becoming more appropriate as the number of observed nodes decreases. This improvement of the physics-based models appears to have no impact on the performance of the hybrid models trained using an MPF.

Importantly, the hybrid models trained using both particle filters outperform the physics-based models regardless of how many nodes are observed.

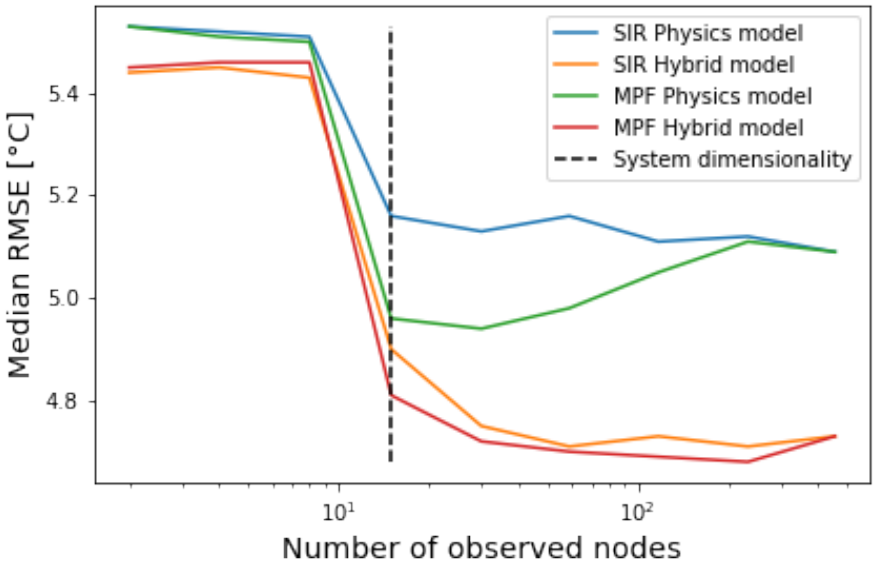

Figure 21: The median of the RMSE calculated across all nodes for different numbers of observed nodes.

Figure 22 shows the median RMSE calculated across the observed nodes of the model when different numbers of nodes are observed. Unlike for the RMSE calculated across all nodes, the RMSE now decreases when the number of observed nodes is reduced. It is probably easier for the data assimilation methods to infer states that fit fewer observed nodes well, rather than a larger number of observed nodes. Importantly, the fact that fewer observed nodes are fit better than a larger amount of observed nodes is not a benefit for the global performance of both the physics-based and hybrid models, as can be seen by comparing Figures 21 and 22. Therefore, a larger number of observed nodes remains desirable because of the benefits this has for global performance, despite the fact that it is a detriment to the performance on the observed nodes. Interestingly, the step in the RMSE that was observed for all nodes in Figure 21 appears to be absent for the hybrid models, though it is present for the physics-based models. As was the case for all nodes, the hybrid models always outperform the physics-based models on the observed nodes.

In Figure 23, the ratio of the median of the RMSE calculated across all nodes to the median of the RMSE calculated across the observed nodes is given for different number of observed nodes. The ratio is always greater for the hybrid models than for the physics-based models, though the ratios for the physics-based and hybrid models do not appear to diverge from each

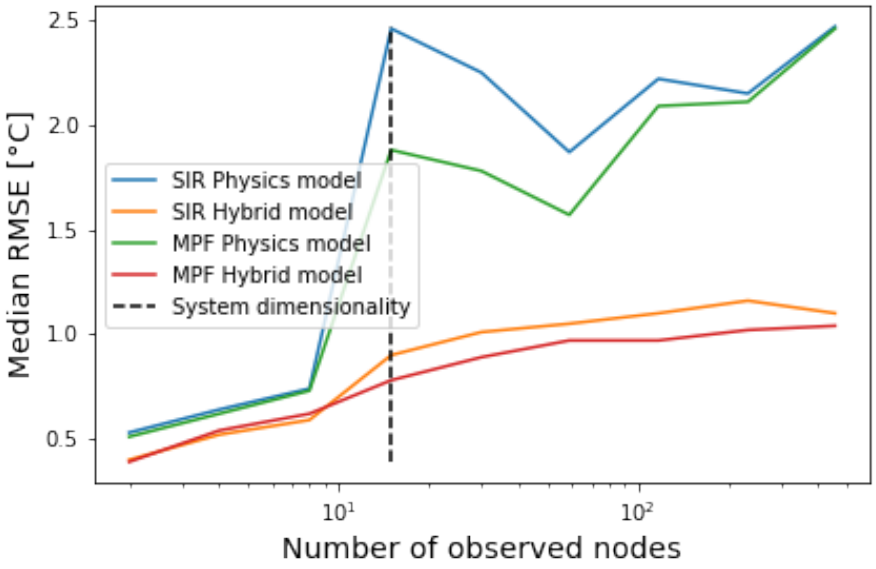

Figure 22: The median of the RMSE calculated across the observed nodes for different numbers of observed nodes.

other. This means that the disparity between the performance on the observed nodes and on all nodes does not become worse for the hybrid models relative to the physics-based models as the number of observed nodes is decreased. Figure 23 confirms for different numbers of observed nodes the observation from Table 13 that improvements in performance achieved by hybrid models relative to physics-based models come at the cost of an increased disparity in the performance on observed and on all nodes.

3.6 Discussion and conclusions

The case study performed in this section was intended to investigate the application of the hybrid model training methodology proposed in section 2. A simulated problem was used in this case study. There are numerous benefits to using a simulated problem for investigating the methodology used in this work. One of these is related to the fact that this work deals with partially observed systems. In a real-world application, one would only have access to the same partial observations of the system that are used to train the hybrid model for evaluating its performance, whereas in a simulated problem, information about the full system can be used in this evaluation. Another benefit is that it is much easier to vary the number of observations of the system and the locations of these observations in a simulated system than in a real-world system. However, the use of a simulated problem may fail to contain important nuances that are

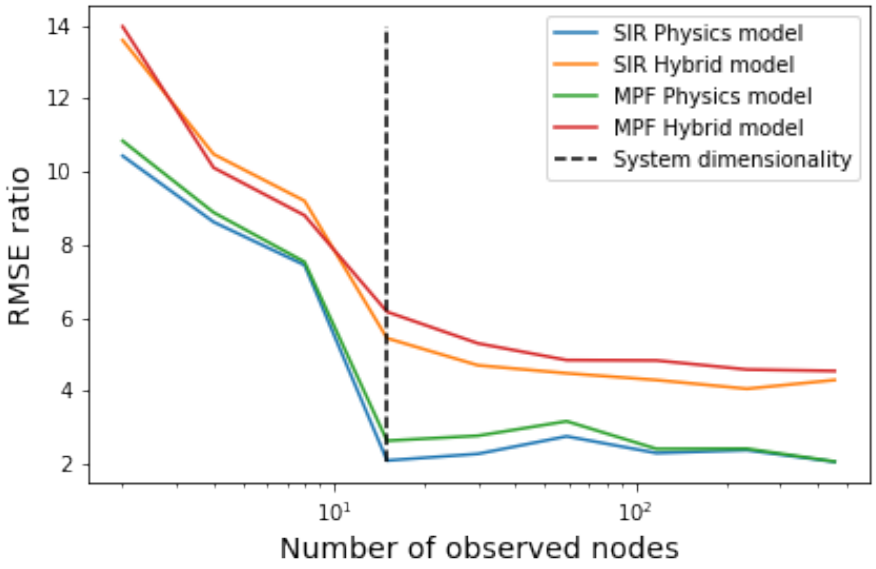

Figure 23: The ratio of the median of the RMSE calculated across all nodes to the median of the RMSE calculated across the observed nodes for different numbers of observed nodes.

present in real-world systems, and this should be taken into account when evaluating the results presented for this case study.

In this case study, the application of the POD dimensionality reduction technique is different from how it is described in section 2 in terms of how mean centring of snapshots and the selection of a sufficient number of POD modes is performed. While the number of POD modes that are used ultimately depends on the threshold ι, the results in section 3.2 suggest that the method as described in section 2 is more efficient for a given number of POD modes. It is possible, and arguably likely, that using the dimensionality reduction method as described in section 2 would increase the efficiency of the surrogate models and of the data assimilation approaches used in this case study if an appropriate threshold is used. Nevertheless, the results from this case study can still demonstrate the feasibility of training hybrid models. The use of a more efficient dimensionality reduction could be beneficial to hybrid models in terms of the relative performance of surrogate and hybrid models, because the data assimilation during hybrid model training could be more efficient. However, it could also be beneficial to the surrogate model, which itself could be more efficient and therefore be closer to the simulated observations.

Another factor of note is the fact that the methodology presented in section 2 and applied in this case study makes use of surrogate models to reduce the computational requirements of physics-based models. For the investigation into data assimilation approaches in section 3.4 and into

hybrid model training in section 3.5, a surrogate model was used as the physics-based model. This means that any improvements relative to the physics-based model that are due to using data assimilation or a hybrid modelling approach are relative to a surrogate model of a high-fidelity physics-based model. It is unknown based on the investigations performed in this case study whether this would be an improvement relative to the high-fidelity physics-based model as well. It would be interesting to conduct an experiment to investigate this, though it would be computationally expensive and challenging to implement using the Ansys software.

The simulated reality from which measurements were generated is different for the investigation into data assimilation approaches that was performed in section 3.4 and for the investigation into hybrid models performed in section 3.5. This is potentially important because the results from section 3.4 were used to motivate the choice of data assimilation parameters in section 3.5. As discussed in section 3.5 this is not necessarily a problem, because repeating computational experiments like those in section 3.4 to find more appropriate parameters in section 3.5 would mean using global information when in a real application only the sensor measurements would be available. The question that remains is whether the two simulated realities are similar enough that this problem still exists, or whether they are too different and the parameters found in section 3.4 are therefore not appropriate for use in section 3.5. A different approach to the selection of data assimilation parameters is used for case study 2 in section 4.5.

It is also important to keep in mind how the architecture selection for the data-driven component of the hybrid models was performed. The training set that was used for this purpose was not generated using any of the hybrid model training algorithms that were evaluated, but is similar in nature to a training set that would be encountered in the DA-O and per-step DA-O algorithms. This is perhaps a good method to avoid biasing the results in favour of a specific training algorithm, though the DA-O and per-step DA-O algorithms could be favoured over the ML-DA algorithm. A second point related to the architecture selection process is that architectures were evaluated based on their performance on the training set. This was done purposefully to investigate the predictive power offered by each architecture, ignoring their generalisation performance. For hybrid models, predictive power is perhaps more important since large datasets can be accumulated over time. Nevertheless, focusing instead on generalisation performance as is common practice could perhaps have led to better results in this case.

Thus far, this section has mentioned some factors that could have had an impact on the results obtained in this case study. The conclusions that can be drawn from these results will now be discussed. The surrogate models that were constructed in this case study indeed managed to provide a reduction in computational cost relative to the high-fidelity physics-based model

by multiple orders of magnitude. Neural network surrogate models trained on the simulation results resulted in the greatest computational saving, but were also the least accurate surrogate models. Neural network models trained on synthetic data performed better in terms of accuracy, but at approximately 3 times greater computational cost when compared to neural networks trained on simulation data. When neural networks are trained to predict the flow rate instead of the resolvent, a computational cost that is 2 orders of magnitude greater results, but depending on the application an improvement in accuracy over the other neural network models is achieved. One of the kernel methods resulted in the best surrogate model for both types of boundary conditions that were considered, but at greater computational cost than the neural network models. If fewer POD modes were to have been used, the computational cost of the kernel method surrogate models would be reduced directly, because they require a different surrogate model for each POD mode. However, since there are ~ 10 POD bases and the kernel method surrogate models had worse computational cost by at least a factor of 100 than the neural network surrogate models predicting the resolvent, this would not affect the conclusion that neural network surrogate models provide the best computational efficiency.

Another important conclusion that can be drawn from this case study is that the use of data assimilation methods can reduce the error in system state estimates compared to using the physics-based model without any sensor information. This is provided that the parameters of the data assimilation technique are chosen appropriately. The ability to outperform the forecasts made by physics-based models without access to sensor information is a prerequisite for being able to train hybrid models by making use of data assimilation techniques.

Likewise, it has been found that hybrid models can outperform physics-based models. This conclusion is unaffected by possibly poor data assimilation hyperparameter and data-driven component architecture choices, and more optimal choices in this regard are likely to be favourable to the hybrid models anyway. Of the hybrid model training algorithms that were implemented, the per-step DA-O algorithm performed best when 30 of the 29077 nodes of the system were observed. However, it is possible that this algorithm was favoured over the next-best performing algorithm, the ML-DA algorithm, by the architecture selection process. Furthermore, the ML-DA algorithm should be more robust to noisy measurements since training is not done directly on observations. The effect of measurement noise was not considered in section 3.5. It is also easier to apply the ML-DA algorithm to any number of observed nodes, since the per-step DA-O algorithm makes use of the pseudo-inverse in the loss function which is no longer full rank when the number of observed nodes decreases below the number of POD bases. Finally, the per-step DA-O algorithm may become computationally prohibitive as the number of sequen-

tial observations is increased because a neural network is trained each time a new observation becomes available. Based on these factors, the ML-DA algorithm is the preferred option for further case studies.

A final finding from this case study is that hybrid models can outperform physics-based models even when the number of observed nodes is reduced to only 2. This result is likely application-specific, but it shows that the sparse observation of a system does not necessarily make the training of hybrid models futile. A larger number of observed nodes was beneficial to the accuracy of the hybrid models that were trained. The results suggest, however, that hybrid model performance eventually becomes constant when more observed nodes are added. This again supports the training of hybrid models for sparsely observed systems.

4 Case study II: half model of process converter freeboard

This case study explores the application of the methodology presented in section 2 to a larger model, which is intended to be reflective of a realistic application. Whereas the model used for the first case study in section 3 was of a small section of the process converter freeboard, the model used in this section is a half model of the process converter freeboard for which real sensor measurements are available. The simulation model used for this case study is discussed in more detail in section 4.1, and section 4.2 introduces the real sensor measurements that are available for this case study. Section 4.3 discusses dimensionality reduction for the model, while section 4.4 considers the construction of surrogate models. Computational experiments with data assimilation and hybrid models using simulated measurements are documented in sections 4.5 and 4.6 respectively. Similar experiments are conducted in sections 4.7 and 4.8 with data assimilation and hybrid models respectively. Finally, section 4.9 presents a discussion of the results obtained for this case study and draws conclusions from them.

4.1 Simulation model

The simulation model used for this case study is a half model of the process converter freeboard. It was constructed by Mr. Roelof Minnaar, and the geometry, mesh, parameters and boundary conditions of the model are derived from his work. As for case study 1, the simulations were performed using Ansys Mechanical, though this time the 2021 R2 version was used. The geometry of the model is shown in Figure 24. The geometry was meshed using shell elements, with the final mesh consisting of 8.1×10^5 nodes. The shell elements are set to allow quadratic thermal variation through their thickness. Each node has three degrees of freedom: the top, bottom and middle temperature. This gives the model a total of 2.4×10^6 degrees of freedom.

The thermal boundary conditions of the model are simple convection boundary conditions that can be divided into three categories: the fireside, waterside and airside boundary conditions. Fireside boundary conditions are applied to all surfaces on the inside of the freeboard, while airside boundary conditions are applied to all surfaces on the outside. The waterside boundary conditions are applied to the inside of the freeboard tubes. No spatial variation in the waterside and airside boundary conditions were considered. The possibility of a coarse spatial variation in the fireside boundary conditions was considered by means of independently specifying the convection coefficient for the differently coloured fireside regions in Figure 24. This results in up to 10 distinct regions of fireside boundary conditions. Throughout this case study, no

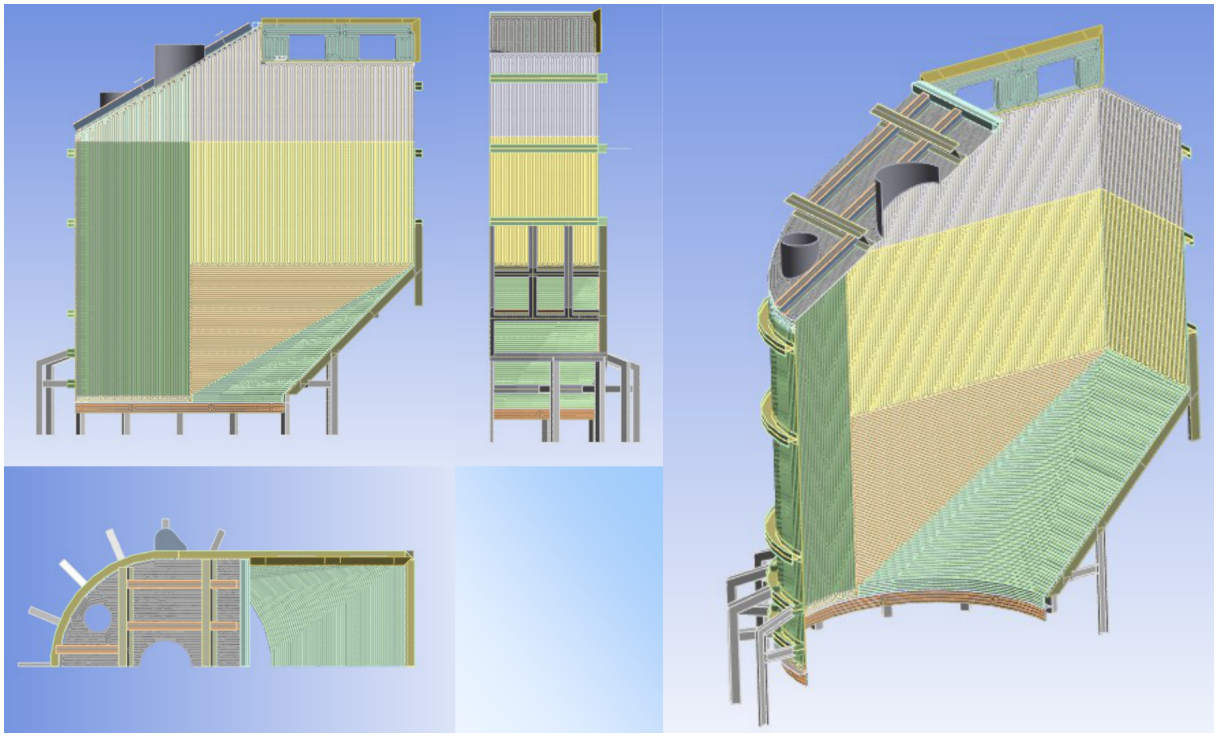

Figure 24: Different views of the simulation model used for this case study.

variation in airside boundary conditions was considered. Likewise, the far-field temperature of the fireside boundary conditions and the convection coefficient of the waterside boundary condition were not varied. Changes in boundary conditions in this case study were therefore achieved through variations in the fireside convection coefficients and the waterside far-field temperature. Table 14 summarises the range of values that were considered for each parameter of the different convection boundary conditions.

Table 14: The range of values that were considered for the parameters of each of the three types of convection boundary conditions applied in the simulation model.

Boundary condition	**Far-field temperature** [$^{\circ}C$]	**Convection coefficient** [$W/m^2\ ^{\circ}C$]
Airside	35	0.535
Fireside	1350	24-425
Waterside	175-250	13000

4.2 Sensor measurements

The process converter is observed by means of a network of sensors that is the result of work conducted by Mr. De Wet Strydom. It is equipped with over 100 sensors to monitor the temper-

ature and strain at various locations during operation. Of these, 17 sensors are thermocouples, 13 of which are installed on the freeboard. Because the simulation model is a half model, only seven of the thermocouples are within the domain of the simulation model. These seven sensor locations are shown in Figure 25. The sampling frequency of the thermocouples is $1\,Hz$, and data are available from over $3000\,h$ of operation. A subset of the available data is used for data assimilation and for the training of hybrid models in sections 4.7 and 4.8 respectively.

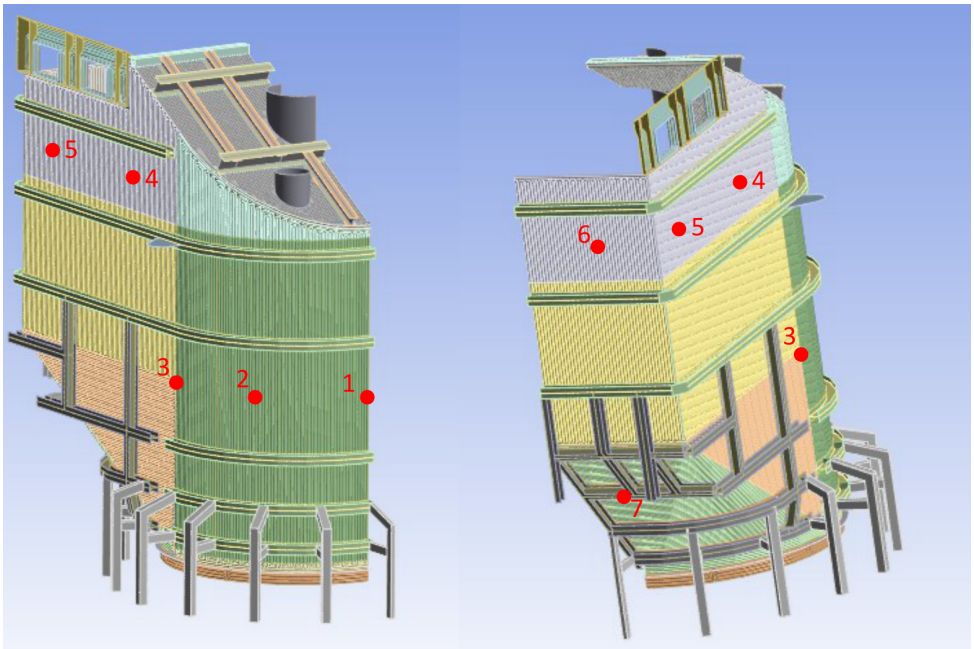

Figure 25: The seven sensor locations at which real measurement data is available. Each location is numbered for reference.

4.3 Dimensionality reduction

Unlike in section 3.2, POD dimensionality reduction in this case study was done exactly as described in section 2.1.1. The main issue that therefore remains to be discussed is the process of generating snapshots for the extraction of POD modes. For the generation of snapshots, transient simulations with 100 time steps of size $1\,s$ were performed. These simulations were performed with different combinations of boundary condition parameters. To obtain different combinations of boundary condition parameters, each boundary condition parameter was given either a high setting, which is the maximum possible value according to Table 14, or a low setting, which is the minimum possible value according to Table 14. In addition to this variation in the boundary condition parameters, two different initial conditions were used. One of these initial conditions is the steady-state condition that is reached when each boundary condition

parameter is in the high setting, and the other is the steady-state condition that is reached when each boundary condition parameter is in the low setting.

A total of 24 different transient simulation runs were performed to obtain the snapshot matrix, which as a result consists of 2400 snapshots. This first of these runs was initialised using the steady-state condition that is reached when each boundary condition parameter is in the high setting, with all boundary condition parameters during the transient simulation being in the low setting. A second run was performed with the steady-state condition that is reached when each boundary condition parameter is in the low setting as initial condition, while all boundary condition parameters in the transient simulation were in the high setting. For the remaining 22 simulation runs, 11 different combinations of boundary condition parameters were obtained by setting only one of the variable boundary condition parameters to the high setting while the others remained in the low setting. For each of these combinations, one simulation run was performed using as initial condition the steady-state condition that is obtained when each boundary condition parameter is in the high setting, and another was performed using the steady-state condition that is obtained when each boundary condition parameter is in the low setting as the initial condition.

The 13 different combinations of boundary conditions described above are intended to include the effect of the variation of each of the boundary condition parameters in the snapshot matrix. The somewhat elaborate methodology that uses these combinations of boundary conditions with different initial conditions is grounded in some experimentation. At first, all simulation runs were initialised using a uniform temperature of $250\,^{\circ}C$. After the extraction of POD modes, a sanity check was performed by evaluating the ability to reconstruct the steady state condition that is reached when each boundary condition parameter is in the high setting, as well as the steady state condition that is reached when each boundary condition parameter is in the low setting. The POD modes extracted using these snapshots were not able to reconstruct these two steady-state conditions well. As a result, it was decided to rather capture the dynamics of the system between these two steady state conditions by using them as initial conditions for the simulation runs used in snapshot generation. Given that conditions close to the two steady-state conditions that had previously been used to evaluate the performance of the dimensionality reduction technique were now represented in the snapshot matrix, and that the reconstruction performance on these conditions would therefore be expected to be good anyway, a new set of steady-state conditions was used instead to evaluate the performance of the dimensionality reduction technique. These steady-state conditions were those reached with the 11 boundary condition parameter combinations in which only one of the parameters is in the high setting.

The ability to reconstruct these steady state conditions using the revised technique was found to be acceptable.

A total of 16 POD modes were extracted using the 2400 snapshots. The number of modes was chosen using equation 11 with a threshold of $\iota = 0.999$. As discussed above, the ability of these 16 POD modes to reconstruct 11 different steady-state conditions was evaluated. This was done using mean squared error and mean absolute error criteria defined by

$$\text{MSE} = \frac{\sum(\mathbf{\Phi}_K \mathbf{\Phi}_K^T \mathbf{V} - \mathbf{V})^2}{O},$$

and

$$\text{MAE} = \frac{\sum|\mathbf{\Phi}_K \mathbf{\Phi}_K^T \mathbf{V} - \mathbf{V}|}{O}.$$

Here O is the dimensionality of the snapshots, $\mathbf{V}$ is the mean centred snapshot matrix and $\mathbf{\Phi}_K$ is the matrix of K POD modes. The square and absolute value operators are elementwise, and the summation takes place over all elements. The average mean squared error across the 11 steady-state conditions that were reconstructed is 81.2 $(^\circ C)^2$, and the average mean absolute error is 1.98 $^\circ C$.

4.4 Surrogate models

This section discusses the construction of surrogate models of the simulation model used for this case study. The inputs of these surrogate models are the 16 POD coefficients obtained after dimensionality reduction as well as the 11 variable boundary condition parameters, resulting in an input space with 27 dimensions. The outputs of the surrogate models are solely the 16 POD coefficients at the next time step. The training data for the surrogate models was obtained by running transient simulations with 10 time steps of size $0.1\ s$.

As in the first case study, the initialisations and boundary conditions for these simulation runs were obtained from a Latin hypercube experimental design. In this case study, the Latin hypercube consisted of 27 dimensions and 512 design points, representing a doubling of the amount of training data available for the surrogate models when compared to the first case study. The Latin hypercube design was again optimised using the LaPSO-L algorithm from [71]. In this case study, 10240 swarms were used in the optimisation algorithm, each with a population of 32. The optimisation algorithm was run for 500 iterations. This represents a significant in-

crease in the computational resources devoted to finding an optimal experimental design when compared to the first case study. An additional factor contributing to increased optimality of the experimental design is that no dimensions were dropped from the Latin hypercube after optimisation, unlike in the first case study.

Like in the first case study, the extremities of the design space needed to be determined. The minimum and maximum values of the 11 variables corresponding to boundary condition parameters are given in Table 14. The remaining 16 variables correspond to the POD coefficients. Their minimum and maximum values correspond to the minimum and maximum values the POD coefficients take in the entire snapshot matrix, as well as in the 13 steady-state conditions corresponding to the 13 different combination of boundary condition parameters that were used in the snapshot generation process.

For this case study, surrogate models were constructed using Gaussian processes, neural networks, support vector regression and learning of the flow rate by a neural network as was done in the first case study. In addition, relevance vector machine surrogate models were constructed using the methodology discussed in section 2.1.2.4. As training data, 448 of the 512 available simulation results were used. The remaining results were used as a test set. The discrete time surrogate models were trained to predict the system evolution over $1\,s$, and the flow rate surrogate models were trained using the simulation result after $1\,s$. The TensorFlow implementation of neural network regression was used to construct the neural network surrogate models. Unlike in the first case study, no synthetic data neural network surrogate model was constructed, and the validation set used for early stopping made up 12.5% of available training data. For the neural network flow rate model, the validation set size remained 20% of the available training data. Whereas only the architecture suggested by equations 8 and 9 was used for the neural network flow rate model in the first case study, this case study also explores architectures that are obtained using the following modifications of equations 8 and 9:

$$L_1 = \sqrt{\tau N(o+2)} + 2\sqrt{\frac{\tau N}{o+2}} \tag{16}$$

and

$$L_2 = o\sqrt{\frac{\tau N}{o+2}}. \tag{17}$$

Here, τ is a multiplier that modifies the training set size N. This multiplier allows the adjustment of the expressiveness afforded by the architecture by adjusting how many training set samples it could replicate exactly. For the flow rate neural network model, 10 time steps were

used in the Runge-Kutta integration.

The performance of each surrogate model was evaluated exactly like it was evaluated in the first case study in section 3.3. Table 15 gives the results for the different surrogate model configurations that were evaluated. The Gaussian process surrogate model with the automatic relevance determination (ARD) kernel performed best in terms of the best output parameter and in terms of the average across all output parameters. In terms of the worst output parameter, the support vector regression (SVR) surrogate model performed best. When a neural network is used to model the flow rate, the resulting surrogate model outperforms the neural network model of discrete time steps, except when the architecture of the neural network flow rate model is chosen using $\tau = 0.01$. The best neural network flow rate model is obtained when the architecture is chosen using $\tau = 0.1$. The re-estimation RVM outperforms the constructive RVM, though it should be noted that the re-estimation RVM uses an average of 143 basis functions to make predictions, whereas the constructive RVM uses only 25 basis functions on average. The constructive RVM is therefore more sparse than the re-estimation RVM.

Table 15: The minimum, average and maximum % RMSE across 16 output parameters for a surrogate model of the transient simulation. The results of the best performing model are printed in italics and underlined.

Surrogate model	**Minimum**	**Average**	**Maximum**
Gaussian process, square exponential	0.0126	0.244	0.642
Gaussian process, Matérn, $\nu = 5/2$	0.0295	0.199	0.574
Gaussian process, ARD	*0.00872*	*0.133*	0.637
Neural network	0.76	0.971	1.38
Support vector regression	0.0571	0.182	*0.284*
RVM, re-estimation	0.0748	0.332	0.556
RVM, constructive	0.12	0.545	0.725
Neural network flow rate, $\tau = 0.01$	1.05	3.59	5.87
Neural network flow rate, $\tau = 0.1$	0.0703	0.294	0.497
Neural network flow rate, $\tau = 0.25$	0.0965	0.327	0.5
Neural network flow rate, $\tau = 1$	0.205	0.404	0.53
Neural network flow rate, $\tau = 10$	0.218	0.435	0.587
Neural network flow rate, $\tau = 100$	0.222	0.45	0.638

Like in the first case study, the computational efficiency of the different surrogate models was evaluated. Unlike in the first case study, no comparison with the simulation model is made. The reason is that simulations were performed utilizing between 16 and 20 cores of an Intel Xeon Gold 5218 CPU, while the surrogate models were evaluated on an Intel i7-11700 CPU. Nevertheless, comparisons between different surrogate models are also informative. The com-

putational implications of the different surrogate models were evaluated using two different tasks. The first task is to replicate simulation results for 100 time steps using a single initial condition. The other task is to replicate 2048 single time steps using different initial conditions. While in the first task the 100 time steps must be performed sequentially, the 2048 time steps in the second task can be computed in parallel. The wall-clock time required to complete these tasks was evaluated using the `timeit` module like in the first case study.

Table 16 gives the wall-clock time required for the different surrogate models on each of the two tasks. To complete both tasks, the Gaussian process surrogate models require at least an order of magnitude more wall-clock time than any other surrogate model type. Still, only the Gaussian process model with the ARD kernel is not faster than real time on task 1. The SVR and RVM models perform the best on task 1 in terms of computational cost, and are an order of magnitude faster than the neural network models on this task. All surrogate models benefit from the possibility of performing the 2048 simulations that make up task 2 in parallel, as none of the wall-clock times have increased by more than a factor of 6 compared to task 1. This is despite the fact that the number of time steps to evaluate is more than 20 times greater than in task 1. In fact, only the SVR and RVM models require more wall-clock time for task 2 than for task 1, with all other models requiring less time. The performance of the neural network and neural network flow rate models with $\tau \leq 1$ are comparable despite the fact that 10 Runge-Kutta integration steps must be performed for the flow rate models. The performance of these neural network models on task 2 is comparable to the performance of the constructive RVM on task 1 in spite of the 20-fold increase in the number of time steps for task 2. It should be noted that no explicit efforts were made to exploit parallel computation for any of the surrogate models, rather the ability of packages such as NumPy [93] and TensorFlow to do this were exploited.

4.5 Data assimilation with simulated measurements

This section presents an investigation into the application of data assimilation techniques to the problem considered in this case study with simulated measurements. The advantage of considering simulated measurements before the real measurements that are available for this case study is that the real measurements represent the only information that is available about the real system, and the only information that can be used to evaluate the performance of the data assimilation techniques. In the first case study, information about the entire system was available because it was a simulation-based study. The aim of the investigation performed in this section is to investigate the performance of data assimilation techniques both at the observed

Table 16: The wall-clock time in seconds required for each surrogate model to perform two different tasks. Task 1 is to replicate simulation results for 100 time steps, and task 2 is to perform 2048 single time steps using different initial conditions. The wall-clock time is for an Intel i7-11700 CPU.

Surrogate model	**Task 1** $[s]$	**Task 2** $[s]$
Gaussian process, square exponential	72	17
Gaussian process, Matérn, $\nu = 5/2$	76	18
Gaussian process, ARD	110	29
Neural network	2.4	0.044
Support vector regression	0.16	1.1
RVM, re-estimation	0.11	0.63
RVM, constructive	0.056	0.31
Neural network flow rate, $\tau = 0.01$	2.7	0.045
Neural network flow rate, $\tau = 0.1$	2.8	0.062
Neural network flow rate, $\tau = 0.25$	2.8	0.058
Neural network flow rate, $\tau = 1$	2.8	0.064
Neural network flow rate, $\tau = 10$	2.9	0.19
Neural network flow rate, $\tau = 100$	4.0	0.47

nodes and over the entire system to see whether there is a relationship between the performance of these techniques at the observed nodes and their performance over the entire system.

In sections 3.4 and 3.5, simulated observations were generated from simulations with constant boundary conditions. Here, time-varying boundary conditions were used in the process of generating simulated observations. To obtain time-varying boundary conditions, boundary condition parameters that vary randomly but smoothly and with a characteristic length scale were specified. This random variation was obtained by sampling from a Gaussian process prior (i.e. a Gaussian process model with no observations on which predictions are conditioned). A square exponential kernel function was used with a length scale $l = 50$. A different function was sampled from the Gaussian process prior for each of the 11 independent boundary condition parameters, and the function values were scaled so that their minimum and maximum values correspond to the minimum and maximum values of each parameter given in Table 14. As an example of how the boundary condition parameters vary with time, see Figure 26, which shows the variation in the waterside far-field temperature over time. With the time variation of each boundary condition parameter available, a 1000 time step simulation with a time step size of $1\,s$ was performed for the generation of observations. This simulation was initialised using the steady-state condition that is reached when all boundary condition parameters are in the low setting, which was one of the initialisations considered in section 4.3.

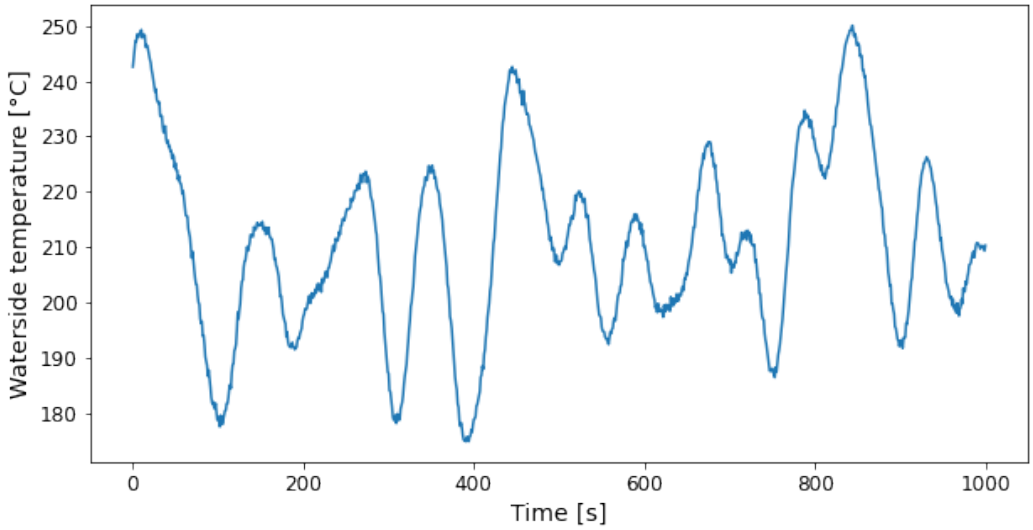

Figure 26: The variation of the waterside far-field temperature with time in the simulation used to generate observations.

The exercise of training hybrid models on simulated data is most fruitful when it is done using simulated observations from locations on the model corresponding to the locations of the actual sensors, which are shown in Figure 25. To simulate measurements from these locations as well as to use the model for data assimilation, these sensor locations must be mapped onto nodes of the model mesh. To do this, the node nearest to each sensor was found, which was then assumed to coincide with the sensor for the purposes of generating observations and for data assimilation. When generating measurements, random noise generated according to $\mathcal{N}(0, 1)$ was added to each measurement to simulate measurement noise.

Two different data assimilation algorithms were considered in this section: the SIR filter and the MPF. In sections 3.4 and 3.5 some knowledge about the initial state of the system was assumed to exist for the purpose of initialising the data assimilation algorithm. Here, no such knowledge is used, except that it is assumed that all parameters are within the range of values used for the design space in section 4.4. The initialisation of each parameter is from a uniform distribution over this range of values. There are 27 parameters that are inferred by the data assimilation procedure, made up of the 16 POD coefficients and the 11 boundary condition parameters. For both the SIR filter and the MPF, ensemble sizes of 2048 were used.

In this case study, the neural network flow rate surrogate model with $\tau = 0.1$ was used as the physics-based model. This choice was made based on its performance in Table 15 and

on its computational properties on task 2 in Table 16. The task 2 performance in Table 16 is representative of the task that is performed at each time step in a particle filter data assimilation algorithm. Note that this choice of physics-based model allows the assimilation of data at a rate that is faster than real time. The physics-based model describes the evolution of the 16 POD coefficients between successive time steps, but does not model the evolution of the boundary condition parameters. The 11 boundary condition parameters were assumed to persist between time steps. As was the case in section 3.4, the system noise was assumed to be distributed according to $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, where the entries of the covariance matrix are given by $Q_{ij} = \delta_{ij}\left(\frac{q_i}{\gamma}\right)^2$. As before, δ_{ij} is the Kronecker delta, q_i is the difference between the minimum and maximum values of the i^{th} parameter, and γ is a parameter.

Unlike in section 3.4, it was not assumed that the parameters of the measurement noise are exactly known. Still, the measurement noise was assumed to be distributed according to $\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, where the entries of the covariance matrix are given by $R_{ij} = r^2\delta_{ij}$ and r is a parameter. Had the approach from section 3.4 been followed, $r = 1$ would have been used since this was the value used to generate measurement noise. Here, however, r is allowed to take on different values.

The investigation conducted in this section consisted of using different combinations of values of γ and r to attempt to find an optimal combination of these parameters. The primary objective of this investigation was to determine whether the performance of the data assimilation algorithms on the observed nodes could serve as a reliable indicator for the performance of the data assimilation algorithms across all nodes, as well as on the boundary condition parameters. To accomplish this, 10 logarithmically spaced values of γ between 10 and 100 and of r between 1 and 20 were used, resulting in 100 different combinations of γ and r. For each combination, 10 data assimilation runs with different initialisations and measurement noise instantiations were used to account for the stochastic nature of performing data assimilation by particle filters. The results presented for the remainder of this section are averaged over these 10 runs.

The RMSE was calculated separately for each time step using equation 15, and then was averaged over the 10 data assimilation runs and 1000 time steps of each run to obtain a single average RMSE metric for each combination of parameters. Note that equation 15 can be modified for calculating the RMSE over a subset of the nodes of the system or for calculating the RMSE over the boundary condition parameters. Figure 27 shows a contour plot of the RMSE calculated both across all nodes and across the observed nodes for all 100 combinations of the γ and r parameters. The results used to make Figure 27 are for the MPF, though similar results are obtained for the SIR filter. Qualitatively, there appear to be minor similarities at best between

these two contour plots in Figure 27, though the minimum RMSE appears to be similarly located. The RMSE calculated across all nodes is more noisy than the RMSE calculated across the observed nodes.

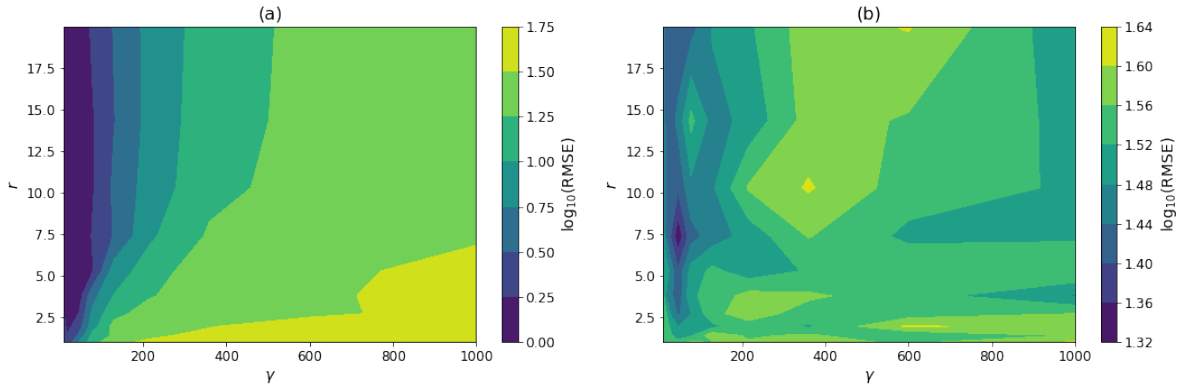

Figure 27: Contour plots of the RMSE calculated across (a) observed nodes and (b) all nodes for different values of the parameters γ and r. Results are for the MPF.

It is possible that the correlation between the RMSE calculated over the observed and over all nodes in Figure 27 is affected or even dominated by the nodes that are most independent of the observed nodes and that can thus be poorly be inferred by the data assimilation algorithm. To investigate this, the covariance of each node with each of the observed nodes was calculated using the snapshots in the snapshot matrix that was used for POD in section 4.3. Before calculating these covariances, the values of each node were standardised (i.e. mean centred and then divided by the standard deviation). The RMSE was then calculated over subsets of all nodes based on this covariance. Figure 28 shows contour plots for the RMSE calculated over the 5% and 60% of the nodes that have the greatest covariance with the observed nodes. To determine which nodes belong to these subsets, the maximum covariance of each node with any of the observed nodes was used. Both Figures 28 (a) and (b) appear qualitatively more similar to Figure 27 (a) than Figure 27 (b), as well as similar to each other.

To obtain a more direct view of the correlation between the RMSE calculated across the observed nodes and the RMSE calculated across all nodes or across subsets of all nodes based on covariance with the observed nodes, Figure 29 shows the RMSE calculated across the observed nodes plotted against the RMSE calculated across all nodes and against the RMSE calculated across the 60% of the nodes that have the greatest covariance with the observed nodes. A visible correlation exists in both cases, though it is clearly stronger when the RMSE is calculated across the 60% of the nodes that have the greatest covariance with the observed nodes. A quantification of the correlation of various RMSE metrics with the RMSE calculated across the observed

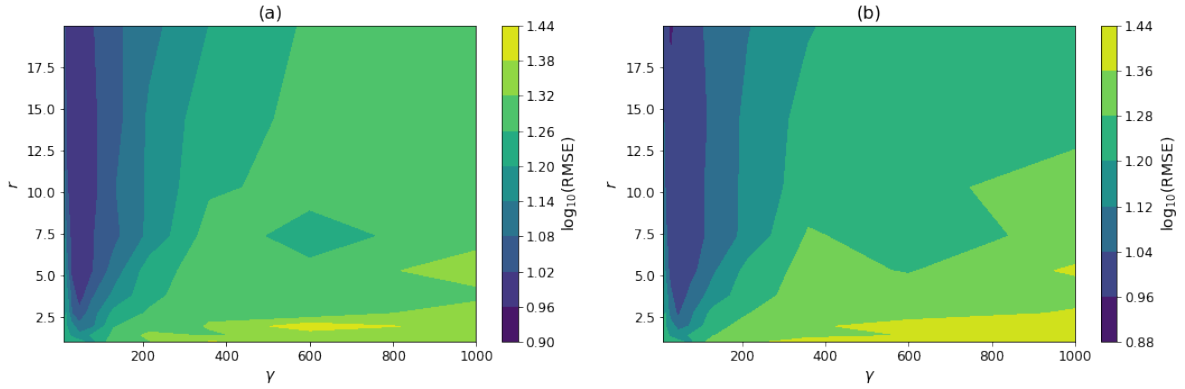

Figure 28: Contour plots of the RMSE calculated across (a) the 5% and (b) the 60% of nodes that have the greatest covariance with the observed nodes for different values of the parameters γ and r. Results are for the MPF.

nodes using the Pearson correlation coefficient is given in Table 17.

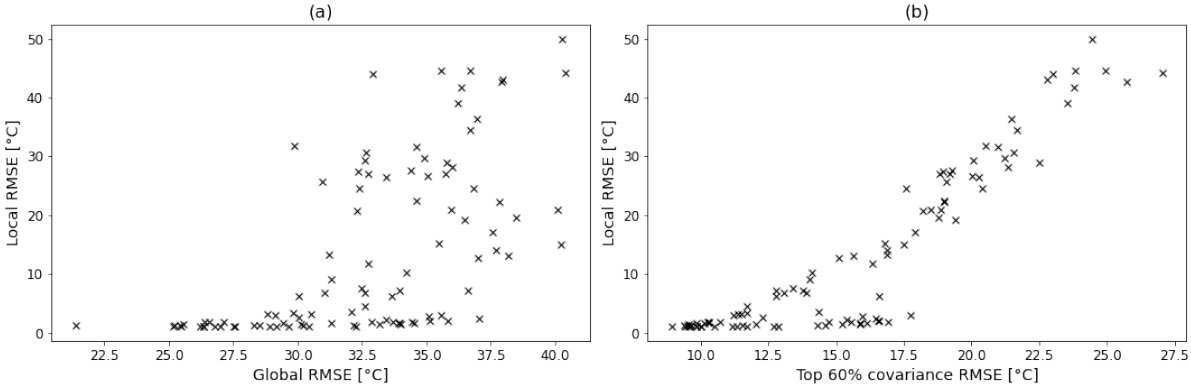

Figure 29: The local RMSE calculated across observed nodes plotted against (a) the global RMSE calculated across all nodes and (b) the RMSE calculated across the 60% of the nodes that have the greatest covariance with the observed nodes. Results are for the MPF.

Table 17 also provides the minimum RMSE calculated over the observed nodes, the boundary conditions and over various subsets of all nodes of the model for any combination of the γ an r parameters. The MPF outperforms the SIR filter in terms of all RMSE metrics. The RMSE calculated across the boundary condition parameters is an order of magnitude worse than the RMSE calculated across the nodes of the model, indicating that it is difficult to infer the boundary condition parameters in the absence of a model of their evolution. The SIR filter shows a better correlation between the RMSE calculated across the observed nodes and other RMSE metrics, except for the RMSE calculated across the boundary condition parameters. The results both in terms of RMSE values and in terms of correlation coefficients change by less

than 5% for all subsets of nodes chosen based on their covariance with the observed nodes. The results show a clear indication that the RMSE calculated over the observed nodes can be a reliable indicator of the RMSE calculated over large fractions of the domain of the model.

Table 17: The RMSE calculated over various subsets of all nodes of the model, as well as the RMSE calculated for the boundary condition parameters for the best combination of data assimilation parameters in each case. Also given is the Pearson correlation coefficient of these RMSE values with the RMSE calculated over observed nodes. This correlation coefficient is calculated over all 100 combinations of parameters. Except in the case of the boundary condition parameters, the RMSE is in units of $^{\circ}C$. Since the boundary condition parameters are a combination of temperatures and convection coefficients, no units are given.

	RMSE		**Correlation**	
RMSE calculated over	**SIR**	**MPF**	**SIR**	**MPF**
All nodes	27.5	21.4	0.58	0.56
Observed nodes	1.55	1.00	1	1
Boundary condition parameters	94.6	74.7	0.72	0.73
Top 5% covariance	12.4	8.93	0.93	0.89
Top 10% covariance	12.4	8.91	0.91	0.87
Top 20% covariance	12.3	8.86	0.90	0.85
Top 40% covariance	12.4	8.97	0.90	0.86
Top 60% covariance	12.5	9.09	0.93	0.88

4.6 Hybrid models with simulated measurements

In this section, the training of hybrid models with simulated sensor measurements is considered. The objective of the investigation conducted in this section is to investigate whether the finding from the first case study, that it is possible to improve on the performance of physics-based models using hybrid models, extends to this case study as well. Another objective is to ascertain whether improvements in performance on the observed nodes extend to improvements on the entire system, which is similar to what was investigated for data assimilation in section 4.5. This investigation used the same series of simulated measurements that was also used in section 4.5.

For this investigation, the γ and r parameters of the data assimilation algorithm that resulted in the best performance on the observed nodes in section 4.5 were used. This is representative of the real situation in which additional information would not be available to choose parameters for optimal performance over the entire system. The parameters chosen based on these considerations are $\gamma = 16.68$ and $r = 14.34$. The value of r is substantially different to the value

$r = 1$ that was used to generate measurement noise. It is believed that this difference may be a result of poor POD reconstruction over some parts of the state trajectory, as well as a result of better sample diversity in particle filtering algorithms when r is larger. The same initialisation of the data assimilation algorithm as in section 4.5 was used. The MPF algorithm was used in this section, based on the fact that it outperformed the SIR filter on the observed nodes in section 4.5. This choice could be further supported by the superior performance of the MPF on all nodes of the model as well, though such a comparison would not be possible with real measurements.

Two different physics-based models were considered. The first is the neural network flow rate model with $\tau = 0.1$ that was also used in section 4.5. While this model has some bias relative to the simulation model that was used to generate the simulated measurements because it is a surrogate model, another model with deliberately introduced bias was also considered. For this second model, bias was introduced by multiplying every second POD coefficient by 0.8 before parameters are input into the neural network flow rate model with $\tau = 0.1$.

The ML-DA algorithm was used in this section. It was chosen based on its performance in section 3.5, as well as based on the fact that there are fewer observed nodes than there are POD coefficients in this case study. The latter property of the problem makes the application of the DA-O algorithms difficult. A further point in favour of the ML-DA algorithm is that the observation time series in this case study can become long, making the need to train a model for each observation in the DA-O algorithms undesirable. The ML-DA algorithm was run for 15 iterations. For each iteration, the first 50 time steps were not considered as part of the training data for the hybrid model to account for burn-in of the data-assimilation algorithm. A neural network model was used as the data-driven component of the hybrid model, and was used to perform resolvent correction. The architecture of the neural network was chosen based on equations 8 and 9, and hyperbolic tangent activation functions were used.

A total of 30 training runs were conducted with each of the two physics-based models that were used in this section. The performance of the hybrid model after each iteration of the ML-DA algorithm was evaluated using the analysis states produced by the data assimilation procedure that is completed as part of the ML-DA algorithm at the subsequent iteration. This evaluation was performed using equation 15, which was modified as discussed in section 4.5 for calculation of the error across different subsets of the nodes of the model. For each iteration of the ML-DA algorithm, the RMSE was averaged over all 1000 time steps as well as over the 30 hybrid model training runs. Figures 30 and 31 use this average RMSE to show the change in performance of the hybrid models as the ML-DA algorithm progresses for the physics-based models without

and with deliberately introduced bias respectively.

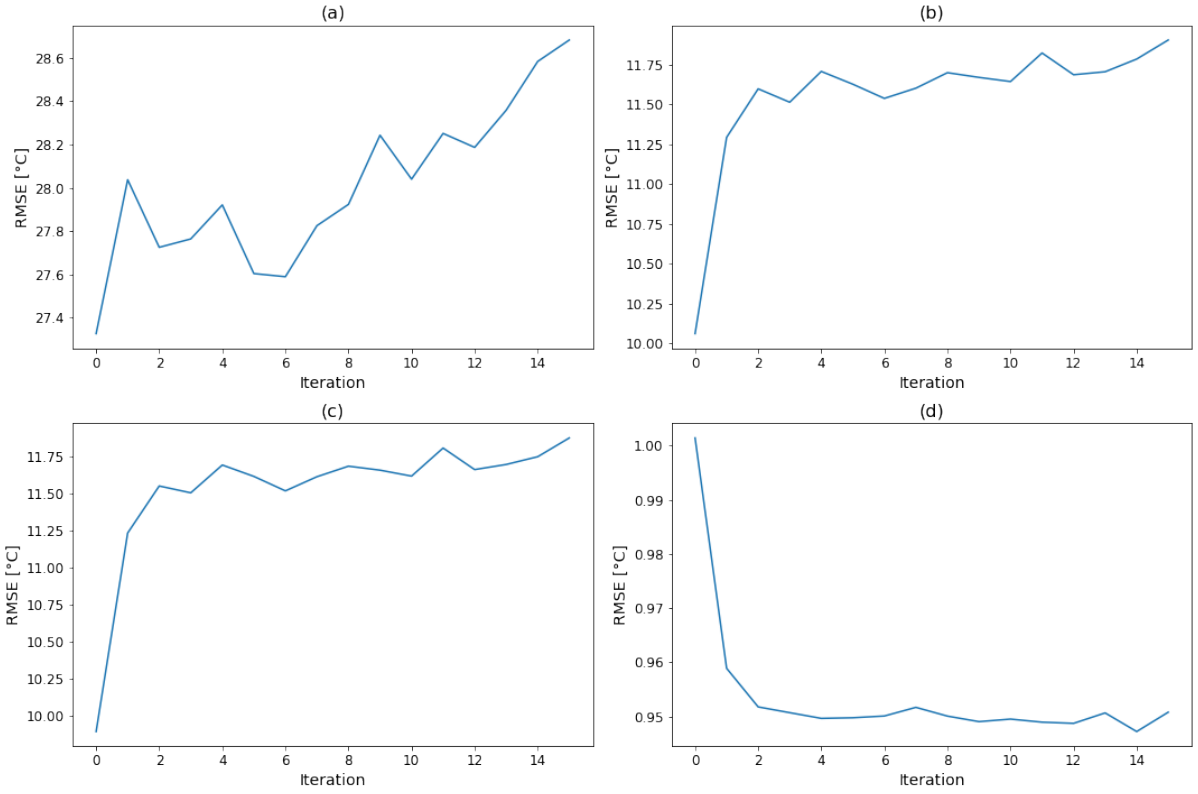

Figure 30: The average RMSE calculated across (a) all nodes, (b) the 60% of the nodes with the greatest covariance with the observed nodes, (c) the 5% of the nodes with the greatest covariance with the observed nodes and (d) the observed nodes at different iterations of the ML-DA algorithm. Iteration 0 corresponds to the physics-based model only, while the results for each subsequent iteration correspond to the hybrid model after that iteration. The results are for the physics-based model without deliberately introduced bias and the accuracy is of the analysis states estimated by the data assimilation algorithm.

As Figure 30 (d) shows, the performance of the hybrid model on the observed nodes improves as the ML-DA training algorithm progresses if the physics model without deliberately introduced bias is used. Most of the improvement has occurred by the time the second iteration of the algorithm has completed. This improvement of the hybrid model does not generalise to any of the larger sets of nodes over which the RMSE was calculated. Rather the performance of the hybrid model for these nodes becomes worse as the ML-DA training algorithm progresses, as can be seen in Figure 30 (a) - (c). Figure 30 (b) and (c) appear to be an inversion of Figure 30 (d) in the sense that most of the degradation in hybrid model performance has occurred after the completion of the second iteration.

As was the case for the physics-based model without deliberate bias introduced, Figure 31 (d)

shows an improvement of the hybrid model on the observed nodes as the ML-DA algorithm progresses when the physics-based model with deliberate bias introduced is used. However, in this case improvement is also observed on all other sets of nodes over which the RMSE was calculated, as shown in Figure 31 (a) - (c). The progression of the improvement is similar in Figure 31 (b) - (d), where the improvement has mostly occurred by the end of the second iteration. Interestingly, when the RMSE is calculated across all nodes, the performance of the hybrid model continuously improves as the ML-DA algorithm progresses despite there being no further improvement on the other subsets of nodes that were considered.

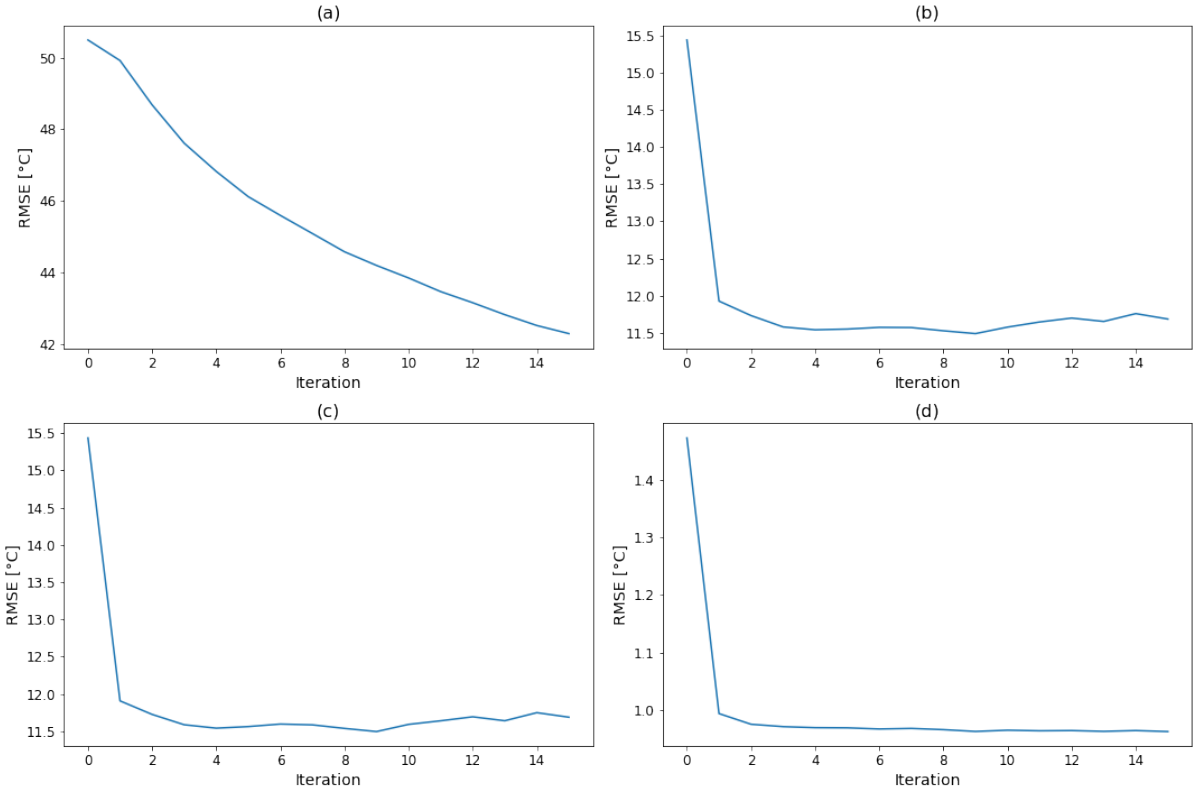

Figure 31: The average RMSE calculated across (a) all nodes, (b) the 60% of the nodes with the greatest covariance with the observed nodes, (c) the 5% of the nodes with the greatest covariance with the observed nodes and (d) the observed nodes at different iterations of the ML-DA algorithm. Iteration 0 corresponds to the physics-based model only, while the results for each subsequent iteration correspond to the hybrid model after that iteration. The results are for the physics-based model with deliberately introduced bias and the accuracy is of the analysis states estimated by the data assimilation algorithm.

Table 18 gives some numerical results on the performance of the hybrid modelling approach pursued in this section. The performance of the physics-based models given in Table 18 is also available at iteration 0 in Figures 30 and 31, and these figures also show the performance of the hybrid models at iteration 15. The results in Table 18 show that there is a 5% reduction in the

RMSE on the observed nodes when a hybrid model is constructed using the unbiased physics-based model. This improvement does not generalise to larger subsets of the nodes of the model, as was already seen in Figure 30. When the physics-based model with deliberately introduced bias is used, the use of a hybrid model results in a 35% reduction in the RMSE on the observed nodes. This improvement generalises to a 25% reduction in the RMSE on the 60% of the nodes with the greatest covariance with the observed nodes and to a 15% reduction in the RMSE on all nodes of the model.

Table 18: A comparison of the performance of the physics-based models with that of the trained hybrid models after iteration 15 of the ML-DA algorithm. The comparison is made in terms of average RMSE, which has units $^{\circ}C$. The table columns are labelled 'unbiased' and 'biased', though it should be noted that the 'unbiased' model is not truly unbiased, but rather lacks deliberately introduced bias.

	Unbiased		**Biased**	
RMSE calculated over	**Physics**	**Hybrid**	**Physics**	**Hybrid**
All nodes	27.3	28.7	50.5	42.3
Observed nodes	1.00	0.951	1.47	0.962
Top 5% covariance	9.89	11.9	15.4	11.7
Top 10% covariance	9.88	11.8	15.6	11.5
Top 20% covariance	9.86	11.8	15.6	11.4
Top 40% covariance	9.92	11.8	15.6	11.5
Top 60% covariance	10.1	11.9	15.4	11.7

The results presented in Figures 30 and 31, as well as those presented in Table 18, show that application of the hybrid modelling methodology presented in section 2 can result in an improvement of the model performance across the observed nodes of the model and possibly also across all nodes of the model for the simulated problem considered in this section. Unfortunately, the results also show that it is possible to obtain an improvement on the observed nodes without obtaining an improvement across all nodes of the model or even across the 5% of the nodes of the model that have the greatest covariance with the observed nodes. This can make it difficult to assess the performance of hybrid models trained with real measurements, where observations of the true state are only available at the observed nodes, and these observations are subject to measurement noise.

To combat this problem, this section also includes an attempt to assess the global performance of models using only the observed nodes. This is done by using the models to make predictions of the evolution of the system over the next 30 time steps using the analysis states estimated by the data assimilation algorithm as initial conditions. For each model, the analysis states that were estimated using that model were used. Each of the first 970 analysis states was used as an initial

condition, resulting in 970 different simulation runs that are 30 time steps long. To calculate the error of these predictions, consider that there are 970 predictions of the state after 30 time steps $\{\mathbf{u}_1^{30}, \mathbf{u}_2^{30}, \ldots, \mathbf{u}_{970}^{30}\}$. Consider also that there are 1000 true states $\{\mathbf{u}_1^t, \mathbf{u}_2^t, \ldots, \mathbf{u}_{1000}^t\}$ obtained from the simulation results that were used to generate the measurements used in this section. The state $\mathbf{u}_k^{30}$ is a prediction of $\mathbf{u}_{k+30}^t$. The error of the prediction can be calculated using equation 15 modified as follows:

$$\text{RMSE}_k = \sqrt{\frac{\sum(\mathbf{u}_{k+30}^t - \mathbf{u}_k^{30})^2}{O}}, \tag{18}$$

where O is the number of nodes in the model. This equation can be modified as needed if subsets of all nodes of the model are considered in the error calculation. Note that the states are referred to by the symbol $\mathbf{u}$ rather than $\mathbf{z}$ to distinguish them from the states used in data assimilation.

The concept behind this second system of evaluating the performance of hybrid models is based on the fact that the full system state (or its lower dimensional representation, which is used in this case) is required as the initialisation for making predictions about the evolution of the system. The accurate prediction of the evolution of the system over multiple time steps also requires that the accuracy of the knowledge of the system state does not degrade over successive time steps. This means that the ability to accurately predict only the observed nodes is insufficient for making accurate predictions over multiple time steps. Better accuracy of the predicted evolution over 30 time steps therefore implies better knowledge of the entire system state, both in the initialisation and in the prediction itself.

The RMSE that was calculated using equation 18 was averaged over all 970 time steps and 30 model training runs to quantify the accuracy of the models at each algorithm iteration. Figure 32 shows the progression of the accuracy of the 30 time step prediction with the progression of the ML-DA algorithm if the unbiased physics-based model is used. Figure 33 shows the same results for when the physics-based model with deliberately introduced bias is used. Figure 32 shows a trend of increasing RMSE as the algorithm progresses for all node subsets across which the error was calculated. This is consistent with the trend across all node subsets except the observed nodes in Figure 30. Similarly, Figure 33 shows a decreasing trend as the algorithm progresses for all node subsets, which is consistent with the trend in Figure 31 for all node subsets.

Importantly, the fact that Figures 32 and 33 show the same trend for all node subsets suggests that the error evaluated across the observed nodes is now a more reliable indicator of the error

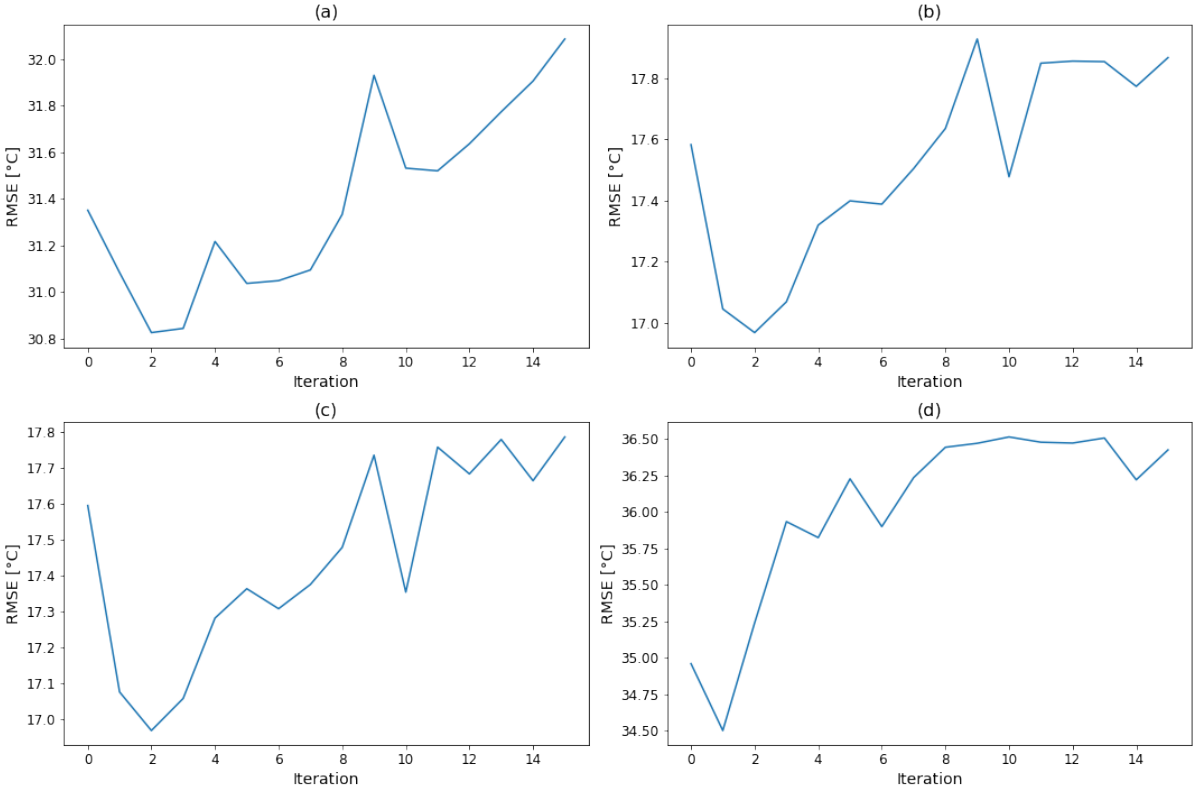

Figure 32: The average RMSE calculated across (a) all nodes, (b) the 60% of the nodes with the greatest covariance with the observed nodes, (c) the 5% of the nodes with the greatest covariance with the observed nodes and (d) the observed nodes at different iterations of the ML-DA algorithm. Iteration 0 corresponds to the physics-based model only, while the results for each subsequent iteration correspond to the hybrid model after that iteration. The results are for the physics-based model without deliberately introduced bias and the accuracy is of the predicted evolution of the system over the next $30\,s$ using the analysis states estimated by the data assimilation algorithm as initial conditions.

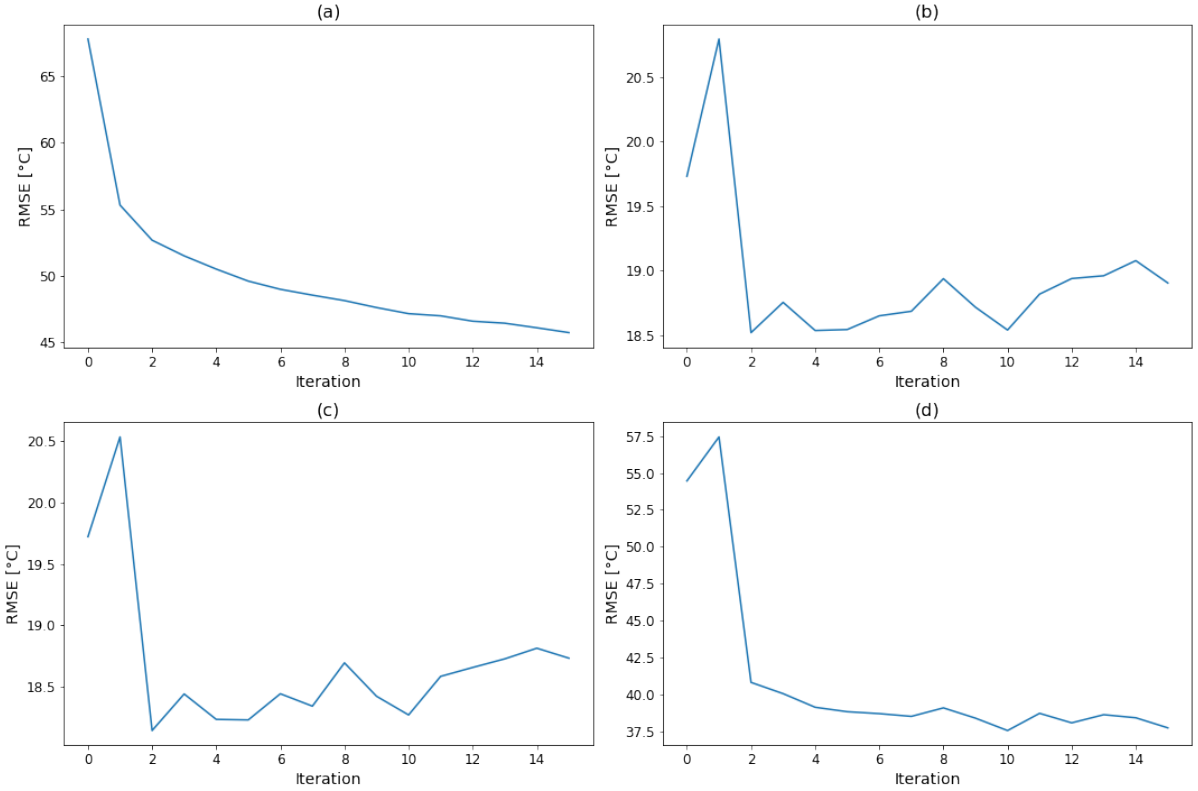

Figure 33: The average RMSE calculated across (a) all nodes, (b) the 60% of the nodes with the greatest covariance with the observed nodes, (c) the 5% of the nodes with the greatest covariance with the observed nodes and (d) the observed nodes at different iterations of the ML-DA algorithm. Iteration 0 corresponds to the physics-based model only, while the results for each subsequent iteration correspond to the hybrid model after that iteration. The results are for the physics-based model with deliberately introduced bias and the accuracy is of the predicted evolution of the system over the next $30\,s$ using the analysis states estimated by the data assimilation algorithm as initial conditions.

evaluated across larger subsets of nodes compared to when the error is evaluated using analysis states. It should be noted, however, that there is some importance attached to the number of time steps over which the evolution is calculated. Evaluation using the predicted evolution over 10 time steps was also tried as an alternative to 30 time steps. When using 10 time steps, the accuracy across the observed nodes improved as the algorithm progressed when the unbiased physics-based model was used, as was the case with analysis states. This undermines the assumption that prediction over multiple time steps requires better estimates of the entire system state, because Figure 30 shows that they are in fact worse. A possible interpretation is that the attributes of the analysis states that caused better predictions of the observed nodes persist in the short term but disappear as predictions are made over longer time windows. This would mean that the evaluation of hybrid model performance using simulations over multiple time steps is still supportable, but that it has possible flaws that must be kept in mind.

4.7 Data assimilation with real measurements

This section investigates the application of data assimilation techniques with real measurement data. Only the MPF data assimilation algorithm is used in this section, given that it performed better than the SIR filter on the simulated data in section 4.5. The same initialisation of the MPF as in sections 4.5 and 4.6 was used. Data was available from six of the seven sensor locations shown in Figure 25 because sensor 7 was lost. Two distinct $1100\,s$ long time series of measurements were used in the data assimilation algorithm, and these are shown in Figure 34. These are labelled time series 1 and time series 2 for easy reference. The sampling frequency is $1\,Hz$.

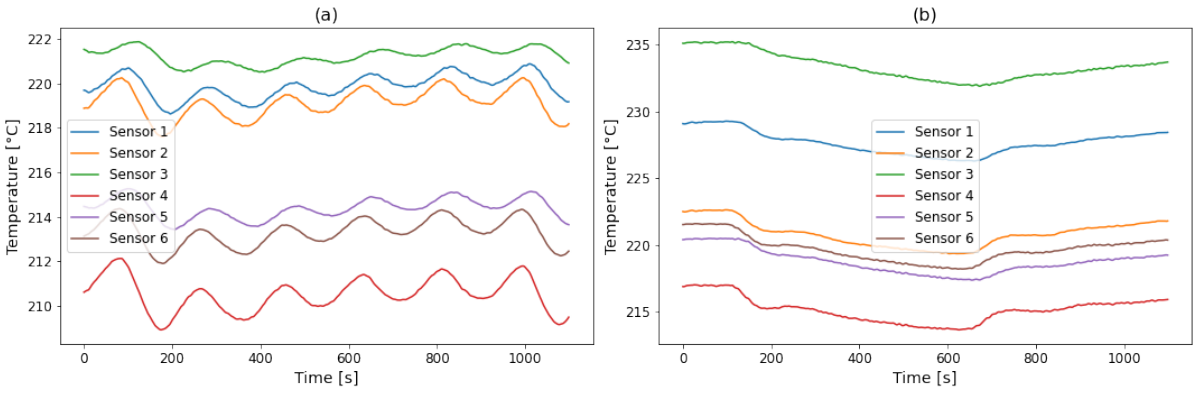

Figure 34: The measurements of the six available sensors over a time of $1100\,s$ for (a) time series 1 and (b) time series 2.

Table 19: The optimal combination of γ and r parameters for time series 1 and time series 2.

Time series	γ	r
1	27.83	5.282
2	27.83	2.714

Data assimilation runs were conducted for the same 10×10 grid of γ and r values that were used in section 4.5, and like in section 4.5, 10 data assimilation runs were conducted for each combination of γ and r. The RMSE was also calculated in the same way as in section 4.5, and is plotted for the different combinations of γ and r in Figure 35 for both measurement time series. The RMSE was calculated over the observed nodes, given that the truth is available for these nodes only. In Figure 35, the results appear to be qualitatively similar to each other for the two time series, as well as qualitatively similar to the results for the simulated measurements in Figure 27 (a). Table 19 gives the optimal combination of γ and r for both time series.

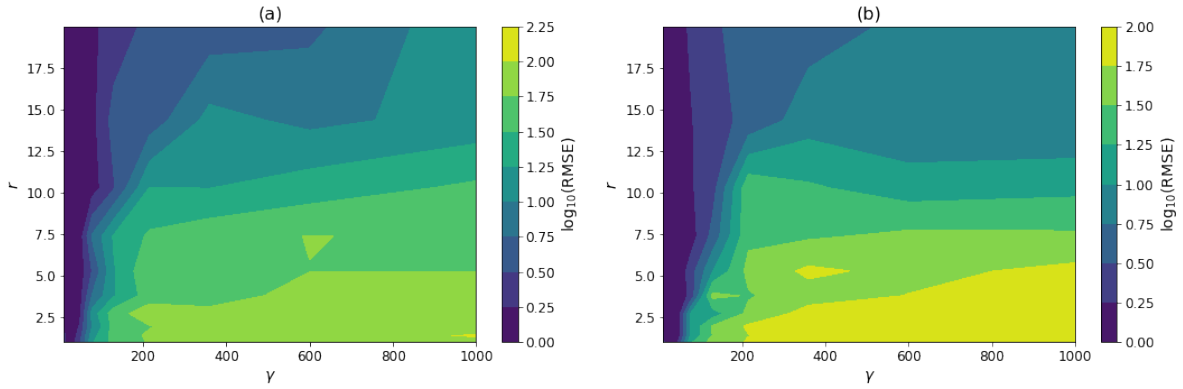

Figure 35: Contour plots of the RMSE calculated across the observed nodes for different values of γ and r for (a) time series 1 and (b) time series 2.

4.8 Hybrid models with real measurements

This section explores the use of real measurements for the training of hybrid models. The availability of sensors was the same as in section 4.7. The $50000\ s$ long time series of observations shown in Figure 36 was used for this section. Training was performed using the first $40000\ s$ of the time series, and the final $10000\ s$ of the time series were kept aside for testing purposes. This means that, unlike in section 4.6, data additional to the data used for training are available for evaluating the performance of the hybrid models. Like in section 4.7, the sampling frequency is $1\ Hz$.

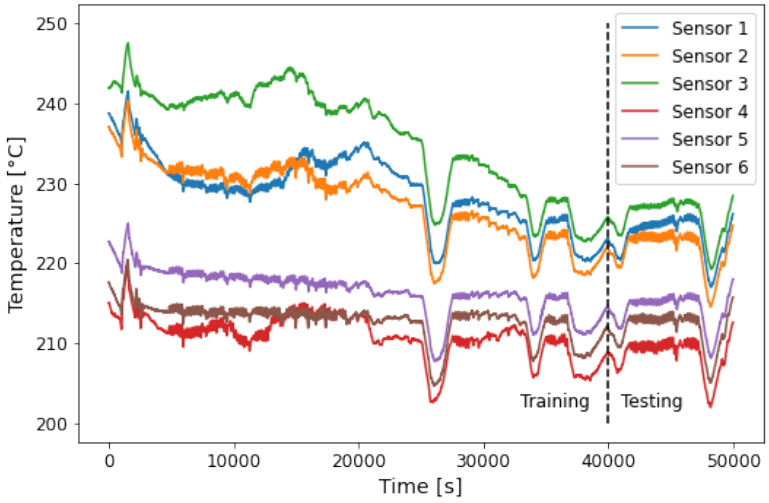

Figure 36: The measurements made by the six available sensors that were used for the training and evaluation of hybrid models. The boundary between the portion of the measurements that were used for training and those that were used for testing is indicated on the figure.

The physics-based model that was used in this section is the neural network flow rate model with $\tau = 0.1$. Hybrid models were trained using the ML-DA algorithm, which ran for 5 iterations. The MPF was used for data assimilation. The parameters that were used for data assimilation were chosen based on the results in Table 19. The system noise parameter $\gamma = 27.83$ was used because it was best for both time series in section 4.7. Judging qualitatively from Figures 34 and 36 the actual measurement noise parameter is likely to be closer to $r = 2.714$ than to $r = 5.282$, and as a result the former was used. Like in section 4.6, the first 50 analysis states were disregarded in the training of hybrid models to account for burn-in of the data assimilation algorithm. A neural network was trained to perform resolvent correction, and the architecture of this neural network was chosen using equations 16 and 17 with $\tau = 0.1$. The parameter $\tau = 0.1$ was used rather than $\tau = 1$ to save computational resources.

A total of 16 training runs were conducted. For each training run, the performance of the physics-based model and the performance of the hybrid model after each iteration of the ML-DA algorithm were evaluated in two ways. The first evaluation was done using the analysis states that are estimated as part of the ML-DA algorithm. These analysis states were used together with a modification of equation 15 to calculate a RMSE across the observed nodes. The second evaluation was done by using the analysis states that are estimated as part of the ML-DA algorithm to predict the evolution of the system over 30 time steps, and then using a modification of equation 18 to calculate the RMSE across the observed nodes.

The RMSE results were averaged over the model runs and over the number of time steps to obtain Figure 37. It shows the progression of the performance of the hybrid model as the ML-DA algorithm progresses. For both RMSE metrics, the performance of the hybrid models improves as the ML-DA algorithm progresses. Most of the improvement occurs by the time iteration 2 has completed, which is similar to the results obtained with simulated measurements in section 4.6.

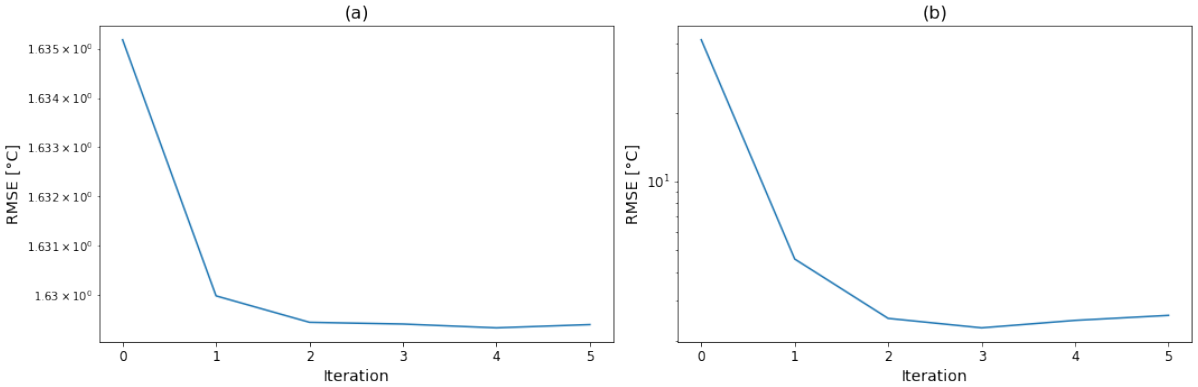

Figure 37: The average RMSE calculated across the observed nodes using (a) the analysis states and (b) predictions of the system evolution over 30 time steps. Iteration 0 corresponds to the physics-based model. Models were trained using all available sensors. Results were calculated using the training observations.

A potential confounding factor in the training of hybrid models is that sensors 4 and 5 in Figure 25 are both placed in a region that in the simulation model always has the same fireside boundary condition parameter. This means that the simulation model always predicts similar temperatures at these two sensors. This would not be a problem if the full simulation model had been used as the physics-based model in the training process. In that case, the hybrid model could learn to predict different temperatures at these sensors. However, a surrogate model is used as the physics-based model. Dimensionality reduction by POD is performed as part of the surrogate model, and the hybrid model also operates in the space of reduced dimensionality. The POD dimensionality reduction does not allow the possibility of there being different temperatures at the two sensors. To show this, 32768 samples were drawn from the distribution that is used to initialise the data assimilation algorithms in this section. The samples were then reconstructed to full dimensionality, and the temperatures at sensor locations 4 and 5 were recorded. The temperatures at these nodes differed by less than $6 \times 10^{-12}\,^{\circ}C$ for all samples.

Because the physics-based model always predicts essentially the same temperature at sensors 4 and 5, problems in training can arise when the actual sensor measurements at the two locations differ. To investigate the effect of this confounding factor, training was re-done using

all available sensors except sensor 5. A total of 16 training runs were again conducted and the performance of the resulting models was evaluated as before. Figure 38 shows the results, and appears to be qualitatively similar to Figure 37. Numerical results are provided in Table 20 to better understand the difference in the results.

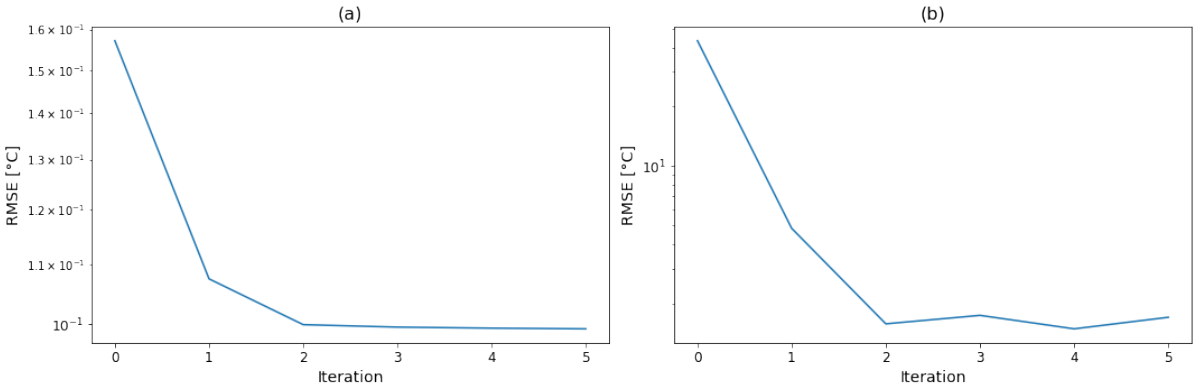

Figure 38: The average RMSE calculated across the observed nodes using (a) the analysis states and (b) predictions of the system evolution over 30 time steps. Iteration 0 corresponds to the physics-based model. Models were trained using all available sensors except sensor 5. Results were calculated using the training observations.

The results in Table 20 for the training observations show that the performance of both the physics-based and hybrid models calculated using the analysis states is an order of magnitude better when sensor 5 is left out. Additionally, the hybrid model improves by less than 1% over the physics-based model when using all available sensors, but by more than 35% when excluding sensor 5. When calculating the RMSE using 30 time step predictions, the hybrid models improve over the physics-based models by a factor 16 when using all sensors and by a factor 25 when excluding sensor 5. The results therefore show that, when models are evaluated using the training observations, the efficiency of hybrid model training increases when sensor 5 is excluded.

When simulated observations were used to train hybrid models in section 4.6, the performance of the models was only evaluated using the same observations that were used to train the models. This means that the generalisation properties of the hybrid models were not investigated. Here, separate testing observations are considered to investigate the generalisation performance of the trained hybrid models. These testing observations directly follow the training observations as indicated in Figure 36. To evaluate the performance of the hybrid models on the testing observations, a data assimilation run was conducted using the testing observations and the physics-based model, as well as the hybrid models after each iteration of all of the 16 training runs that were conducted. The two RMSE metrics were calculated in the same manner as

on the training observations.

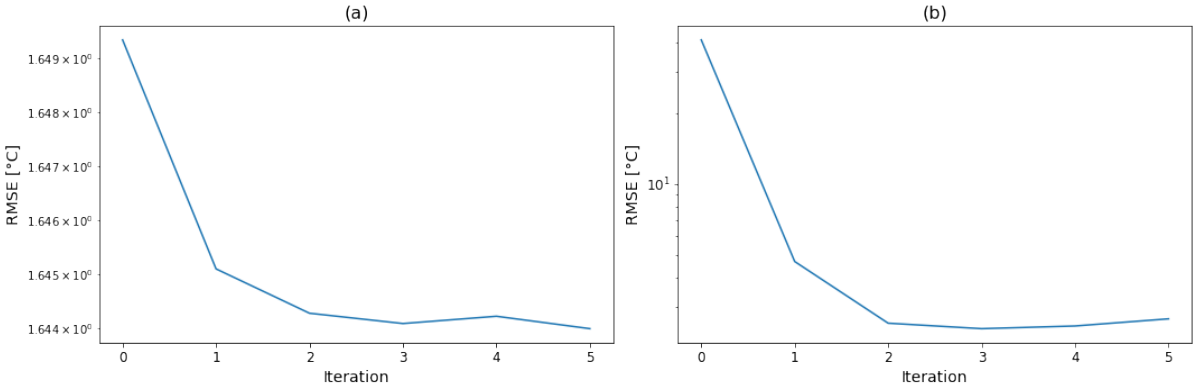

Figure 39: The average RMSE calculated across the observed nodes using (a) the analysis states and (b) predictions of the system evolution over 30 time steps. Iteration 0 corresponds to the physics-based model. Models were trained using all available sensors . Results were calculated using the testing observations.

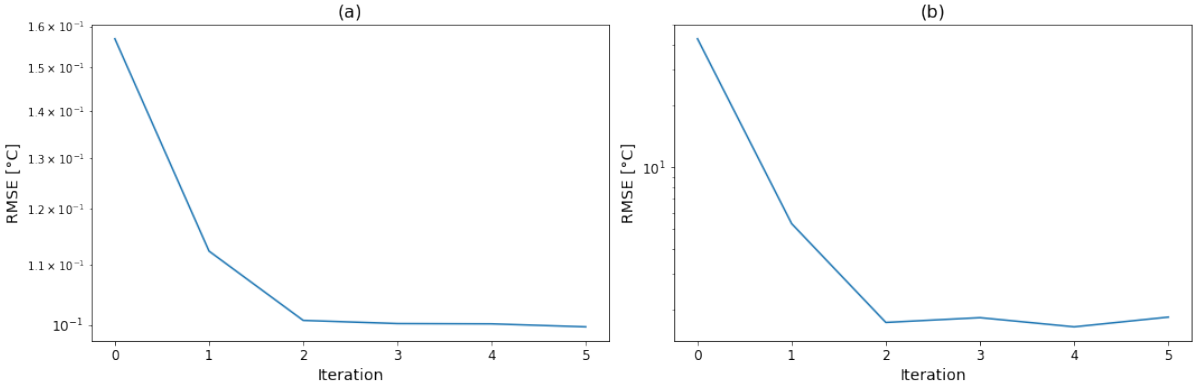

Figure 40: The average RMSE calculated across the observed nodes using (a) the analysis states and (b) predictions of the system evolution over 30 time steps. Iteration 0 corresponds to the physics-based model. Models were trained using all available sensors except sensor 5. Results were calculated using the testing observations.

For the case in which all sensors are used, Figure 39 shows the progression of the two RMSE metrics calculated using the testing observations as the ML-DA algorithm progresses. Similar results are shown in Figure 40 for the case in which sensor 5 is excluded. The results for the two cases appear qualitatively similar to the results that were obtained using the training observations and that were illustrated in Figures 37 and 38. Table 20 gives the RMSE results on the testing observations for the physics-based model and for the trained hybrid model after iteration 5 of the ML-DA algorithm. When all sensors are used, the hybrid model represents less

than a 1% improvement over the physics based model when the RMSE is calculated using analysis states, as was the case on the training observations. A factor 15 improvement is obtained when the RMSE is calculated using 30 time step predictions. When sensor 5 is excluded, an improvement in the RMSE calculated using the analysis states of 35% is achieved, and a factor 23 improvement in the RMSE calculated using the 30 time step prediction is obtained. These results are similar to the results obtained using the training observations, though the results on the testing observations are slightly worse, as expected. This indicates good generalisation performance of the trained hybrid models.

Table 20: A comparison of the performance of the physics-based models with that of the trained hybrid models after iteration 5 of the ML-DA algorithm. The comparison is made in terms of the RMSE (units $^{\circ}C$) calculated using analysis states and in terms of the RMSE calculated using predictions of system evolution over 30 time steps. The RMSE was calculated using the measurements that were used for training and separately using the measurements set aside for testing.

	All		**All, except number 5**	
RMSE type	**Physics**	**Hybrid**	**Physics**	**Hybrid**
Analysis states, training	1.64	1.63	0.157	0.0993
30-step prediction, training	41.7	2.58	43.2	1.70
Analysis states, testing	1.65	1.64	0.157	0.0997
30-step prediction, testing	41.2	2.67	42.6	1.85

4.9 Discussion and conclusion

As was found in case study 1, the surrogate models that were constructed for this case study were indeed capable of emulating the results of the simulation model. On average, the Gaussian process surrogate model with the automatic relevance determination kernel function performed best on this task, while of the neural network surrogate models the neural network flow rate model with $\tau = 0.1$ performed best. The computational requirements of the surrogate modelling approaches were evaluated using two tasks, one being the sequential prediction of the evolution of model states and the other being the parallel prediction of the evolution of model states. On the first task, the SVR and RVM models performed best, while on the second task the neural network models performed best. The second task is more representative of the application of the surrogate models in data assimilation algorithms, and therefore has more weight for this work.

The performance of the MPF and the SIR filter data assimilation algorithms was studied with

simulated measurements to investigate whether the observed nodes of the model can be used to make a good evaluation of the performance of the data assimilation algorithms on larger subsets of the nodes of the model. To do this, different combinations of the system and measurement noise parameters of the algorithms were used. Across 100 different combinations of data assimilation parameters, the best parameter combination for the MPF achieved better performance than the best parameter combination for the SIR filter for all combinations of state variables that were used to evaluate the performance of the data assimilation algorithms. For the MPF, the RMSE calculated across all nodes of the model correlated with the RMSE calculated across the observed nodes with a correlation coefficient of 0.56. The RMSE calculated across the 60% of the nodes with the greatest covariance with the observed nodes correlated with the RMSE across the observed nodes with a correlation coefficient of 0.88. This indicates that when the performance of the data assimilation approaches is evaluated on the observed nodes, this is representative of the performance across large portions of the domain of the system. However, the parameter combination that achieves optimal performance on the observed nodes will not necessarily achieve optimal performance on other node subsets.

Hybrid models were trained using the same simulated measurements that were used to evaluate the performance of data assimilation algorithms. Two cases were considered: the physics-based model was used with and without deliberately introduced bias. In both cases, the trained hybrid models represented an improvement over the physics based models when the RMSE is calculated using analysis states and across the observed nodes. However, when the bias-free physics-based model is used the hybrid model does not improve on the physics based model when the performance is calculated across larger subsets of the nodes of the model. This shows, as in section 3.5, that a hybrid model can improve over a physics-based model on the observed nodes but not on the entire model.

This last observation means that the evaluation of model performance on the observed nodes is inadequate for comparing the performance of different models over their entire domain. In an attempt to combat this, a method for calculating the RMSE of the predictions that models make of system evolution over 30 time steps was proposed. This method relies on the assumption that an improvement in the accuracy of these 30 time step predictions requires an improvement in the prediction of the entire system state. When using this second RMSE metric, the trend in the performance of hybrid models during training was the same for the observed nodes and for larger node subsets. This supports the reliability of the method, but it must still be considered possible to achieve an improvement in performance on the observed nodes while the performance over the entire model domain degrades.

The hybrid models that were trained using simulated data were not tested for their generalisation properties. They were evaluated using the same simulated data that were used to train them. It is therefore possible that the improvements made by hybrid models relative to physics-based models apply only to the training data and do not generalise.

The MPF data assimilation algorithm was applied with real measurements collected using sensors installed on the process converter. Since the observations are the only knowledge of the system that is available, the performance of the MPF was evaluated using these observations. Two different measurement time series were used, and 100 different combinations of system and measurement noise parameters were evaluated. For both measurement time series a common system noise parameter was optimal, while the best measurement noise parameter differed. It must be kept in mind when evaluating the performance of these data assimilation algorithms and later the performance of hybrid models using real observations, that the observations are not actually noise free. When simulated observations were used, access to the noise-free observations was possible, but this is not possible with real observations.

As a final investigation, hybrid models were trained using real observations. Two cases were again considered. The first case uses all available sensors, and the second case excludes sensor 5. The reason for the exclusion of sensor 5 is that the model is not flexible enough to distinguish it from sensor 4, leading to contradictory information during training when the two sensor readings are different. In both cases, the hybrid model improves on the physics-based model when performance is evaluated using the analysis states and when it is evaluated using 30 time step predictions. The improvement in performance is greater when sensor 5 is omitted. The improvement that is made in both cases according to the 30 time step prediction metric is greater than an order of magnitude. This improvement, together with its magnitude, indicates that there is an improvement in performance not only on the observed nodes but also on larger regions of the model domain. It is, however, also possible that 30 time steps is insufficient time for global performance to contribute significantly to the performance on this metric.

The fact that the performance improvement of the hybrid model over the physics-based model is greater when sensor 5 is excluded points to a limitation of the approach. The POD dimensionality reduction used by the surrogate and hybrid models limits the flexibility of the models so that it is not possible for either model to predict different temperatures for sensors 4 and 5. This problem would not exist if the original simulation model had been used as the physics-based model or the original dimensionality had been used for the hybrid model. Either of these alternatives would likely lead to an increase in the computational cost, however. Allowing the fireside boundary conditions to vary with finer resolution could also increase flexibility. The number

of sensors and the number of regions with independent boundary conditions do appear to be well matched, however. Allowing a more flexible variation in the fireside boundary conditions would likely require more sensors to exploit this flexibility.

When the performance of the hybrid models trained with real measurements was evaluated on a set of testing observations, the results were similar to when their performance was evaluated using the set of training observations. This indicates good generalisation performance of the trained hybrid models. However, it should be noted that the testing observations directly follow the training observations. It is unclear from the investigations performed in this section whether the performance of the trained hybrid models would generalise to observations that are separated in time from the training observations.

5 Conclusion

In this work, the training of hybrid models of a process converter freeboard was investigated. The investigations that were performed can be split into two case studies, and were designed to evaluate the hybrid model training methodology presented in section 2. In the first case study, a small simulation model was used to explore different aspects of the methodology. The investigations performed in case study 1 were done using only simulated data. Surrogate models of the simulation model were constructed, and it was found that these can fulfil their purpose of reducing the computational cost associated with the simulation model, while emulating the results of the simulation model.

The surrogate models that were constructed for case study 1 were then used to investigate the performance of hybrid models trained using various algorithms. It was found that hybrid models can improve over physics based models in terms of performance evaluated using the RMSE of the analysis states estimated using a data assimilation run. While the per-step DA-O algorithm performed best in terms of the RMSE, the ML-DA algorithm was used in the subsequent studies conducted in this work. The ML-DA algorithm has advantages over the per-step DA-O algorithm in terms of computational cost and in terms of ease of application to different numbers of observed nodes. The ML-DA algorithm is also potentially less sensitive to measurement noise than the per-step DA-O algorithm.

Using the ML-DA algorithm, the impact of varying the number of observed nodes was studied. It was found that while it is advantageous in terms of model performance to use a larger number of observed nodes, hybrid models still represent an improvement in terms of RMSE over physics-based models when as few as two nodes of the model used for case study 1 are observed. This improvement is observed across both the observed nodes and across all nodes of the model. This indicates that the construction of hybrid models is feasible even for sparsely observed systems.

In the second case study, the methodology presented in section 2 was applied to a larger model for which real measurements were available. In this case study, some of the findings from the first case study were used. The surrogate modelling approach was again able to emulate the results of the simulation model. Using the surrogate models that were constructed, the performance of data assimilation algorithms was studied using simulated data. Specifically, it was investigated whether the performance of data assimilation approaches can meaningfully be evaluated using only knowledge of the observed nodes. It was found that there is a strong

correlation between the performance of the data assimilation approaches on the observed nodes and their performance on large regions of the model domain. As a result, data assimilation approaches with real measurements were evaluated on the observed nodes for the purpose of parameter selection.

Simulated observations were also used to investigate the training of hybrid models. Only the ML-DA algorithm was used for hybrid model training in the second case study. The performance of these hybrid models was evaluated using the RMSE calculated using analysis states estimated during data assimilation. It was found that the hybrid models represented an improvement over the physics-based model across all nodes only when bias was deliberately introduced in the physics-based model. When the RMSE was calculated using analysis states, and no bias was deliberately introduced in the physics-based model, the hybrid models were an improvement on the observed nodes but not on larger subsets of the nodes of the model. This shows that it is insufficient to evaluate the performance of hybrid models using the observed nodes and analysis states. A second method of calculating an RMSE was therefore proposed. It involves predictions of the evolution of the system over 30 time steps, and the comparison of these predictions to the actual states of the system after 30 time steps. When this second RMSE metric is used, an increasing trend in the RMSE on the observed nodes is accompanied by an increasing trend in the RMSE on all nodes, and likewise a decreasing trend in the RMSE on the observed nodes is accompanied by a decreasing trend in the RMSE on all nodes. The second RMSE metric is therefore potentially better at evaluating the performance of the system across all nodes using only information from the observed nodes.

The final investigation that was performed involved the use of real measurements to train hybrid models. It was found that hybrid models can improve over physics-based models in terms of the RMSE calculated across the observed nodes. This is the case when the RMSE is calculated using analysis states estimated by data assimilation, and when the RMSE is calculated using 30 time step predictions. The performance of the hybrid models was evaluated using the training observations and also using testing observations. The results were similar in both cases, indicating that the performance of the hybrid models can generalise to new situations.

The generalisation of the hybrid models was evaluated using a set of testing observations that directly follow the observations that were used for training. While this is sufficient to show that the performance of the hybrid models that were trained can generalise to new situations, it does not provide a full picture of the generalisation properties of the hybrid models. Additional investigations are needed to reveal whether the hybrid models generalise to observations that are separated from the training observations in time, which may not be the case if the system is

not stationary.

The investigations conducted in this work show that hybrid models can be an improvement over physics-based models when the performance of models is evaluated using the observed nodes. They cannot show directly that hybrid models are an improvement on larger subsets of nodes. The observation that hybrid models are an improvement when the RMSE is calculated using 30 time step predictions provides some evidence that the hybrid models are an improvement for more than just the observed nodes. This relies on the assumption that improved predictions over multiple time steps require better prediction of the entire system state. It could be the case, however, that 30 time steps is too short for global improvements to be a requirement for making better predictions for the observed nodes. When all available sensors are used to train hybrid models, a factor 15 improvement in the performance on 30 time step predictions is obtained despite an improvement of less than 1% on the observed nodes when the evaluation is done using analysis states. This indicates that improvements are made on more than just the observed nodes. In summary, therefore, the improvements in performance on 30 time step predictions point toward the hybrid model being an improvement on more nodes than just the observed nodes. Alternative explanations cannot be ruled out, however. Nevertheless, the results definitely do show that the hybrid models allow better prediction of the evolution of the system over 30 time steps. Predictions like these may themselves be useful in some applications.

The performance of the hybrid model relative to the physics-based model is improved when only five of the six available sensors are used for training. When the real sensor measurements at two specific locations of the six sensor locations are different, this appears to be contradictory from the point of view of the model. The reason is that the use of POD dimensionality reduction does not allow either the surrogate model or the hybrid model to predict different temperatures at these two sensors. Both sensors are located in a region of the model with common fireside boundary condition parameters, and because there are little long-range effects in the model, the two sensor locations always have the same temperature in the snapshot matrix. Adding more regions with independent fireside boundary condition parameters could allow both sensors to be used effectively. In this work, the regions of independent fireside boundary condition parameters were chosen independently of the sensor locations. For future work, it could be beneficial to choose these regions with the sensor locations in mind. Interestingly, the lack of long-range effects in the model that was mentioned in this paragraph is evidence against the improvements of the hybrid models being an improvement on more than just the observed nodes.

In addition to the lack of flexibility caused by the POD dimensionality reduction, there are other

possible problems with the use of surrogate models as physics-based models. One of these is that the improvements of the hybrid models relative to the physics-based model are therefore relative to a surrogate model of a higher-fidelity physics-based model. It is possible that the hybrid models constructed in this way are not an improvement over the higher-fidelity physics-based model. The use of this higher-fidelity physics-based model in the construction of the hybrid model would require much more computational resources, however. Additionally, even if the hybrid model is only an improvement relative to the surrogate model, this would still mean that an improved model with low computational cost is available which can itself be beneficial.

Another possible problem with the use of surrogate models as physics based models is that it could undermine one of the goals of hybrid models, namely to combine physics-based models with data-driven models to benefit from the advantages of both while minimising the disadvantages of both. Because data-driven surrogate models are used in this work, the hybrid models in this work are essentially a combination of two data-driven models. Both still have the disadvantages of data-driven models, such as poor generalisation to novel situations. Nevertheless, one can interpret the hybrid models presented in this work as using information from physics-based models in their construction, as opposed to training a data-driven model from scratch. The possible problems with using surrogate models to construct hybrid models suggest that investigations into the implications of such use of surrogate models are necessary.

While the majority of this section has been dedicated to a discussion of the results obtained from the computational investigations that were performed in this work, it is also worth considering what contributions have been made to the solution of the overall problem discussed in section 1.4 and illustrated in Figure 3. In this work, the training of hybrid models was investigated in the context of a complex engineering problem in the form of a process converter freeboard. The freeboard is transient in nature and continuous sensor measurements and a large FE model of it are available. The methodology that was used in this work was developed to address two problems that present themselves when training hybrid models of a complex system such as the process converter freeboard. The first problem is that FE models are computationally expensive, which was addressed by constructing surrogate models of the FE models. The second problem is that the freeboard is observed at far fewer locations than there are nodes in the FE model. This second problem was addressed through the use of sequential data assimilation.

The use of surrogate models is beneficial not only because it makes the training of hybrid models more computationally tractable, but also because it enables the use of the trained hybrid models in real time applications such as digital twins for monitoring and predicting structural integrity. In the process converter freeboard application considered in this work, the temperature

field predicted by the hybrid models must first be used to determine the structural response of the freeboard, after which damage models can be applied to predict structural integrity. The methodology used in this work is, however, sufficiently general for any application in which a model of the evolution of a system is available together with observations of the system. It can therefore be applied to systems where there is a different driving force behind structural response or where the evolution of the structural response is directly modelled.

The use of data assimilation in the training of hybrid models has been documented in the literature, though simplified problems were considered in these previous applications. Likewise, in the engineering literature, surrogate models have previously been used to construct hybrid models. However, in these applications the initial conditions are usually known and the hybrid model is only easily applicable on a limited time grid. The methodology used in this work can handle uncertain initial conditions and is suitable for real-time applications such as digital twins, where pre-determined time grids are impractical. This work applies the use of data assimilation in the training of a hybrid model to a large engineering problem, adding the use of surrogate models to reduce the computational cost of such training. This brings an approach to the engineering literature that can naturally handle uncertain initial conditions and new observations that become available continuously.

A final appraisal of this work is that the hybrid models that were considered represent progress toward achieving the ideals of a digital twin, while more work is needed to fully understand their potential and their limitations. In a broad sense, they make it possible to account for both known and unknown physics in the modelling process, which can enable better synchronisation between the physical and virtual entities of a digital twin. Data assimilation is the means by which information is transferred from the physical entity to the virtual entity, while surrogate models reduce the computational cost required to incorporate this information in the virtual entity during hybrid model training. The investigations that were conducted in this work show that the hybrid models can indeed lead to better synchronisation between physical and virtual entities local to the observations, though it is unclear under which conditions this improvement extends to the whole system. Additionally, while it is clear that the use of surrogate models reduces the computational cost of the hybrid models, it is unclear what additional effects it has.

References

[1] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary Perspectives on Complex Systems*. Springer, 2017, pp. 85–113.

[2] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the digital twin: A systematic literature review," *CIRP Journal of Manufacturing Science and Technology*, vol. 29, pp. 36–52, 2020.

[3] H. Brandtstaedter, C. Ludwig, L. Hübner, E. Tsouchnika, A. Jungiewicz, and U. Wever, "Digital twins for large electric drive trains," in *2018 Petroleum and Chemical Industry Conference Europe (PCIC Europe)*. IEEE, 2018, pp. 1–5.

[4] E. J. Tuegel, A. R. Ingraffea, T. G. Eason, and S. M. Spottswood, "Reengineering aircraft structural life prediction using a digital twin," *International Journal of Aerospace Engineering*, vol. 2011, 2011.

[5] A. Rasheed, O. San, and T. Kvamsdal, "Digital twin: Values, challenges and enablers from a modeling perspective," *IEEE Access*, vol. 8, pp. 21 980–22 012, 2020.

[6] S. Boschert and R. Rosen, "Digital twin—the simulation aspect," in *Mechatronic Futures*. Springer, 2016, pp. 59–74.

[7] E. N. Lorenz, "Designing chaotic models," *Journal of the Atmospheric Sciences*, vol. 62, no. 5, pp. 1574–1587, 2005.

[8] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. R. Hunt, M. Girvan, and E. Ott, "Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 4, p. 041101, 2018.

[9] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating scientific knowledge with machine learning for engineering and environmental systems," *ACM Computing Surveys (CSUR)*, 2021.

[10] C. Steyn and K. Brooks, "De-bottlenecking of the Anglo Platinum Converting Process utilising advanced process control," *IFAC-PapersOnLine*, vol. 50, no. 2, pp. 1–6, 2017.

[11] M. Bocquet, J. Brajard, A. Carrassi, and L. Bertino, "Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization," *Foundations of Data Science*, vol. 2, no. 1, p. 55, 2020.

[12] M. Bocquet, A. Farchi, and Q. Malartic, "Online learning of both state and dynamics using ensemble Kalman filters," *Foundations of Data Science*, vol. 3, no. 3, p. 305, 2021.

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

[14] M. Bocquet, J. Brajard, A. Carrassi, and L. Bertino, "Data assimilation as a learning tool to infer ordinary differential equation representations of dynamical models," *Nonlinear Processes in Geophysics*, vol. 26, no. 3, pp. 143–162, 2019.

[15] A. Farchi, P. Laloyaux, M. Bonavita, and M. Bocquet, "Using machine learning to correct model error in data assimilation and forecast applications," *Quarterly Journal of the Royal Meteorological Society*, vol. 147, no. 739, pp. 3067–3084, 2021.

[16] A. Farchi, M. Bocquet, P. Laloyaux, M. Bonavita, and Q. Malartic, "A comparison of combined data assimilation and machine learning methods for offline and online model error correction," *Journal of Computational Science*, vol. 55, p. 101468, 2021.

[17] S. Scher and G. Messori, "Generalization properties of feed-forward neural networks trained on Lorenz systems," *Nonlinear Processes in Geophysics*, vol. 26, no. 4, pp. 381–399, 2019.

[18] P. D. Dueben and P. Bauer, "Challenges and design choices for global weather and climate models based on machine learning," *Geoscientific Model Development*, vol. 11, no. 10, pp. 3999–4009, 2018.

[19] A. Wikner, J. Pathak, B. R. Hunt, I. Szunyogh, M. Girvan, and E. Ott, "Using data assimilation to train a hybrid forecast system that combines machine-learning and knowledge-based components," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 5, p. 053114, 2021.

[20] T. Arcomano, I. Szunyogh, J. Pathak, A. Wikner, B. R. Hunt, and E. Ott, "A machine learning-based global atmospheric forecast model," *Geophysical Research Letters*, vol. 47, no. 9, p. e2020GL087776, 2020.

[21] J. A. Weyn, D. R. Durran, and R. Caruana, "Can machines learn to predict weather? Using deep learning to predict gridded 500-hpa geopotential height from historical weather data," *Journal of Advances in Modeling Earth Systems*, vol. 11, no. 8, pp. 2680–2693, 2019.

[22] I. Ayed, E. de Bézenac, A. Pajot, J. Brajard, and P. Gallinari, "Learning dynamical systems from partial observations," *arXiv preprint arXiv:1902.11136*, 2019.

[23] U. Forssell and P. Lindskog, "Combining semi-physical and neural network modeling: An example of its usefulness," *IFAC Proceedings Volumes*, vol. 30, no. 11, pp. 767–770, 1997.

[24] M. C. Kennedy and A. O'Hagan, "Bayesian calibration of computer models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.

[25] A. Daw, A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided neural networks (PGNN): An application in lake temperature modeling," *arXiv preprint arXiv:1710.11431*, 2017.

[26] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, "Reservoir observers: Model-free inference of unmeasured variables in chaotic systems," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 4, p. 041102, 2017.

[27] G. A. Gottwald and S. Reich, "Combining machine learning and data assimilation to forecast dynamical systems from noisy partial observations," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 10, p. 101103, 2021.

[28] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[29] E. N. Lorenz and K. A. Emanuel, "Optimal sites for supplementary weather observations: Simulation with a small model," *Journal of the Atmospheric Sciences*, vol. 55, no. 3, pp. 399–414, 1998.

[30] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[31] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proceedings F (Radar and Signal Processing)*, vol. 140, no. 2. IET, 1993, pp. 107–113.

[32] P. J. Van Leeuwen, H. R. Künsch, L. Nerger, R. Potthast, and S. Reich, "Particle filters for high-dimensional geoscience applications: A review," *Quarterly Journal of the Royal Meteorological Society*, vol. 145, no. 723, pp. 2335–2365, 2019.

[33] P. J. Van Leeuwen, "Nonlinear data assimilation in geosciences: an extremely efficient particle filter," *Quarterly Journal of the Royal Meteorological Society*, vol. 136, no. 653, pp. 1991–1999, 2010.

[34] M. Ades and P. J. Van Leeuwen, "The equivalent-weights particle filter in a high-dimensional system," *Quarterly Journal of the Royal Meteorological Society*, vol. 141, no. 687, pp. 484–503, 2015.

[35] S. Reich, "A nonparametric ensemble transform method for Bayesian inference," *SIAM Journal on Scientific Computing*, vol. 35, no. 4, pp. A2013–A2024, 2013.

[36] S. Nakano, G. Ueno, and T. Higuchi, "Merging particle filter for sequential data assimilation," *Nonlinear Processes in Geophysics*, vol. 14, no. 4, pp. 395–408, 2007.

[37] S. J. Godsill, A. Doucet, and M. West, "Monte Carlo smoothing for nonlinear time series," *Journal of the American Statistical Association*, vol. 99, no. 465, pp. 156–168, 2004.

[38] D. Xiao, F. Fang, A. G. Buchan, C. C. Pain, I. M. Navon, and A. Muggeridge, "Non-intrusive reduced order modelling of the Navier–Stokes equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 293, pp. 522–541, 2015.

[39] R. Białecki, A. Kassab, and A. Fic, "Proper orthogonal decomposition and modal analysis for acceleration of transient FEM thermal analysis," *International Journal for Numerical Methods in Engineering*, vol. 62, no. 6, pp. 774–797, 2005.

[40] S. Walton, O. Hassan, and K. Morgan, "Reduced order modelling for unsteady fluid flow using proper orthogonal decomposition and radial basis functions," *Applied Mathematical Modelling*, vol. 37, no. 20-21, pp. 8930–8945, 2013.

[41] L. Sirovich, "Turbulence and the dynamics of coherent structures. I. Coherent structures," *Quarterly of Applied Mathematics*, vol. 45, no. 3, pp. 561–571, 1987.

[42] D. Xiao, F. Fang, C. Pain, and G. Hu, "Non-intrusive reduced-order modelling of the Navier–Stokes equations based on RBF interpolation," *International Journal for Numerical Methods in Fluids*, vol. 79, no. 11, pp. 580–595, 2015.

[43] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[44] J. L. Elman and D. Zipser, "Learning the hidden structure of speech," *The Journal of the Acoustical Society of America*, vol. 83, no. 4, pp. 1615–1626, 1988.

[45] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[46] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[47] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[48] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California University San Diego La Jolla Institute for Cognitive Science, Tech. Rep., 1985.

[50] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[51] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[52] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *IEEE International Conference on Neural Networks*. IEEE, 1993, pp. 586–591.

[53] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[55] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[56] D. Stathakis, "How many hidden layers and nodes?" *International Journal of Remote Sensing*, vol. 30, no. 8, pp. 2133–2147, 2009.

[57] F. Itano, M. A. d. A. de Sousa, and E. Del-Moral-Hernandez, "Extending MLP ANN hyper-parameters optimization by using genetic algorithm," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[58] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274–281, 2003.

[59] R. Pascanu, G. Montufar, and Y. Bengio, "On the number of response regions of deep feed forward networks with piece-wise linear activations," *arXiv preprint arXiv:1312.6098*, 2013.

[60] J. Du and Y. Xu, "Hierarchical deep neural network for multivariate regression," *Pattern Recognition*, vol. 63, pp. 149–157, 2017.

[61] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[62] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[63] C. E. Rasmussen, H. Nickisch, and C. K. I. Williams. (2018) Documentation for GPML Matlab code version 4.2. Accessed: 09/11/2021. [Online]. Available: http://www.gaussianprocess.org/gpml/code/matlab/doc/index.html

[64] J. A. Snyman and D. N. Wilke, *Practical mathematical optimization: Basic optimization theory and gradient-based algorithms*, 2nd ed. Cham, Switzerland: Springer, 2018.

[65] P.-H. Chen, R.-E. Fan, and C.-J. Lin, "A study on SMO-type decomposition methods for support vector machines," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 893–908, 2006.

[66] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[67] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *Journal of Machine Learning Research*, vol. 1, no. Jun, pp. 211–244, 2001.

[68] A. Faul and M. Tipping, "Analysis of sparse Bayesian learning," *Advances in Neural Information Processing Systems*, vol. 14, 2001.

[69] M. E. Tipping and A. C. Faul, "Fast marginal likelihood maximisation for sparse Bayesian models," in *International Workshop on Artificial Intelligence and Statistics*. PMLR, 2003, pp. 276–283.

[70] F. A. Viana, "A tutorial on Latin hypercube design of experiments," *Quality and Reliability Engineering International*, vol. 32, no. 5, pp. 1975–1985, 2016.

[71] R.-B. Chen, D.-N. Hsieh, Y. Hung, and W. Wang, "Optimizing Latin hypercube designs by particle swarm," *Statistics and Computing*, vol. 23, no. 5, pp. 663–676, 2013.

[72] Z. Yue, Y. Ding, H. Zhao, and Z. Wang, "Mechanics-guided optimization of an LSTM network for real-time modeling of temperature-induced deflection of a cable-stayed bridge," *Engineering Structures*, vol. 252, p. 113619, 2022.

[73] F. Jiang, Y. Ding, Y. Song, F. Geng, and Z. Wang, "Digital twin-driven framework for fatigue life prediction of steel bridges using a probabilistic multiscale model: Application to segmental orthotropic steel deck specimen," *Engineering Structures*, vol. 241, p. 112461, 2021.

[74] F. Gomez, Y. Narazaki, V. Hoskere, B. F. Spencer, and M. D. Smith, "Bayesian inference of dense structural response using vision-based measurements," *Engineering Structures*, vol. 256, p. 113970, 2022.

[75] Y. Huang, J. L. Beck, and H. Li, "Hierarchical sparse Bayesian learning for structural damage detection: Theory, computation and application," *Structural Safety*, vol. 64, pp. 37–53, 2017.

[76] S. Nikolopoulos, I. Kalogeris, and V. Papadopoulos, "Machine learning accelerated transient analysis of stochastic nonlinear structures," *Engineering Structures*, vol. 257, p. 114020, 2022.

[77] M. K. Ramancha, M. A. Vega, J. P. Conte, M. D. Todd, and Z. Hu, "Bayesian model updating with finite element vs surrogate models: Application to a miter gate structural system," *Engineering Structures*, vol. 272, p. 114901, 2022.

[78] C. Lin, T. Li, S. Chen, L. Yuan, P. van Gelder, and N. Yorke-Smith, "Long-term viscoelastic deformation monitoring of a concrete dam: A multi-output surrogate model approach for parameter identification," *Engineering Structures*, vol. 266, p. 114553, 2022.

[79] S. E. Azam and S. Mariani, "Online damage detection in structural systems via dynamic inverse analysis: A recursive Bayesian approach," *Engineering Structures*, vol. 159, pp. 28–45, 2018.

[80] Y. A. Yucesan, A. Von Zuben, F. Viana, and J. Mahfoud, "Estimating parameters and discrepancy of computer models with graphs and neural networks," in *AIAA Aviation 2020 Forum*, 2020, p. 3123.

[81] M. K. Ramancha, J. P. Conte, and M. D. Parno, "Accounting for model form uncertainty in Bayesian calibration of linear dynamic systems," *Mechanical Systems and Signal Processing*, vol. 171, p. 108871, 2022.

[82] D. Higdon, J. Gattiker, B. Williams, and M. Rightley, "Computer model calibration using high-dimensional output," *Journal of the American Statistical Association*, vol. 103, no. 482, pp. 570–583, 2008.

[83] D. Higdon, C. Nakhleh, J. Gattiker, and B. Williams, "A Bayesian calibration approach to the thermal problem," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 29-32, pp. 2431–2441, 2008.

[84] N. C. Kumar, A. K. Subramaniyan, L. Wang, and G. Wiggs, "Calibrating transient models with multiple responses using Bayesian inverse techniques," in *Turbo Expo: Power for Land, Sea, and Air*, vol. 55263. American Society of Mechanical Engineers, 2013.

[85] S. Pawar, S. E. Ahmed, O. San, and A. Rasheed, "Data-driven recovery of hidden physics in reduced order modeling of fluid flows," *Physics of Fluids*, vol. 32, no. 3, p. 036602, 2020.

[86] scikit-learn. (2022) sklearn.neural_network.mlpregressor. Accessed: 23/08/2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

[87] TensorFlow. (2022) TensorFlow v2.8.0. Accessed: 23/08/2022. [Online]. Available: https://www.tensorflow.org/versions/r2.8/api_docs/python/tf

[88] C.-C. Chang and C.-J. Lin. (2022) LIBSVM – A library for support vector machines. Accessed: 23/08/2022. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[89] The SciPy community. (2021) Statistics (scipy.stats). Accessed: 15/09/2022. [Online]. Available: https://docs.scipy.org/doc/scipy-1.7.1/reference/tutorial/stats.html

[90] R. Minnaar, “Thermal and structural analyses of heat exchanger,” Advanced Structural Mechanics, Tech. Rep., 2017.

[91] Ansys. (2021) Ansys Student - free software download. Accessed: 16/11/2021. [Online]. Available: https://www.ansys.com/academic/students/ansys-student

[92] Python Software Foundation. (2022) timeit — measure execution time of small code snippets. Accessed: 30/08/2022. [Online]. Available: https://docs.python.org/3/library/timeit.html

[93] NumPy Developers. (2021) NumPy 1.20.3 release notes. Accessed: 05/10/2022. [Online]. Available: https://numpy.org/devdocs/release/1.20.3-notes.html

A Additional data assimilation results for case study I

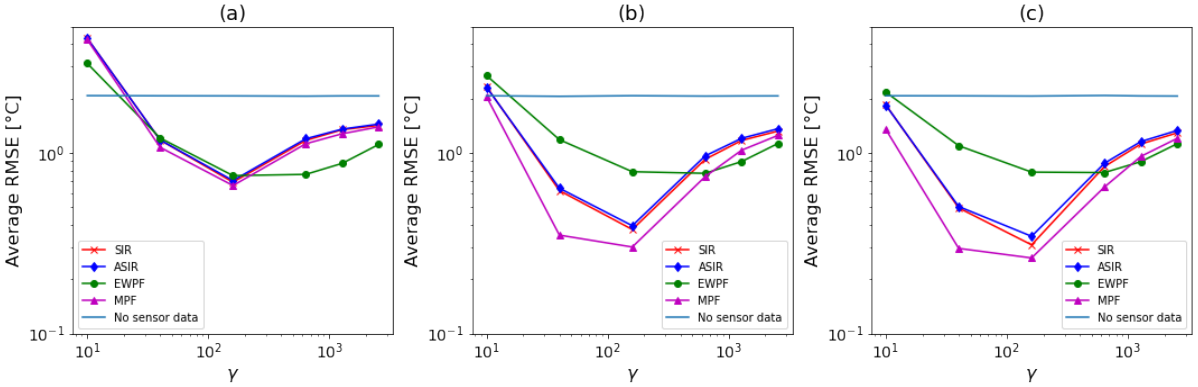

Figure 41: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 1 as a function of the system noise parameter γ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. The initialisation noise parameter was $\kappa = 10$ and the measurement noise parameter was $r = 0.5\ ^\circ C$.

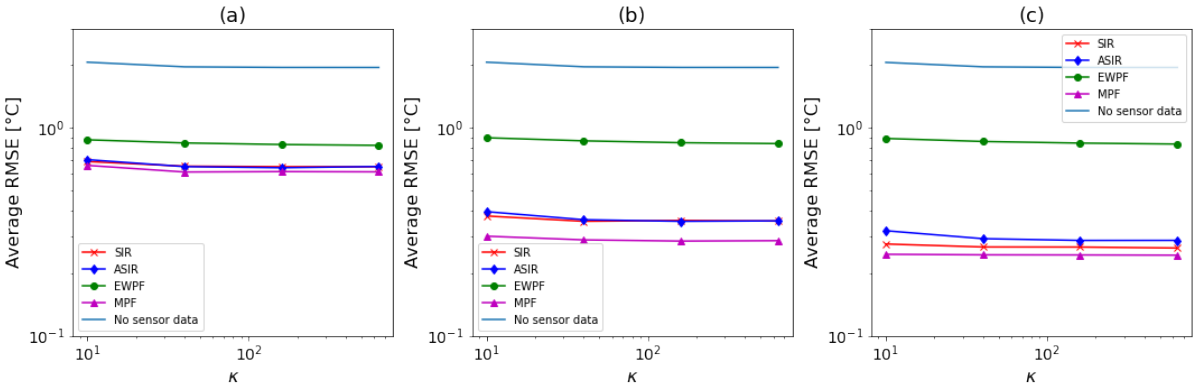

Figure 42: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 1 as a function of the initialisation noise parameter κ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The measurement noise parameter was set to $r = 0.5\ ^\circ C$.

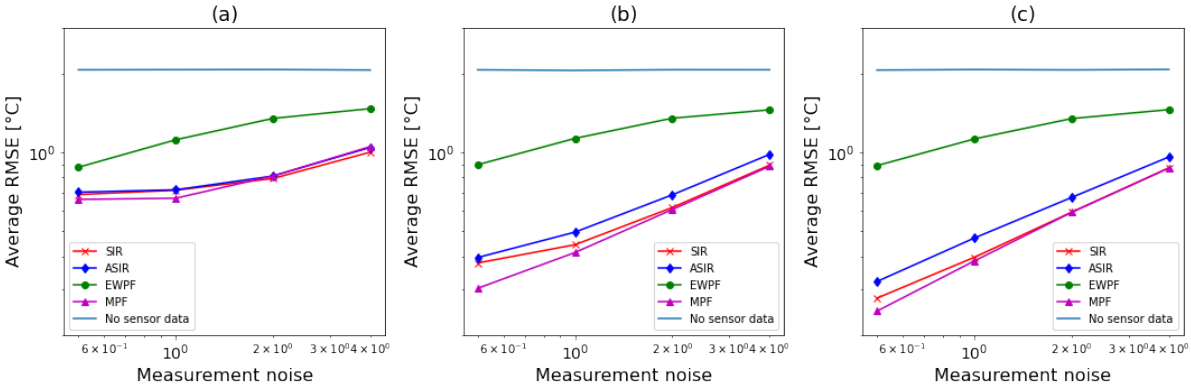

Figure 43: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 1 as a function of the initialisation noise parameter r. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$.

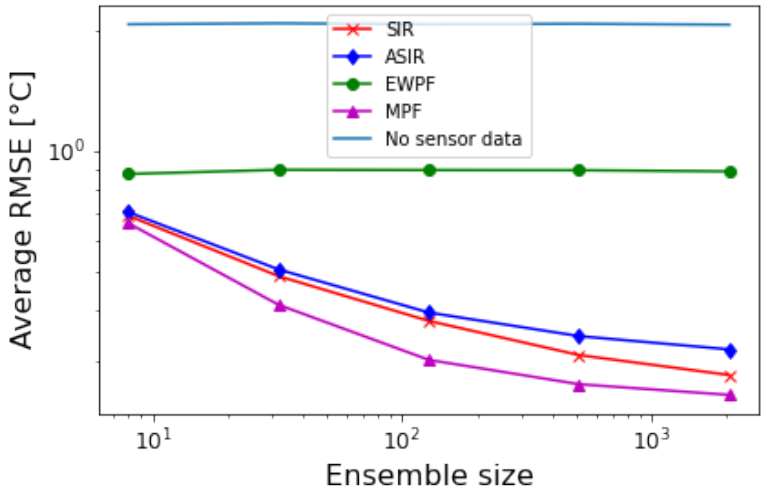

Figure 44: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 1 as a function of the ensemble size. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$, and the measurement noise parameter was $r = 0.5\ ^\circ C$.

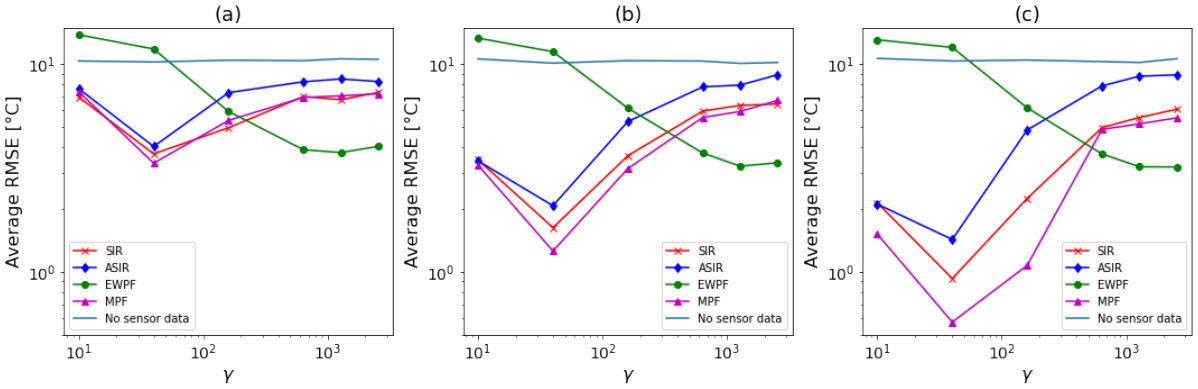

Figure 45: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 2 as a function of the system noise parameter γ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. The initialisation noise parameter was $\kappa = 10$ and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

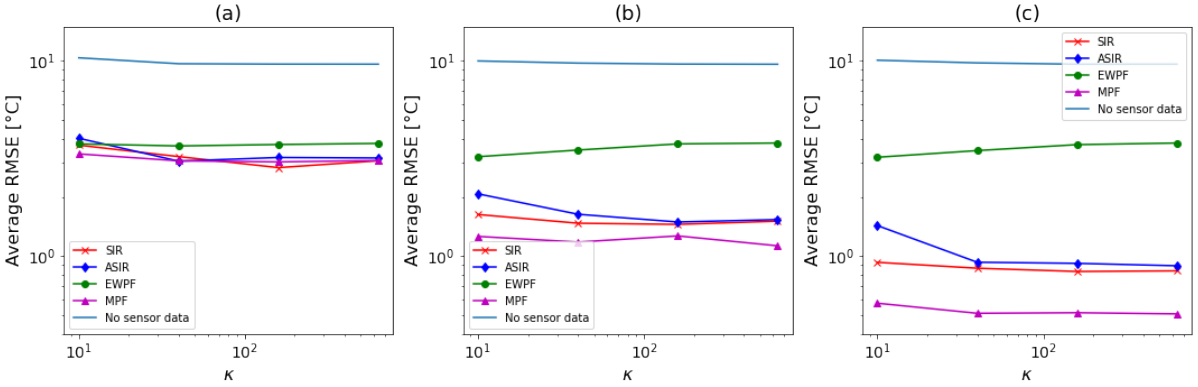

Figure 46: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 2 as a function of the initialisation noise parameter κ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The measurement noise parameter was set to $r = 0.5\ ^{\circ}C$.

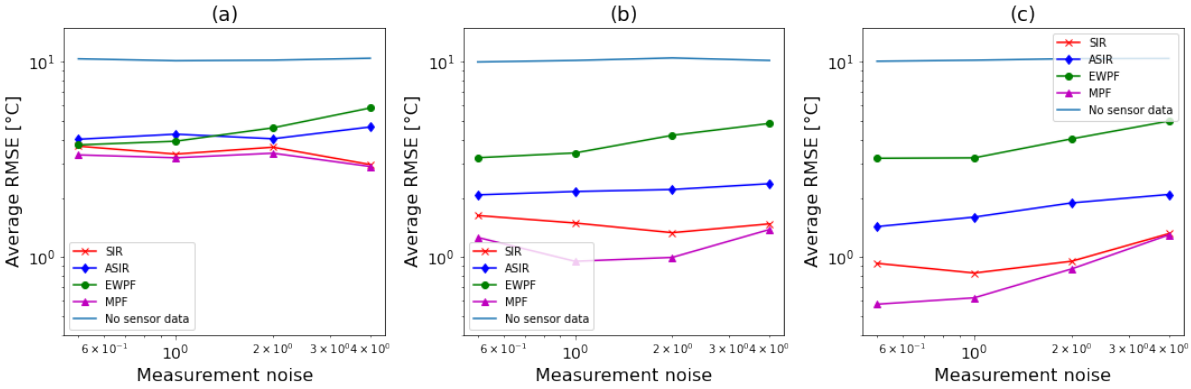

Figure 47: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 2 as a function of the initialisation noise parameter r. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$.

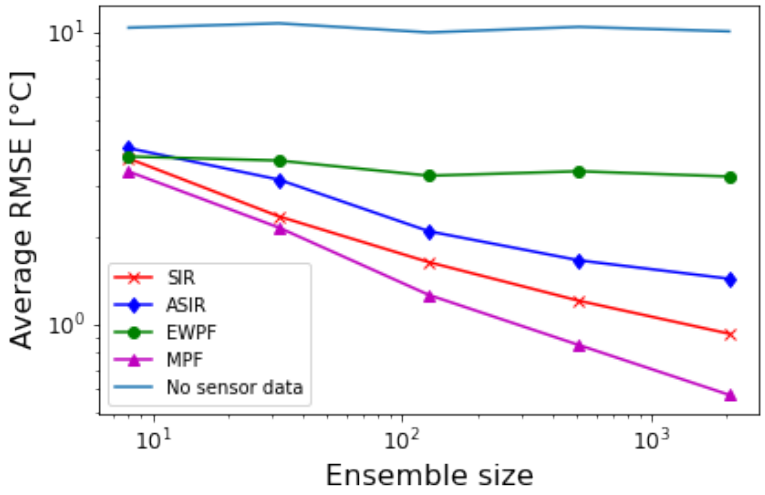

Figure 48: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 2 as a function of the ensemble size. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$, and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

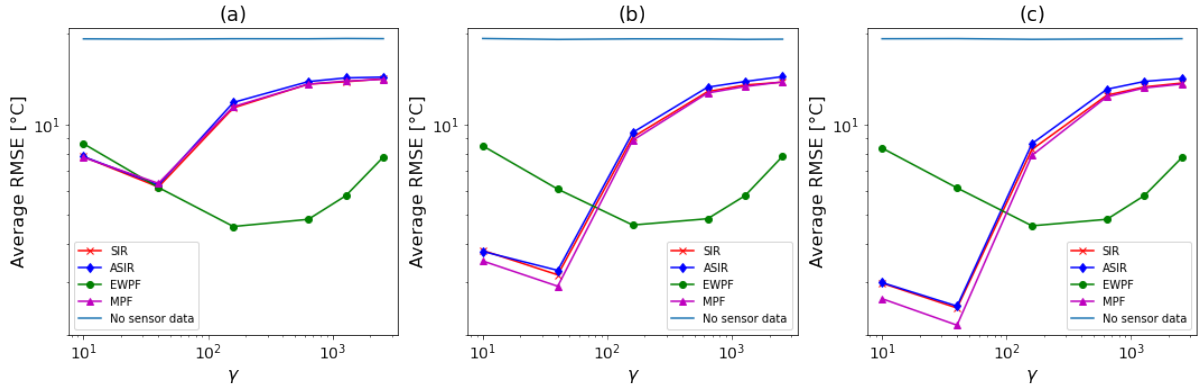

Figure 49: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 2 as a function of the system noise parameter γ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. The initialisation noise parameter was $\kappa = 10$ and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

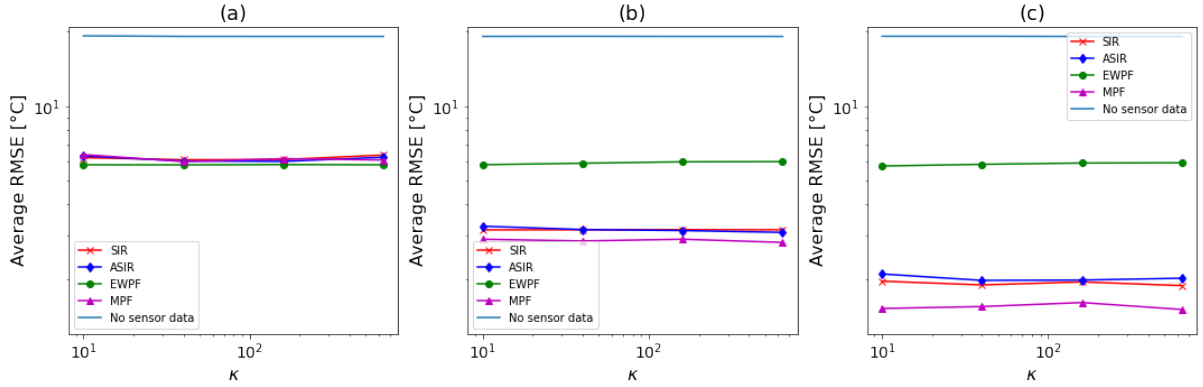

Figure 50: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 2 as a function of the initialisation noise parameter κ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The measurement noise parameter was set to $r = 0.5\ ^{\circ}C$.

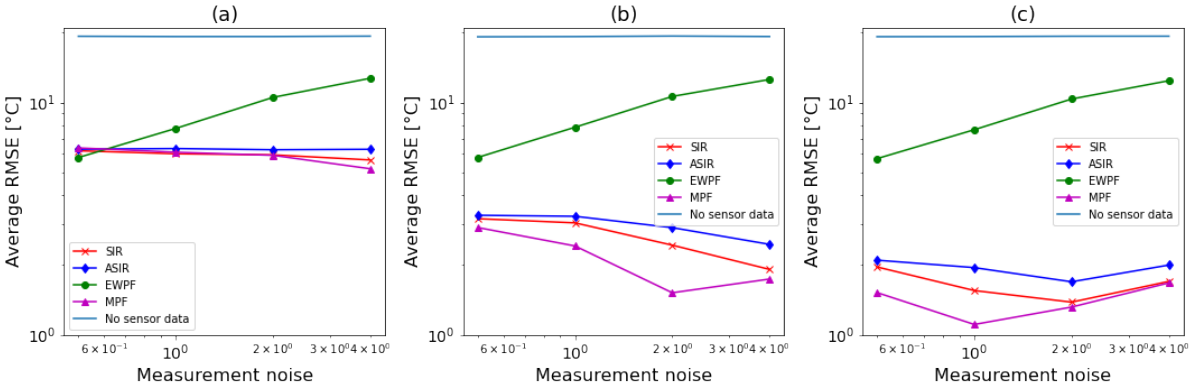

Figure 51: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 2 as a function of the initialisation noise parameter r. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$.

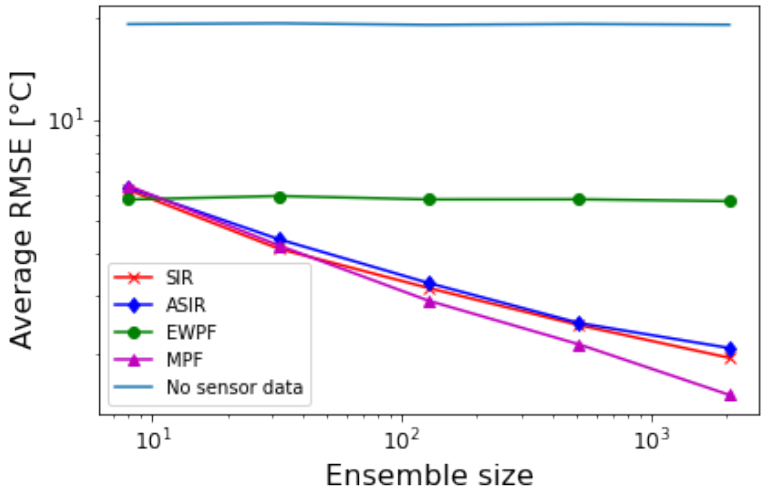

Figure 52: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 2 as a function of the ensemble size. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$, and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

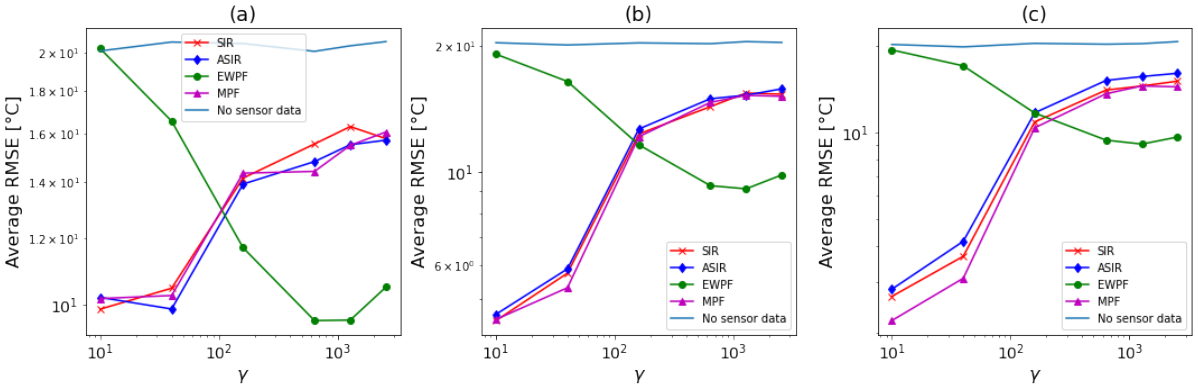

Figure 53: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 3 as a function of the system noise parameter γ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. The initialisation noise parameter was $\kappa = 10$ and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

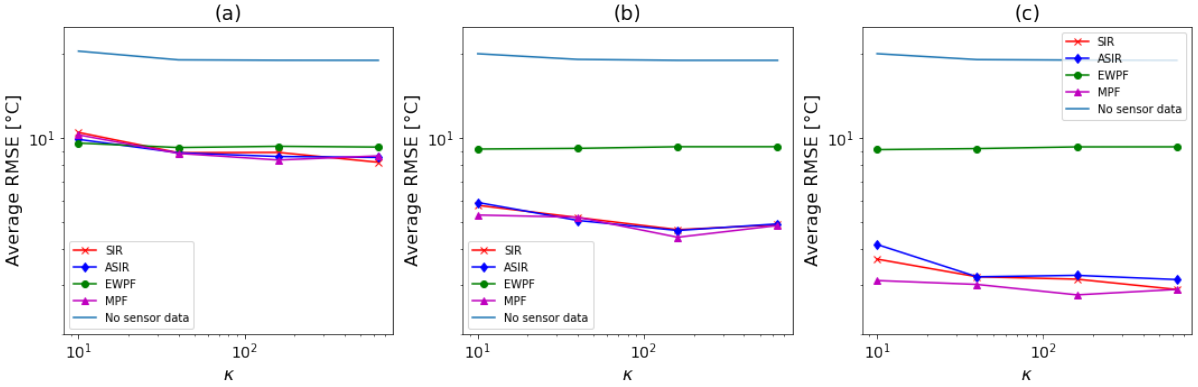

Figure 54: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 3 as a function of the initialisation noise parameter κ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The measurement noise parameter was set to $r = 0.5\ ^{\circ}C$.

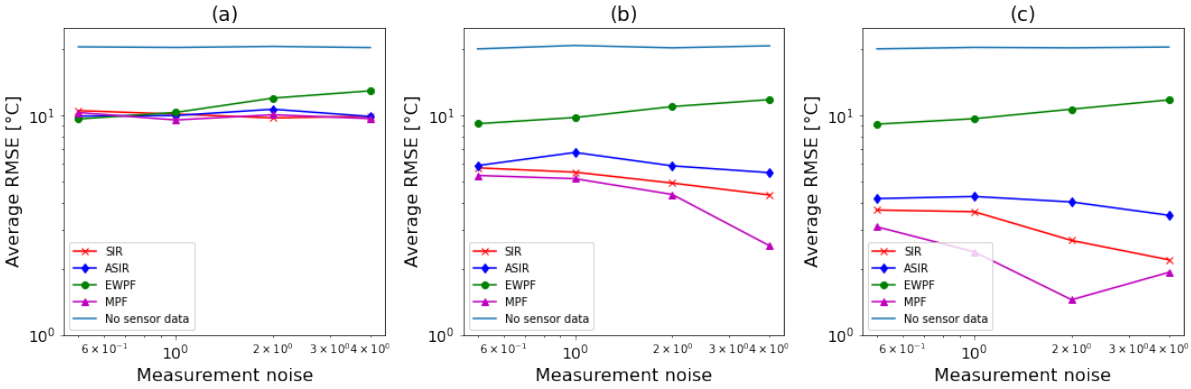

Figure 55: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 3 as a function of the initialisation noise parameter r. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$.

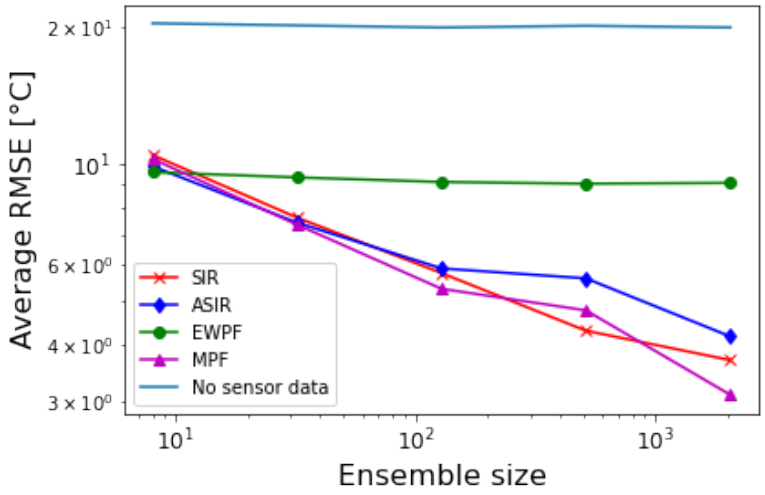

Figure 56: The average RMSE of the different data assimilation approaches over the first 10 time steps for situation 3 as a function of the ensemble size. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$, and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

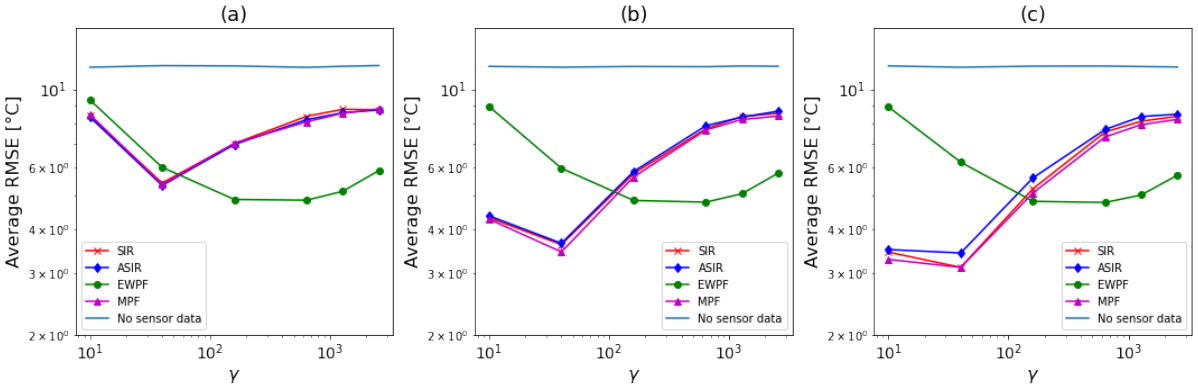

Figure 57: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 3 as a function of the system noise parameter γ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. The initialisation noise parameter was $\kappa = 10$ and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.

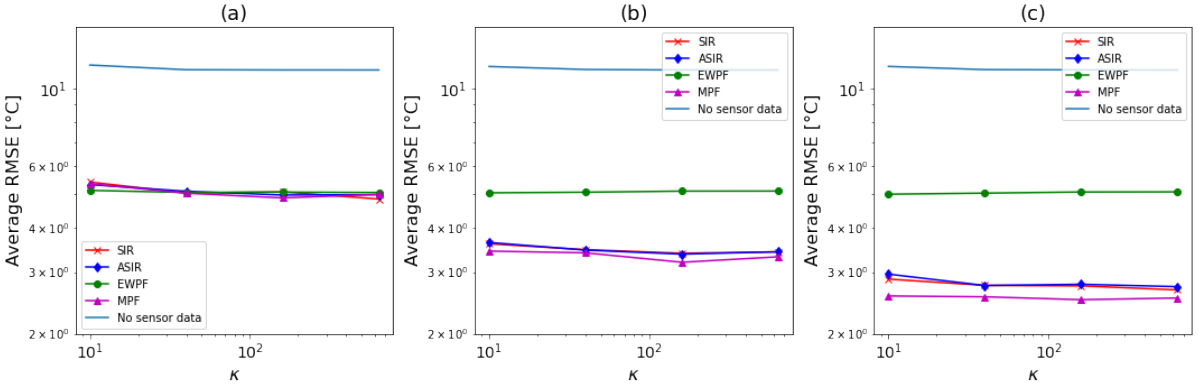

Figure 58: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 3 as a function of the initialisation noise parameter κ. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The measurement noise parameter was set to $r = 0.5\ ^{\circ}C$.

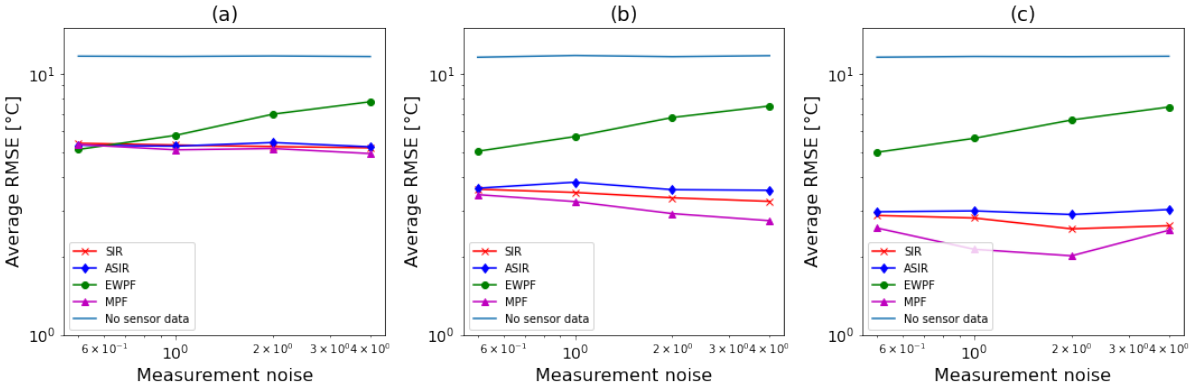

Figure 59: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 3 as a function of the initialisation noise parameter r. Results are shown for ensemble sizes of (a) 8, (b) 128 and (c) 2048. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 40$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$.

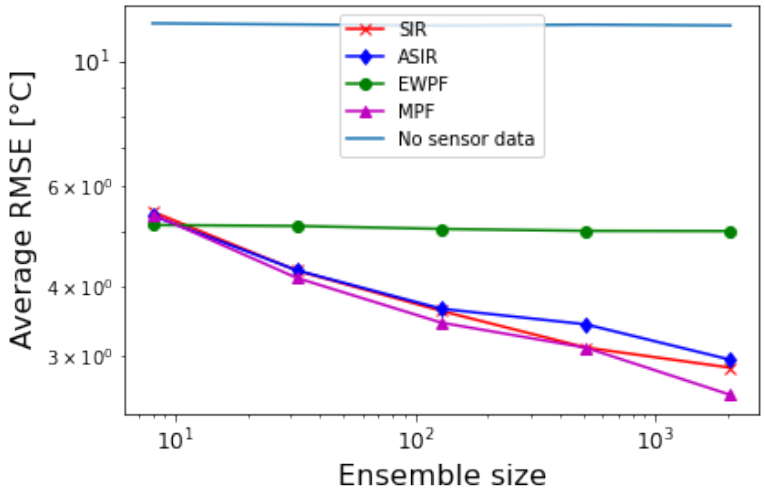

Figure 60: The average RMSE of the different data assimilation approaches over all 90 time steps for situation 3 as a function of the ensemble size. For the SIR, ASIR and MPF filters the system noise parameter was set to $\gamma = 160$ and for the EWPF it was set to $\gamma = 1280$. The initialisation noise parameter was $\kappa = 10$, and the measurement noise parameter was $r = 0.5\ ^{\circ}C$.