Online Model Error Correction With Neural Networks in the Incremental 4D-Var Framework

Alban Farchi1, Marcin Chrust2, Marc Bocquet1, Patrick Laloyaux2, and Massimo Bonavita3

1CERA, École des Ponts and EDF R&D, Île-de-France, France, 2ECMW, Reading, UK

Abstract Recent studies have demonstrated that it is possible to combine machine learning with data assimilation to reconstruct the dynamics of a physical model partially and imperfectly observed. The surrogate model can be defined as an hybrid combination where a physical model based on prior knowledge is enhanced with a statistical model estimated by a neural network (NN). The training of the NN is typically done offline, once a large enough data set of model state estimates is available. By contrast, with online approaches the surrogate model is improved each time a new system state estimate is computed. Online approaches naturally fit the sequential framework encountered in geosciences where new observations become available with time. In a recent methodology paper, we have developed a new weak-constraint 4D-Var formulation which can be used to train a NN for online model error correction. In the present article, we develop a simplified version of that method, in the incremental 4D-Var framework adopted by most operational weather centers. The simplified method is implemented in the European Center for Medium-Range Weather Forecasts (ECMW) Object-Oriented Prediction System, with the help of a newly developed Fortran NN library, and tested with a two-layer two-dimensional quasi geostrophic model. The results confirm that online learning is effective and yields a more accurate model error correction than offline learning. Finally, the simplified method is compatible with future applications to state-of-the-art models such as the ECMWF Integrated Forecasting System.

Plain Language Summary We have recently proposed a general framework for combining data assimilation (DA) and machine learning (ML) techniques to train a neural network for online model error correction. In the present article, we develop a simplified version of this online training method, compatible with future applications to more realistic models. Using numerical illustrations, we show that the new method is effective and yields a more accurate model error correction than the usual offline learning approach. The results show the potential of incorporating DA and ML tightly, and pave the way toward an application to the Integrated Forecasting System used for operational numerical weather prediction at the European Centre for Medium-Range Weather Forecasts.

1. Introduction: Machine Learning for Model Error Correction

In the geosciences, data assimilation (DA) is used to increase the quality of forecasts by providing accurate initial conditions (Asch et al., 2016; Carissi et al., 2018; Evensen et al., 2022; Kalnay, 2003; Law et al., 2015; Reich & Cotter, 2015). The initial conditions are obtained by combining all sources of information in a mathematically optimal way, in particular information from the dynamical model and information from sparse and noisy observations. There are two main classes of DA methods. In variational DA, the core of the methods is to minimize a cost function, usually using gradient-based optimization techniques, to estimate the system state. Examples include 3D- and 4D-Var. In statistical DA, the methods relies on the sampled error statistics to perform sequential updates to the state estimation. The most popular examples are the ensemble Kalman filter (EnKF) and the particle filter.

Most of the time, DA methods are applied with the perfect model assumption: this is called strong-constraint DA. However, despite the significant effort provided by the modelers, geoscientific models remain affected by errors (Dee, 2005), for example, due to unresolved small-scale processes. This is why there is a growing interest of the DA community in weak-constraint (WC) methods, that is, DA methods relaxing the perfect model assumption (Trémolet, 2006). This has led, for example, to the iterative EnKF in the presence of additive noise (Sakov et al., 2018) in statistical DA, and to the forcing formulation of WC 4D-Var (Laloyaux et al., 2020a) in variational DA. In practice, the DA control vector has to be extended to include the model error in addition to the system state. The downside of this approach is the potentially significant increase of the problem’s dimension since the
model trajectory is not anymore described uniquely by the initial condition. Generic definitions of model error can be found, for example, in Harlim et al. (2021) and Levine and Stuart (2022). In the present work, we do not make any specific assumption on the model error. This means that, following the classification of Levine and Stuart (2022), we consider the most general case of non-Markovian model error.

In parallel, following the renewed impetus of machine learning (ML) applications (Chollet, 2018; Goodfellow et al., 2016; LeCun et al., 2015), data-driven approaches are more and more frequent in the geosciences. The goal of these approaches (e.g., Arcomano et al., 2020; Brunton et al., 2016; Dueben & Bauer, 2018; Fablet et al., 2018; Hamilton et al., 2016; Lguensat et al., 2017; Pathak et al., 2018a; Scher & Messori, 2019; Weyn et al., 2019, among many others) is to learn a surrogate of the dynamical model using supervised learning, that is, by minimizing a loss function which measures the discrepancy between the surrogate model predictions and an observation data set. In order to take into account sparse and noisy observations, ML techniques can be combined with DA (Abarbanel et al., 2018; Arcucci et al., 2021; Bocquet et al., 2019, 2020; Brajard et al., 2020). The idea is to take the best of both worlds: DA techniques are used to estimate the state of the system from the observations, and ML techniques are used to estimate the surrogate model from the estimated state. In practice, the hybrid DA and ML methods can be used both for full model emulation (e.g., in the previous references) and model error correction (Bolton & Zanna, 2019; Bonavita & Laloyaux, 2020; Brajard et al., 2021; Chen et al., 2022; Farchi, Bocquet, et al., 2021; Farchi, Laloyaux, et al., 2021; Gagne et al., 2020; Jia et al., 2019; Pathak et al., 2018b; Rasp et al., 2018; Watson, 2019; Wikner et al., 2020). In the first case, the surrogate model is entirely learned from observations, while in the latter case, the surrogate model is hybrid: a physical, knowledge-based model is corrected by a statistical model, for example, a neural network (NN), which is learned from observations. Even though from a technical point of view it can arguably be more difficult to implement, model error correction has many advantages over full model emulation: by leveraging the long history of numerical modeling, one can hope to end up with an easier learning problem (Farchi, Laloyaux, et al., 2021; Watson, 2019).

Most of the current hybrid DA-ML methods use disjoint, non-sequential (in time) learning strategies: disjoint because the system state and the surrogate model (or model error correction) are estimated separately, and “non-sequential” because the surrogate model is estimated only once, once the entire training data set is available. There are two main reasons for these choices. First, separating the system state and the surrogate model estimations means that it is possible to leverage the full potential of the ML variational tools to estimate the surrogate model. Second, surrogate modeling requires a large amount of data to provide accurate results, therefore it makes sense to wait for as much as data as possible before starting the surrogate model estimation. Nevertheless, joint and/or sequential learning has in paper several advantages over disjoint, non-sequential learning:

- Sequential learning fits the standard sequential DA approach in the geosciences. Each time a new batch of observations becomes available, the surrogate model parameters can be corrected.
- Joint estimation is in general more consistent, and hence potentially more accurate, than separate estimation.
- With non-sequential learning, the training only starts once a sufficiently large data set is available. With sequential learning, the training begins from the first batch of observations, which means that improvements can be expected before having a large data set.
- With sequential learning, since the surrogate model is constantly updated, it can adapt to new (previously unseen) conditions. An example could be, in the case of model error correction, an update of the physical model to correct. Another example could be slowly varying effects on the dynamics (e.g., seasonality).

In this article, offline learning refers to disjoint, non-sequential learning and online learning refers to joint, sequential learning. For example, with this definition, WC 4D-Var is an online model error correction method.

Fundamentally, online learning is very similar to parameter estimation in DA: the goal is to estimate at the same time the system state and some parameters—in this case the surrogate model parameters. Several example of online learning methods have recently emerged. Bocquet et al. (2021) and Malartic et al. (2022) have developed several variants of the EnKF to perform a joint estimation of the state and the parameters of surrogate model which fully emulates the dynamics. Gottwald and Reich (2021) have used a very similar approach for the parameters of an echo state network used as surrogate model. Finally, Farchi, Bocquet, et al. (2021) have derived a variant of WC 4D-Var to perform a joint estimation of the state and the parameters of a NN which corrects the tendencies of a physical model. In this article, we revisit the method of Farchi, Bocquet, et al. (2021). A simplified method is derived, compatible with future applications to more realistic models. The method is implemented in the Object-Oriented Prediction System (OOPS) framework developed at the European Center for Medium-Range
Weather Forecasts (ECMWF), and tested using the two-layer quasi-geostrophic (QG) channel model developed in OOPS. To us, this is a final step before considering an application with the Integrated Forecasting System (IFS, Bonavita et al., 2017), since the IFS will soon rely on OOPS for its DA part.

The article is organized as follows. Section 2 presents the methodology. The QG model is described in Section 3. The experimental results are then presented in Section 4 for offline learning, and in Section 5 for online learning. Finally, conclusions are given in Section 6.

2. A Simplified Incremental Neural Network Variant of Weak-Constraint 4D-Var

2.1. Strong-Constraint 4D-Var

Suppose that we follow the evolution of a system using a series of observations taken at discrete times. In the classical 4D-Var, the observations are gathered into time windows \((y_0, \ldots, y_L)\). The integer \(L \geq 0\) is the window length, and \(y_k \in \mathbb{R}^{N_y}\), the \(k\)th batch of observations, contains all the observations taken at time \(t_k\), for \(k = 0, \ldots, L\). For convenience, we assume that the time interval between consecutive observation batches \(t_{k+1} - t_k = \Delta t\) is constant. This assumption is not fundamental; it just makes the presentation much easier. Within the window, the system state \(x_k \in \mathbb{R}^{N_x}\) at time \(t_k\) is obtained by integrating the model in time from \(t_0\) to \(t_k\):

\[
x_k = \mathcal{M}_{k,0} (x_0),
\]

where \(\mathcal{M}_{k,0} : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}\) is the resolvent of the dynamical (or physical) model from \(t_0\) to \(t_k\). Moreover, the observations are related to the state by the observation operator \(H_k : \mathbb{R}^{N_x} \to \mathbb{R}^{N_y}\) via

\[
y_k = H_k(x_k) + v_k,
\]

where \(v_k\) is the observation error at time \(t_k\), which could be a random vector. Let us make the assumption that the observation errors are independent from each other.

The 4D-Var cost function is defined as the negative log-likelihood:

\[
J^c(x_0) \triangleq - \ln p(x_0|y_0, \ldots, y_L),
\]

\[
\alpha = - \ln p(x_0) - \ln p(y_0, \ldots, y_L|x_0),
\]

\[
\alpha = - \ln p(x_0) - \sum_{k=0}^{L} \ln p(y_k|x_0).
\]

where conditional independence of the observation vectors on \(x_0\) was used. The background \(p(x_0)\) is Gaussian with mean \(x_0^b\) and covariance matrix \(B\), and the observation errors \(v_k\) are also Gaussian distributed with mean \(0\) and covariance matrices \(R_v\), in such a way that \(J^c\) becomes:

\[
J^c(x_0) = \frac{1}{2} \|x_0 - x_0^b\|^2_B + \frac{1}{2} \sum_{k=0}^{L} \|y_k - H_k \circ \mathcal{M}_{k,0}(x_0)\|^2_R,
\]

where we have dropped the constant terms and where the notation \(\|\cdot\|^2_M\) stands for the squared Mahalanobis norm \(v^\top M v\).

This formulation is called strong-constraint 4D-Var because it relies on the perfect model assumption Equation 1. In practice, Equation 4 is minimized using scalable gradient-based optimization methods to provide the analysis \(x_0^c\). In cycled DA, the model is then used to propagate \(x_0^c\) till the start of the next window, yielding thus a value for the background state \(x_0^b\).

2.2. Weak-Constraint 4D-Var

2.2.1. Generic State Formulation

Recognizing that the model is not perfect, we can replace the strong constraint Equation 1 by the more general model evolution
\[ x_{k+1} = \mathcal{M}_{k+1:1}(x_k) + w_k, \] (5)

where \( w_k \in \mathbb{R}^{N_t} \) is the model error from \( t_k \) to \( t_{k+1} \), potentially random. Let us make the assumption that the model errors are independent from each other and independent from the background errors. This implies that the model evolution satisfies the Markov property.

The updated cost function now depends on all states inside the window:

\[ J^{wc}(x_0, \ldots, x_L) \equiv -\ln p(x_0, \ldots, x_L | y_0, \ldots, y_L), \] (6a)
\[ \propto -\ln p(x_0, \ldots, x_L) - \ln p(y_0, \ldots, y_L | x_0, \ldots, x_L), \] (6b)
\[ \propto -\ln p(x_0) - \sum_{k=0}^{L-1} \ln p(x_{k+1} | x_k) - \sum_{k=0}^{L} \ln p(y_k | x_k). \] (6c)

With the Gaussian assumptions of Section 2.1 and the additional hypothesis that the model errors \( w_k \) also follow a Gaussian distribution with mean \( w^b_k \) and covariance matrices \( Q_k \), \( J^{wc} \) becomes

\[ J^{wc}(x_0, \ldots, x_L) = \frac{1}{2} \| x_0 - x^b_0 \|^2_{B^{-1}} + \frac{1}{2} \sum_{k=0}^{L} \| x_{k+1} - \mathcal{M}_{k+1:1}(x_k) - w^k \|^2_{Q_k^{-1}} + \frac{1}{2} \sum_{k=0}^{L} \| y_k - H_k(x_k) \|^2_{R_k}, \] (7)

where we have once again dropped the constant terms. This formulation is called WC 4D-Var (Trémolet, 2006) because it relaxes the perfect model assumption Equation 1, which means that the analysis \((x^a_0, \ldots, x^a_L)\) is not any more a trajectory of the model. However, this comes at a price: the dimension of the problem has increased from \( N_t \) to \( L N_t \).

2.2.2. Constant Forcing Formulation

This dimensionality increase can be mitigated by making additional assumptions. For example, one can assume that the model error is constant throughout the window, that is,

\[
\begin{align*}
    w_0 &= \ldots = w_{L-1} \triangleq w, \\
    w^b_0 &= \ldots = w^b_{L-1} \triangleq w^b, \\
    Q_0 &= \ldots = Q_{L-1} \triangleq LQ.
\end{align*}
\] (8a–8c)

In this case, the trajectory \((x_0, \ldots, x_L)\) is fully determined by \((w, x_0)\):

\[ x_k = \mathcal{M}_{k+1:1}(x_k) + w = \mathcal{M}_{k+1:1}(\mathcal{M}_{k:1}(x_{k-1}) + w) + w = \ldots \triangleq \mathcal{M}^{wc}_{k+1:0}(w, x_0), \] (9)

with \( x \mapsto \mathcal{M}^{wc}_{k+1:0}(w, x) \) being the resolvent of the \( w \)-debiased model from \( t_k \) to \( t_{k+1} \). The Gaussian cost function \( J^{wc} \) Equation 7 can hence be written

\[ J^{wc}(w, x_0) = \frac{1}{2} \| x_0 - x^b_0 \|^2_{B^{-1}} + \frac{1}{2} \| w - w^b \|^2_{Q^{-1}} + \frac{1}{2} \sum_{k=0}^{L} \| y_k - H_k \circ \mathcal{M}^{wc}_{k+1:0}(w, x_0) \|^2_{R_k}. \] (10)

This approach is called forcing formulation of WC 4D-Var (Fisher et al., 2011; Laloyaux et al., 2020a; Trémolet, 2006) and is the one that is implemented at ECMWF (Laloyaux et al., 2020b). By construction, the perfect model assumption Equation 1 is relaxed, but the analysis \((w^a_0, x^a_0)\) yields a trajectory of the \( w^a \)-debiased model. In cycled DA, this \( w^a \)-debiased model is used to propagate \( x^b_0 \) till the start of the next window to provide the background state \( x^b_0 \). However this time, a background is also needed for model error \( w^a \). The simplest option is to use \( w^a \) as is, in other words make the assumption that the dynamical model for model error is persistence.

Hereafter, the forcing formulation of WC 4D-Var is simply called WC 4D-Var.

2.3. A Neural Network Formulation of Weak-Constraint 4D-Var

In this section, we introduce a NN-based formulation of WC 4D-Var, hereafter referred to as NN 4D-Var. The generic formulation, originally derived by Farchi, Bocquet, et al. (2021), is presented in Section 2.3.1 without modification. In Section 2.3.2, we develop an incremental formulation for NN 4D-Var, to make it compatible with operational applications. In Section 2.3.3, we show how incremental NN 4D-Var can be used for model error
In Section 2.3.4, we consider the simplified case of an autonomous NN, and in particular we show the connection between the resulting simplified incremental NN 4D-Var and WC 4D-Var. Finally in Section 2.3.5, we describe the details of the implementation in the OOPS framework.

In detail, Section 2.3.1 corresponds to what is described by Farchi, Bocquet, et al. (2021), and the subsequent developments are new.

### 2.3.1. Generic Formulation

Following the approach of Farchi, Bocquet, et al. (2021), we now assume that the dynamical model is parametrized by a set of parameters \( \mathbf{p} \in \mathbb{R}^{N_p} \) constant over the window, in such a way that the model integration Equation 1 becomes

\[
x_k = \mathcal{M}^{\text{nn}}_{k:0}(\mathbf{p}, x_0),
\]

where \( x \mapsto \mathcal{M}^{\text{nn}}_{k:0}(\mathbf{p}, x) \) is the resolvent of the \( \mathbf{p} \)-parametrized model from \( t_0 \) to \( t_k \). Using the state augmentation principle (Jazwinski, 1970), the model parameters \( \mathbf{p} \) can be included in the control variables and hence be estimated in DA. If we further assume that the background for model parameters and system state are independent, and that the background for model parameters is Gaussian with mean \( \mathbf{p}^b \) and covariance matrix \( \mathbf{P} \), then the Gaussian cost function Equation 4 becomes

\[
J^{\text{nn}}(\mathbf{p}, x_0) = \frac{1}{2} \left\| \mathbf{x}_0 - \mathbf{x}_0^b \right\|_2^2 + \frac{1}{2} \left\| \mathbf{p} - \mathbf{p}^b \right\|_2^2 + \frac{1}{2} \sum_{k=0}^{L-1} \left\| \mathbf{y}_k - \mathbf{H}_k \circ \mathcal{M}^{\text{nn}}_{k:0}(\mathbf{p}, x_0) \right\|_2^2.
\]

This formulation is called NN 4D-Var because in the present article, the set of parameters \( \mathbf{p} \) are typically the weights and biases of a NN. Nevertheless, we would like to emphasize the fact that this formulation is not restricted only to NNs and can be used to estimate any parameters. The similarity between Equations 10 and 12 is clear, which is why NN 4D-Var should be seen as another formulation of WC 4D-Var. By construction, the perfect model assumption Equation 1 is once again relaxed, but this time the analysis \((\mathbf{p}^a, \mathbf{x}_0^a)\) yields a trajectory of the \( \mathbf{p}^a \)-parametrized model. In cycled DA, this \( \mathbf{p}^a \)-parametrized model is used to propagate the analysis state \( \mathbf{x}_0^a \) till the start of the next window to provide the background state \( \mathbf{x}_0^b \). Once again, a background is also needed for model parameters \( \mathbf{p}^b \). The simplest option is to use \( \mathbf{p}^a \) as is, in other words make the assumption that the evolution model for model parameters is persistence.

Even though there are a lot of similarities between NN 4D-Var and the WC 4D-Var, two essential differences should be highlighted:

1. The model error \( \mathbf{w} \) lies in the state space \( \mathbb{R}^{N_x} \) while the model parameters lies in the parameter space \( \mathbb{R}^{N_p} \), which has consequences on the design of the covariance matrices \( \mathbf{Q} \in \mathbb{R}^{N_x \times N_x} \) and \( \mathbf{P} \in \mathbb{R}^{N_p \times N_p} \).
2. More importantly, \( \mathcal{M}^{\text{nn}}_{k:0} \) and \( \mathcal{M}^{\text{wc}}_{k:0} \) may have different functional forms. In particular, in the first case the model error \( \mathbf{w} \) is constant while in the second case, it is the model parameters \( \mathbf{p} \) which are constant.

### 2.3.2. Incremental Formulation

In practice, most operational centers using variational DA, including ECMWF, rely on an incremental approach with outer and inner loops for the minimization (Courtier et al., 1994). Within the each outer loop, the cost function is linearized about the first-guess trajectory, and the linearized cost function is then minimized in the inner loop, typically using the conjugate gradient algorithm. Originally, incremental 4D-Var was devised to reduce the computational cost of 4D-Var and make it practical for operational application (Courtier et al., 1994). More recently, the incremental formulation has been shown to be crucial for dealing with nonlinear observations and model behavior in operational implementations (Bonavita et al., 2018).

Therefore, to adapt NN 4D-Var to operational constraints, we need to derive an incremental formulation. Using the change of variables \((\mathbf{\delta p}, \mathbf{\delta x}_0) \triangleq (\mathbf{p} - \mathbf{p}^b, \mathbf{x}_0 - \mathbf{x}_0^b)\), where \((\mathbf{p}^b, \mathbf{x}_0^b)\) is the first guess, we have

\[
J^{\text{inc}}(\mathbf{p}, x_0) = J^{\text{nn}}(\mathbf{p}^b + \mathbf{\delta p}, \mathbf{x}_0^b + \mathbf{\delta x}_0),
\]

\[
= \frac{1}{2} \left\| \mathbf{x}_0^b + \mathbf{\delta x}_0^b \right\|_2^2 + \frac{1}{2} \left\| \mathbf{p}^b + \mathbf{\delta p} \right\|_2^2 + \frac{1}{2} \sum_{k=0}^{L-1} \left\| \mathbf{y}_k - \mathbf{H}_{k \circ \mathcal{M}^{\text{nn}}_{k:0}} (\mathbf{p}^b + \mathbf{\delta p}, \mathbf{x}_0^b + \mathbf{\delta x}_0) \right\|_2^2.
\]

\[
\approx \frac{1}{2} \left\| \mathbf{x}_0^b + \mathbf{\delta x}_0^b \right\|_2^2 + \frac{1}{2} \left\| \mathbf{p}^b + \mathbf{\delta p} \right\|_2^2 + \frac{1}{2} \sum_{k=0}^{L-1} \left\| \mathbf{d}_k - \mathbf{H}_{k \circ \mathcal{M}^{\text{nn}}_{k:0}} (\mathbf{\delta p}, \mathbf{\delta x}_0) \right\|_2^2.
\]
Algorithm 1. Gradient of the incremental cost function $\tilde{J}^m$ Equation 13d.

Input: $\delta p$ and $\delta x_0$
1: $z_0 \leftarrow R^{-1}_0 (H_0 \delta x_0 - d_0)$
2: for $k = 1$ to $L$ do
3: $\delta x_k \leftarrow M_{k:k-1}^m (\delta p, \delta x_{k-1})^T$ $\triangleright$ TL of the dynamical model $M_{k:k-1}^m$
4: $z_k \leftarrow R^{-1}_k (H_k \delta x_k - d_k)$
5: end for
6: $\delta x_L \leftarrow 0$ $\triangleright$ AD variable for system state
7: $\delta p_L \leftarrow 0$ $\triangleright$ AD variable for model parameters
8: for $k = L$ to 1 do
9: $\delta x_k \leftarrow H_k^T z_k + \delta x_k$
10: $(\delta \hat{p}_{k-1}, \delta x_{k-1})^T \leftarrow (M_{k:k-1}^m)^T \delta x_k + (\delta \hat{p}_k, 0)^T$ $\triangleright$ AD of the dyn. model $M_{k:k-1}^m$
11: end for
12: $\delta x_0 \leftarrow H_k^T z_0 + \delta x_k$
13: $\delta \hat{x}_0 \leftarrow B^{-1} (x_0^1 - x_0^0 + \delta x_k) + \delta x_k$
14: $\delta \hat{p} \leftarrow P^{-1} (p' - p^0 + \delta p) + \delta \hat{p}_0$
Output: $\nabla_{\delta p} \tilde{J}^m = \delta \hat{p}$ and $\nabla_{\delta x_0} \tilde{J}^m = \delta \hat{x}_0$

$\Delta \tilde{J}^m(\delta p, \delta x_0)$. (13d)

where $d_i \triangleq y_i - H_k \circ M_{k:0}^m (p', x_i^0)$, $H_k$ is the tangent linear (TL) operator of $H_k$ taken at $M_{k:0}^m (p', x_i^0)$, and $M_{k:0}^m$ is the TL operator of $M_{k:0}^m$ taken at $(p', x_i^0)$. The linearized or incremental cost function $\tilde{J}^m$ is sometimes also called the quadratic cost function because it has the advantage of being quadratic in $\delta p$ and $\delta x_0$, where the conjugate gradient algorithm could be very efficient for the minimization. The key ingredient for this minimisation is the gradient of $\tilde{J}^m$, which is computed using Algorithm 1.

Now, in order to derive a practical implementation of this method, we need to make $M_{k:0}^m$ explicit, which is the purpose of the following sections.

2.3.3. Incremental Formulation for Model Error Correction

In the present article, we want to use NN 4D-Var for model error correction. Let us consider the case where the parametrized model is written

$x_{k+1} = M_{k+1:k}(p, x_k) = M_{k+1:k}(x_k) + F(p, x_k), \quad (14)$

where $F$ is a NN correction added to $M_{k+1:k}$, the resolvent of the (non-corrected) physical model from $t_k$ to $t_{k+1}$, and $p$ are the parameters of this NN.

In this specific example, Algorithm 1 becomes Algorithm 2, where the following notation has been used: $F_{k+1:k}^p$ and $F_{k+1:k}^x$ are the TL operators of $F$ with respect to $p$ and $x$, respectively, both taken at $(p', x_k)$, and $M_{k+1:k}^p$ is the TL operator of $M_{k+1:k}$ taken at $(p', x_k)$.

2.3.4. Simplified Incremental Formulation for Model Error Correction

Following the approach of Section 2.2.2, we assume that the NN is autonomous, that is, the NN correction is constant throughout the window. The model evolution Equation 14 can hence be written

$M_{k+1:k}^m(p, x_k) = M_{k+1:k}(x_k) + w, \quad w = F(p, x_0). \quad (15)$

This evolution model can then be plugged into the cost function $J^m$ Equation 12, which yields a simplified variant of NN 4D-Var where the NN is used only once per cycle. Furthermore, comparing this to Equation 9, we conclude that

$M_{k+1:k}^m(p, x_0) = M_{k+1:k}^m(F(p, x_0), x_0). \quad (16)$
This means that it will be possible to build this new method on top of the currently implemented WC 4D-Var framework, which is a major practical advantage.

With these assumptions, the gradient of the incremental cost function is simplified and can now be computed using Algorithm 3, in which the following notation has been used: $F^p$ and $F^x$ are the TL operators of $F$ with respect to $p$ and $x$, respectively, both taken at $(p^0, x^0)$. In this algorithm, lines 2 to 14 corresponds to the gradient of the incremental cost function of the WC 4D-Var cost function (without the background terms).

### 2.3.5. Practical Implementation in the OOPS Framework

In the present work, we have chosen to implement the simplified incremental NN 4D-Var formulation (presented in Section 2.3.4), which is based on the model evolution given by Equation 15. This choice is driven by practical considerations. Indeed, in order to implement this variant we could reuse most of the framework already in place for WC 4D-Var (specifically in the ECMWF IFS) and we only needed to provide:

- The forward operator $F$ of the NN to compute the nonlinear trajectory at the start of each outer iteration;
- The TL operators $F^p$ and $F^x$ of the NN for line 1 of Algorithm 3;
- The adjoint (AD) operators $[F^p]_T$ and $[F^x]_T$ of the NN for lines 15 and 16 of Algorithm 3.

Implementing the (non-simplified) incremental NN 4D-Var formulation (presented in Section 2.3.3) would have required significant additional coding effort. Indeed, the use of the NN in that case is more intrusive than in the simplified formulation: it has to be applied at each step in both the TL and AD loops, and hence it has to be linearized $L$ times (once for each model step throughout the window) instead of just once. This is why, in this work, we have not implemented the (non-simplified) incremental NN 4D-Var formulation but we could do so in further studies.

From a technical perspective, all the additional operators have to be computed in the model core, where the components of the system state are available. In OOPS, the model core is implemented in Fortran, which implies that we need a ML library in Fortran. The only one that we could find, namely the Fortran–Keras Bridge (FKB, Ott et al., 2020), does not provide all the required operators. For this reason, we have implemented our own ML library in Fortran, called Fortran neural networks (FNN, Farchi et al., 2022). In this library, we have manually implemented, for each layer that we need, functions for the forward, but also the TL and adjoint operators with respect to both NN parameters and NN input. We have then included the FNN library in OOPS and added the interface between OOPS and FNN for two forecast models, OOPS-QG and OOPS-IFS. Finally, we have included the NN parameters in the control variables in OOPS, in such a way that they can be estimated using the simplified incremental NN 4D-Var method.
In the following sections, we validate the simplified incremental NN 4D-Var method using OOPS-QG, as a final test before considering an operational application. In particular, we want to confirm that it is able to make an accurate online estimation of model error. Hereafter, the simplified incremental NN 4D-Var method is simply called NN 4D-Var.

### 3. The Quasi-Geostrophic Model
#### 3.1. Brief Model Description

The QG model in the present article is the same as the one used by Fisher and Gürol (2017) and Laloyaux et al. (2020a) and later by Farchi, Laloyaux, et al. (2021). In the following, we only outline the model description. More details about this model can be found in Fisher and Gürol (2017) and Laloyaux et al. (2020a).

The QG model’s equations express the conservation of the (non-dimensional) potential vorticity $q$ for two layers of constant potential temperature in the $x$–$y$ plane. The potential vorticity is related to the stream function $\psi$ through a specific variant of Poisson’s equation. The domain is periodic in the $x$ direction, and with fixed boundary conditions for $q$ in the $y$ direction. We use a horizontal discretization of 40 grid points in the $x$ direction and 20 in the $y$ direction. In OOPS, the control vector $\mathbf{x}$ contains all values of the stream function $\psi$ for both levels, that is, a total of $N_x = 1600$ variables.

#### 3.2. The Reference and Perturbed Setups

In the test series reported in Sections 4 and 5, we rely on twin experiments. The synthetic truth is generated using the reference setup described by Farchi, Laloyaux, et al. (2021). Model error is then introduced by using a perturbed setup, in which the values of both layer depths and the integration time steps have been modified, as reported in Table 1. Note that, by contrast with the perturbed setup of Farchi, Laloyaux, et al. (2021), the orography term has not been changed, because we have found that the model error setup is sufficiently challenging as is and an orography perturbation does not add meaningful complexity here. An illustration of the model and model error is provided in Figure 1.
3.3. Neural Network Architecture for Model Error Correction

By construction, NN 4D-Var (both the original and simplified incremental formulations) is very similar to parameter estimation, which is very challenging when the number of parameters is high. For this reason, it is important to use prior knowledge on the physics to come up with NN architectures that are *parameter efficient*, that is, NNs that are as expressive as possible with as few parameters as possible.

A typical *parameter efficient* architecture is the monomial architecture introduced by Bocquet et al. (2019), in which the model tendencies are parametrized by a set of regressors (the monomials) and then integrated in time to build the resolvent between two time steps. For completeness, we mention that this architecture is actually very similar to the one originally proposed by Lang et al. (2016), and later used by Mojgani et al. (2022a, 2022b).

In the present article, we follow another approach, introduced by Bonavita and Laloyaux (2020) for the IFS. In this case, the NN is applied independently for each atmospheric column and for several groups of variables: mass (temperature and surface pressure), wind (vorticity and divergence), and humidity. Horizontal and temporal variations are taken into account by adding latitude, longitude, time of the day, and month of the year to the set of predictors. This choice is imposed by operational constraints—variables in different columns may come from different processors when using parallelism. It also makes sense because a significant amount of the model error in the IFS comes from the parametrization of physical processes, which is applied in vertical model columns (Polichtchouk et al., 2022), and because in this configuration, the amount of samples is multiplied by the number of vertical columns in the data, which is highly beneficial to the training. Furthermore, it has been shown that the performance of simple vertical NNs is roughly similar to that of non-vertical convolutional neural networks in a realistic model error correction problem (Laloyaux et al., 2022).

| Parameter               | Reference setup | Perturbed setup |
|-------------------------|-----------------|-----------------|
| Top layer depth         | 6,000 m         | 5,750 m         |
| Bottom layer depth      | 4,000 m         | 4,250 m         |
| Integration time step   | 10 min          | 20 min          |

Table 1. Set of Parameters for the Reference Setup (Middle Row) and the Perturbed Setup (Right Row)

![Figure 1](image_url). Snapshots of the quasi-geostrophic model, equally distributed in time over 96 hr. The first two rows illustrate the stream function (the model state) for the top- ($\psi_2$, first row) and the bottom layer ($\psi_1$, second row). The last two rows illustrate the 24 hr model error for the top- ($w_2$, third row) and the bottom layer ($w_1$, fourth row). The 24 hr model error is defined as the difference between a 24 hr forecast with the true model and a 24 hr forecast with the perturbed model, both starting from the initial condition given by the first two rows.
The QG model has only two vertical layers and one variable, the stream function \( \psi \), and it is autonomous, that is, the model does not explicitly depend on time. This means that our NN for model error correction, independently applied to all 40 \( \times \) 20 columns, has four predictors:

1. \( \psi_1 \), the bottom layer stream function;
2. \( \psi_2 \), the top layer stream function;
3. \( \sin [2 \pi (\theta - 1/2)/40] \), where \( \theta \) is the longitude index between 1 and 40;
4. \( \sin [\pi (\lambda - 1/2 - 10)/20] \) where \( \lambda \) is the latitude index between 1 and 20;

and two predictands:

1. \( w_1 \), the model error estimate for the bottom layer stream function;
2. \( w_2 \), the model error estimate for the top layer stream function.

Note that, in the zonal direction, the sinus function is used here to make the NN aware of the periodicity. Moreover, the learning of deep neural networks is known to be biased toward low spatial frequencies (Rahaman et al., 2019)

The importance of this issue has been recently highlighted by Mojgani et al. (2022a). Following Mojgani et al. (2022a), we could solve this issue by using as predictor additional frequencies of \( \theta \) and \( \lambda \) such as, for example, \( \sin [2k \pi (\theta - 1/2)/40] \) for \( k = 2, \ldots, 40 \) and \( \sin [\pi (\lambda - 1/2 - 10)/20] \) for \( k = 2, \ldots, 20 \). Nevertheless, we chose not to use this approach for several reasons. First, adding a bunch of extra predictors would increase a lot the number of parameters of the NN, which is what we would like to avoid. Second, we expect (a) \( \psi_1 \) and \( \psi_2 \) to be much more important predictors than \( \theta \) and \( \lambda \), and (b) the model error (and hence \( w_1 \) and \( w_2 \)) to be large scale (Laloyaux et al., 2020a), which means that a spectral bias toward low frequencies would not be a major issue.

We have tested several NNs, and ended up with the following sequential architecture, illustrated in Figure 2: (a) a first internal dense layers with 16 neurons and with the tanh activation function; (b) a second internal dense layers with 16 units and with the tanh activation function as well; (c) one output dense layer with 2 units and no activation function. This NN has a total of \( (2 \times 4 + 4) + (4 \times 4 + 4) + (4 \times 2 + 2) = 386 \) parameters, which is significantly less than the number of variables (1,600).

To stay within the scope of the simplified incremental NN 4D-Var defined in Section 2.3.4, we assume that the NN correction is constant throughout the window, and that it is added after every model time step (i.e., every 20 min in our case) as it is enforced in the current implementation of WC 4D-Var. According to the classification of Farchi, Laloyaux, et al. (2021) and Farchi, Bocquet, et al. (2021), this approach is a resolvent correction (as opposed to a tendency correction), because it is added after the integration scheme. However, a classical resolvent correction would add the correction after every window, in other words much less frequently than after every model time step. Hence, the spirit of the present correction is closer to that of a tendency correction.

### 4. Offline Learning Results

We begin the numerical experiments by using offline learning to train the NN. Offline learning here serves two purposes: it provides a baseline for comparison as well as a pre-trained NN for online learning.

#### 4.1. Observation and Data Assimilation Setup

In the present test series, we use for the QG model the same initial condition as Farchi, Laloyaux, et al. (2021). After a first relaxation run of 256 days, the state is perturbed and a second relaxation run of 256 days is performed to provide the initial state for the DA experiment. At this point, observations are available every 2 hr, starting at 01:00 every day, at 30 fixed locations, whose distribution mimics the coverage provided by (polar-orbiting) satellite soundings. The observation operator is simply a bilinear interpolation of the stream function at the observation locations. The observations are independently perturbed using a Gaussian noise with zero mean and standard deviation equal to 0.2 (about 4% of the true model variability).

We start by assimilating the observations using cycled strong-constraint 4D-Var, with consecutive windows of 24 hr starting at 00:00 each. Hence, there are 12 batches of observations, for a total of 360 observations per window. The observation error covariance matrix is set to \( R = 0.2^2 I \) to be consistent with how the synthetic observations are produced. For the first cycle, the background state \( x_0 \) is set to be the initial condition before
the two relaxation runs. For the following cycles, the background state is obtained by forecasting the previous analysis state. Finally, the background error covariance matrix is set to $B = b^2C$, where $C$ is a short-range correlation matrix, the same as the one used by Farchi, Laloyaux, et al. (2021), and where $b$ is the standard deviation, a free parameter. The accuracy of the estimations is measured with the instantaneous root-mean-squared error (RMSE) between the estimate and the truth for all 1,600 state variables, possibly averaged over time. In particular, the first-guess (respectively analysis) RMSE is defined in this article as the instantaneous RMSE between the first-guess (or analysis) trajectory, the trajectory originated from the first-guess (or analysis) at the start of the window, and the true trajectory, averaged over the entire DA window. The time-averaged first-guess (respectively analysis) RMSE is then defined as this first-guess (or analysis) RMSE averaged over a sufficiently large number of cycles. For completeness, note that in all our experiments, we choose the background state as first-guess at the start of each window.

In order to be close to operational conditions, we tune the value of $b$ to minimize the time-averaged first-guess RMSE. Preliminary experiments (not detailed here) have shown that, for the present DA setup, the optimal value is $b = 0.4$. With this value, we run a cycled DA experiment of $N_t^{\text{total}} = 2,100$ cycles. The results of the first $N_t^{\text{spinup}} = 51$ cycles are dropped as spin-up process of the experiment. Then, for each remaining cycles $t = 1, \ldots, N_t^{\text{data}} = 2049$, we keep $x_0^a(t)$ and $x_0^a(t)$, respectively the first-guess and the analysis at the start of the $t$th window.

### 4.2. Neural Network Training

As shown by Farchi, Laloyaux, et al. (2021), the analysis increment $x_0^a(t) - x_0^a(t)$ can be chosen as a proxy of the model error for a 1-window-long integration, provided that the analysis is a reasonably accurate estimation of the true state:

$$x_0^a(t+1) - x_0^a(t+1) = x_0^a(t+1) - \mathcal{M}_t \left(x_0^a(t)\right) \approx x_0^a(t+1) - \mathcal{M}_t \left(x_0^a(t)\right),$$

where $\mathcal{M}_t$ corresponds to the resolvent of the model between the start of the $t$th window and the start of the $(t+1)$-th window, and where $x_0^a(t)$ is the true state of the system at the start of the $t$th window. However, as explained in Section 3.3, the NN correction is added after every model time step, which means that we need a proxy of the model error for a 1-step integration. Without further knowledge on the model error dynamics, we assume a uniform linear growth of model error in time and hence we rescale the analysis increments by a factor $\delta t/\Delta T = 1/72$, where $\delta t = 20$ min is the model time step and $\Delta T = 24$ hr is the window length.

To summarize, we use the following data set for the training of the NN:

$$\left\{x_0^a(t) \rightarrow \frac{\delta t}{\Delta t} \left[x_0^a(t+1) - x_0^a(t+1)\right], \quad t = 1, \ldots, N_t^{\text{data}} - 1 = 2048 \right\}.$$  

Note the time lag between the input $x_0^a(t)$ and the output $\delta t/\Delta T \left(x_0^a(t+1) - x_0^a(t+1)\right)$. Indeed, the analysis increment $x_0^a(t+1) - x_0^a(t+1)$ of the $(t+1)$-th window does inform about the model error during the $t$th window, which is exactly what we need according to the model formulation described in Section 2.3.4. Also note that we have chosen to use the analysis $x_0^a(t)$ as predictor, but we could have equivalently chosen the first-guess $x_0^a(t)$.

Preliminary experiments (not illustrated here) have shown that both choices yield similar results. Since the NN is applied independently to each atmospheric column, there are actually $40 \times 20 = 800$ samples per pair (analysis $\rightarrow$ analysis increment). Finally, in order to accelerate the convergence, the input and output of the training data set are standardized before the training, using independent normalization coefficients for each variable.

In order to evaluate the sensitivity to the length of the data set, we train the NN using only the last $N_t^{\text{train}}$ pairs (analysis $\rightarrow$ analysis increment) for several values of $N_t$. Among all these $N_t^{\text{train}}$ pairs, the first 7/8th form the training data set and the last 1/8th the validation data set. The test data set is formed by $N_t^{\text{test}} = 2048$ pairs (truth $\rightarrow$ true model error) originated from a different trajectory of the model. With this setup, the NN is trained for a maximum of 1,024 epochs using Adam algorithm (Kingma & Ba, 2015), a variant of the stochastic gradient descent, with the typical learning rate $10^{-3}$. The loss function is the mean-squared error (MSE). To accelerate the training, we use a relatively large batch size (1,024) as well as an early stopping callback on the validation MSE with a patience of 256 epochs. After triggering the early stopping callback, we restore the optimal NN parameters, that is, the NN parameters which minimize the validation MSE. Finally after the
training, we compute the test MSE. This experiment is repeated 16 times with different sets of trajectories for training and testing and different random seeds for the training. For comparison, we have also performed the exact same set of experiments but with dense and perfect observations, that is, when the analysis is equal to the true state. This second set of experiments illustrates the full predictive power of the NN representation of the model error.

Figure 3 shows the evolution of the test MSE as a function of the length of the training data set \( N_{\text{train}} \). The score is normalized by the averaged squared norm of the model error, in such a way that it is equal to 1 when the NN predicts a zero model error. In all experiments, the normalized test MSE is lower than 1. This means that, on average, the model error prediction is useful. When using the truth, both training and test data sets are statistically equivalent. The normalized test MSE decreases with the size of the training data set \( N_{\text{train}} \). The final value is 0.334 for \( N_{\text{train}} = 2048 \), but the score is already very close (0.351) for \( N_{\text{train}} = 128 \). The residual error for a large training data set comes from the limited predictive power of the NN. We have checked that better scores can easily be obtained when using larger, non column-wise NNs. Unsurprisingly, when using the analysis the normalized test MSE is significantly higher (0.735 at best) and stops improving for \( N_{\text{train}} \geq 256 \). The primary reason for these discrepancies is the fact that the statistical moments (e.g., the time average and time standard deviation) are not the same between the analysis increments and the true model error. In particular, the average analysis increment norm is lower than the average model error norm. This means that the NN trained with the analysis generally underestimates the model error. This is consistent with what has been found by Crawford et al. (2020) and Farchi, Laloyaux, et al. (2021).

In both cases (NN trained with the truth and NN trained with the analysis), we observe in Figure 3 that the accuracy saturates for large values of the size of the training data set. This means that our data set is large enough to train the model in both cases.

4.3. Accuracy of the Hybrid Model

Now that the NN has been trained, we would like to test the hybrid model in forecast and DA experiments.

4.3.1. Corrected Data Assimilation

We start with DA experiments using the exact same setup as in Section 4.1, but with a true state taken from a different trajectory of the model. Four 4D-Var variants are compared:

1. **SC**: strong-constraint (Section 2.1) with the physical model (no model error correction).
2. **WC**: weak-constraint (Section 2.2.2) with the physical model—in this case the model error correction comes from the constant, online estimated forcing.
3. **SC-Ht**: strong-constraint (Section 2.1) with the hybrid model, where the NN correction has been trained offline with the truth using the largest data set \( N_{\text{train}} = 2048 \).
Table 2

| Variant | 4D-Var constraint | Model error correction | First-guess RMSE | Analysis RMSE |
|---------|-------------------|------------------------|------------------|--------------|
| SC      | Strong (Section 2.1) | —                     | 0.350 (0.020)    | 0.157 (0.003) |
| WC      | Weak (Section 2.2.2) | Constant, online estimated | 0.271 (0.016)    | 0.128 (0.003) |
| SC-Ht   | Strong (Section 2.1) | NN trained offline with the truth | 0.263 (0.018)    | 0.133 (0.003) |
| SC-Ha   | Strong (Section 2.1) | NN trained offline with the analysis | 0.265 (0.020)    | 0.144 (0.003) |

Note: Time-averaged first-guess and analysis RMSE for the four 4D-Var variants presented in Section 4.3.1. For each variant, we report the mean (main numbers) and standard deviation (in parentheses) values over the 128 experiments.

4. SC-Ha: strong-constraint (Section 2.1) with the hybrid model, where the NN correction has been trained offline with the analysis using the largest data set \( N_{\text{train}} = 2048 \).

In all cases, we use the same background error covariance matrix \( B \) as in Section 4.1, because we want to highlight the benefit of each approach without the need to re-tune \( B \). The initial background state \( x_0 \) corresponds to the background obtained after a spin-up of 32 DA cycles with strong-constraint 4D-Var. For WC 4D-Var, we need to provide in addition (a) the initial background for model error \( w^b(0) \), and (b) the background error covariance matrix for model error \( Q \). We choose to use \( w^b(0) = 0 \) and \( Q = q^2 \hat{C} \), where \( \hat{C} \) is a long-range correlation matrix, the same as the one used by Laloyaux et al. (2020a), and where \( q \) is the standard deviation, another free parameter. We choose \( q = 0.004 \) in order to minimize the time-averaged first-guess RMSE. In each case, we run a cycled DA experiment of \( N_{\text{ assim}} = 257 \) cycles, which we empirically consider to be sufficiently long. The results of the first 33 cycles are dropped as spin-up. For the remaining 224 cycles, we compute the first-guess and analysis RMSE.

Each experiment is repeated 128 times with different trajectories for the synthetic truth. Note that in the second and third case, the 128 repetitions are equally spread over the 16 trained NN obtained in Section 4.2: experiments 1 to 8 use the first trained NN, experiments 9 to 16 use the second, experiments 17 to 24 use the third, etc.

The time-averaged first-guess and analysis RMSE are reported in Table 2. The results show the efficiency of model error corrections: in all cases, the first-guess and the analysis are more accurate with model error correction (WC/SC-Ht/SC-Ha) than without (SC). As expected, the model error correction provided by the NN is more efficient when the NN has been trained with the truth (SC-Ht) than when it has been trained with the analysis (SC-Ha). Furthermore, using the offline correction provided by the NN (SC-Ht/SC-Ha) yields in both cases a more accurate first-guess but a less accurate analysis than using the online correction computed with WC 4D-Var (WC).

4.3.2. Corrected Forecast

To conclude this first test series, we evaluate the accuracy of the model in the four cases described in Section 4.3.1. To this end, we extend the previous set of experiments. After each analysis cycle, we compute a 32-day forecast starting from the DA analysis using the same model as in the 4D-Var cost function. In the case of WC 4D-Var (WC), the constant, online estimated forcing is used throughout the entire forecast. In the case of strong-constraint 4D-Var with the hybrid model (SC-Ht/SC-Ha), the NN correction is also used throughout the entire forecast, but in a flow-dependent way: the correction values are updated at a 1-day frequency using the forecasted state. With these specifications, the error in the first day of forecast corresponds to the analysis error and the error in the second day of forecast corresponds to the first-guess error. Figure 4 shows the evolution of the forecast RMSE, averaged over the last 32 DA cycles and over the 128 repetitions of the experiments, as a function of the forecast lead time.

With WC 4D-Var (WC), the model error correction is calibrated over the DA window, that is, over the first day. Overall, the correction is efficient and yields a more accurate forecast than with the non-corrected model (SC). After several days, the true model error has significantly evolved and this initial error estimate gets less accurate. This is why the reduction of the forecast error vanishes after several days. Also note that the model has a periodic behavior, with a period around 16 days. This means that, after 16 days, the model state (and hence the model error) is roughly the same as at the beginning, which explains the forecast error reduction around day 16 and around day 32.
By contrast, when using the hybrid model (SC-Ht/SC-Ha), the model error correction is flow-dependent (updated every day). This yields overall an even more accurate forecast than with WC 4D-Var (WC). In the first few days, the correction accumulates and positively interacts with the physical model, which is why the forecast error reduction increases over time. After several days however, the model error correction becomes less efficient, because the forecasted state—the most important predictor of the NN—has become significantly different from the true state. At this point, the model error correction does not any more yield a forecast error reduction. Worse, it even increases the forecast errors. This explains the quick increase of the forecast errors after 10 days when the NN is trained with the truth (SC-Ht) and after 15 days when the NN is trained with the analysis (SC-Ha). In an operational perspective, it would be interesting to progressively mitigate the model error correction over time, but this is beyond the scope of the present study. Surprisingly, the validity period of the model error correction is longer for SC-Ha (NN trained with the analysis) than for SC-Ht (NN trained with the truth). This could be due to the fact that a NN trained with the analysis underestimates the model error: if the model error estimate is pointing in the wrong direction, it is better to have an underestimated model error (Crawford et al., 2020). Finally, after about 13 days, the forecast is more accurate with SC-Ha. We believe that this result is related to the limited predictive power of the chosen NN. Indeed, we have checked that with larger NNs, the accuracy of the forecast is almost always more accurate with SC-Ht than with SC-Ha.

5. Online Learning Results

In the present section, we test the simplified incremental NN 4D-Var presented in Section 2.3.4, as an online model error correction method, using the same QG model as in the offline experiments.

5.1. Data Assimilation Setup

In this last test series, we use the same DA setup as in Sections 4.1 and 4.3.1. Once again, the true state stems from a different trajectory. We keep the same initial background state \( x_0 \) and background error covariance matrix \( B \) as in Section 4.3.1, once again to highlight the benefit of each approach without the need to re-tune \( B \). In addition, we need to provide (a) the initial background for model parameters \( p_0 \) and (b) the background error covariance matrix for model parameters \( P \). For \( p_0 \), we choose to use the parameters of the NN that has been trained offline with the analysis, in other words we use offline learning as a pre-training step for online learning. Hence we hope to immediately see the potential benefits of online learning. Finally, without any prior knowledge on the model parameters, we use \( P = p I \), where \( p \) is the standard deviation, a free parameter. After several preliminary tests, we have chosen \( p = 0.02 \). Following the approach of Section 4.3.2, at each DA cycle, we compute a 32-day forecast starting from the DA analysis using the hybrid model with the updated parameters. Finally, once again, each experiment is repeated 128 times with as many different trajectories for the synthetic truth. In the follow-
ing paragraphs, we use the label $\text{NN-Ha}$ to refer to this fifth 4D-Var variant: “NN” refers to the 4D-Var variant (simplified incremental NN 4D-Var), “H” refers to the hybrid model, and “a” refers to the fact that the NN has been pre-trained offline with the analysis.

5.2. Temporal Evolution of the Forecast Errors

Figure 5 shows the temporal evolution of the errors in the first day of forecast (the analysis), in the second day of forecast (the first-guess), and in the eighth day of forecast (which corresponds to a medium-range forecast). The evolution in all three cases is very similar. At the start of the experiment, the forecast errors with $\text{NN-Ha}$ (NN trained online) are close to those with $\text{SC-Ha}$ (NN trained offline with the analysis). This was expected because in the $\text{NN-Ha}$ variant, we have initialized the parameters of the NN using the parameters obtained by offline training with the analysis. The added positive effect of the online NN training is then rapidly visible. After a few cycles, the forecast errors have decreased. This improvement is quicker for shorter forecast horizons. For the medium-range errors, we even see an increase at the start of the experiments before they eventually decrease, after several dozens of cycles. At the end of the experiments, the forecast is significantly more accurate with $\text{NN-Ha}$ than with $\text{SC-Ha}$, which is what we hoped for. In some cases (first-guess and medium range), the forecast is even better with $\text{NN-Ha}$ than $\text{SC-Ht}$ (NN trained offline with the truth). This results may seem at first somewhat surprising because, unless there has been some optimization issues, the NN trained offline with the truth should provide the most accurate model error predictions. However, one must keep in mind that two essential simplifications have been made:

1. The model error growth is assumed to be linear in time (Section 4.2);  
2. The model error correction is constant over the DA window (Section 2.3.4).

This explains why the NN trained offline with the truth is suboptimal in the DA and forecast experiments considered here. The first assumption could be circumvented by using samples of the true model error for a $\delta t = 20$ min forecast (obviously, this would not be possible when training with the truth) but the second assumption is intrinsic to the simplified incremental NN 4D-Var formulation. This second assumption allows us to build NN 4D-Var as a relatively simple extension of the currently implemented WC 4D-Var, but it has a negative impact on the forecast that we will illustrate in the following section.

5.3. Focus on the Second Day of Forecast

Figure 6 shows the temporal evolution of the errors in the second day of forecast in two cases: (a) the NN correction is updated every day (as has been done previously—this corresponds to the first-guess errors) or (b) it is kept constant throughout the entire forecast. The forecast errors with the NN ($\text{SC-Ht}/\text{SC-Ha}/\text{NN-Ha}$) are systematically lower in the second case than in the first. Indeed, in the 4D-Var variants considered here, the NN correction is constant over the DA window, hence the forecast model is more consistent with the 4D-Var analysis when the NN correction is not updated.

Of course, this argument is only valid for short-range forecasts. Indeed, because the (true) model error evolves over time, after several days the initial error estimate gets less accurate (as explained in Section 4.3.2 in the discussion on the accuracy of the forecast with WC). For this reason, it is important for the medium- and long-range forecast accuracy to regularly update the NN correction.

Therefore, we believe that implementing NN 4D-Var without the assumption of a constant model error over the window (i.e., the non-simplified incremental NN 4D-Var formulation presented in Section 2.3.3) should have a positive impact on the analysis, but also in the forecast. However, as explained before, a direct implementation of such a formulation would not be trivial, as it could not be built directly on top of the existing WC 4D-Var. Although we have not attempted it in this study, we will consider it in further studies.

5.4. Forecast Errors at the End of the Experiments

Finally, Figure 7 shows the evolution of the forecast RMSE, averaged over the last 32 cycles and over the 128 repetitions of the experiments, as a function of the forecast lead time. The errors are the same as the ones in Sections 5.2 and 5.3, but aggregated and shown in a different way. For the $\text{NN-Ha}$ variant, the forecast errors up to day 10 are consistent with the description in Section 5.2. After day 10, the forecast errors increase accelerate,
which indicates that the NN correction is not any more valid. This is the same phenomenon as what has been described in Section 4.3.2 for SC-Ht and SC-Ha, but this time, the error increase is earlier and quicker. Once again, we believe that this result is related to the limited predictive power of the chosen NN. However, using a larger and deeper NN (i.e., with more parameters) is not necessarily a good strategy with online learning. Indeed, based on preliminary experiments, we conclude that if the number of parameters is large, the background error covariance matrix for parameters (called $P$ in Section 2.3) must be small to avoid a quick divergence of the method. The downside of this choice is that it naturally slows down the learning process. This is why, with online learning, it is important to keep the number of parameters as small as possible, as explained by (Farchi, Bocquet, et al., 2021). Hence, the use of online learning could initially be limited to the correction of short-term forecasts.

Figure 5. Forecast scores for the online experiments. Evolution of the forecast root-mean-squared error, averaged over the 128 experiments and over the first day of forecast (PT0S-P1D, top panel), over the second day of forecast (P1D-P2D, middle panel), or over the eighth day of forecast (P8D-P9D, bottom panel), as a function of time for the five 4D-Var variants: SC in blue, WC in orange, SC-Ht in green, SC-Ha in red, and NN-Ha in teal. The thin lines report the instantaneous values and the thick ones report the running-average over 32 cycles.
6. Conclusions

In this article, we have developed a new variant of WC 4D-Var, in which a set of parameters can be jointly estimated alongside the system state. The new method is called NN 4D-Var to emphasize the fact that it is used in
this article to estimate the coefficients (weights and biases) of a NN. It can be seen as a simplified incremental variant of the original NN 4D-Var method introduced by Farchi, Bocquet, et al. (2021), dedicated to model error correction. It is assumed that the NN provides a correction to a physical model, added after each integration, and constant over the DA window. These simplifications make the method very similar to the forcing formulation of WC 4D-Var, and hence easier to implement on top of an existing implementation of WC 4D-Var, such as the one available in the OOPS framework.

In the second part of the article, we have provided a numerical illustration of the new, simplified incremental NN 4D-Var algorithm in conditions which are as close as possible to operational. The illustrations use twin experiments with OOPS-QG, a two-layer two-dimensional QG model. A simple yet non-trivial model error setup is introduced, where the layer depths and integration step of the model are perturbed. The model error correction is computed using a small, dense NN acting on vertical columns, like the one used for an operational model by Bonavita and Laloyaux (2020). The NN is first trained offline, using the analyses and analysis increments of a DA experiment with the non-corrected model, following the method originally introduced by Brajard et al. (2020). The corrected model is then used in forecast and DA experiments, and provides in both cases significant improvements in the scores as already shown by Farchi, Laloyaux, et al. (2021). Then, the NN is trained online using the new, simplified incremental NN 4D-Var algorithm. The results confirm the findings of Farchi, Bocquet, et al. (2021) for the original NN 4D-Var algorithm. With proper tuning of the background error covariance matrices, an online, joint estimation of the system state and the NN parameters is possible. As new observations become available, the model error correction becomes more accurate, which translates into lower analysis, first-guess, and short- to mid-term forecast errors than in the offline training case.

The results also illustrate two limitations of the simplified incremental NN 4D-Var method. The first is related to the assumption of a constant model error throughout the window. This is necessary to build the new method on top of an existing WC 4D-Var implementation, but we believe that relaxing this simplification could improve the analysis and short-term forecast errors. This could be the topic of further studies on the method. The other limitation is somewhat more fundamental: the online training process is slower as the number of parameters to estimate is larger, as already highlighted by Farchi, Bocquet, et al. (2021). This underlines the importance of choosing parameter-efficient NNs.

At this point, we estimate that the simplified incremental NN 4D-Var method is mature enough for more realistic applications, for example, with the IFS. Implementing the new formulation in this operational model will only require developing an interface to the NN library with all the algorithmic developments already in place in the OOPS framework. For such application, we would typically use the vertical NN architecture of Bonavita and Laloyaux (2020), for which the number of parameters is much lower than the number of system state variables. In this case however, the main difficulty would come from the fact that the true state of the system is unknown, which makes the evaluation much harder because the diagnostics should be based on observations. Nevertheless, we should be able to rely on the test suite developed by ECMWF to evaluate the potential benefits of proposed upgrades to the operational assimilation and forecast systems.

Finally, the current implementation of the simplified incremental NN 4D-Var method in OOPS is dedicated to model error correction only, that is, the NN is trained for model error correction only. Nevertheless, there is no obstacle to use this method to train the NN for other tasks (e.g., observation bias correction) provided that we are able to model their effect on the 4D-Var cost function.

Data Availability Statement

The numerical experiments in this article rely on OOPS and the FNN library (version 1.0.0). The source code of OOPS is property of ECMWF and is not publicly available. The source code of FNN (Farchi et al., 2022) is preserved at https://doi.org/10.5281/zenodo.7245291, available via the MIT license and developed openly at https://github.com/cerea-daml/fnn.
Acknowledgments
The authors are grateful to two anonymous reviewers for insightful comments and suggestions. AF thanks Tobias Finn for feedback on the manuscript. AF has benefited from a visiting grant of the ECMWF. CEREA is a member of Institut Pierre–Simon Laplace.

References
Abarbanel, H. D. I., Rozdeba, P. J., & Shirman, S. (2018). Machine learning: Deepest learning as statistical data assimilation problems. Neural Computation, 30(8), 2025–2055. https://doi.org/10.1162/neco_a_01094

Arcomano, T., Szunyogh, I., Pathak, J., Wikner, A., Hunt, B. R., & Ott, E. (2020). A machine learning–based global atmospheric forecast model. Geophysical Research Letters, 47(9), e2020GL087776. https://doi.org/10.1029/2020GL087776

Arcucci, R., Zhu, J., Hu, S., & Guo, Y.-K. (2021). Deep data assimilation: Integrating deep learning with data assimilation. Applied Sciences, 11(3), 1114. https://doi.org/10.3390/app11031114

Asch, M., Bocquet, M., & Nodet, M. (2016). Data assimilation: Methods, algorithms, and applications (No. 11). SIAM, Society for Industrial and Applied Mathematics.

Bocquet, M., Brajard, J., Carrassi, A., & Bertino, L. (2019). Data assimilation as a learning tool to infer ordinary differential equation representations of dynamical models. Nonlinear Processes in Geophysics, 26(3), 143–162. https://doi.org/10.5194/npg-26-143-2019

Bocquet, M., Brajard, J., Carrassi, A., & Bertino, L. (2020). Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization. Foundations of Data Science, 2(1), 55–80. https://doi.org/10.3934/fods.2020004

Bocquet, M., Farchi, A., & Malaric, Q. (2021). Online learning of both state and dynamics using ensemble Kalman filters. Foundations of Data Science, 3,2639–8001_2019_0_24, 305–330. https://doi.org/10.3934/fds.2020015

Bolton, T., & Zanna, L. (2019). Applications of deep learning to ocean data inference and subgrid parameterization. Journal of Advances in Modeling Earth Systems, 11(1), 376–399. https://doi.org/10.1029/2018MS001472

Bonavita, M., & Laloyaux, P. (2020). Machine learning for model error inference and correction. Journal of Advances in Modeling Earth Systems, 12(12). https://doi.org/10.1029/2020MS002322

Bonavita, M., Lean, P., & Holm, E. (2018). Nonlinear effects in 4D-Var. Nonlinear Processes in Geophysics, 25(3), 713–729. https://doi.org/10.5194/npg-25-713-2018

Bonavita, M., Trémolet, Y., Holm, E., Lang, S., Christ, M., Janiskova, M., et al. (2017). A strategy for data assimilation. (p. 800). https://doi.org/10.21957/TX1EPDP2p

Brajard, J., Carrassi, A., Bocquet, M., & Bertino, L. (2020). Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model. Journal of Computational Science, 44, 101171. https://doi.org/10.1016/j.jocs.2020.101171

Brajard, J., Carrassi, A., Bocquet, M., & Bertino, L. (2021). Combining data assimilation and machine learning to infer unresolved scale parametrization. Philosophical Transactions of the Royal Society A: Mathematical, Physical & Engineering Sciences, 379(2194), 20200086. https://doi.org/10.1098/rsta.2020.0086

Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations equations from data by sparse identification of nonlinear dynamical systems. Proceedings of the National Academy of Sciences of the United States of America, 113(15), 3932–3937. https://doi.org/10.1073/pnas.1517384113

Carrassi, A., Bocquet, M., Bertino, L., & Evensen, G. (2018). Data assimilation in the geosciences: An overview of methods, issues, and perspectives. Wiley Interdisciplinary Reviews: Climate Change, 9(5), e535. https://doi.org/10.1002/wcc.535

Chen, T.-C., Penny, S. G., Whitaker, J. S., Frolov, S., Pincus, R., & Tulich, S. N. (2022). Correcting systematic and state-dependent errors in the observations of dynamical models. Journal of Advances in Modeling Earth Systems, 14(9), 3729–3745. https://doi.org/10.1175/JMRI-D-20-0008.1

Dee, D. P. (2005). Bias and data assimilation. Quarterly Journal of the Royal Meteorological Society, 131(615), 3323–3343. https://doi.org/10.1012/njxg.05.137

Dueben, P. D., & Bauer, P. (2018). Challenges and design choices for global weather and climate models based on machine learning. Geoscientific Model Development, 11(10), 3999–4009. https://doi.org/10.5194/gmd-11-3999-2018

Evensen, G., Vossepoel, F. C., & van Leeuwen, P. J. (2022). Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem. Springer Nature.

Fablet, R., Otsala, S., & Herzet, C. (2018). Bilinear residual neural network for the identification and forecasting of geophysical dynamics. In 2018 26th European signal processing conference (EUSIPCO) (pp. 1477–1481). IEEE. https://doi.org/10.23919/EUSIPCO.2018.8553492

Farchi, A., Bocquet, M., Laloyaux, P., Bonavita, M., & Malaric, Q. (2021). A comparison of combined data assimilation and machine learning methods for offline and online model error correction. Journal of Computational Science, 55, 101468. https://doi.org/10.1016/j.jocs.2021.101468

Farchi, A., Christ, M., Bocquet, M., Laloyaux, P., & Bonavita, M. (2022). The Fortran neural network (FNN) library. https://doi.org/10.5281/zenodo.7245291

Farchi, A., Laloyaux, P., Bonavita, M., & Bocquet, M. (2021). Using machine learning to correct model error in data assimilation and forecast applications. Quarterly Journal of the Royal Meteorological Society, 147(739), 3067–3084. https://doi.org/10.1002/qj.4116

Fisher, M., & Gürol, S. (2017). Parallelization in the time dimension of four-dimensional variational data assimilation: Parallelization of 4D-Var. 4-dimensional Quarterly Journal of the Royal Meteorological Society, 143(703), 1136–1147. https://doi.org/10.1002/qj.2997

Fisher, M., Trémolet, Y., Auvinein, H., Tan, D., & Poli, P. (2011). Weak-constraint and long-window 4D-Var. (p. 655). https://doi.org/10.21957/9i4d4d4qq

Fogarty, M. J., Christensen, H. M., Subramanian, A. C., & Monahan, A. H. (2020). Machine learning for stochastic parameterization: Generative adversarial networks in the Lorenz ‘96 model. Journal of Advances in Modeling Earth Systems, 12(3). https://doi.org/10.1029/2019MS001896

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. The MIT Press.

Gottwald, G. A., & Reich, S. (2021). Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation. Physica D: Nonlinear Phenomena, 423, 132911. https://doi.org/10.1016/j.physd.2021.132911

Hamilton, F., Berry, T., & Sauer, T. (2016). Ensemble Kalman filtering without a model. Physical Review X, 6(1), 011021. https://doi.org/10.1103/PhysRevX.6.011021

Harlim, J., Jiang, S. W., Liang, S., & Yang, H. (2021). Machine learning for prediction with missing dynamics. Journal of Computational Physics, 428, 109022. https://doi.org/10.1016/j.jcp.2020.109022

Jazwinski, A. H. (1970). Stochastic processes and filtering theory (No. 64). Academic Press.
