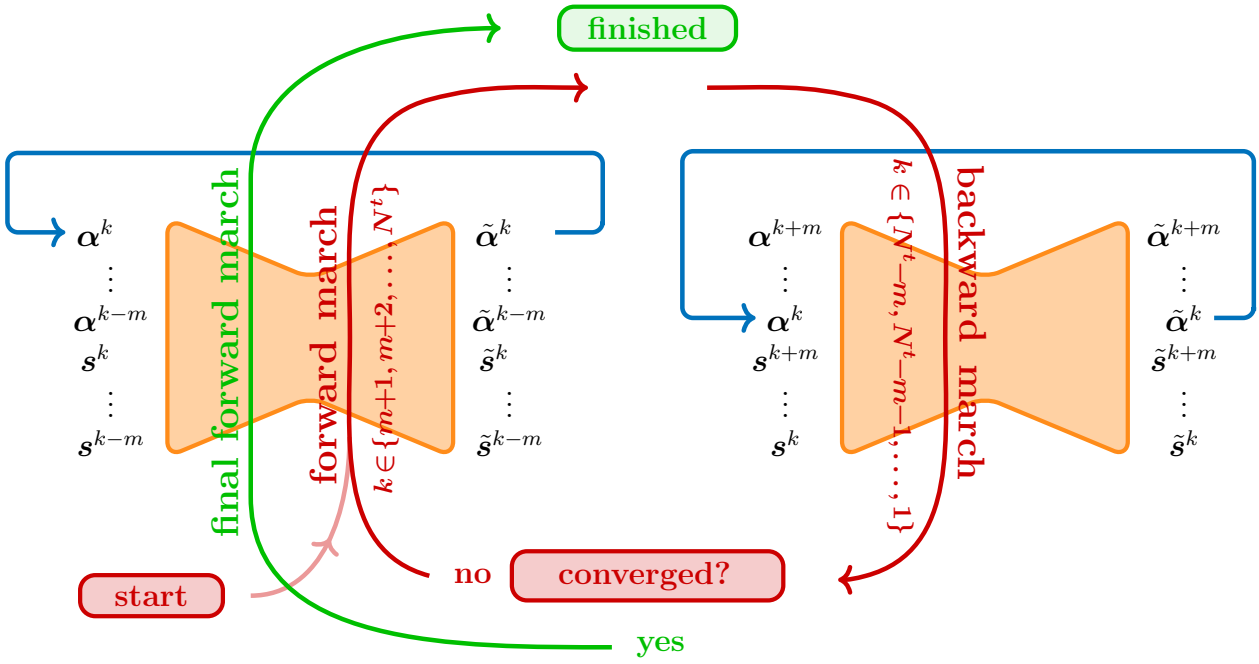
Graphical Abstract

**Data Assimilation with Machine Learning for Dynamical Systems: Modelling Indoor Ventilation**

Claire E. Heaney, Jieyi Tang, Jintao Yan, Donghu Guo, Jamesson Ipock, Sanjana Kaluvakollu, Yushen Lin, Danhui Shao, Boyang Chen, Laetitia Mottet, Prashant Kumar, Christopher C. Pain



# Highlights

**Data Assimilation with Machine Learning for Dynamical Systems: Modelling Indoor Ventilation**

Claire E. Heaney, Jieyi Tang, Jintao Yan, Donghu Guo, Jamesson Ipock, Sanjana Kaluvakollu, Yushen Lin, Danhui Shao, Boyang Chen, Laetitia Mottet, Prashant Kumar, Christopher C. Pain

- Presentation of a method which combines data assimilation (DA) with machine learning for efficiency.
- The approach is an alternative to 4D variational DA, and uses both solution variables and observations as the input to a neural network.
- A demonstration of the data assimilation method for a dual-twin experiment.
- The assimilation of observations measured in a school classroom in north London.

# Data Assimilation with Machine Learning for Dynamical Systems: Modelling Indoor Ventilation

Claire E. Heaney[a,b,*], Jieyi Tang[c,1], Jintao Yan[c,1], Donghu Guo[c,2], Jamesson Ipock[c,2], Sanjana Kaluvakollu[c,2], Yushen Lin[c,d,2], Danhui Shao[c,2], Boyang Chen[a], Laetitia Mottet[a,e], Prashant Kumar[f,g], Christopher C. Pain[a,b,h]

[a] *Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, South Kensington Campus, London, SW7 2AZ, UK*

[b] *Centre for AI-Physics Modelling, Imperial-X, Imperial College London, White City Campus, London, W12 7SL, UK*

[c] *Department of Earth Science and Engineering, Imperial College London, South Kensington Campus, London, SW7 2AZ, UK*

[d] *Department of Electrical and Electronic Engineering, The University of Manchester, Oxford Road, Manchester, M13 9PL, UK*

[e] *Inria, National Institute for Research in Computer Science and Control, Bordeaux, France*

[f] *Global Centre for Clean Air Research (GCARE), School of Sustainability, Civil and Environmental Engineering, University of Surrey, Guildford, GU2 7XH, Surrey, UK*

[g] *Institute for Sustainability, University of Surrey, Guildford, GU2 7XH, Surrey, UK*

[h] *Data Assimilation Laboratory, Data Science Institute, Imperial College London, South Kensington Campus, London, SW7 2AZ, UK*

**Abstract**

Data assimilation is a method of combining physical observations with prior knowledge (for instance, a computational simulation) in order to produce an improved model; that is, improved over what the physical observations or the computational simulation could offer in isolation. Recently, machine learning techniques have been deployed in order to address the significant computational burden that is associated with the procedures involved in data assimilation.

In this paper we propose an approach that uses a non-intrusive reduced-order model (NIROM) as a surrogate for a high-resolution model thereby saving computational effort. The mismatch between observations and the surrogate model is propagated forwards and backwards in time in a manner similar to 4D-variational data assimilation methods. The observations and prior are reconciled in a new way which takes full advantage of the neural network used in the NIROM and also means that there is no need to form the sensitivities explicitly when propagating the mismatch. Instead, the observations are part of the input and output of the network.

Modelling the air quality in a school classroom is the test case for our demonstration. Firstly, the data assimilation approach is shown to perform very well in a dual-twin type experiment, and secondly, the approach is used to assimilate observations collected from a classroom in Houndsfield Primary School with predictions from the NIROM.

[*] Corresponding author

*Email address:* c.heaney@imperial.ac.uk (Claire E. Heaney)

[1] Indicates an equal contribution

[2] Indicates an equal contribution

*Keywords:* machine learning, data assimilation, adversarial neural network, indoor fluid dynamics modelling

## 1. Introduction

### *1.1. Background*

Data assimilation (DA) aims to combine observations from sensors with predictions from a prior model (often from computer simulations) in order to obtain a more accurate prediction of the state of a system. In recent decades, DA has made a significant impact in earth science applications, for instance, its use has led to a noteable increase in the accuracy of medium range weather forecasts [1]. There are two types of data assimilation in common useage: ensemble methods and variational methods [2, 3], as well as many hybrid methods [4, 5]. Ensemble approaches use statistical methods to form the prior model, whereas variational methods minimise a functional which calculates the difference between the prior and the observations [3, 4, 6]. Variational DA can be split into 3D Variational (3D-Var) DA and 4D Variational (4D-Var) DA. The former disregards the time dependency of the observations, whereas the latter takes this into account and is regarded as state-of-the-art DA for operational numerical weather prediction [6–8].

One issue for practitioners of DA is its high computational cost. For ensemble methods, this arises from the need to perform many forward simulations. For variational methods, it arises from the need to march forwards and backwards in time, until the observations and the computer predictions are reconciled. Machine learning (ML) methods are a natural tool with which one can alleviate some of the computational burden of DA, and recently there has been considerable interest in combining ML with DA. In this paper, we wish to exploit ML techniques, through a reduced-order model framework, in order to assimilate data efficiently. We propose an approach which assimilates data through time, propagating the mismatch both forwards and backwards in time in a manner similar to 4D-Var methods. We exploit the architecture and design of the ML model to develop a DA algorithm which does not require differentiation of the model (although this is not generally an issue for ML models). Instead, the observations are part of the input and output of the network.

### *1.2. Related work*

Similarities between variational data assimilation and machine learning techniques have been elucidated by a number of authors [8–11]. One such example of this is that both techniques minimise a functional: in DA, the functional describes the mismatch between the observations and the computer prediction, and in ML, the functional that is minimised during training (for supervised or self-supervised learning paradigms) describes the difference between the learned output of a network and the desired output. Both DA and ML apply gradient descent methods to minimise these functionals. A second similarity shared by DA and ML is the concept of the adjoint model, used in DA to march backwards in time, and also used in ML where it is described as the backpropagation algorithm. For a general discussion of these points, see [8–11].

A number of recent articles demonstrate the promise of combining Machine Learning (ML) and DA. In some work, forward and adjoint models are replaced by fast-running ML-based surrogates, which can

capitalise on the ability to differentiate the adjoint using the backpropagation functionality commonly found within ML libraries [6, 7, 12, 13]. Other work attempts to represent the model error rather than the model itself with ML surrogates [14–18]. Considering ensemble DA approaches, several examples exist of using ML to facilitate DA [19–25]: both Proper Orthogonal Decomposition (POD) [19] and autoencoder networks [20, 21] have been used to identify a low dimensional space; Multilayer Perceptrons [22, 23] and Long Short-Term Memory networks have been used to make predictions in time [19, 20, 24]; and residual blocks have been used to predict measurements from geophysical logs [25]. Test cases include Lorenz systems [21, 24]; modelling the surface temperature of the sea [19]; modelling the movement of $CO_2$ in an indoor environment [20]; global weather predictions [22] and inversion of electromagnetic measurements from boreholes [25].

Turning to some examples of variational methods which use ML (the focus of this paper), Chennault et al. [13] used a surrogate model (based on a multi-layer perceptron) to replace the forward and adjoint models, and incorporated this in a 4D-Var DA method applied to a Lorenz problem. They found that using information from the adjoint resulted in an improved surrogate model, especially when the amount of training data was limited. Farchi et al. [17] constructed a 4D-Var method that attempts to learn the error in a Computational Fluid Dynamics (CFD) model from noisy and sparse observations applied to a two-layer 2D quasi-geostrophic model. The DA and ML models are used alternately, producing significantly better results than using the original model. Hatfield et al. [7] used a surrogate model based on a multi-layer perceptron to replace a discretised system modelling gravity wave drag. Applied to non-orographic gravity wave drag in an atmospheric model, no statistically significant differences were found between the ML-based results and the HFM results. For low dimensional problems such as some Lorenz problems, applying a dimensionality reduction method such as POD is not necessary [24] and was not done by Hatfield et al. [7], Chennault et al. [13], Farchi et al. [17]. For high-dimensional problems, incorporating POD into the surrogate model can save a significant amount of computational effort [26–28]. Gong et al. [29] implemented a 3D-Var DA method in latent space using a combination of POD and an autoencoder to find the low dimensional space, and a nearest neighbour method to model the parameter dependence. The original high-fidelity model (HFM) had 5000 grid points, represented by 30 latent variables, and over 24,000 combinations of material parameters were used. Maulik et al. [6] used POD and an LSTM to construct a surrogate model with which to predict the geopotential height of a 500 hPa pressure surface. With just over 10,000 spatial degrees of freedom represented by 5 POD modes, observations at 5000 spatial locations are assimilated. The use of surrogate models for forward and adjoint calculations results in a reduction in computational effort over classical methods by 4 orders of magnitude. Silva et al. [30] used a Generative Adversarial Network (GAN) to predict the spread of a virus through an idealised town with a compartmental model modified to include spatial variation. They were able to exploit the adjoint-like nature of the GAN to assimilate data, so that given the number of infections, they could estimate two $R_0$ numbers, one for each of two classes of person, for example. Convergence was achieved relatively quickly, as the method was able to resolve the mismatch and update the model parameters accurately within a few forwards and backwards iterations. Regazzoni et al. [31] tackle a multiscale problem with a slightly different approach to the previously mentioned work. They use DA to identify model parameters that led to the results at the faster scale. A machine learning method is used to model the behaviour over the slower timescale.

For many, a method of choice for learning time-dependent behaviour is an LSTM network. However, the predictions of these networks have sometimes been shown to be unphysical [32–34], either diverging or reaching a steady state after a certain time. Within our surrogate model, along with POD, we propose the use of an adversarial autoencoder (AAE) [35] to model the time-dependent behaviour, as it includes an adversarial layer which attempts to keep predictions close to those seen in the training data. Good results for prediction have been seen in [36, 37] using a similar network, although, in those papers, the network architecture was slightly modified so that the output had a different dimension to the input (i.e., was an encoder-decoder rather than an autoencoder).

### *1.3. Contribution*

A new method for data assimilation is proposed in which data is assimilated whilst marching forwards and backwards in time, in a manner similar to 4D-Var DA. To ease the computational burden, a surrogate model is used, which combines POD (for dimensionality reduction) and an AAE (for predicting the evolution in time of the system). Instead of calculating the gradient of the mismatch between the prior and the observations, the AAE takes as an input both the observations and the POD coefficients (which represent the evolution of the system) and is able to provide predictions for the POD coefficients which are consistent with the observations. Whilst performing time marching, a mismatch functional is evaluated, and, if this increases, relaxation is applied. Similar to Amendola et al. [20], Peyron et al. [21], Gong et al. [29], for example, we assimilate data in the low-dimensional space identified by a dimensionality reduction method. In those articles, an autoencoder is used to find the reduced space, whereas in this paper, POD is used.

The test case used in this paper investigates the air flows and air quality within a naturally ventilated classroom in Houndsfield Primary School, north London. During winter, a measurement campaign was undertaken, which collected observations of $CO_2$, relative humidity and temperature from 18 locations [38]. Measurements of pollution are also available from this dataset, but are not used in this study. An air purifier is present in the classroom, which has the dual purpose of improving air quality, in particular by removing pollution particles, and also reducing the number of particles that may contain the SARS-CoV-2 virus [38]. In this study we also compare $CO_2$ with a viral load tracer since the former is often used as a proxy for the virus concentration [39]. Subsequently, a CFD solver was used to obtain predictions of the indoor flows and air quality within the classroom. A ML-based surrogate model is constructed from solutions of the CFD model (snapshots) and two methods of predicting in time with the surrogate are investigated. A method for DA is then developed using the surrogate and a set of observations (generated from the CFD model). A dual-twin experiment is carried out to test how well the approach can assimilate data. Finally, observations collected from the classroom over the period of an hour are assimilated into the ML surrogate model.

The main contributions of this paper are (i) to present a method which combines data assimilation (4D-Var) with machine learning; (ii) to compare two methods for making predictions in time with the adversarial autoencoder; (iii) to demonstrate the data assimilation approach for a dual-twin experiment; (iv) to assimilate observations taken from a classroom in Houndsfield Primary School, north London. The novelty of this work is twofold. It comes from having the sensor values as direct inputs and outputs of the AAE, and also using the AAE for prediction and DA. As a consequence of this, no differentiation of

the surrogate adjoint model is needed, even though the proposed method has similarties with the 4D-Var method.

The remainder of this article is arranged as follows. The methodology is presented in Section 2 and results are shown in Section 3. Section 4 draws conclusions and discusses possible directions for future work.

## 2. Methodology

A non-intrusive reduced-order model is used to reduce the computational cost of the forward model, thereby making data assimilation tractable. The two main elements of this method are explained in Section 2.1 (finding a low-dimensional space which can represent well the CFD solutions) and Section 2.2 (building a model which can predict the behaviour of the system in the low-dimensional space). The proposed method for data assimilation is presented in Section 2.3.

### *2.1. Dimensionality Reduction*

Proper Orthogonal Decomposition (POD) was used to reduce the dimension of the high-fidelity results generated by the CFD code that solved the discretisation of the Navier-Stokes Equations. For the test case used in this paper, the seven fields of interest are velocity (one field for each of the three components), relative humidity, temperature, carbon dioxide levels and a field which represents the potential viral load in the air. The latter was based on carbon dioxide levels but included a half-life to take account of how long a virus might remain active in the air. We chose the value of 35 minutes for this, which is within the possible range of values for SARS-CoV-2 [40]. Each field was normalised or scaled so that the values at each node lie within the range [-1,1] in order to take account of the different units of the seven solution fields and the different physical process being represented by each field. Therefore, the snapshots matrix has dimension $7N$ by $M$, where $N$ is the number of nodes, the number of fields is 7, and $M$ is the number of snapshots used in building the surrogate model. Using Singular Value Decomposition, the snapshots matrix $\boldsymbol{\Phi}$ is decomposed in the following manner

$$\left[\boldsymbol{\phi}^1\ \boldsymbol{\phi}^2\ \cdots\ \boldsymbol{\phi}^M\ \right] =: \boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{1}$$

where $\boldsymbol{\phi}^k \in \mathbb{R}^{7N}$ is a column vector containing the seven normalised solution fields at time level $k$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices containing the left and right singular vectors respectively, and the singular values $\{\sigma_i\}_{i=1}^{M}$ are found on the diagonal of the matrix $\boldsymbol{\Sigma}$. An empirical measure of how much information is captured by using $N^{\mathrm{POD}}$ of the possible $M$ basis functions is given by

$$\frac{\sum_{i=1}^{N^{\mathrm{POD}}} \sigma_i^2}{\sum_{i=1}^{M} \sigma_i^2}\ . \tag{2}$$

Once $N^{\mathrm{POD}}$ has been chosen, the first $N^{\mathrm{POD}}$ columns of $\boldsymbol{U}$ are taken as the basis functions, which are stored in the matrix $\boldsymbol{R} \in \mathbb{R}^{7N \times N^{\mathrm{POD}}}$. Choosing the basis functions to be the left singular vectors minimises the least square error of the projection error of the snapshots. Of the total number of snapshots, denoted by $N^s$, $M$ snapshots are used to find the low-dimensional space and are also used to determine the mappings related to the normalisation of data. The remaining $N^s - M$ snapshots are used as unseen

or test data. All the snapshots are projected onto the low-dimensional space by the basis functions as follows

$$\boldsymbol{\alpha}^k = \boldsymbol{R}^T \boldsymbol{\phi}^k \qquad \forall k\,, \tag{3}$$

where $\boldsymbol{\alpha}^k$ is a vector containing the POD coefficients associated with time level $k$. Once the POD coefficients for the snapshots in the training dataset are obtained, each POD coefficient is scaled or normalised so that its values over time lie in the range [0,1]. This process is known as normalisation in machine learning terminology and is considered good practice when training neural networks. For more details on the theory of POD and the optimality of the basis functions see [41].

### *2.2. Prediction*

An adversarial autoencoder [35] is chosen to learn the behaviour of the system through time. An autoencoder is a neural network which attempts to learn the identity map through two networks, an encoder and a decoder. In addition to this, the adversarial autoencoder contains a discriminator network, the purpose of which is to encourage the latent space to follow a (given) prior distribution ($P_{\text{prior}}$). This means that the latent space is less likely to have gaps and will perform better for unseen data [35]. See Figure 1 for a schematic diagram of an adversarial autoencoder, including the encoder, $\mathcal{E}^{\text{nc}}$, the decoder, $\mathcal{D}^{\text{ec}}$, and discriminator, $\mathcal{D}^{\text{is}}$. The discriminator is used during training, but is not needed for prediction (i.e., inference), for which the map learned by the AAE can be written as:

$$\tilde{\boldsymbol{x}} = f^{\text{AAE}}(\boldsymbol{x}) \equiv \mathcal{D}^{\text{ec}}\left(\mathcal{E}^{\text{nc}}(\boldsymbol{x})\right)\,, \tag{4}$$

where $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ represent the input and output of the network respectively. During training, there are three steps undertaken for each subset of training data (mini-batch). First, the reconstruction error is minimised (the difference between the input of the encoder and the output of the decoder); second, the discriminator is trained with samples generated by the encoder (labelled as false) and samples from the prior distribution (labelled as true); and finally, the encoder is trained to fool the discriminator. This training procedure can be summarised as

$$\min_{\mathcal{E}^{\text{nc}},\mathcal{D}^{\text{ec}}} \mathbb{E}\left(||\boldsymbol{x}-\tilde{\boldsymbol{x}}||_2^2\right) + \min_{\mathcal{E}^{\text{nc}}}\max_{\mathcal{D}^{\text{is}}}\left(\mathbb{E}_{\boldsymbol{z}\sim P_{\text{prior}}}[\log \mathcal{D}^{\text{is}}(\boldsymbol{z})] + \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}[\log(1-\mathcal{D}^{\text{is}}(\mathcal{E}^{\text{nc}}(\boldsymbol{x})))]\right)\,, \tag{5}$$

where $\mathbb{E}$ is the expectation, $\boldsymbol{z} \sim P_{\text{prior}}$ is a sample from the desired distribution and $\boldsymbol{x} \sim P_{\text{data}}$ is a sample from the training data (POD coefficients and sensor values). The first term represents the reconstruction error (involving the encoder and decoder), and the second term represents the adversarial training (involving the encoder and the discriminator). The input of the AAE is made up of a sequence of $m+1$ POD coefficients (representing the CFD model) and sensor values (representing the observations). To generate the sensor values used in training, the CFD model is evaluated at the sensor locations. When predicting, only the first $m$ sensor values need to be used along with the POD coefficients at $m$ time levels, and the algorithm will predict the POD coefficients and the sensor values at the subsequent time level. When assimilating data, actual observations at the sensor locations can be used, therefore $m+1$ sensor values and POD coefficients at $m$ time levels will be used to predict the POD coefficients at the subsequent time level.

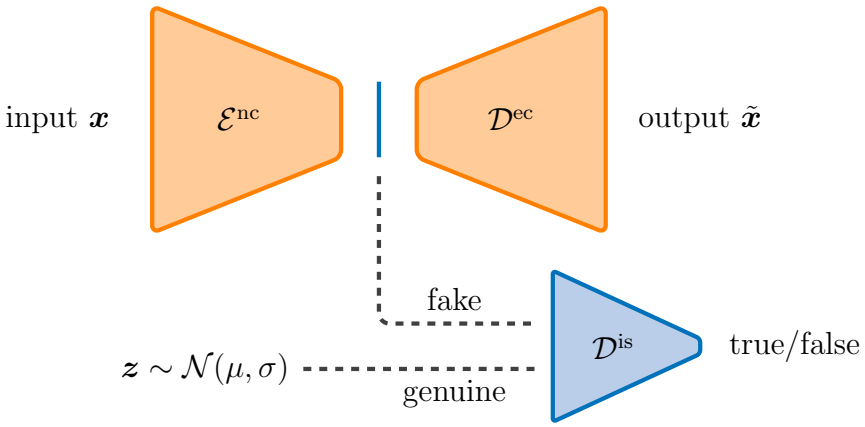



Figure 1: Schematic diagram of an adversarial autoencoder. The encoder is represented by $\mathcal{E}^{\text{nc}}$; the blue line represents the adversarial layer (and is the output of the encoder); the decoder network $\mathcal{D}^{\text{ec}}$ maps the values in the adversarial layer to the output; the input to the discriminator $\mathcal{D}^{\text{is}}$ is either a (genuine) sample from the prior distribution (here, a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with mean $\mu$ and variance $\sigma$) or a (fake) sample from the output of the encoder.

Autoencoders are typically used for compression. In order to use this type of network to predict in time, two strategies were tested, both using a time-marching approach. Both methods can be applied to the same trained AAE. When being used for prediction, the map learned by the AAE can be written as

$$\tilde{\boldsymbol{X}} = f^{\text{AAE}}(\boldsymbol{X}), \text{ where } \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}^k \\ \boldsymbol{x}^{k-1} \\ \vdots \\ \boldsymbol{x}^{k-m} \end{pmatrix}, \ \tilde{\boldsymbol{X}} = \begin{pmatrix} \tilde{\boldsymbol{x}}^k \\ \tilde{\boldsymbol{x}}^{k-1} \\ \vdots \\ \tilde{\boldsymbol{x}}^{k-m} \end{pmatrix}, \text{ and } \boldsymbol{X}, \tilde{\boldsymbol{X}} \in \mathbb{R}^{(m+1)\times F}. \tag{6}$$

$F$ represents the number of features (here, the number of POD coefficients plus the number of sensor values observed at each sensor multiplied by the number of sensors), $\boldsymbol{x}^k$ represents the POD coefficients and sensor values at time level $k$ and $\tilde{\boldsymbol{x}}^k$ represents the corresponding part of the output. Here, $\boldsymbol{x}^k$ and $\tilde{\boldsymbol{x}}^k$ are row vectors. The input and output of the AAE are 2D arrays, $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$ respectively, as convolutional layers are used within the encoder and decoder. Once the AAE has been trained to reproduce these values as closely as it can (whilst simultaneously attempting to make the latent space Gaussian), we can make predictions in time, using either of the following two methods.

The first method ("constrained prediction") requires $m$ time levels ($k-1$, $k-2$, …, $k-m$), from what is sometimes referred to as a "look-back window", in order to make a prediction for the $k$th time level. Both known and unknown values make up the input of the AAE. As $\boldsymbol{x}^k$ is unknown initially, we approximate it with $\boldsymbol{x}^{k-1}$. The values in Equation (6) are passed through the (trained) AAE resulting in an updated approximation for $\boldsymbol{x}^k$, written as $\tilde{\boldsymbol{x}}^k$. Disregarding the other outputs of the AAE (for the known or previous time levels), we iterate (by repeatedly passing the updated values for $\boldsymbol{x}^k$ and the unchanged known values through the AAE) until the values of $\boldsymbol{x}^k$ do not change within a given tolerance. We then take values at time levels ($k$, $k-1$, …, $k-m+1$) to be the known values and, from these, make a prediction for time level $k+1$, and so on. This procedure is described in Algorithm 1 and Figure 2.

**Algorithm 1** Constrained Prediction. By constraining the values of the sensors and POD coefficients at the previous (known) time levels, we can approximate the values of the sensors and POD coefficients at the future time level. The quantity represented by $||\cdot||_2$ is the $L_2$ norm, which, is related to the mean square error.

Given the maximum number of iterations ($N^{its}$); the trained AAE ($f^{\text{AAE}}$); a tolerance for the stopping criterion ($\varepsilon$); and $m$ previous solutions (POD coefficients and sensor values), $\boldsymbol{x}^{k-1}$, $\boldsymbol{x}^{k-2}$, $\ldots$, $\boldsymbol{x}^{k-m}$, this algorithm will make predictions for the POD coefficients and sensor values at future time levels.
**for** time level $k \in$ desired range of time levels **do**
    *!! approximate the solution at time level k*
    $\boldsymbol{x}^{k,0} = \boldsymbol{x}^{k-1}$
    **for** iteration $i = 1, 2, \ldots, N^{its}$ **do**

$$\begin{pmatrix} \tilde{\boldsymbol{x}}^{k,i-1} \\ \tilde{\boldsymbol{x}}^{k-1} \\ \vdots \\ \tilde{\boldsymbol{x}}^{k-m} \end{pmatrix} = f^{\text{AAE}} \begin{pmatrix} \boldsymbol{x}^{k,i-1} \\ \boldsymbol{x}^{k-1} \\ \vdots \\ \boldsymbol{x}^{k-m} \end{pmatrix} \tag{7}$$

        $\boldsymbol{x}^{k,i} = \tilde{\boldsymbol{x}}^{k,i-1}$
        **if** $\left(||\tilde{\boldsymbol{x}}^{k,i-1} - \boldsymbol{x}^{k,i-1}||_2^2 < \varepsilon \text{ or } i == N^{its}\right)$ **then**
            $\boldsymbol{x}^k = \boldsymbol{x}^{k,i}$
            exit iteration loop
        **end if**
    **end for**
**end for**

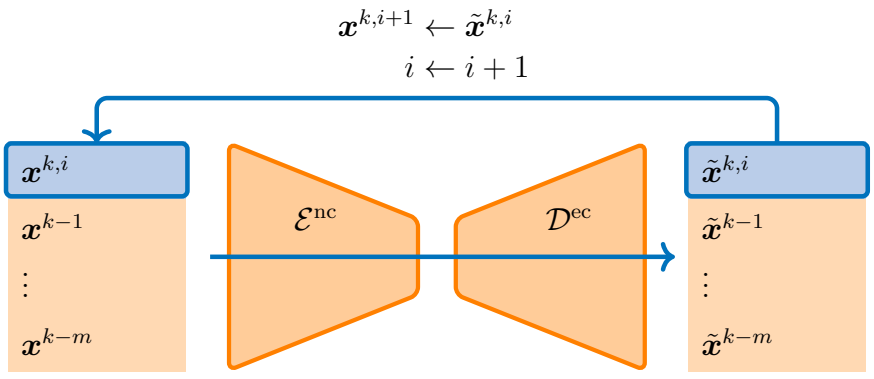



Figure 2: Schematic diagram illustrating the method of constrained prediction. The input includes known values ($\boldsymbol{x}^{k-1}, \boldsymbol{x}^{k-2}, \ldots, \boldsymbol{x}^{k-m}$) (shaded in orange), and unknown values $\boldsymbol{x}^{k,i}$ (shaded with blue). The approximation of the solution, $\boldsymbol{x}^{k,i}$, is fed into the AAE along with the known values and the output of the AAE, $\tilde{\boldsymbol{x}}^{k,i}$, is taken to be the updated solution. This is fed back into the input of the AAE with the known values and the iteration process continues until the solution does not change to within a given tolerance.

The second method ("prediction by backpropagation") uses the trained decoder to make predictions in

a manner similar to that described in Silva et al. [30]. Again, the method requires $m$ previous time levels to make a prediction for the the solution at a future time level. A set of randomly chosen latent variables ($\tilde{\boldsymbol{z}}^k$) is decoded and the difference between the output corresponding to the known values ($\tilde{\boldsymbol{x}}^{k-1}$, $\tilde{\boldsymbol{x}}^{k-2}$, $\ldots$, $\tilde{\boldsymbol{x}}^{k-m}$) and the known values themselves ($\boldsymbol{x}^{k-1}$, $\boldsymbol{x}^{k-2}$, $\ldots$, $\boldsymbol{x}^{k-m}$) is minimised with respect to the latent variables. This effectively finds the latent variables which result in an output close to the values at the $m$ known time levels. Once this has converged, the $(m+1)$th value in the output is taken as the prediction of the POD coefficients and sensor values at the future time level, time level $k$. The process is repeated for successive time levels as described in Algorithm 2 and Figure 3.

We note that, in the explanation of these two time-marching schemes, as we are predicting only and not assimilating data, we predict the future value of both the POD coefficients and the sensor values. When assimilating data, the future value of the sensor values will be known, so $\boldsymbol{s}^k$ can be treated as the other known variables. We also note that both prediction methods require $m$ known values in order to make a prediction for the subsequent time level. Initially, these $m$ values come from the high-fidelity model, however, after predicting for $m$ time levels, predictions are based on previous predictions from the AAE and no more values are needed from the high-fidelity model.

**Algorithm 2** Prediction by backpropagation. By minimising the difference between the output of the decoder and the solutions (POD coefficients and sensor values) at the known time levels, a set of the latent variables which produce the known values can be obtained. The decoder can then be used to obtain a prediction for the solution at time level $k$.

Given the trained decoder ($\mathcal{D}^{\text{ec}}$) and $m$ previous solutions (POD coefficients and sensor values), $\boldsymbol{x}^{k-1}$, $\boldsymbol{x}^{k-2}$, ..., $\boldsymbol{x}^{k-m}$, this algorithm will make predictions for the POD coefficients and sensor values at future time levels.

**for** time level $k \in$ desired range of time levels **do**

*!! set all d latent variables to random values at time level k*

$\tilde{\boldsymbol{z}}^k = \mathcal{N}^d(0,1)$

*!! evaluate the decoder*

$$\begin{pmatrix} \tilde{\boldsymbol{x}}^k \\ \tilde{\boldsymbol{x}}^{k-1} \\ \vdots \\ \tilde{\boldsymbol{x}}^{k-m} \end{pmatrix} = \mathcal{D}^{\text{ec}}\left(\tilde{\boldsymbol{z}}^k\right) \tag{8}$$

*!! minimise the difference between the output of the decoder and the desired output with respect to the latent variables — note that this is only done for the known values*

$$\boldsymbol{z}^k = \underset{\forall \tilde{\boldsymbol{z}}^k}{\arg\min} \left( \begin{pmatrix} \tilde{\boldsymbol{x}}^{k-1} \\ \tilde{\boldsymbol{x}}^{k-2} \\ \vdots \\ \tilde{\boldsymbol{x}}^{k-m} \end{pmatrix} - \begin{pmatrix} \boldsymbol{x}^{k-1} \\ \boldsymbol{x}^{k-2} \\ \vdots \\ \boldsymbol{x}^{k-m} \end{pmatrix} \right) \tag{9}$$

*!! evaluate the decoder for an input of* $\boldsymbol{z}^k$

$\mathcal{D}^{\text{ec}}(\boldsymbol{z}^k)$

*!! Discard the AAE's approximation of the solution at the previous time levels, and keep the approximation for time level k.*

$\boldsymbol{x}^k = \tilde{\boldsymbol{x}}^k$

**end for**

### *2.3. Data Assimilation*

In this section, we describe the method used to perform data assimilation in the reduced space (i.e. the low-dimensional space found by POD). Results for time series predictions presented in Section 3.3 will show that both the constrained prediction and prediction by backpropagation methods outlined in the previous section give similar results, so the first method (constrained prediction) is used here for its simplicity and fast convergence.

In order to perform data assimilation with the AAE as trained in the previous section, let us assume that we have observations for all the sensor values with which we trained the AAE (that is $CO_2$, relative humidity, velocities, temperature). When predicting in time with the AAE (and not performing DA),

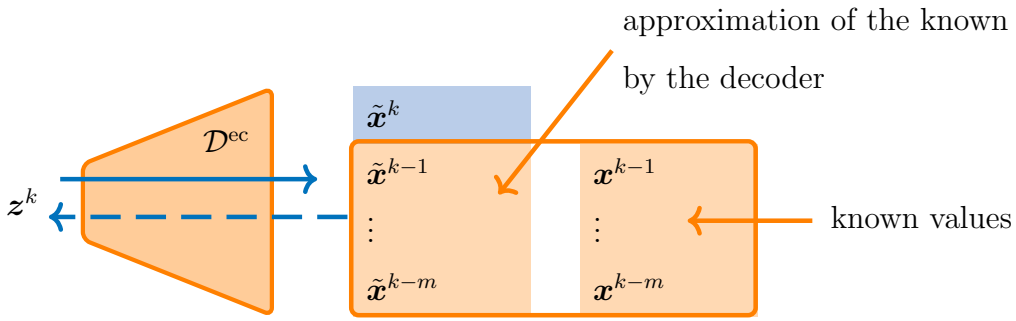



Figure 3: A schematic diagram of the "prediction by backpropagation" method. The difference between known solutions and solutions predicted by the decoder is minimised by backpropagating the difference through the decoder network. This finds a set of latent variables that corresponds to the known values. The remaining values in the output of the decoder are taken to be the solutions (POD coefficients and sensors values) at time level $k$ ($\tilde{\boldsymbol{x}}^k$).

the known values were the previous values of the POD coefficients and sensor values at $m$ time levels, and the unknown values were the POD coefficients and sensor values at the future time level, level $k$ (see Algorithm 1). Now, as we are assimilating data as well as predicting in time, the known values include the future value of the observations. For the two different cases of prediction and data assimilation, the known and unknown values are shown in Table 1. Algorithm 1 is rewritten in order to assimilate data with

| | known values | unknown values |
|---|---|---|
| prediction (only) | $\{\boldsymbol{\alpha}^{k'}\}_{k'=k-m}^{k-1}$ and $\{\boldsymbol{s}^{k'}\}_{k'=k-m}^{k-1}$ | $\boldsymbol{\alpha}^k$ and $\boldsymbol{s}^k$ |
| data assimilation | $\{\boldsymbol{\alpha}^{k'}\}_{k'=k-m}^{k-1}$ and $\{\boldsymbol{s}^{k'}\}_{k'=k-m}^{k}$ | $\boldsymbol{\alpha}^k$ |

Table 1: The known and unknown values used for prediction and for data assimilation.

the AAE in Algorithm 3. So, given an initial set of POD coefficients from time levels $(1, 2, \ldots, m)$ and a set of observations throughout time, $\{\boldsymbol{s}\}_{k=1}^{N^t}$, we approximate the POD coefficients at time level $m+1$ using Algorithm 3. The first time we pass the input through the AAE the first time, we approximate $\boldsymbol{\alpha}^k$ by $\boldsymbol{\alpha}^{k-1}$. Once we have a converged solution for the POD coefficients at the future time level, $\boldsymbol{\alpha}^k$, we march forwards in time until the final time level is reached, $N^t$, say. Now we march backwards in time, treating as known the observations $\{\boldsymbol{s}^{k'}\}_{k'=N^t-m}^{N^t}$ and POD coefficients $\{\boldsymbol{\alpha}^{k'}\}_{k'=N^t-m+1}^{N^t}$, and as unknown, the solution $\boldsymbol{\alpha}^{N^t-m}$. Once we have a converged solution at time level $N^t-m$, we continue marching back in time until we reach the first time level. We march forwards and backwards in time until the method converges, see Figure 4. There are two additions we make to this general procedure. First, rather than minimising the least squares mismatch over time, we rely on the AAE to produce solutions consistent with the known or fixed values, thereby producing solutions which we expect to reconcile the mismatch between the observations and the predictions. We do, however, make use of a functional based on the

least squares mismatch over time, defined as follows:

$$\mathcal{M} = \mathcal{M}^{\text{fwd}} + \mathcal{M}^{\text{bwd}} \tag{10}$$

$$\mathcal{M}^{\text{fwd}} = \sum_{k=m+1}^{N^t} \left(\hat{\boldsymbol{s}}^k - \tilde{\boldsymbol{\phi}}^k(\boldsymbol{x}_p)\right)^T \boldsymbol{W} \left(\hat{\boldsymbol{s}}^k - \tilde{\boldsymbol{\phi}}^k(\boldsymbol{x}_p)\right) \tag{11}$$

$$\mathcal{M}^{\text{bwd}} = \sum_{k=1}^{N^t-m} \left(\hat{\boldsymbol{s}}^k - \tilde{\boldsymbol{\phi}}^k(\boldsymbol{x}_p)\right)^T \boldsymbol{W} \left(\hat{\boldsymbol{s}}^k - \tilde{\boldsymbol{\phi}}^k(\boldsymbol{x}_p)\right) \tag{12}$$

where $\hat{\boldsymbol{s}}^k$ are the sensor observations at time level $k$, $\boldsymbol{x}_p$ are the sensor locations, $N^t$ is the number of time levels over which data is assimilated, $\boldsymbol{W}$ is the covariance matrix, here, taken as the identity matrix and $\tilde{\boldsymbol{\phi}}^k(\boldsymbol{x}_p)$ is the current approximation of the sensor values at time level $k$. This is calculated by using the inverse of Equation (3),

$$\tilde{\boldsymbol{\phi}}^k = \boldsymbol{R}\tilde{\boldsymbol{\alpha}}^k \quad \forall k\,, \tag{13}$$

and evaluating $\tilde{\boldsymbol{\phi}}^k$ at $\boldsymbol{x}_p$ by interpolating from the nearest nodal values. This can be done by pre-multiplying $\tilde{\boldsymbol{\phi}}^k$ by an interpolation matrix which will be sparse, containing non-zero values associated with the nearest nodes. The functional is evaluated after every backward march. After the second backward march (when there are at least two functional evaluations available), if the functional has increased in value (and therefore the match to the observed data has become less good), we apply relaxation over the next iteration, as follows:

$$\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^{k,\,\text{old}} + \gamma\left(\tilde{\boldsymbol{\alpha}}^k - \boldsymbol{\alpha}^{k,\,\text{old}}\right) \tag{14}$$

where $\gamma \in [0, 1]$ controls the amount of relaxation applied, $\boldsymbol{\alpha}^{k,\,\text{old}}$ is the solution approximated during the previous backwards march, $\tilde{\boldsymbol{\alpha}}^k$ is the solution predicted by the AAE, and $\boldsymbol{\alpha}^k$ is the approximation to the solution at time level $k$ (after relaxation has been applied). Initially, the value of $\gamma$ is set to be one, and when the functional increases the value of $\gamma$ is halved.

The second addition made to the basic procedure of marching forwards and backwards in time is prioritisation. By introducing a parameter $\beta_{j\ell} \in [0, 2]$ for each observed value $\ell$ at each sensor $j$, we can control the importance of each observation. For example, if the observation for field $\ell$ at the $j$th sensor at time level $k$ is represented by $\hat{\boldsymbol{s}}^k_{j\ell}$, then we prioritise as follows:

$$\boldsymbol{s}^k_{j\ell} = \beta_{j\ell}\hat{\boldsymbol{s}}^k_{j\ell} + (1 - \beta_{j\ell})\tilde{\boldsymbol{s}}^k_{j\ell}\,, \tag{15}$$

where the tilde sign ($\tilde{}$) represents the output of the AAE and $\boldsymbol{s}^k_{j\ell}$ represents the approximation to this sensor value which will be used as in input to the AAE at the next iteration. The observation itself, $\hat{\boldsymbol{s}}^k_{j\ell}$, is fixed. If this observation is to be ignored, then $\beta_{j\ell}$ can be set to zero. If the AAE is to be relied upon to enforce the observation, then $\beta_{j\ell}$ can be set to one. If more priority is to be given to this observation then $\beta_{j\ell} > 1$. This is a very flexible way of ignoring some observations and prioritising other observations without having to retrain the neural network. See Algorithm 4 for a description of the DA algorithm proposed in this paper. $\beta_{j\ell} = 1$ might be seen as the default prioritisation. Figure 5 illustrates the DA process. The blue loop represents the inner iteration which, for the forward march, obtains the POD coefficients at the future time level, $\boldsymbol{\alpha}^k$ given the previous $m$ values of POD coefficients, $\{\boldsymbol{\alpha}^{k-1}, \boldsymbol{\alpha}^{k-2}, \ldots, \boldsymbol{\alpha}^{k-m}\}$ and observations $\{\boldsymbol{s}^k, \boldsymbol{s}^{k-1}, \ldots, \boldsymbol{s}^{k-m}\}$. The forward march iterates from $k = m + 1$ to $N^t$ with an increment of 1, and once complete, the backward march begins, where the

solution at time level $k$ is obtained from future solutions $\{\boldsymbol{\alpha}^{k+m}, \boldsymbol{\alpha}^{k+m-1}, \ldots, \boldsymbol{\alpha}^{k+1}\}$ and observations $\{\boldsymbol{s}^{k+m}, \boldsymbol{s}^{k+m-1}, \ldots, \boldsymbol{s}^{k}\}$. The backward march iterates from $k = N^t - m$ to 1 with a decrement of 1. One forward and one backward march makes up an outer iteration which is indicated in red. When the outer iterations converge, one final forward march is performed before the algorithm finishes.

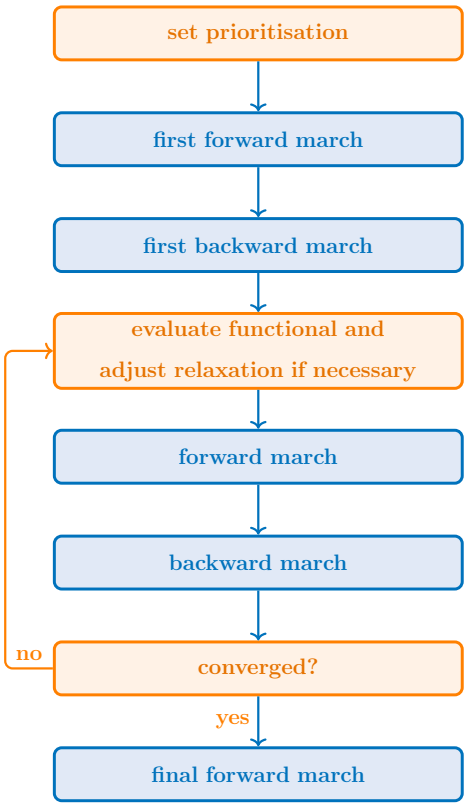



Figure 4: Forward and backward marching that occurs when performing the data assimilation.

**Algorithm 3** Constrained Prediction for use with DA. By constraining the values of the sensors and POD coefficients at the previous (known) time levels, and the sensor values at the future time level, we can approximate the POD coefficients at the future time level.

This algorithm will make improved predictions for the POD coefficients for each time level in the set of desired values $\mathcal{K}$, given the maximum number of iterations ($N^{its}$); the trained AAE ($f^{\text{AAE}}$); a tolerance for the stopping criterion ($\varepsilon$); $m$ previous solutions (POD coefficients), $\{\boldsymbol{\alpha}^{k'}\}_{k'=k-m}^{k-1}$; $m+1$ sensor values, $\{\boldsymbol{s}^{k'}\}_{k'=k-m}^{k}$; the range of time levels $\mathcal{K}$; and the relaxation parameter $\gamma$.
**for** time level $k \in \mathcal{K}$ **do**
    *!! approximate the solution at time level k*
    **if** `forward` **then**
        $\boldsymbol{\alpha}^{k,0} = \boldsymbol{\alpha}^{k-1}$
    **else**
        $\boldsymbol{\alpha}^{k,0} = \boldsymbol{\alpha}^{k+1}$
    **end if**
    **for** iteration $i = 1, 2, \ldots, N^{its}$ **do**
        form $\boldsymbol{X}$ and pass through the autoencoder
        $\tilde{\boldsymbol{X}} = f^{\text{AAE}}(\boldsymbol{X})$
        *!! update the approximation of the POD coefficients using values in* $\tilde{\boldsymbol{X}}$
        $\boldsymbol{\alpha}^{k,i} = \tilde{\boldsymbol{\alpha}}^{k,i-1}$
        **if** $\left(||\tilde{\boldsymbol{\alpha}}^{k,i-1} - \boldsymbol{\alpha}^{k,i-1}||_2^2 < \varepsilon \text{ or } i == N^{its}\right)$ **then**
            $\boldsymbol{\alpha}^{k} = \boldsymbol{\alpha}^{k,i}$
            exit iteration loop
        **end if**
    **end for**
**end for**

**Algorithm 4** This algorithm details the data assimilation algorithm proposed here, which marches forwards and backwards through time, adjusting the relaxation parameter, $\gamma$, according to the value of the mismatch functional.

set prioritisation coefficients $\beta_{j\ell}$ for all the sensor values
set $\mathcal{M}^0 = 10^{10}$
set $\gamma = 1$
**for** iteration $i = 1, \ldots, N^{its}$ of outer loop **do**
    *!! forward march with* $\mathcal{K} = \{m+1, m+2, \ldots, N^t\}$ *using Algorithm 3*
    **for** $k = m+1$, $m+2$, $\ldots$, $N^t$ **do**
        predict $\boldsymbol{\alpha}^k$ given $\{\boldsymbol{\alpha}^{k'}\}_{k'=k-m}^{k-1}$ and $\{\boldsymbol{s}^{k'}\}_{k'=k-m}^{k}$.
    **end for**
    *!! backward march with* $\mathcal{K} = \{N^t - m, N^t - m - 1, \ldots, 1\}$ *using Algorithm 3*
    **for** $k = N^t - m$, $N^t - m - 1$, $\ldots$, $1$ **do**
        predict $\boldsymbol{\alpha}^k$ given $\{\boldsymbol{\alpha}^{k'}\}_{k'=k-m}^{k-1}$ and $\{\boldsymbol{s}^{k'}\}_{k'=k-m}^{k}$.
    **end for**
    *!! relaxation*
    evaluate the functional, $\mathcal{M}$, at iteration $i$ using Equation (10)
    **if** $(\mathcal{M}^i > \mathcal{M}^{i-1})$ **then**
        $\gamma \leftarrow 0.5\gamma$
    **end if**
    *!! check convergence*
**end for**
*!! final forward march with* $\mathcal{K} = \{m+1, m+2, \ldots, N^t\}$ *using Algorithm 3*
**for** $k = m+1, \ldots, N^t$ **do**
    predict $\boldsymbol{\alpha}^k$ given $\{\boldsymbol{\alpha}^{k'}\}_{k'=k-m}^{k-1}$ and $\{\boldsymbol{s}^{k'}\}_{k'=k-m}^{k}$
**end for**

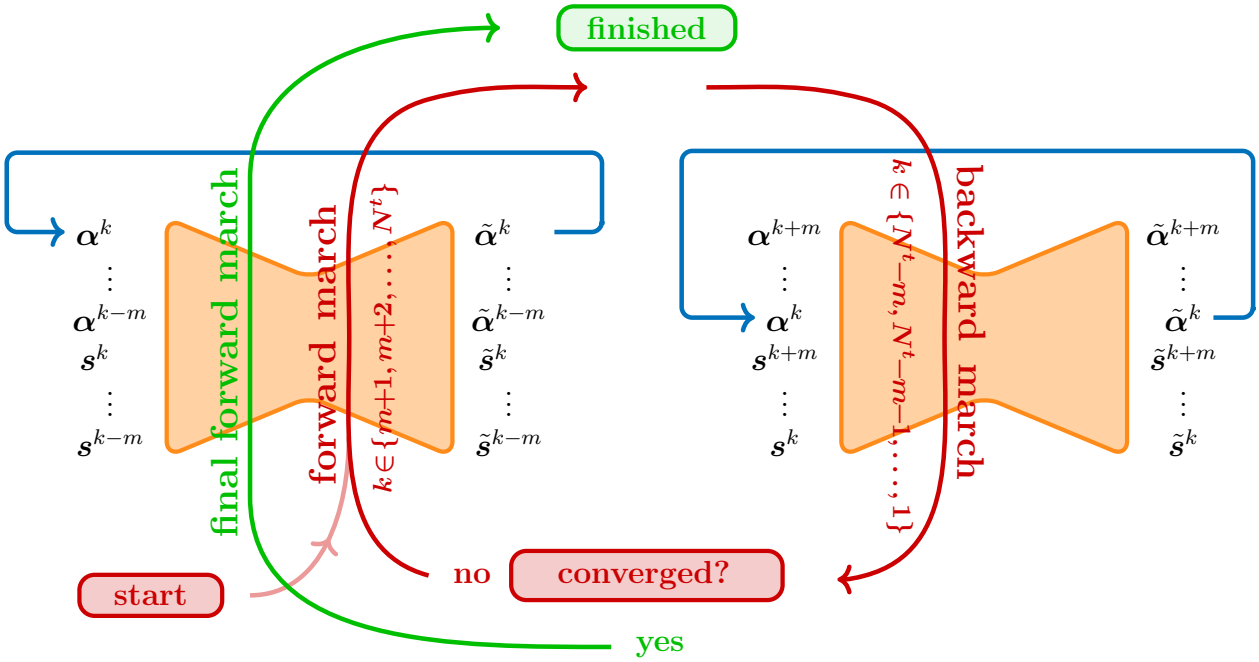



Figure 5: A schematic diagram of the data assimilation process.

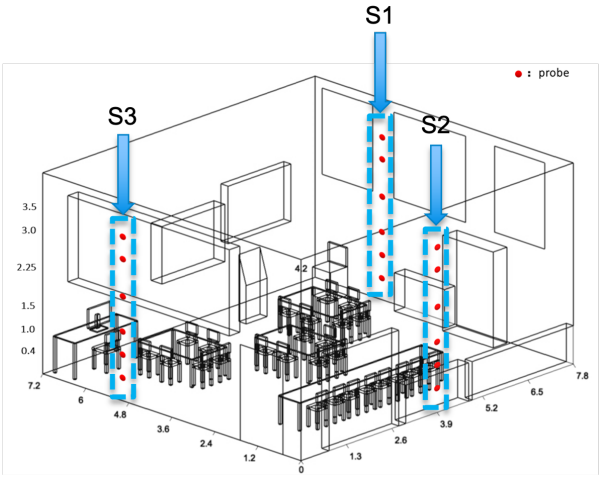
## 3. Results

First we present a description of the test case, explaining where the observations were taken and how the test case was modelled numerically. In Section 3.2 we give details on how the dimension of the CFD results was reduced; in Section 3.3 we present predictions by the surrogate model of the air flows and air quality in the room. These predictions are based on the constrained method of prediction. In Section 3.4 we describe how the data assimilation approach was tested; first, on a dual-twin type experiment and second, with observations from a classroom.

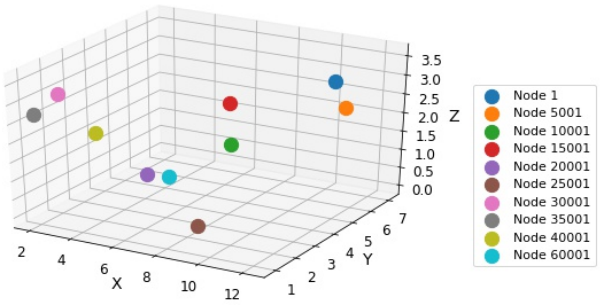
### *3.1. Test Case*

A classroom in Houndsfield Primary School, north London, is used to demonstrate the proposed methods. As part of a measurement campaign to investigate indoor air quality, observations of $CO_2$, relative humidity and temperature were collected at 18 locations over a two week period [38]. From this, we select observations collected over an hour whilst the classroom was occupied by 26 students and one teacher. The measurements have been time-averaged over 1 min. Heating was supplied by a radiator and a Dyson fan, the latter was also used to clean the air and remove the virus-laiden particles from the air. Figure 6 shows a schematic diagram of the classroom, including desks, chairs, windows, the door to an internal corridor, the fan, the radiator (on the same wall as the sensors labelled S1) and the sensor locations. The classroom measures 7.2 m by 7.8 m with a height of 4.2 m. The door to the internal corridor was open (seen in Figure 6a on the same wall as sensors labelled S3) and the top-hinged windows on the right were also open (seen in Figure 6a opposite the door to the internal corridor and near the sensors labelled S1. Another view of this can be seen in Figure 7, which includes the teacher and students.

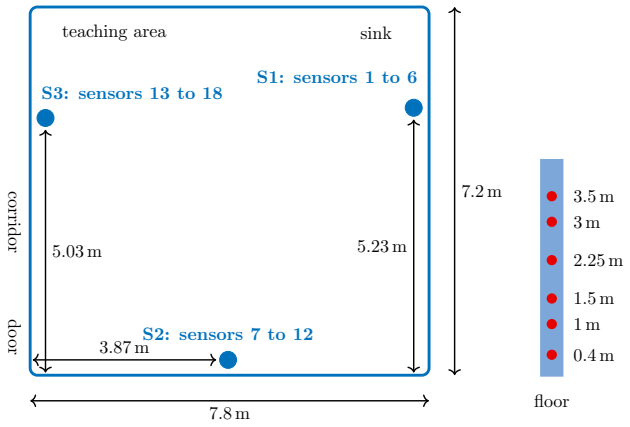
In addition to collecting observations, a Large Eddy Simulation (LES) was performed to generate high-fidelity numerical results of the air flows and air quality in the classroom by using the open source CFD code Fluidity [42, 43]. Based on the finite element method, this code solves the Navier-Stokes equations for unsteady, incompressible and viscous flow with an advanced mesh capability, which includes unstructured, anisotropic and adaptive meshing [44]. Fluidity is able to predict the evolution of components in the air (i.e., $CO_2$, water vapour, temperature and virus concentration) by the transportation of scalar fields as tracers. The principle behind the LES model is based on fully resolving large-scale turbulent structures and modelling the less energetic small-scale structures through a subgrid-scale model. With this method, Fluidity has been able to resolve turbulent features of urban flows and the dispersion of passive tracers down to length scales of 10 cm and time scales of less than a second [45–47]. Thermal stratification is modelled by the Boussinesq approximation which assumes that the predominant influence of density variation is on the buoyancy forces. The geometry of the classroom is created using the Indoor Geometry Generator (IGG) [48], which is a software package that allows easy definition of objects such as furniture, doors, and windows, as well as people sitting or standing. Figure 7 shows the layout of the room used for the CFD simulations. The dimensions and locations of desks, students, fan and radiator are identical to those of the classroom. To replicate the indoor-outdoor exchange or air, there are two additional buffer regions attached to the left and right of the classroom. Natural ventilation is simulated by imposing inflow and outflow boundary conditions on certain walls of the buffer regions, which allows air to go in or out of the room through the door which leads to the internal corridor (left) and through the top-hinged windows (right). In Figure 7, the inlets are shaded yellow and the outlets are shaded green. Inlet




(a) Layout of the classroom and its contents




(b) Node locations for some time history plots




(c) Sensor locations

Figure 6: Classroom geometry, sensor and node locations used for time history plots. (a) and (c) Location of the 18 sensors used to collect $CO_2$, relative humidity and temperature data. (Sensors 1 to 6 share the same $x$ and $y$ coordinates, with the sensor index increasing with height. Similarly for sensors 7 to 12 and 13 to 18.) See Kumar et al. [38] for more details.

velocities are specified on the nearest facing planes of the buffer regions of 0.5 m/s (left) and 1.0 m/s (right). Open boundary conditions are specified on the far walls of the buffer regions (shaded green), by setting non-hydrostatic pressure to zero. On all other external boundaries and internal walls, floors and ceilings, we use a natural boundary condition of zero shear stress with a normal velocity of zero. For any remaining surfaces, such as the surface of desks, objects and people, a no-slip boundary condition is applied. Regarding boundary conditions for temperature, zero heat flux was applied on the walls, floor, furniture and ceiling. Within the classroom, a Dirichlet temperature boundary condition was applied to the radiator enforcing a temperature of 32 °C. In order to account for the body heat of individuals, a Robin or heat transfer boundary condition was applied imposing a temperature of 34 °C and using a surface heat transfer coefficient of $10\,\mathrm{W\,m^{-2}\,K^{-1}}$. A Dirichlet temperature boundary condition was also applied at the largest monitor with a temperature of 40 °C, which can be seen in Figure 6a on the desk near the array of sensors labelled S3. Dirichlet boundary conditions are also adopted at the 'mouth' and 'nose' area of people for other passive scalar fields (i.e., $CO_2$, relative humidity, water vapour and virus concentration), in order to reproduce the inhalation and exhalation process. Initial conditions are given in Table 2 and boundary conditions for $CO_2$, relative humidity and viral load are consistent with these values. The simulation is started impulsively, so the value of velocity is adjusted until it satisfies continuity during the first time step. An adaptive time stepping scheme is applied in this simulation, which means that the time step is chosen so that a Courant number of 10 is obtained. The anisotropic unstructured mesh is adapted so that the smallest element edge length is 0.1 m and the largest element edge length is 2.5 m.

| | left buffer | school classroom | right buffer |
|---|---|---|---|
| $CO_2$ (ppm) | 400 | 430 | 400 |
| temperature (K) | 279 | 289 | 279 |
| relative humidity (%) | 40 | 40 | 40 |
| viral load (copies) | 0 | 0 | 0 |
| velocity | | *impulsively started* | |

Table 2: The initial conditions used in the CFD simulation. The values are constant throughout each region.

Solutions of the high-fidelity model were generated every 5 s over a period of one hour. Although the solutions were generated using mesh adaptivity, in order to apply POD, all the solutions were interpolated onto the unstructured mesh that was generated at the end of the simulation, which had 192,060 nodes. Using solutions on different-sized meshes would be possible, but would require the use of space-filling curves [49], spatially varying kernels [50] or graph neural networks [51]. Here we choose to interpolate all the solutions onto a single (unstructured) mesh, as a fast and straightforward option whilst we investigate the capability of the newly proposed DA method. The solutions or snapshots were separated into two groups: a training or seen dataset corresponding to a time window of $[5, 2880]$ seconds (576 time levels); and a test or unseen dataset corresponding to a window of $[2885, 3600]$ seconds (144 time levels). Within these time windows, solutions are available every 5 s.

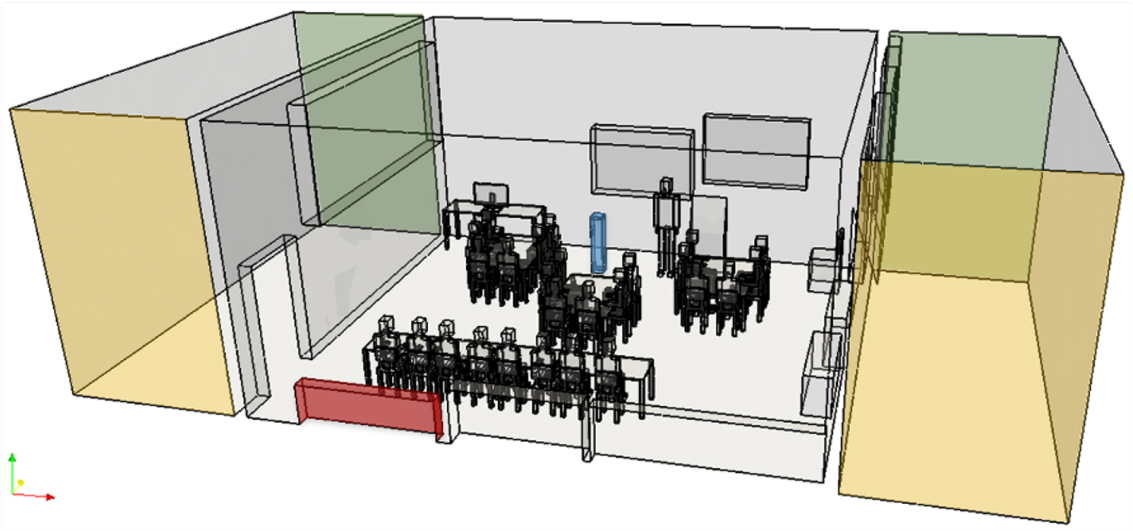


Figure 7: Geometry of the classroom at Houndsfield Primary School used in the CFD simulations showing the furniture, 26 students and the teacher. The blue rectangular box represents a Dyson fan; the red rectangular box represents a radiator. The two yellow surfaces denote two inlets with imposed velocities of 0.5 m/s (left) and 1.0 m/s (right), where incoming air has the temperature of 279 K. The two green surfaces denote two outlets with open pressure conditions.

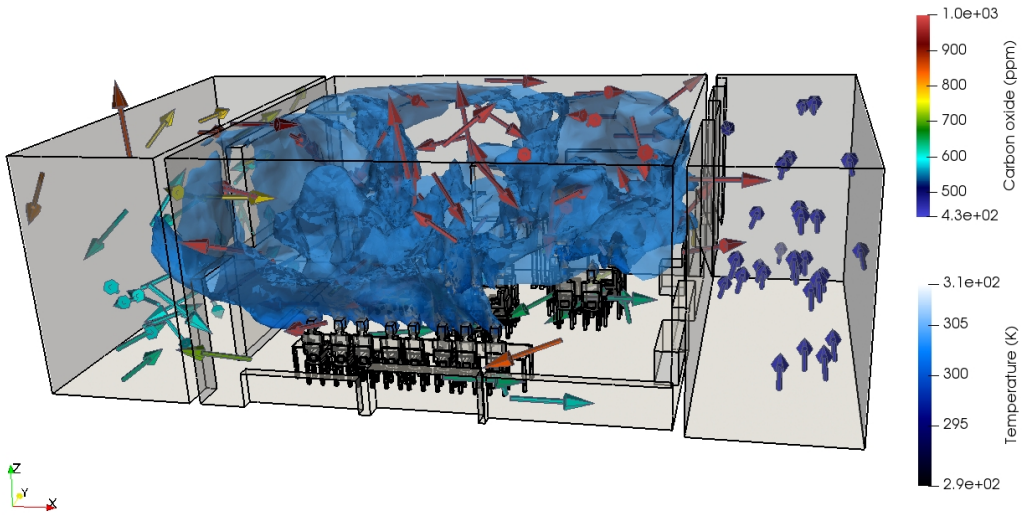


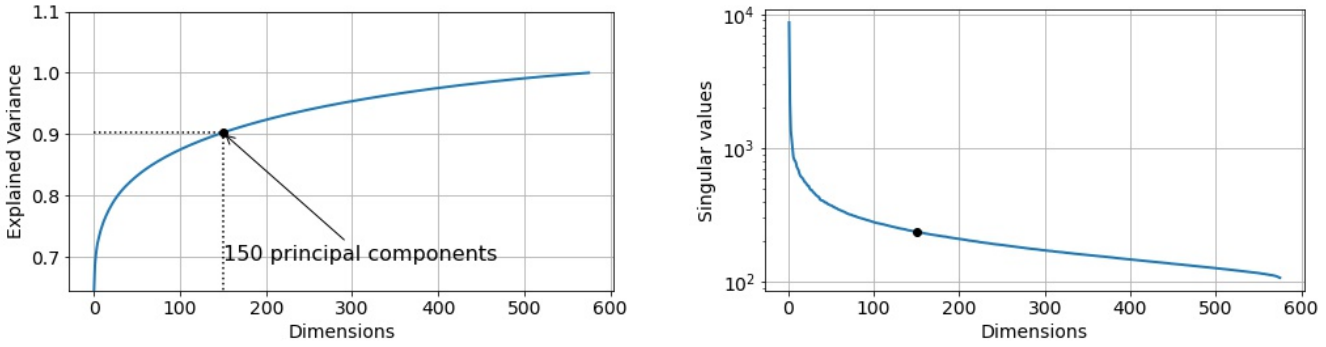

Figure 8: Thermal $CO_2$ plume within the naturally ventilated classroom simulated after one hour. The iso-surface represents a level of 800 ppm of $CO_2$ and is coloured by temperature; arrows represent air flows and are coloured by $CO_2$ level.

### *3.2. Dimensionality Reduction*

POD is applied to the training dataset to reduce the dimension of the problem. Seven solution fields ($CO_2$, velocities, temperature, relative humidity and viral load) defined on an unstructured mesh of 192,060 nodes are approximated by 150 POD basis functions and coefficients, which represents capturing 90.2% of the information contained by the snapshots in the training dataset. This corresponds to a compression ratio of 1,344,420 (7 fields, each of 192,060 nodes) to 150, or almost 9000 to 1. Before applying POD, the seven fields are normalised so their values lie in the range [-1,1]. After applying POD, further normalisation is applied to the POD coefficients, so their values lie in the range [0,1], as they will be used to train a neural network to predict the evolution of the POD coefficients. This is standard machine learning practice. Once calculated, these normalisation functions are applied, as needed, to any unseen data. Figure 9a shows the variance captured by the singular values (or principal components), and Figure 9b shows how the information captured by the POD basis functions decays with increasing numbers of basis functions. Although 150 POD coefficients capture over 90% of the information, at this stage the singular values are decaying slowly. Retaining more coefficients would improve the accuracy, however, this would make training the networks more difficult, so using 150 coefficients was seen as a good compromise between accuracy and tractability of training the networks. The POD coefficients associated with the training dataset,

$$\boldsymbol{\alpha}^k = \boldsymbol{R}^T \boldsymbol{\phi}^k \quad \forall k \in \text{ training dataset}\,, \tag{16}$$

will be used to train the neural network that will predict in time and assimilate data.




(a) Explained variance

(b) Singular values

Figure 9: POD is applied to the snapshots in the training dataset in order to reduce the dimension from 1,344,420 to 150 (a compression ratio of almost 9000). Retaining 150 POD basis functions is equivalent to capturing 90.2% of the information held in the snapshots in the training dataset. The so-called explained variance is plotted on the left, and the decay of the singular values is plotted on the right.

### *3.3. Prediction*

For prediction, an adversarial autoencoder with convolutional layers is used. For training the network, the data is assembled in the form of 2D arrays, using (normalised) POD coefficients and normalised sensor values at a number of successive time levels. The dimension of the input to the network is the number of consecutive time levels used by the number of POD coefficients + (number of sensors × number of fields observed at each sensor):

$$9 \text{ by } 150 + (18 \times 6)\,, \quad \text{that is} \quad 9 \text{ by } 258\,. \tag{17}$$

See Table 3 for a diagram of the structure of the 2D data. For flexibility, we allow for six fields of sensor data. In the dual-twin experiment we use data from the high-fidelity model as the sensor data (by interpolating the high-fidelity model at the sensor locations). When assimilating the observations from the classroom, we have observations for $CO_2$, temperature and humidity, and we set the other values by interpolating the high-fidelity model at the sensor locations. The adversarial autoencoder network

| | $\alpha_1$ | $\alpha_2$ | $\cdots$ | $\alpha_{149}$ | $\alpha_{150}$ | $(\boldsymbol{s_1})_1$ | $(\boldsymbol{s_1})_2$ | $\cdots$ | $(\boldsymbol{s_1})_6$ | $(\boldsymbol{s_2})_1$ | $\cdots$ | $(\boldsymbol{s_{18}})_6$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t^k$ | | | | | | | | | | | | |
| $t^{k-1}$ | | | | | | | | | | | | |
| $t^{k-2}$ | | | | | | | | | | | | |
| $\vdots$ | | | | | | | | | | | | |
| $t^{k-8}$ | | | | | | | | | | | | |

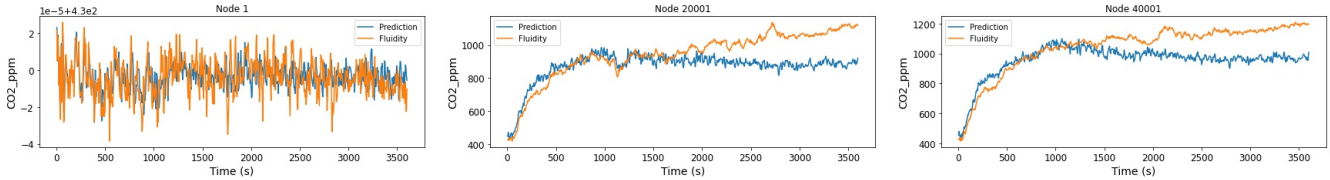
9 by 258

Table 3: The 2D structure of the data used for training, with 9 rows (samples) and 258 columns (features). The $i$th POD coefficient is represented by $\alpha_i$. Values associated with the $j$th sensor are represented by $\boldsymbol{s_j}$. Each sensor will have 6 values, one for each of the six fields $\{(\boldsymbol{s_j})_\ell\}_{\ell=1}^{6}$.

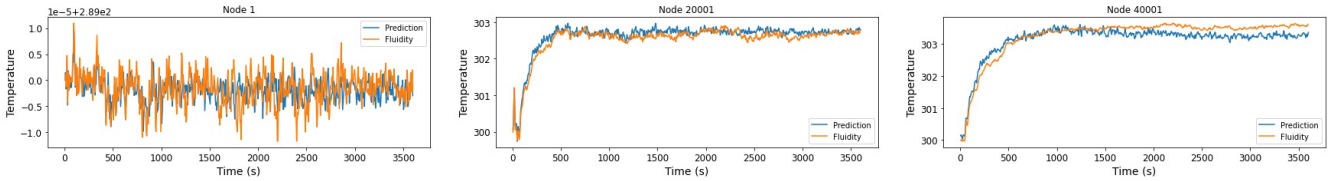
was implemented in TensorFlow [52] and a schematic diagram of the network can be seen in Figure 1. After optimisation, the best values and settings for the hyperparameters were found and are shown in Tables A.4 and A.5.

Figures 10a to 10g demonstrate how well the non-intrusive reduced-order model, referred to here as PredAAE, predicts for the seven fields of interest ($CO_2$, velocities, temperature, relative humidity and viral load). The orange curves represent time series results from the high-fidelity model and the blue curves are the time series predictions made by the PredAAE using the constrained prediction method. The time series were collected at the location of three of the nodes of the unstructured mesh (see Figure 6c). The training data is taken from the time interval [5,2880] seconds. The test data is taken from the remaining time [2885, 3600] seconds. PredAAE predicts well over the training data for velocities and temperature, but for $CO_2$, humidity and viral load, the predictions start to stray from the high-fidelity model results. This difference can be seen when the predictions are still within the training data time interval. A similar collection of plots can be seen in Figure 11 for when the PredAAE method uses the backpropagation method. The results shown here are broadly similar to those for the constrained method presented in Figure 10, although for relative humidity and viral load, there is some improvement in the accuracy of the predicitons. We emphasise here, that (for both constrained and backpropagation methods) the reduced-order model uses solutions at 8 previous time levels to approximate the solution at a future time level. The model therefore requires 8 time levels as initial conditions, but once the model has made predictions for 8 time levels, the predictions are based solely on outputs of the model and do not require any further CFD solutions.

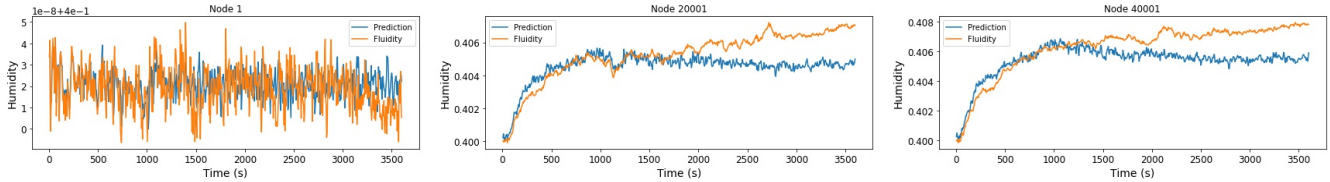
A comparison of the convergence of the two prediction methods can be seen in Figure 12, plotting the mean square error between predictions at successive iterations. This shows that the constrained method converges very quickly (within a few iterations), whereas the backpropagation method takes longer to converge but converges to a lower error.



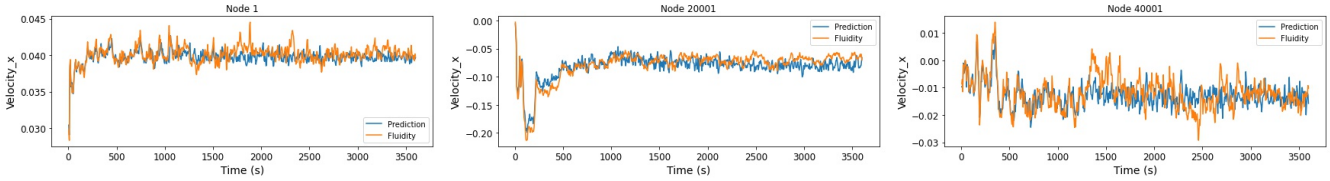

(a) $CO_2$



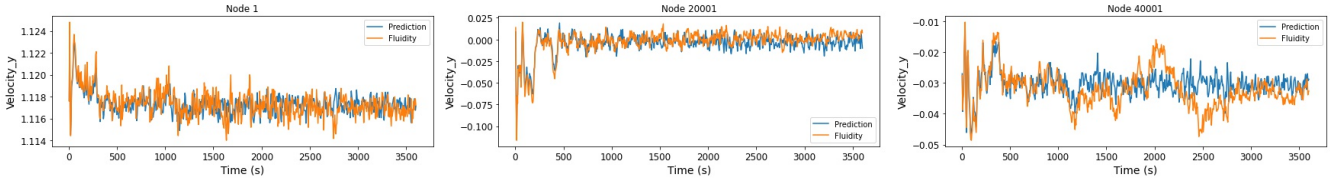

(b) Temperature



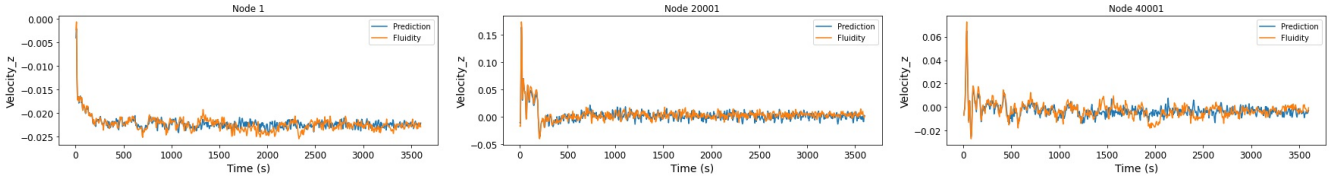

(c) Humidity



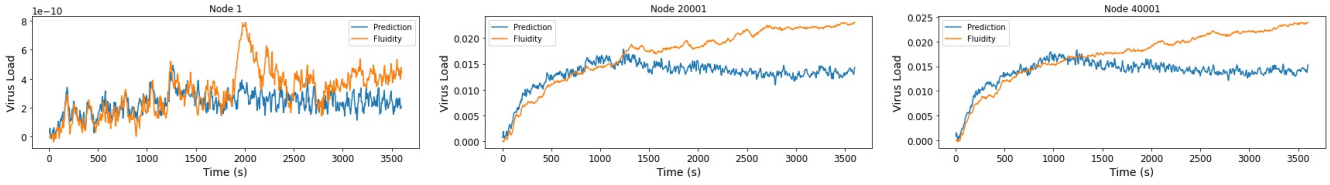

(d) $x$ component of velocity




(e) $y$ component of velocity

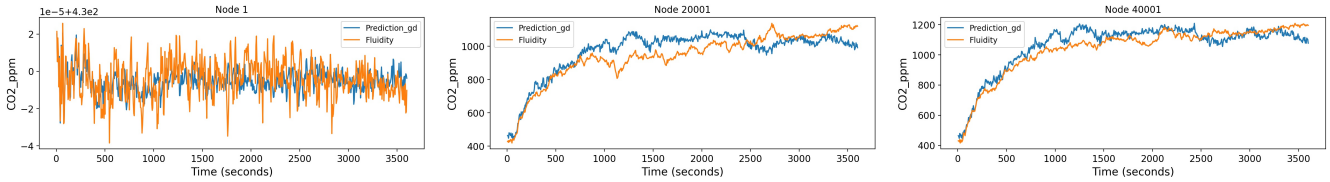



(f) $z$ component of velocity

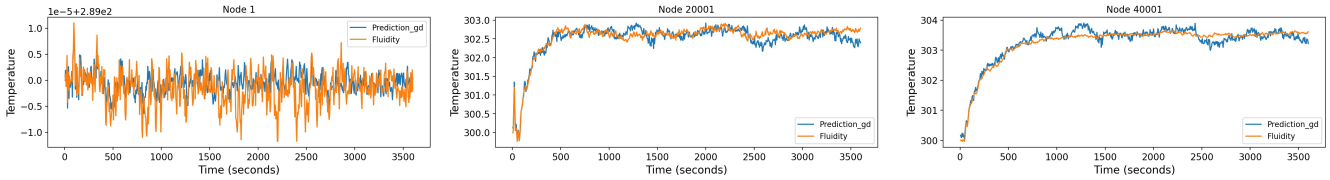



(g) Viral load

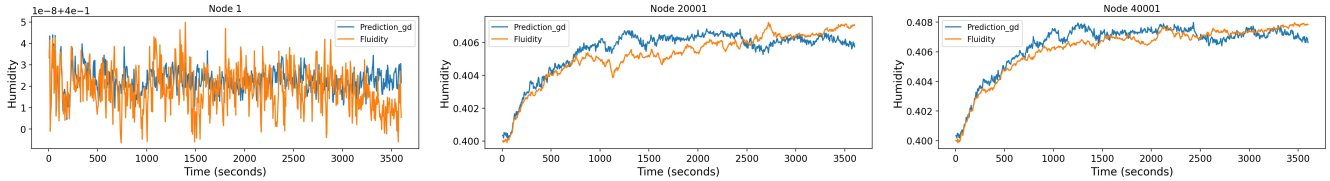
Figure 10: Time series plots of the seven fields at three locations corresponding to nodes 1, 20001 and 40001 (see Figure 6c). Orange represents the high-fidelity model results and blue represents the PredAAE prediction with the constrained method.



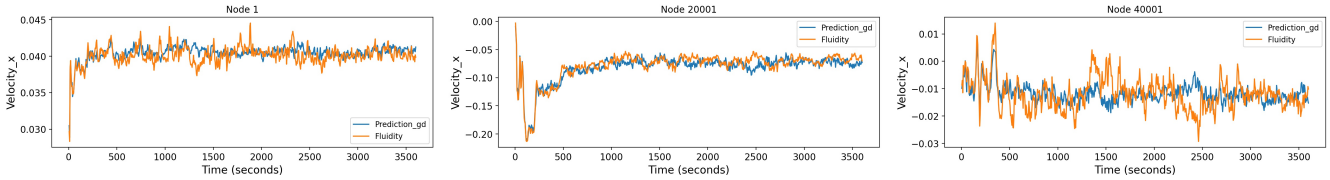

(a) $CO_2$



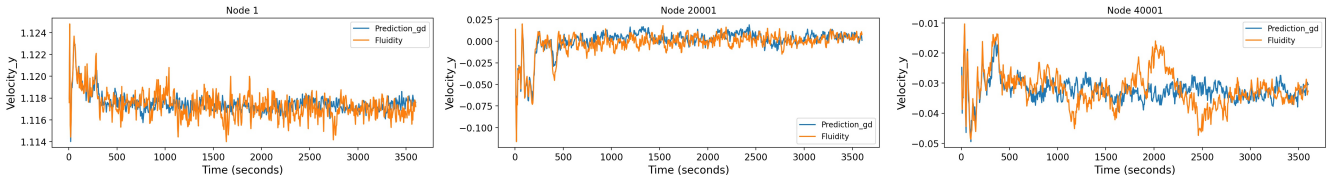

(b) Temperature



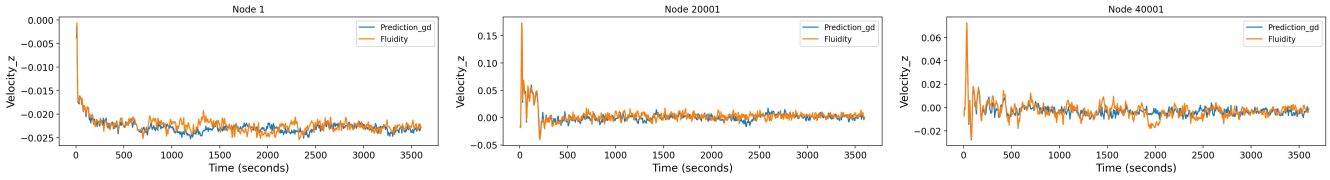

(c) Humidity



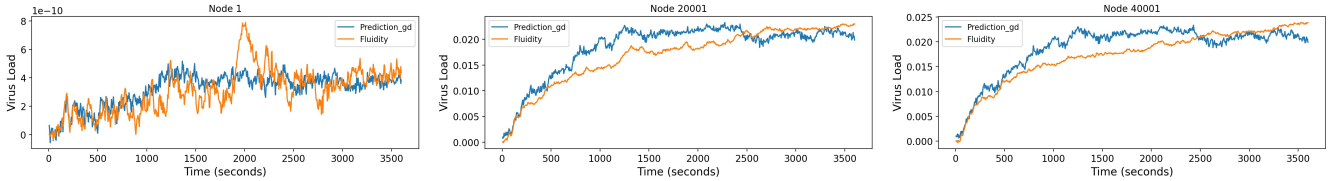

(d) $x$ component of velocity




(e) $y$ component of velocity

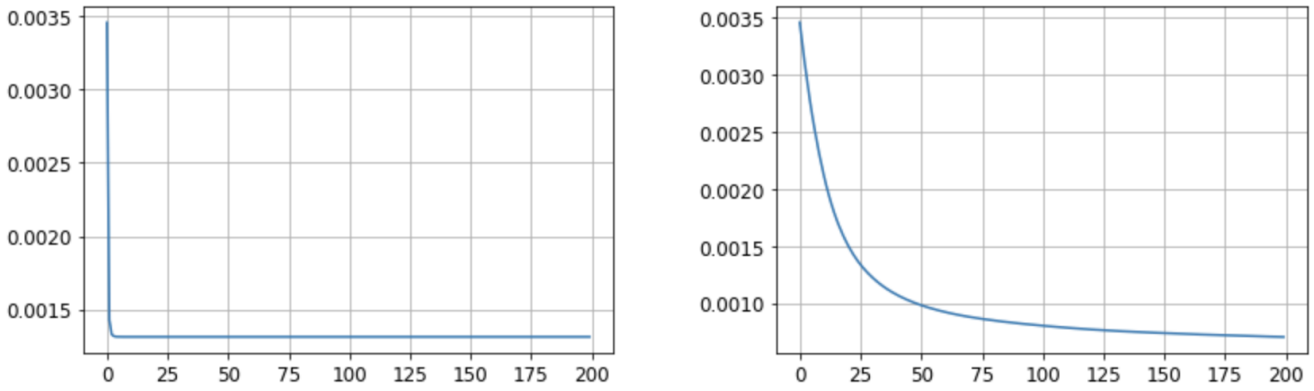



(f) $z$ component of velocity




(g) Viral load

Figure 11: Time series plots of the seven fields at three locations corresponding to nodes 1, 20001 and 40001 (see Figure 6). Orange represents the high-fidelity model results and blue represents the PredAAE prediction with the backpropagation method.




(a) Constrained method

(b) Backpropagation method

Figure 12: Comparison between the two methods of predicting in time with the PredAAE. The mean square error between solutions at successive iterations is shown on the $y$ axis, the number of iterations on the $x$ axis.

### 3.4. Assimilating data

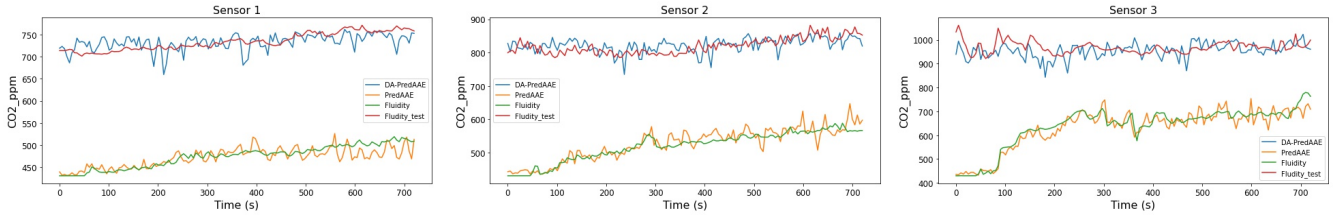
Once trained, the PredAAE reduced-order model is used to assimilate data. We investigate two scenarios in this paper: (1) assimilating data from the test dataset, where the sensor values are provided by the high-fidelity model results from 2885 to 3600 seconds, and the POD coefficients are taken from $t \in \{5, 10, \ldots 40\}$ and marched forward to 720 s (sometimes referred to as a dual twin experiment); (2) assimilating observations collected from a classroom.

#### 3.4.1. Scenario 1: dual-twin experiment

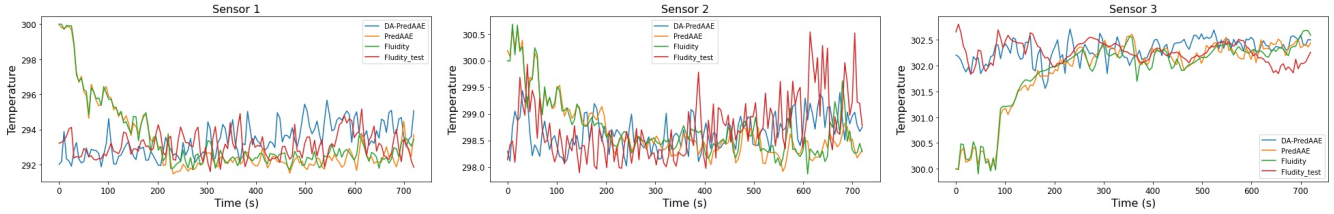
To test the capability of the data assimilation algorithm, we perform a dual-twin experiment. The AAE expects an input of POD coefficients and sensor values at one particular time level. For the dual-twin experiment, the input of the AAE is consists of sensor data from time 2885 s to 3600 s (taken from the test dataset) and the POD coefficients taken from 5 s to 720 s (using "initial conditions" from 5 s to 40 s and predicting/assimilating data with the AAE up to 720 s). After assimilating the sensor values, the aim is to see whether the POD coefficients (having been mapped to the solution space) are consistent with the solutions from 2885 s to 3600 s. If so, the data assimilation method is judged to be successful. Figure 13 shows the solutions for $CO_2$, temperature and humidity after assimilating data from the sensors taken during the test dataset. This illustrates clearly that before data assimilation, the predictions (from the POD coefficients, mapped to the solution space) are aligned with the CFD model from the time interval $[5, 720]$, whereas after assimilation, the predictions are closely aligned with the CFD solution from the time interval $[2885, 3600]$. This tells us that the data assimilation algorithm is working well. Results are also shown at a number of nodes, see Figure 14, from which the same conclusion can be drawn, that before data assimilation, the predictions are consistent with the initial conditions and the CFD solutions from the time interval $[5, 720]$, whereas after assimilation, the predictions are closely aligned with the behaviour associated with the observations, i.e. the CFD solution from the time interval [2885,3600]. The prediction for $CO_2$ (Figure 14a) broadly follows the trend seen in the relative humidity (Figure 14c) and the viral load (Figure 14f), which has a similar source from the breath to $CO_2$ but has a half-life of SARS-CoV-2 of 35 minutes. A little more variation can be seen in $CO_2$ than the viral load or relative humidity though. This suggests that relative humidity, $CO_2$ and viral load all share the same source (exhalation) whereas temperature, which does not follow the same trend, has a different predominant source (heat from the radiator). The spatial variation for the dual twin simulations can be seen in Figure 15, which shows an iso-surface of $CO_2$ for the value 1000 ppm at two times: at 100 s (or 2985 s) and at 500 s (or 3385 s). Three different cases are shown: the predictions of PredAAE (with no data assimilation), predictions of PredAAE with data assimilation (DA-PredAAE) and the original high-fidelity model. This again shows that, after data assimilation, the prediction agrees closely with the CFD model taken from the time interval that is consistent with the observations rather than the initial conditions.

#### 3.4.2. Scenario 2: assimilating observations from experiments

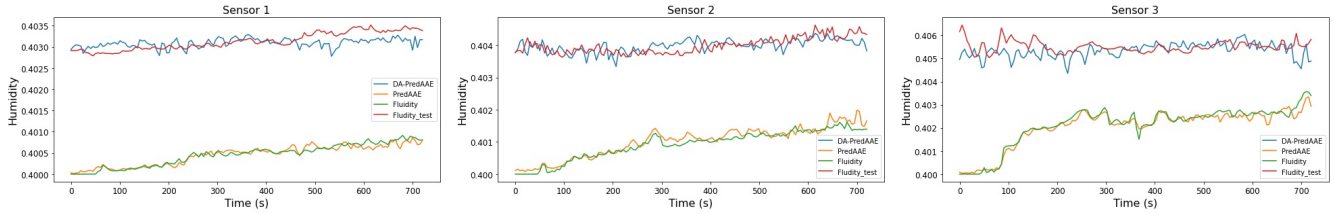
In this section, we assimilate observations collected with sensors from a classroom in Houndsfield Primary School into predictions of the surrogate model. Sensor values for $CO_2$, humidity and temperature are available, however, sensor values for the velocity fields were not measured and are taken from the high-fidelity model results instead. These values are required in the input (and output) of the AAE, but, by



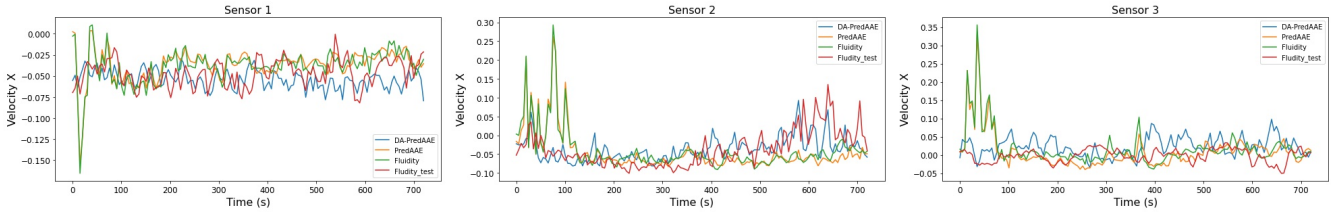

(a) $CO_2$



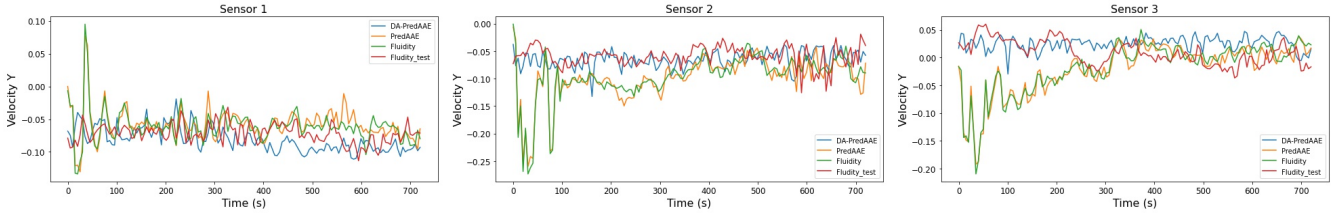

(b) Temperature



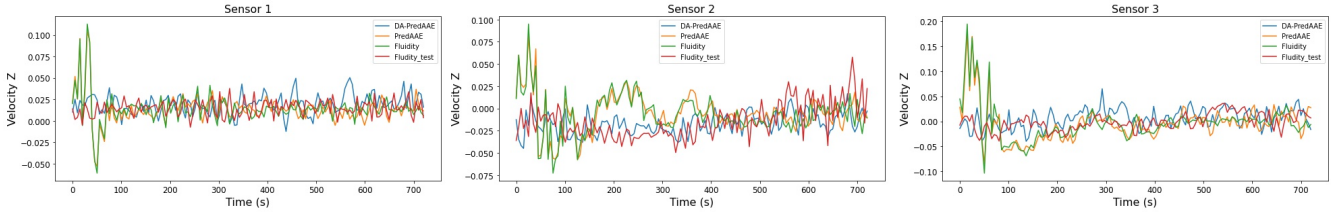

(c) Humidity




(d) $x$ component of velocity

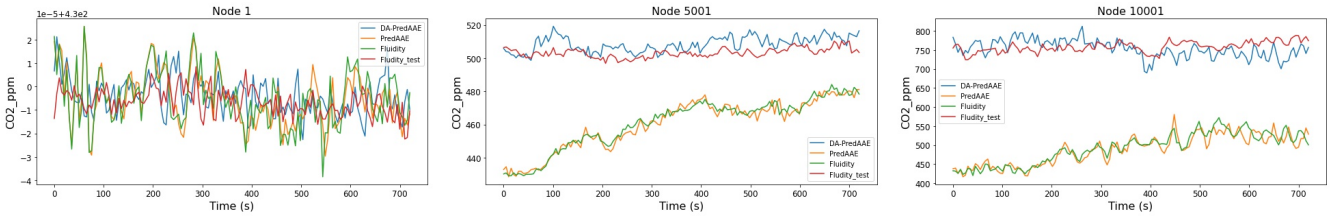



(e) $y$ component of velocity

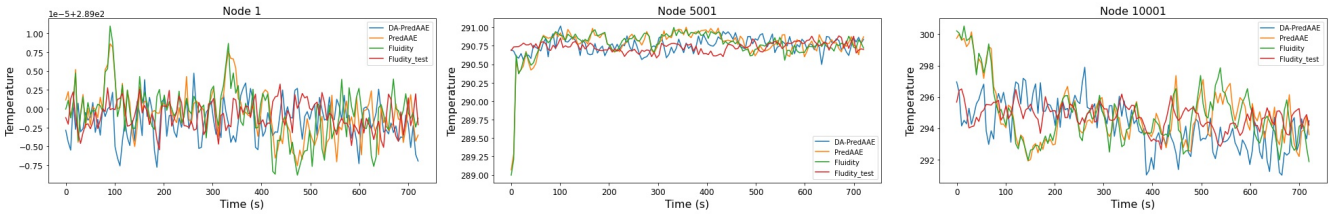



(f) $z$ component of velocity

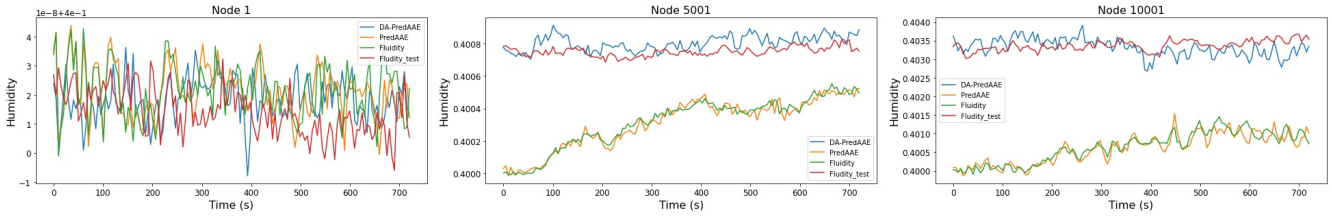
Figure 13: Time histories plotted at sensor locations for the dual twin experiment. The AAE is used to predict for time interval $[5, 720]$ seconds but given the sensor data from the time interval $[2885, 3600]$. The above plots show that after assimilating the sensor data the predictions of the AAE match closely those from $[2885, 3600]$.



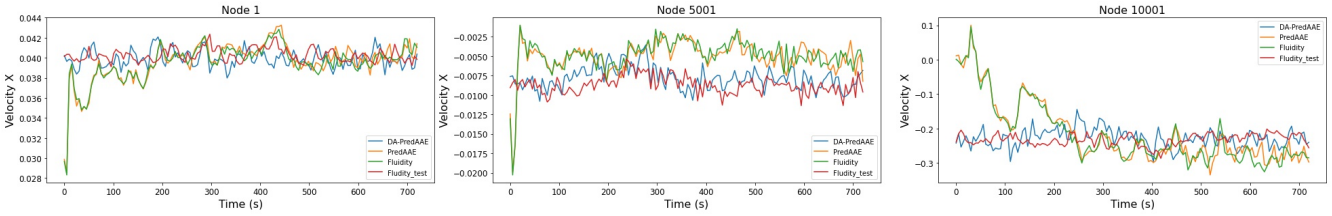

(a) $CO_2$



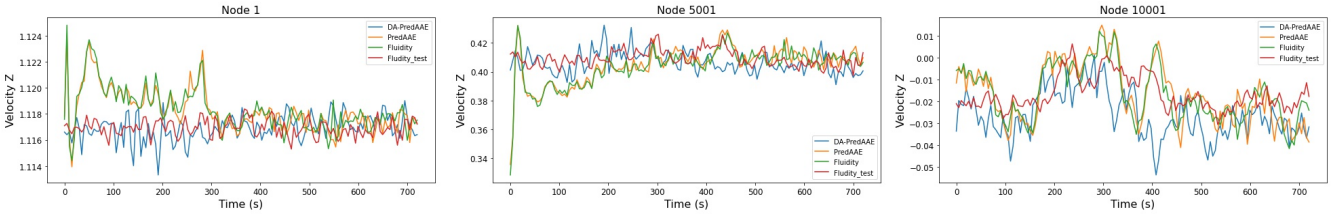

(b) Temperature



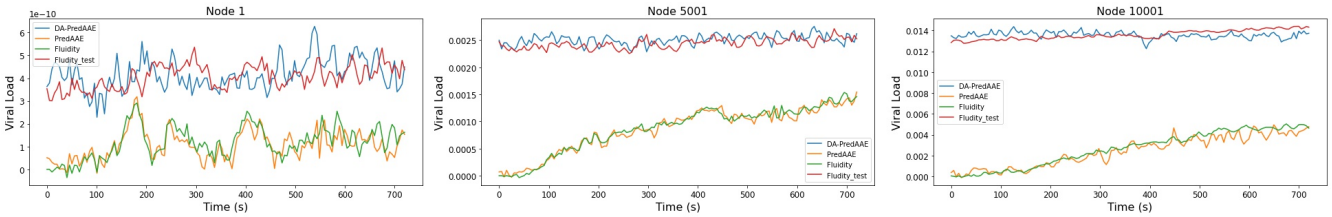

(c) Humidity




(d) $x$ component of velocity




(e) $z$ component of velocity




(f) Viral load

Figure 14: Time histories plotted at a number of nodal locations for the dual twin experiment (see Figure 6c). The AAE is used to predict for time interval $[5, 720]$ seconds but given the sensor data from the time interval $[2885, 3600]$. The above plots show that after assimilating the sensor data the predictions of the AAE match closely those from $[2885, 3600]$.

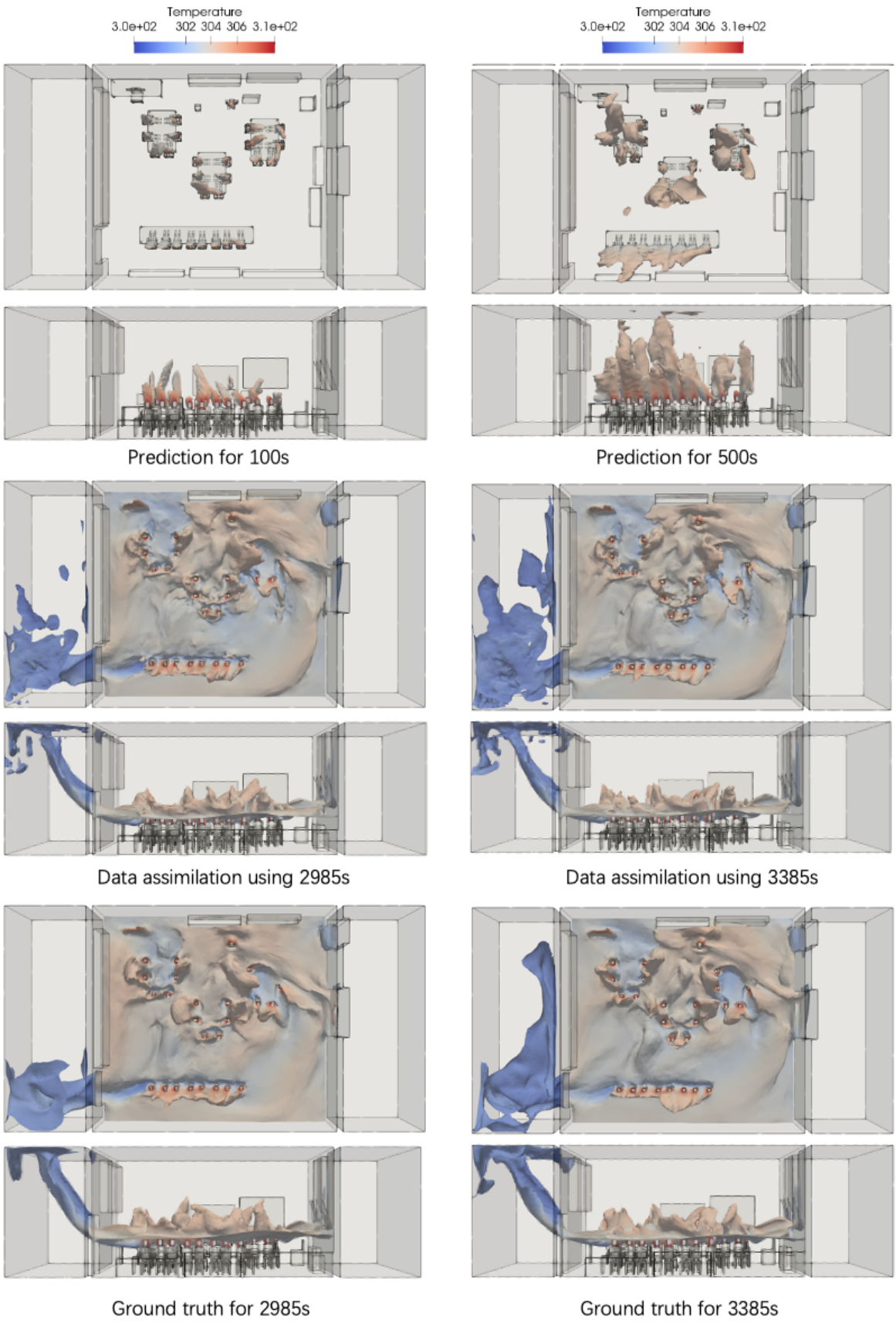
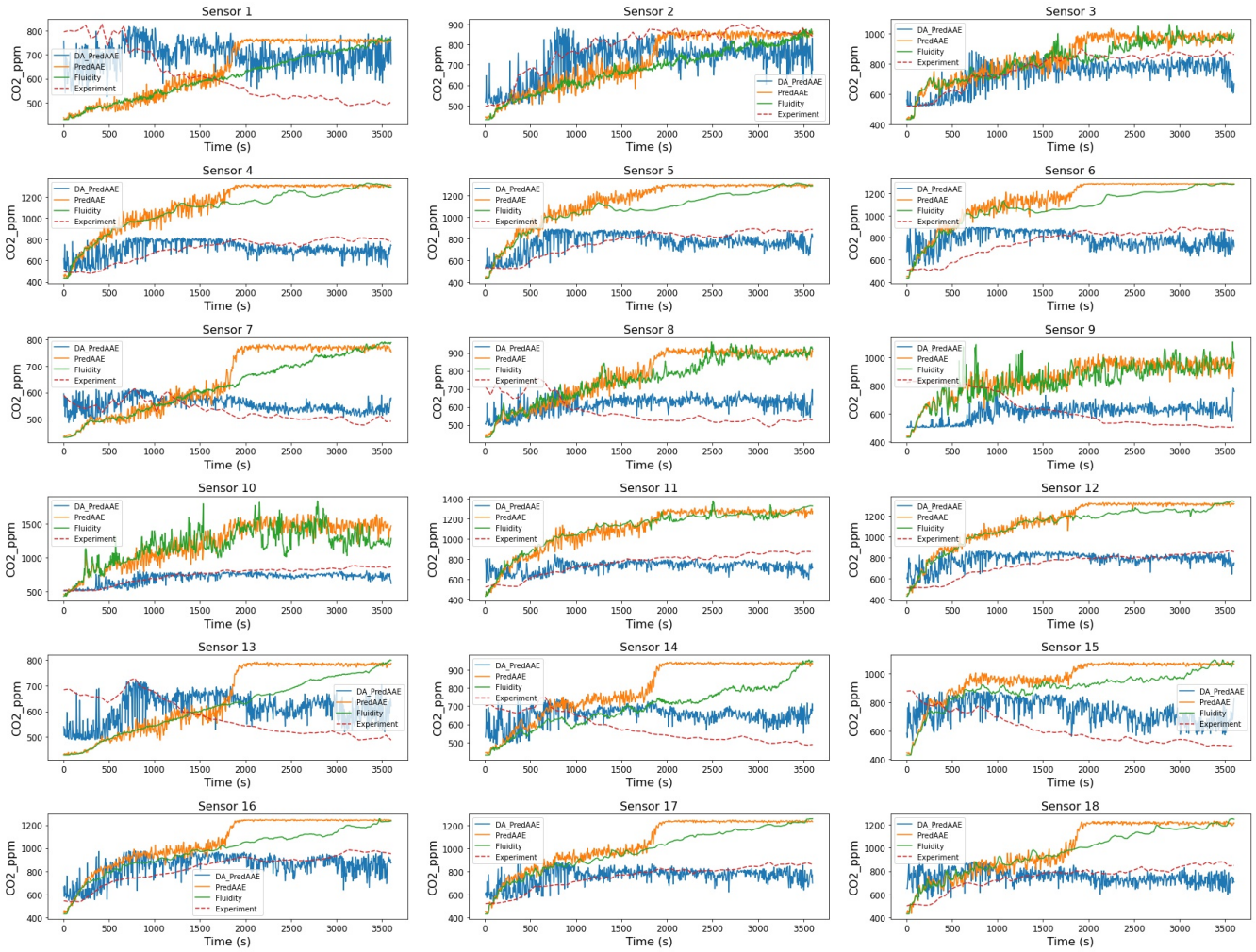



Figure 15: Spatial variation for the dual twin experiment at 100 s/2985 s (left) and 500 s/3385 s (right). An iso-surface of $CO_2$ at 1000 ppm is coloured by temperature. At two points in time (shown on the left and right), we compare the prediction of PredAAE (top plots) with results from data assimilation DA-PredAAE (centre) with the CFD results (bottom).

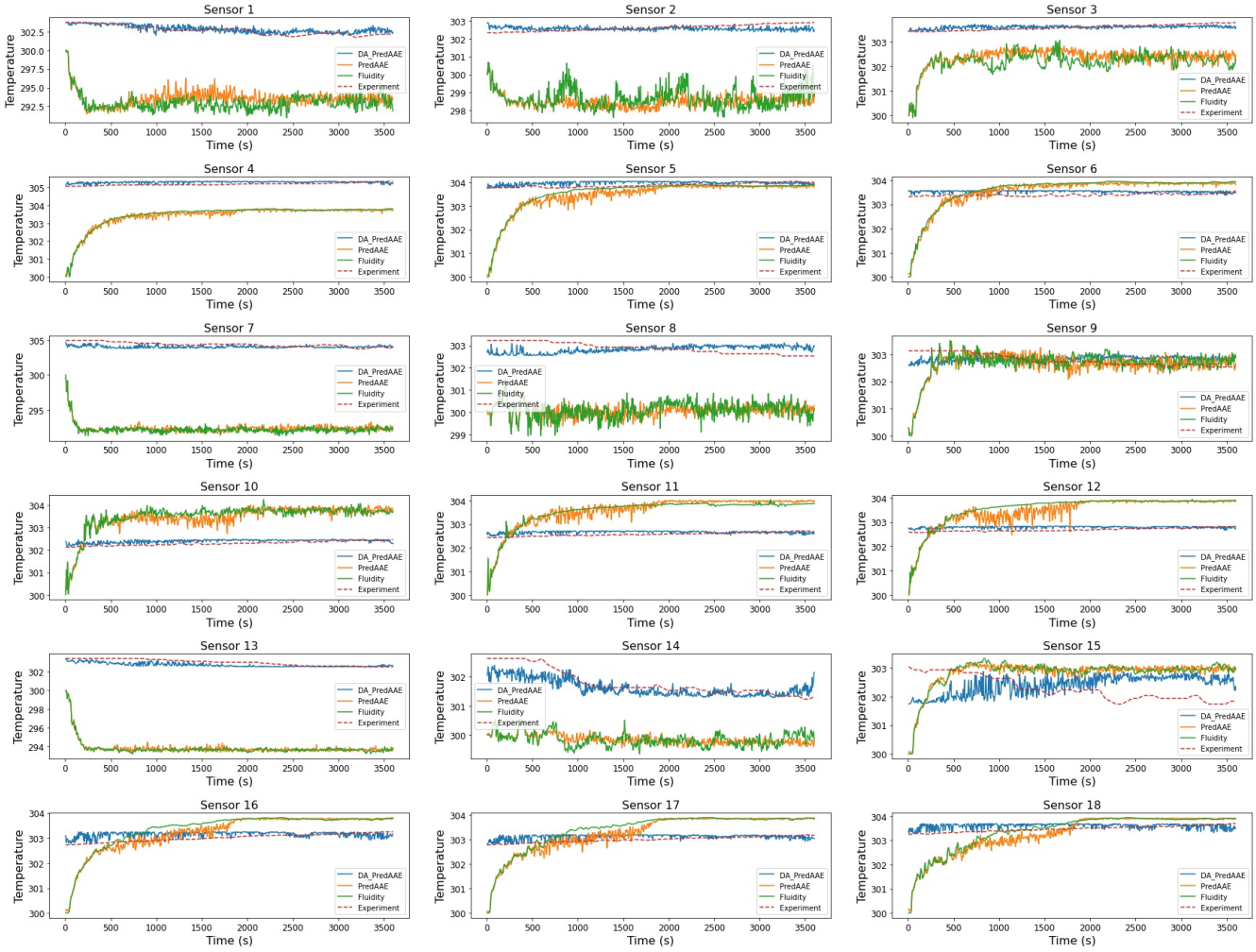
setting the prioritisation parameter to zero for these fields, are not assimilated. The algorithm we propose has the flexibility to switch on and off which variables are assimilated. The solutions resulting from the DA process might be assumed to be a good estimate of the fields within the classroom and an interpolation of the sensor data in space and time. Figure 16 shows the results from the high-fidelity model, the reduced-order model without DA (PredAAE), the reduced-order model with DA (DA-PredAAE) and the sensor values in time series plots, at the sensor locations. In all cases, the predictions of PredAAE are close to the results from the high-fidelity model. For $CO_2$, temperature and humidity, assimilating data (DA-PredAAE) successfully brings the model predictions closer to the sensor values, especially for $CO_2$ and temperature. Figure 17 plots time series predictions from the high-fidelity model, predictions from the reduced-order model without data assimilation (PredAAE) and predictions from the reduced-order model with data assimilation (DA-PredAAE), at a number of nodal locations. Here one can see that the predictions of PredAAE are close to the high-fidelity model, and that, after assimilating the data, the predictions deviate from the high-fidelity model, showing that the data assimilation as had an effect. Figure 18 shows the spatial variation at time 1000 s for the high-fidelity model, PredAAE and DA-PredAAE. Although we do not know the conditions associated with the sensor values from the school (e.g. wind direction), we can see from Figure 16 that after assimilation, the predictions are closer to the sensor values, and we expect that the predictions shown in Figure 18 are more consistent with the conditions in the classroom, given the previous results which demonstrate the ability of the proposed method to assimilate data (Section 3.4.1).

## 4. Conclusions

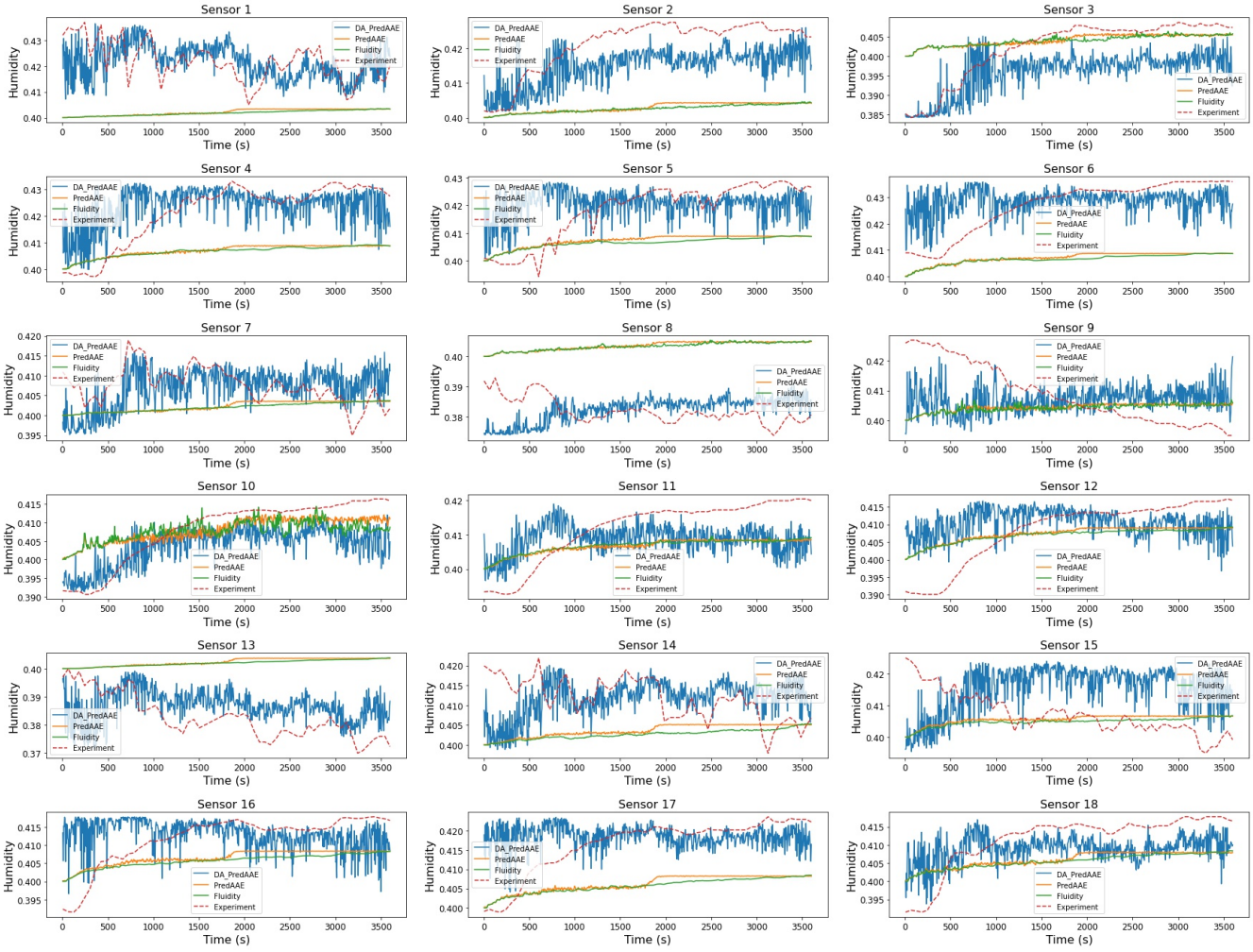
We have proposed a new method which assimilates data whilst marching forwards and backwards in time, as is done in 4D-Variational approaches. To make the method computationally tractable a surrogate model is used, which combines proper orthogonal decomposition, (POD) to obtain a low-dimensional space, and an adversarial autoencoder (AAE), to predict how the POD coefficients evolve in time. The input (and output) of the AAE consists of POD coefficients and sensor values, both over a sequence of $m + 1$ time levels. Two methods of predicting with the AAE were presented: the "constrained" method involves repeatedly passing solutions through the AAE until convergence is reached; the second "backpropagation" method uses a backpropagation or adjoint algorithm to find which latent variables produce the first $m$ values of the output. Once convergence is reached, the remaining values are taken to be the POD coefficients at the future time level. A school classroom is used as a test case here. A Computational Fluid Dynamics Large Eddy Simulation (CFD-LES) code is used to generate high-resolution solutions over the period of an hour, on which the surrogate model is based. The results were good for both the constrained method and the backpropagation method, although some deviation in the solution is seen towards the end of the one hour period for both surrogate models when compared with the CFD model. For the data assimilation, a type of dual-twin experiment was performed, in which the AAE was given POD coefficients from $m$ time levels starting from time level 1 and marched forward in time for 120 time levels whilst assimilating observations from the test dataset at the end of the hour time period (time levels 577 to 720). The algorithm is able to assimilate the data and predict outputs that are very close to the CFD results in the test dataset (despite starting with POD coefficients from time level 1). After this successful demonstration of the data assimilation algorithm, observations that had




(a) $CO_2$ time series at the 18 sensors

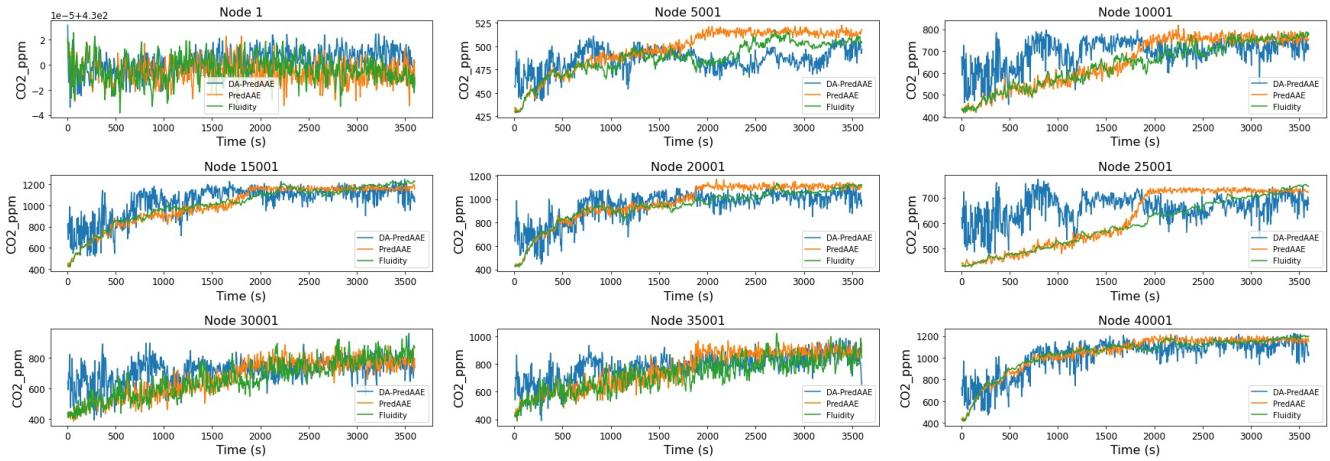



(b) Temperature time series at the 18 sensors

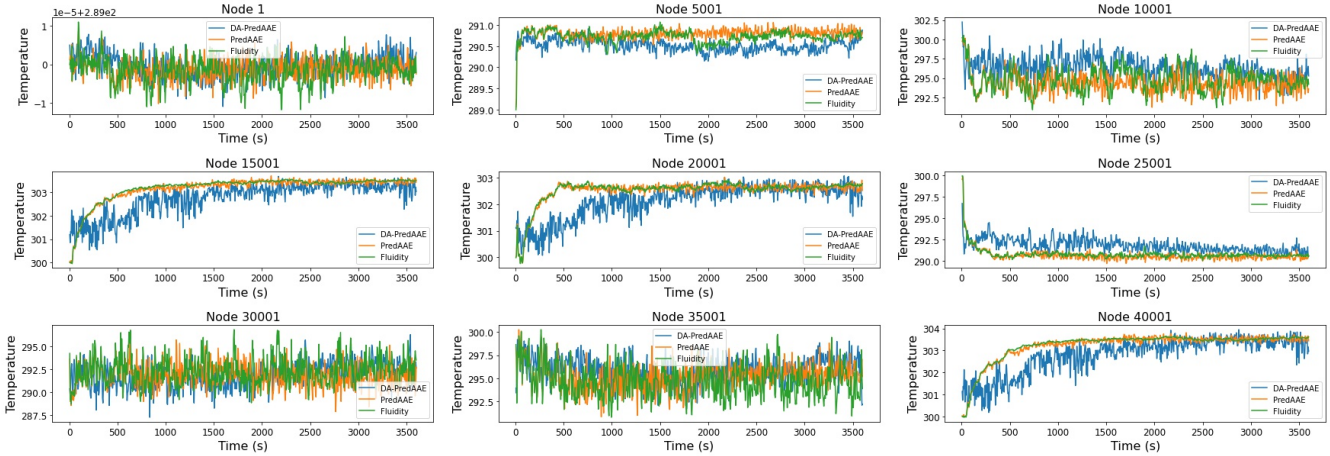



(c) Humidity time series at the 18 sensors

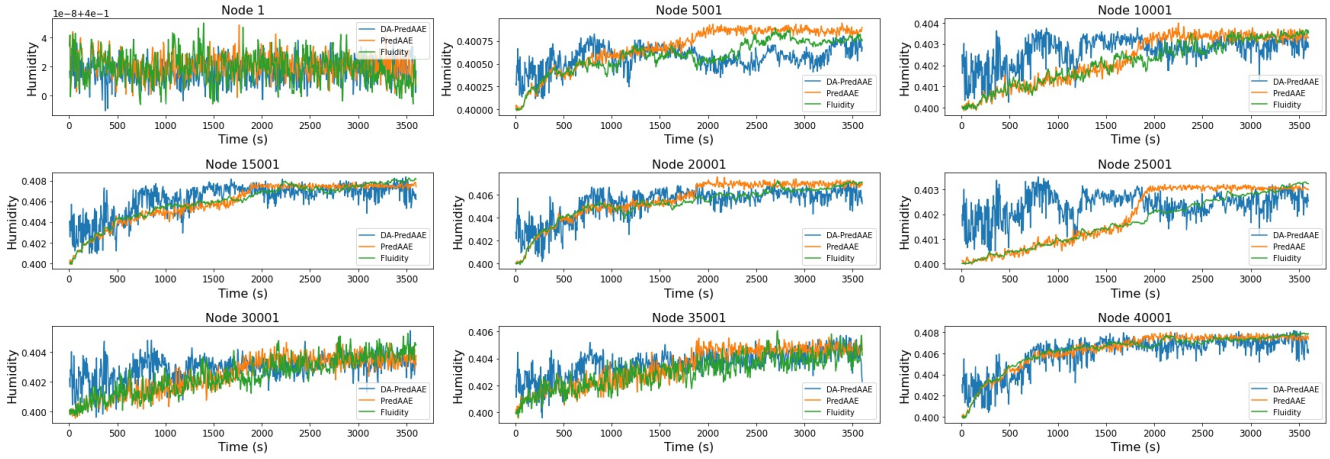
Figure 16: Assimilating data from the classroom in Houndsfield primary school. The above plots compare results from the high-fidelity model (green) with PredAAE predictions (orange), PredAAE prediction with data assimilation (blue) and the sensor observations (red). See Figure 6 for the location of the sensors.




(a) $CO_2$ time series




(b) Temperature time series




(c) Humidity time series

Figure 17: Assimilating data from the classroom in Houndsfield primary school. The above plots compare results from the high-fidelity model (green) with PredAAE predictions (orange) and PredAAE prediction with data assimilation (blue). The time series are taken at locations of various nodes — nodes 1, 5001, 10001, 15001, 20001, 25001, 30001, 35001 and 40001 (see Figure 6).

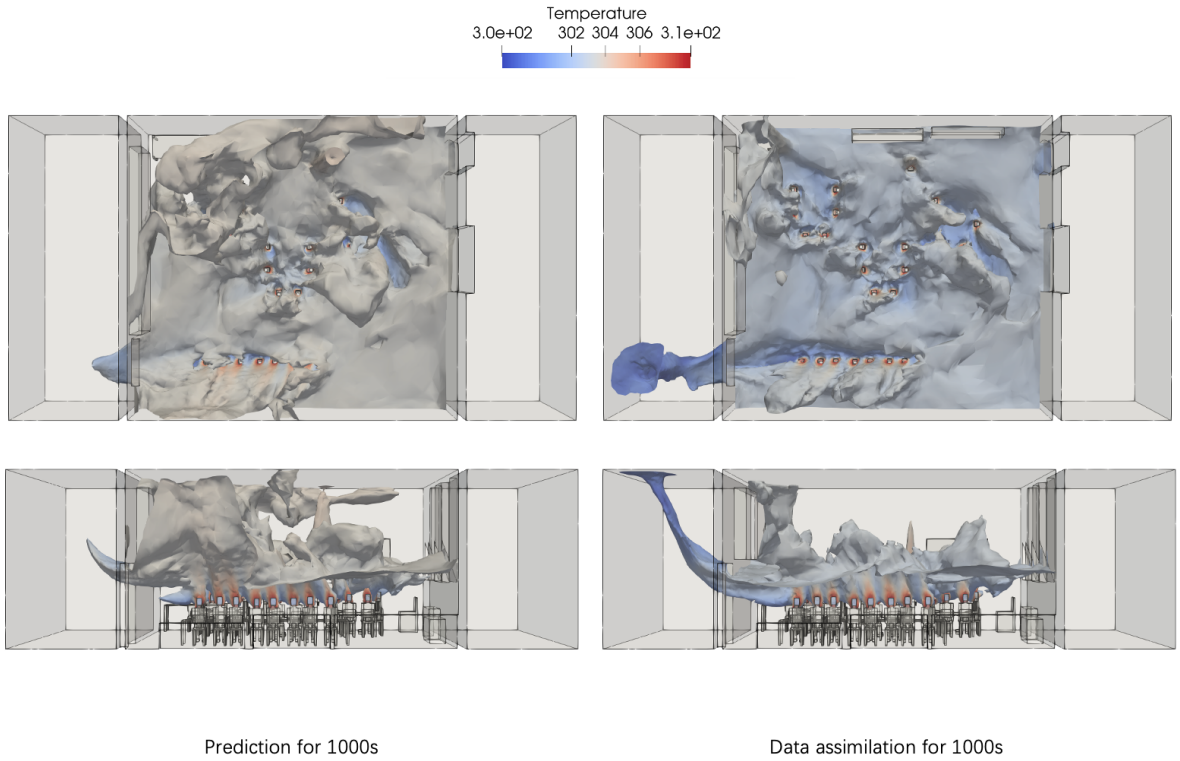



Figure 18: Assimilating data from the classroom in Houndsfield Primary School. Spatial variation of the iso-surface of $CO_2$ at 1000 ppm coloured by the temperature. The prediction without data assimilation is shown on the left at 1000 s and the prediction with data assimilation is shown on the right also at 1000 s.

been collected from a classroom in Houndsfield Primary School were assimilated and results presented.

## Acknowledgements

We would like to acknowledge the Applied Computational Science & Engineering MSc course and the Environmental Data Science and Machine Learning MSc course at Imperial College London. Seven of the co-authors were students on these courses and worked on projects related to the methods presented in this article. We also thank the University of Surrey's Global Centre for Clean Air Research (GCARE) team members Dr Arvind Tiwari, Nidhi Rawat, Dr Sarkawt Hama, Dr Gopinath Kalaiarasan and Dr Ana Paula Mendes Emygdio for collecting the classroom data [38]. We are grateful to Imperial College Research Computing Service (http://doi.org/10.14469/hpc/2232) for providing computational resources.

The authors would like to acknowledge the following EPSRC grants: RELIANT, Risk EvaLuatIon fAst iNtelligent Tool for COVID19 (EP/V036777/1); CO-TRACE, COvid-19 Transmission Risk Assessment Case Studies — education Establishments (EP/W001411/1); INHALE, Health assessment across biological length scales (EP/T003189/1); MAGIC, Managing Air for Green Inner Cities (EP/N010221/1); the PREMIERE programme grant (EP/T000414/1); and MUFFINS (EP/P033180/1).

## CRediT authorship contribution statement

**CEH:** methodology, software, writing (original draft, review and editing), supervision. **JT, JY:** software, writing (review and editing), analysis, visualisation. **DG, JI, SK, YL, DS:** software, writing (review and editing), analysis. **BC:** software, writing (original draft, review and editing), supervision. **LM:** writing (review and editing), supervision, software. **PK:** data acquisition, writing (review and editing), funding acquisition. **CCP:** conceptualisation, methodology, software, writing (review and editing), supervision, funding acquisition.

## Appendix A. Hyperparameters of the Adversarial Autoencoder

The hyperparameter settings used in the adversarial autoencoder are shown in Tables A.4 and A.5.

| | optimal values | grid search |
|---|---|---|
| latent space | 512 | 128, 256, 512, 1024 |
| encoder layers | 3 | 2, 3, 4, 5 |
| decoder layers | 2 | 2, 3, 4, 5 |
| kernel size | $(3, 3)$ | (2,2), (3,3), (5,5) |
| dropout rate | 0.2 | 0, 0.2, 0.3 |
| optimizer | Adam | Adam, Adamax, RMSProp, SGD |
| activation functions: | | LeakyReLU, ReLU, sigmoid |
| hidden layers in encoder and decoder | LeakyReLU | |
| hidden layers in discriminator | ReLU | |
| output layers | sigmoid | |
| epochs | 10,000 | 5,000, 10,000, 20,000 |
| learning rate: | | $10^{-3}$, $10^{-4}$, $4 \times 10^{-5}$, $10^{-5}$, $5 \times 10^{-5}$ |
| autoencoder | $10^{-4}$ | |
| encoder-discriminator | $5 \times 10^{-5}$ | |

Table A.4: Hyperparameter settings for the predictive AAE. The values over which a grid search was performed are on the right, the optimal values are on the left.

## References


[1] P. Bauer, A. Thorpe, G. Brunet, The quiet revolution of numerical weather prediction, Nature 525 (2015) 47–55. doi:10.1038/nature14956.

[2] E. Kalnay, Atmospheric Modeling, Data Assimilation and Predictability, Cambridge University Press, 2003. doi:10.1017/CBO9780511802270.

[3] D. G. Cacuci, I. M. Navon, M. Ionescu-Bujor, Computational methods for data evaluation and assimilation, Chapman and Hall (CRC), 2016.

| autoencoder | | discriminator | |
|---|---|---|---|
| input | $9 \times 258$ | | |
| conv2D | $9 \times 258 \times 64$ | | |
| conv2D | $5 \times 129 \times 32$ | | |
| conv2D | $5 \times 65 \times 16$ | | |
| reshape | 5200 | | |
| dense | 512 $\longrightarrow$ | input | 512 |
| dense | 5200 | dense | 256 |
| reshape | $5 \times 65 \times 16$ | dense | 128 |
| conv2D | $5 \times 65 \times 32$ | dense | 64 |
| conv2D | $5 \times 129 \times 64$ | output | 1 |
| output | $9 \times 258$ | | |

Table A.5: Architecture of the predictive AAE. The size of the 2D convolutional layers ('conv2D') written in the table gives the two dimensions of the feature maps, and the third number denotes the number of channels or filters used.


[4] M. Asch, M. Bocquet, M. Nodet, Data Assimilation: Methods, Algorithms and Applications, SIAM, 2016.

[5] R. N. Bannister, A review of operational methods of variational and ensemble-variational data assimilation, Quarterly Journal of the Royal Meteorological Society 143 (2017) 607–633. doi:10.1002/qj.2982.

[6] R. Maulik, V. Rao, J. Wang, G. Mengaldo, E. Constantinescu, B. Lusch, P. Balaprakash, I. Foster, R. Kotamarthi, Efficient high-dimensional variational data assimilation with machine-learned reduced-order models, Geoscientific Model Development 15 (2022) 3433–3445. doi:10.5194/gmd-15-3433-2022.

[7] S. Hatfield, M. Chantry, P. Dueben, P. Lopez, A. Geer, T. Palmer, Building Tangent-Linear and Adjoint Models for Data Assimilation With Neural Networks, Journal of Advances in Modeling Earth Systems 13 (2021) e2021MS002521. doi:10.1029/2021MS002521.

[8] A. J. Geer, Learning earth system models from observations: machine learning or data assimilation, Philosophical Transactions of the Royal Society A 379 (2021) 20200089. doi:10.1098/rsta.2020.0089.

[9] M. Bocquet, J. Brajard, A. Carrassi, L. Bertino, Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization, Foundations of Data Science 2 (2020) 55–80. doi:10.3934/fods.2020004.

[10] M. Sonnewald, R. Lguensat, D. C. Jones, P. D. Dueben, J. Brajard, V. Balaji, Bridging observations, theory and numerical simulation of the ocean using machine learning, Environmental Research Letters 17 (2021). doi:10.1088/1748-9326/ac0eb0.

[11] C. Buizza, C. Quilodrán Casas, P. Nadler, J. Mack, S. Marrone, Z. Titus, C. Le Cornec, E. Heylen, T. Dur, L. Baca Ruiz, C. Heaney, J. A. Daz Lopez, K. S. Kumar, R. Arcucci, Data Learning: Integrating Data Assimilation and Machine Learning, Journal of Computational Science 58 (2022) 101525. doi:10.1016/j.jocs.2021.101525.

[12] J. Brajard, A. Carrassi, M. Bocquet, 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 44 (202) 101171.

[13] A. Chennault, A. A. Popov, A. N. Subrahmanya, R. Cooper, A. Karpatne, A. Sandu, Adjoint-Matching Neural Network Surrogates for Fast 4D-Var Data Assimilation, arXiv preprint, arxiv:2111.08626 (2021). doi:10.48550/arxiv.2111.08626.

[14] C. Q. Casas, R. Arcucci, P. Wu, C. Pain, Y.-K. Guo, A Reduced Order Deep Data Assimilation model, Physica D: Nonlinear Phenomena 412 (2020) 132615. doi:10.1016/j.physd.2020.132615.

[15] M. E. Levine, A. M. Stuart, A framework for machine learning of model error in dynamical systems, arXiv preprint, arxiv:2107.06658 (2021). doi:10.48550/arXiv.2107.06658.

[16] J. Brajard, A. Carrassi, M. Bocquet, L. Bertino, Combining data assimilation and machine learning to infer unresolved scale parametrization, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 379 (2021) 20200086. doi:10.1098/rsta.2020.0086.

[17] A. Farchi, P. Laloyaux, M. Bonavita, M. Bocquet, Using machine learning to correct model error in data assimilation and forecast applications, Quarterly Journal of the Royal Meteorological Society 147 (2021) 3067–3084. doi:10.1002/qj.4116.

[18] M. R. Ebers, K. M. Steele, J. N. Kutz, Discrepancy Modeling Framework: Learning missing physics, modeling systematic residuals, and disambiguating between deterministic and random effects, arXiv preprint, arXiv:2203.05164 (2022). doi:10.48550/arxiv.2203.05164.

[19] S. Pawar, O. San, Equation-free surrogate modeling of geophysical flows at the intersection of machine learning and data assimilation, arXiv preprint, arxiv:2205.13410 (2022). doi:10.48550/arxiv.2205.13410.

[20] M. Amendola, R. Arcucci, L. Mottet, C. Q. Casas, S. Fan, C. Pain, P. Linden, Y.-K. Guo, Data Assimilation in the Latent Space of a Convolutional Autoencoder, in: M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, P. M. Sloot (Eds.), Computational Science – ICCS 2021, 2021, pp. 373–386.

[21] M. Peyron, A. Fillion, S. Gürol, V. Marchais, S. Gratton, P. Boudier, G. Goret, Latent space data assimilation by using deep learning, Quarterly Journal of the Royal Meteorological Society 147 (2021) 3759–3777. doi:10.1002/qj.4153.

[22] R. S. Cintra, H. F. de Campos Velho, Data Assimilation by Artificial Neural Networks for an Atmospheric General Circulation Model, in: A. El-Shahat (Ed.), Advanced Applications for Artificial Neural Networks, IntechOpen, Rijeka, 2018. doi:10.5772/intechopen.70791.

[23] F. P. Härter, H. F. d. C. Velho, Multilayer Perceptron Neural Network in a Data Assimilation Scenario, Engineering Applications of Computational Fluid Mechanics 4 (2010) 237–245. doi:10.1080/19942060.2010.11015313.

[24] S. Pawar, O. San, A. Rasheed, I. M. Navon, A nonintrusive hybrid neural-physics modeling of incomplete dynamical systems: Lorenz equations, International Journal on Geomathematics 12 (2021) 17. doi:10.1007/s13137-021-00185-z.

[25] M. H. Rammay, S. Alyaev, A. H. Elsheikh, Probabilistic model-error assessment of deep learning proxies: an application to real-time inversion of borehole electromagnetic measurements, Geophysical Journal International 230 (2022) 1800–1817. doi:10.1093/gji/ggac147.

[26] C. López, E. Hernández-García, Low-dimensional dynamical system model for observed coherent structures in ocean satellite data, Physica A: Statistical Mechanics and its Applications 328 (2003) 233–250. doi:10.1016/S0378-4371(03)00505-3.

[27] Y. Cao, J. Zhu, I. M. Navon, Z. Luo, A reduced-order approach to four-dimensional variational data assimilation using proper orthogonal decomposition, International Journal for Numerical Methods in Fluids 53 (2007) 1571–1583. doi:10.1002/fld.1365.

[28] D. N. Daescu, I. M. Navon, Efficiency of a POD-based reduced second-order adjoint model in 4D-Var data assimilation, International Journal for Numerical Methods in Fluids 53 (2007) 985–1004. doi:10.1002/fld.1316.

[29] H. Gong, S. Cheng, Z. Chen, Q. Li, C. Quilodrn-Casas, D. Xiao, R. Arcucci, An efficient digital twin based on machine learning SVD autoencoder and generalised latent assimilation for nuclear reactor physics, Annals of Nuclear Energy 179 (2022) 109431. doi:10.1016/j.anucene.2022.109431.

[30] V. L. Silva, C. E. Heaney, Y. Li, C. C. Pain, Data Assimilation Predictive GAN (DA-PredGAN): applied to a spatio-temporal compartmental model in epidemiology, Journal of Scientific Computing 94 (2021) 1–31. doi:10.1007/s10915-022-02078-1.

[31] F. Regazzoni, D. Chapelle, P. Moireau, Combining data assimilation and machine learning to build data-driven models for unknown long time dynamics—Applications in cardiovascular modeling, International Journal for Numerical Methods in Biomedical Engineering 37 (2021) e3471. doi:10.1002/cnm.3471.

[32] C. Quilodrán-Casas, R. Arcucci, L. Mottet, Y.-K. Guo, C. C. Pain, Adversarial autoencoders and adversarial LSTM for improved forecasts of urban air pollution simulations (2021). arXiv:2104.06297.

[33] R. Maulik, B. Lusch, P. Balaprakash, Non-autoregressive time-series methods for stable parametric reduced-order models, Physics of Fluids 32 (2020) 087115. doi:10.1063/5.0019884.

[34] S. Fresca, A. Manzoni, POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, Computer Methods in Applied Mechanics and Engineering 388 (2022) 114181. doi:10.1016/j.cma.2021.114181.

[35] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, Adversarial autoencoders, arXiv preprint, arxiv:1511.05644 (2015). doi:10.48550/arXiv.1511.05644.

[36] C. E. Heaney, Z. Wolffs, J. A. Tómasson, L. Kahouadji, P. Salinas, A. Nicolle, I. M. Navon, O. K. Matar, N. Srinil, C. C. Pain, An AI-based non-intrusive reduced-order model for extended domains applied to multiphase flow in pipes, Physics of Fluids 34 (2022) 055111. doi:10.1063/5.0088070.

[37] C. E. Heaney, X. Liu, H. Go, Z. Wolffs, P. Salinas, I. M. Navon, C. C. Pain, Extending the Capabilities of Data-Driven Reduced-Order Models to Make Predictions for Unseen Scenarios: Applied to Flow Around Buildings, Frontiers in Physics 10 (2022) 910381. doi:10.3389/fphy.2022.910381.

[38] P. Kumar, N. Rawat, A. Tiwari, Micro-characteristics of a naturally ventilated classroom air quality under varying air purifier placements, Environmental Research (2022) 114849. doi:10.1016/j.envres.2022.114849.

[39] R. K. Bhagat, M. S. Davies Wykes, S. B. Dalziel, P. F. Linden, Effects of ventilation on the indoor spread of COVID-19, Journal of Fluid Mechanics 903 (2020) F1. doi:10.1017/jfm.2020.720.

[40] P. Dabisch, M. Schuit, A. Herzog, K. Beck, S. Wood, M. Krause, D. Miller, W. Weaver, D. Freeburger, I. Hooper, B. Green, G. Williams, B. Holland, J. Bohannon, V. Wahl, J. Yolitz, M. Hevey, S. Ratnesar-Shumate, The influence of temperature, humidity, and simulated sunlight on the infectivity of SARS-CoV-2 in aerosols, Aerosol Science and Technology 55 (2021) 142–153. doi:10.1080/02786826.2020.1829536.

[41] P. Holmes, J. Lumley, G. Berkooz, C. Rowley, Turbulence, Coherent Structures, Dynamical Systems and Symmetry, Cambridge University Press, 2012.

[42] Applied Modelling and Computation Group, Imperial College London, Fluidity repository, https://fluidityproject.github.io/, 2016. Accessed 2 October 2022.

[43] Fluidity manual, 2016. https://fluidityproject.github.io/, accessed 2 October 2022.

[44] C. Pain, A. Umpleby, C. de Oliveira, A. Goddard, Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations, Computer Methods in Applied Mechanics and Engineering 190 (2001) 3771–3796. doi:10.1016/S0045-7825(00)00294-2.

[45] D. Pavlidis, G. J. Gorman, J. L. M. A. Gomes, C. C. Pain, H. ApSimon, Synthetic-Eddy Method for Urban Atmospheric Flow Modelling, Boundary-Layer Meteorology 136 (2010) 285–299. doi:10.1007/s10546-010-9508-x.

[46] H. Woodward, M. Stettler, D. Pavlidis, E. Aristodemou, H. ApSimon, C. Pain, A large eddy simulation of the dispersion of traffic emissions by moving vehicles at an intersection, Atmospheric Environment 215 (2019) 116891. doi:10.1016/j.atmosenv.2019.116891.

[47] L. Mottet, J. Song, A. C. Short, S. Chen, J. Wu, W. Yu, J. Xiong, Q. Zhang, J. Ge, M. Liu, R. Yao, B. Li, The hot summer-cold winter region in China: Challenges in the low carbon adaptation of residential slab buildings to enhance comfort, Energy and Buildings 223 (2020) 110181. doi:10.1016/j.enbuild.2020.110181.

[48] L. Mottet, Indoor Geometry Generator Manual, https://www.researchgate.net/publication/349350000_Indoor_Geometry_Generator_IGG_Manual, 2021. Accessed 2 October 2022.

[49] C. E. Heaney, Y. Li, O. K. Matar, C. C. Pain, Applying Convolutional Neural Networks to Data on Unstructured Meshes with Space-Filling Curves, arXiv preprint, arxiv:2011.14820 (2020). doi:10.48550/arxiv.2011.14820.

[50] Y. Zhou, C. Wu, Z. Li, C. Cao, Y. Ye, J. Saragih, H. Li, Y. Sheikh, Fully convolutional mesh autoencoder using efficient spatially varying kernels, in: H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, 2020, pp. 9251–9262. URL: https://proceedings.neurips.cc/paper/2020/file/68dd09b9ff11f0df5624a690fe0f6729-Paper.pdf.

[51] J. Tencer, K. Potter, A Tailored Convolutional Neural Network for Nonlinear Manifold Learning of Computational Physics Data Using Unstructured Spatial Discretizations, SIAM Journal on Scientific Computing 43 (2021) A2581–A2613. doi:10.1137/20M1344263.

[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. doi:10.5281/zenodo.4724125, software available from tensorflow.org.