scientific reports

Check for updates

OPEN

The deep latent space particle filter for real-time data assimilation with uncertainty quantification

Nikolaj T. Mücke[1,3]✉, Sander M. Bohté[2,4] & Cornelis W. Oosterlee[3]

In data assimilation, observations are fused with simulations to obtain an accurate estimate of the state and parameters for a given physical system. Combining data with a model, however, while accurately estimating uncertainty, is computationally expensive and infeasible to run in real-time for complex systems. Here, we present a novel particle filter methodology, the Deep Latent Space Particle filter or *D-LSPF*, that uses neural network-based surrogate models to overcome this computational challenge. The D-LSPF enables filtering in the low-dimensional latent space obtained using Wasserstein AEs with modified vision transformer layers for dimensionality reduction and transformers for parameterized latent space time stepping. As we demonstrate on three test cases, including leak localization in multi-phase pipe flow and seabed identification for fully nonlinear water waves, the D-LSPF runs orders of magnitude faster than a high-fidelity particle filter and 3-5 times faster than alternative methods while being up to an order of magnitude more accurate. The D-LSPF thus enables real-time data assimilation with uncertainty quantification for the test cases demonstrated in this paper.

Keywords Particle filter, Transformers, Wasserstein autoencoders, Partial differential equations, Data assimilation

Virtual representations of physical systems like digital twins have proven to be invaluable tools for monitoring, predicting, and optimizing the performance of intricate systems, ranging from industrial machinery to biological processes[1]. The efficacy of digital twins relies however on the accurate assimilation of real-time data into the simulations to ensure accurate calibration and state estimation in situations where the state and its dynamics are not known. Importantly, to confidently rely on the information provided, data assimilation should be accompanied by a quantification of the associated uncertainty originating from both measurements and model errors[2,3].

Performing data assimilation with uncertainty quantification in real time for high-dimensional systems, such as discretized partial differential equations (PDEs), is computationally infeasible due to the need to compute large ensembles of solutions. Approaches such as (ensemble) Kalman filtering[4] aim to overcome this computational bottleneck by assuming Gaussian distributed prior, likelihood, and posterior distributions[5], which is restrictive in practical situations where these assumptions do not hold and when the problems are highly nonlinear[6,7]. As an alternative, iterative ensemble smoother methods have recently emerged[8]. These methods speed up the computations by utilizing efficient nonlinear solvers and can handle stronger nonlinearities. However, unlike filtering methods, they assimilate all data at once and are thereby not immediately suitable for online data assimilation. Particle filters, on the other hand, can approximate any distribution sequentially provided there is a sufficient number of particles in the ensemble[9,10]. The necessary ensemble size for particle filters however often makes it infeasible to run in real-time for complex, high-dimensional systems. Therefore, there is a need for methods to speed up the computations of ensembles.

To speed up the computation of ensembles, surrogate models are used to approximate the forward problem by replacing the full order model with a computationally cheaper alternative. Surrogate models based on proper orthogonal decomposition and dynamic mode decomposition have been developed with reasonable success[7]. However, for nonlinear, hyperbolic, and/or discontinuous problems, advanced surrogates are necessary to achieve the desired speed-up[11]. Therefore, deep learning approaches have received increased attention in efforts to obtain significant speed-ups without sacrificing essential accuracy[11–14].

[1]Scientific Computing, Centrum Wiskunde & Informatica, 1098 XG Amsterdam, The Netherlands. [2]Machine Learning, Centrum Wiskunde & Informatica, 1098 XG Amsterdam, The Netherlands. [3]Mathematical Institute, Utrecht University, 3584 CS Utrecht, The Netherlands. [4]Swammerdam Institute of Life Sciences (SILS), University of Amsterdam, 1098 XH Amsterdam, The Netherlands. ✉email: nikolaj.mucke@cwi.nl

Deep learning-based surrogate models can be designed in various ways. One approach is to make use of latent space representations. Here, high-dimensional states are mapped onto a low-dimensional latent space such that the expensive computations, such as time stepping, can be performed cheaply in this latent space. The autoencoder (AE)[15] is the principal enabling architecture for this. In the AE, an encoder network reduces the original data into a latent representation and a decoder subsequently reconstructs the original data from the latent representation. Since Ballard (1987)[15], many extensions and improvements have been developed, such as adding probabilistic priors to the latent space[16,17]. In the context of surrogate modeling for physical systems, the focus has been on ensuring that AEs learn latent representations that are suitable for downstream tasks, such as time stepping, via various regularization techniques[12,13,18]. The success of such regularization is important when embedding the surrogate model into a data assimilation framework.

Utilizing neural network surrogate models for speeding up data assimilation has been explored in various studies, see also Cheng et al. (2023)[19] for a review. In[20–23], generative deep learning has been used for high-dimensional state- and parameter estimation. However, none of these approaches performed sequential assimilation of the data. In Brunton et al. (2016)[24], deep learning was used for model discovery. In such applications, the model is a-priori unknown and is learned from observations, typically requiring a huge number of observations in space and time to recover the complete, unknown state; to this point it is not clear how these methods perform with real-time data assimilation. Similarly, in Maddison et al. (2017)[25] and Moretti et al. (2019)[26] a particle filter approach to data assimilation has been used to formulate a variational objective for training a latent space model. Hence, the latent model is trained while data is arriving, which severely limits real-time assimilation for high-dimensional, nonlinear problems with limited observations. In Silva et al. (2023)[27], a GAN set-up, combined with proper orthogonal decomposition, was used for sequential data assimilation with an approach similar to randomized maximum likelihood. However, unlike particle filters, convergence of such methods is not ensured for nonlinear cases[5].

Yet, while many deep learning-based surrogate models have been used to speed up data assimilation, there is limited work on such approaches using particle filters[28,29]. In Gonczarek and Tomczak (2016)[28] a back-constrained Gaussian process latent variable model is used to parameterize both the dimensionality reduction and latent space dynamics. In Yang et al. (2022)[29], a particle filter using a latent space formulation was presented. The approach evaluated the likelihood by iterative closest point registration fitness scores and the latent time stepping was mainly linear. Since we are only dealing with highly nonlinear PDE-based problems with very few observations in space and time in this paper, we have chosen not implement and compare with these methods, as it is unlikely that linear time stepping will be sufficient.

Closest to our work are the works of Cheng et al. (2023), Zhang et al. (2022), Peyron et al. (2021), and Chen et al. (2023)[30–33], where real-time data assimilation through deep learning was also the aim. Autoencoders are used for dimensionality reduction and a secondary neural network is used for time stepping in the latent space. Subsequently, they perform data assimilation in the latent space. However, in Cheng et al. (2023)[30] and Zhang et al. (2022)[31] variational data assimilation is employed, therefore not quantifying the corresponding uncertainty, while in Peyron et al. (2021)[32] and Chen et al. (2023)[33] variations of the ensemble Kalman filter are used, thereby restricting the involved distributions to be Gaussians.

Here, we propose a deep learning framework for performing particle filtering in real-time using latent-space representations: the Deep Latent Space Particle Filter, or *D-LSPF*, targeting complex nonlinear data assimilation problems modeled by PDEs. For this, we develop a novel extension to the vision transformer layer for dimensionality reduction of the high-dimensional state in an AE setup. A transformer-based network is then used for parameterized time stepping, which enables filtering in the latent space as well parameter estimation. To ensure that the latent space has the appropriate desirable properties, we combine several regularization techniques such as divergence and consistency regularization[17,18]. The proposed methodology differs from earlier methods in multiple ways. It focuses on real-time data assimilation by training the neural networks in an offline stage before assimilating the incoming data. Furthermore, by utilizing particle filters instead of ensemble Kalman filters and variational approaches, the uncertainty can be quantified independent of what distributions the observations, prior, or posterior follow. Moreover, the use of rather involved neural network architectures facilitates the use of the present methodology for nontrivial problems, including highly nonlinear, high-dimensional, and even discontinuous problems. We showcase the D-LSPF on three distinct test problems with varying characteristics, such as discontinuity, few observations in space and time, parameter estimation, and highly oscillatory real-world data. In all cases, the D-LSPF demonstrates significant speed-ups compared to alternative methods without sacrificing accuracy. This promises to enable true or near real-time data assimilation for new, more complex classes of problems, with direct applications in engineering such as leak localization as well as seabed and wave height estimation.

The paper is organized as follows. In the second section we describe the problem setting. This consists of a brief description of the Bayesian filtering problem and an overview of the particle filter. In the next section, we present the D-LSPF. Firstly, we outline the latent filtering problem, followed by a description of the latent space regularized AE using the novel transformer-based dimensionality reduction layers, and parameterized time stepping. Lastly, we showcase the performance of the D-LSPF on three test cases, namely the viscous Burgers equation, harmonic wave generation over a submerged bar, and leak localization for multi-phase flow in a pipe. The D-LSPF is compared with a high-fidelity particle filter and the Reduced-Order Autodifferentiable Ensemble Kalman Filter (ROAD-EnKF)[33].

Problem setting

We consider problems that are modeled by time-dependent PDEs. Such problems consist of a state, typically made up by several quantities such as velocity and pressure, and parameters, source terms, boundary conditions, and initial conditions. Since the model won't be perfect and true values of the parameters are rarely known, the problem needs to be accompanied by observations coming from a series of sensors. However, sensors deliver noisy data and are often scarcely placed in the domain of interest, so that the data needs to be assimilated into the model to yield an accurate estimate of the state and parameters.

Consider a time and spatially discretized PDE, with accompanying observations:

$$\begin{aligned} \boldsymbol{q}_n &= F(\boldsymbol{q}_{n-1}; \boldsymbol{m}_{n-1}) + \boldsymbol{\xi}_{n-1}, & \boldsymbol{\xi}_{n-1} &\sim P_\xi(\boldsymbol{\xi}_{n-1}), \\ \boldsymbol{m}_n &= G(\boldsymbol{m}_{n-1}) + \boldsymbol{\zeta}_{n-1}, & \boldsymbol{\zeta}_{n-1} &\sim P_\zeta(\boldsymbol{\zeta}_{n-1}), \\ \boldsymbol{y}_n &= h(\boldsymbol{q}_n) + \boldsymbol{\eta}_{n-1}, & \boldsymbol{\eta}_{n-1} &\sim P_\eta(\boldsymbol{\eta}_{n-1}), \end{aligned} \tag{1}$$

where F is a (nonlinear) operator advancing the state, G is an operator advancing the parameters, $\boldsymbol{q}_n(\boldsymbol{m}_n) \in \mathbb{R}^{N_x}$ is a parameter-dependent state at time step n, $\boldsymbol{m}_n \in \mathbb{R}^{N_m}$ are the parameters, $\boldsymbol{y}_n \in \mathbb{R}^{N_o}$ is an observation vector at time step n, $h : \mathbb{R}^{N_x} \to \mathbb{R}^{N_o}$ is the observation operator, $\boldsymbol{\xi}_n$ is the model error, $\boldsymbol{\zeta}_n$ is the parameter error, and $\boldsymbol{\eta}_n$ is the observation noise. Note that when parameters are constant in time, we use $G(\boldsymbol{m}_n) = \boldsymbol{m}_n$. To simplify notation, we introduce the combined state-parameter variable, $\boldsymbol{u}_n = (\boldsymbol{q}_n, \boldsymbol{m}_n)$. We will refer to $\boldsymbol{u}_n$ as the augmented state and introduce the notation for time series, $\boldsymbol{u}_{0:n} = (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_n)$. We further assume that the probability density functions exist and will therefore continue with the derivations using the densities.

The goal is to compute the posterior density of the augmented state given observations, i.e., $\rho(\boldsymbol{u}_{0:N_t}|\boldsymbol{y}_{0:N_t})$. Based on the formulation as a filtering problem, it can be solved sequentially as observations become available. This leads to the filtering distribution, $\rho(\boldsymbol{u}_{n+1}|\boldsymbol{y}_{0:n+1})$. Bayes' theorem then gives us:

$$\rho(\boldsymbol{u}_n|\boldsymbol{y}_{0:n}) = \frac{\rho(\boldsymbol{y}_n|\boldsymbol{u}_n)\rho(\boldsymbol{u}_n|\boldsymbol{y}_{0:n-1})}{\rho(\boldsymbol{y}_n|\boldsymbol{y}_{0:n-1})}. \tag{2}$$

The problem at hand is to compute equation (2) as observations become available. The posterior density is not analytically tractable, so we must resort to numerical approximations.

We will make use of the particle filter, also referred to as sequential Monte Carlo method, where one aims to sample from the posterior instead of computing it[34]. The approximation is performed by creating an ensemble of augmented states (particles) and advancing each augmented state in time using the prior distribution. The posterior is then approximated by the empirical density, made up of N particles,

$$\rho^N(\boldsymbol{u}_n|\boldsymbol{y}_{0:n}) = \sum_{i=1}^{N} w_n^i \delta\left(\boldsymbol{u}_n - \boldsymbol{u}_n^i\right), \tag{3}$$

where $\boldsymbol{u}_n^i$ represents particle i at time step n and w_n^i is its corresponding weight. δ is the Dirac delta function, which gives $w_n^i \delta(x) = w_n^i$ for $x = 0$ and zero otherwise. $\rho^N(\boldsymbol{u}_n|\boldsymbol{y}_{0:n})$ can therefore be considered a discrete approximation of the true posterior. The computation of the weights is done by the specific choice of particle filter. A common choice is the bootstrap filter which makes use of importance sampling, originally described in[9].

The bootstrap filter assimilates data by advancing each particle using the prior distribution and assigning a weight to each. The weights are the normalized likelihoods, computed by evaluating the observation noise density at the residual between the observations and the particles. The weights are then normalized and used to resample the particles using a multinomial distribution with replacement. Note, however, that resampling when a new observation becomes available can lead to poor variablity in the ensemble. Therefore, resampling only occurs when the effective sampling size, $ESS = 1/\sum_{i=0}^{N}(w_n^i)^2$, is below a certain threshold, λ_{ESS}, typically chosen as $N/2$.

We will refer to the particle filter solution of equation (1) as the high-fidelity (HF) solution as it is the most accurate solution available.

The prior distribution is sampled by time stepping in the underlying discretized PDE and adding random noise. For high-dimensional problems, it is generally not feasible to run a particle filter in real-time as several thousands of particles are needed to accurately approximate the posterior, since the true model error is typically unknown. Therefore, reduced order models are often employed to speed up the computations at the cost of accuracy and training time.

Methodology

Here, we present the proposed methodology for real-time data assimilation with particle filters—the Deep Latent Space Particle Filter (D-LSPF).

At its heart, we represent the high-fidelity state in a more compact and cheaper to compute latent space and perform the data assimilation in the latent space. We then compute a posterior distribution over the latent state after which we transform back to the high-fidelity space to obtain the high-fidelity posterior. For this, we employ an autoencoder (AE) to reduce to the latent state, which we combine with a latent time stepping model to advance the latent state. AEs consist of two neural networks: An encoder, $\phi_{\text{enc}} : \boldsymbol{q} \mapsto \boldsymbol{z}$, that reduces the dimension of the data to a latent state, and a decoder, $\phi_{\text{dec}} : (\boldsymbol{z}, \boldsymbol{m}) \mapsto \tilde{\boldsymbol{q}}$, that reconstructs the data. The AE is trained by minimizing the Mean Squared Error (MSE) loss between the input and the reconstruction—for the parameter dependent cases, we include the parameters in the decoder, which increases the reconstruction accuracy[35]. The encoder and the decoder are then used to represent equation (1) in the latent space:

$$\begin{aligned} \boldsymbol{z}_n &= f(\boldsymbol{z}_{n-1}; \boldsymbol{m}_{n-1}) + \hat{\boldsymbol{\xi}}_{n-1}, & \hat{\boldsymbol{\xi}}_n &\sim P_{\hat{\xi}}(\hat{\boldsymbol{\xi}}_n), \\ \boldsymbol{m}_n &= G(\boldsymbol{m}_{n-1}) + \boldsymbol{\zeta}_{n-1}, & \boldsymbol{\zeta}_n &\sim P_{\zeta}(\boldsymbol{\zeta}_n), \\ \boldsymbol{y}_n &= h(\phi_{\text{dec}}(\boldsymbol{z}_n, \boldsymbol{m}_n)) + \boldsymbol{\eta}_n, & \boldsymbol{\eta}_n &\sim P_{\eta}(\boldsymbol{\eta}_n), \end{aligned} \tag{4}$$

Equation (4) differs from Eq. (1) in three ways:

- $\boldsymbol{q}_n$ is replaced by $\boldsymbol{z}_n$—we advance the latent state instead of the high-fidelity state in time;
- $\boldsymbol{\xi}_n$ is replaced by $\hat{\boldsymbol{\xi}}_n$—the latent time stepping model introduces a model error that differs from the high-fidelity model error;
- $\boldsymbol{q}_n = \phi_{\text{dec}}(\boldsymbol{z}_n)$ is added—we need to decode the latent state to get synthetic observations in the high-fidelity space. Note that the decoder may also take the parameters as input. This is the case when a supervised AE is used. If an unsupervised AE were used, the parameters would not be used as input to the decoder.

It is worth noting that the error terms $\boldsymbol{\zeta}_{n-1}$ and $\boldsymbol{\eta}_n$ are not affected by the transformation to the latent state. This is because both the model parameters and the observations are still in the same space as in Eq. (1). One could, however, modify the observation noise to account for the potential errors associated with the decoder. This is subject to future work.

With the augmented latent state, $\boldsymbol{a}_n = (\boldsymbol{z}_n, \boldsymbol{m}_n)$, the high-fidelity posterior density is replaced by the latent posterior density, $\rho(\boldsymbol{a}_{0:N_t}|\boldsymbol{y}_{0:N_t})$. Formulating the problem as a filtering problem, the sequentially defined posterior density is given by:

$$\rho(\boldsymbol{a}_n|\boldsymbol{y}_{0:n}) = \frac{\rho(\boldsymbol{y}_n|\boldsymbol{a}_n)\rho(\boldsymbol{a}_n|\boldsymbol{y}_{0:n-1})}{\rho(\boldsymbol{y}_n|\boldsymbol{y}_{0:n-1})}. \tag{5}$$

The latent prior density is then computed by:

$$\rho(\boldsymbol{a}_n|\boldsymbol{y}_{0:n-1}) = \int \rho(\boldsymbol{a}_n|\boldsymbol{a}_{n-1})\rho(\boldsymbol{a}_{n-1}|\boldsymbol{y}_{0:n-1})\mathrm{d}\boldsymbol{a}_{n-1}, \tag{6}$$

which is an integral of much lower dimension than the high-fidelity equivalent. The latent likelihood is computed by:

$$\rho(\boldsymbol{y}_n|\boldsymbol{a}_n) = \rho_{\eta_n}\big(\boldsymbol{y}_n - h(\phi_{\text{dec}}(\boldsymbol{a}_n))\big), \tag{7}$$

which is faster to evaluate than the high-fidelity equivalent, since $\phi_{\text{dec}}(\boldsymbol{a}_n))$ is fast to compute once f and ϕ_{dec} have been trained. Eqs. (5), (6), and (7) are approximated using the particle filter, as for the high-fidelity equations. The computationally expensive part of the particle filter, namely the time stepping, is performed efficiently in the latent space.

Here, we make use of the bootstrap particle filter; other types of particle filter algorithms could possibly be used instead. An outline of the D-LSPF algorithm is shown in Algorithm 1 and in Fig. 1.

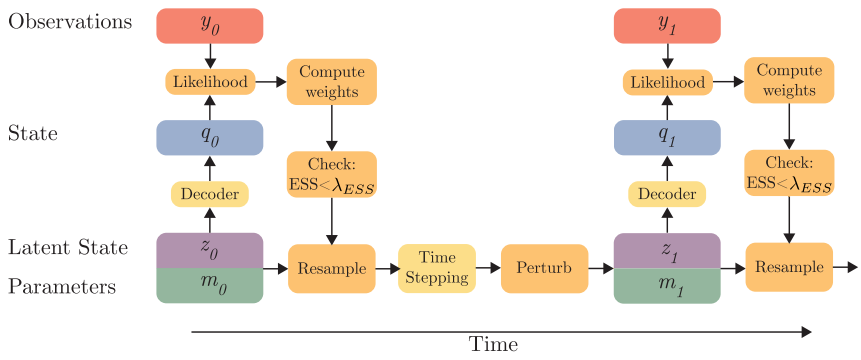

Figure 1. Schematic of the D-LSPF.

Input: Trained autoencoder=$(\phi_{\text{enc}}, \phi_{\text{dec}})$, trained time stepping network=f, ensemble size=N

Compute resample threshold, $\lambda_{ESS} = N/2$;

Encode N initial conditions, $\{z_0^i\}_{i=1}^N = \{\phi_{\text{enc}}(u_0^i)\}_{i=1}^N$;

Initialize weights, $\{w_0^i\}_{i=1}^N = \{1/N\}_{i=1}^N$;

while *new data,* y_n*, arrives* **do**

  Time-step latent states, $\{z_n^i\}_{i=1}^N = \{f(z_{n-1}^i) + \hat{\xi}^i_{n-1}\}_{i=1}^N$;

  Decode latent states, $\{u_n^i\}_{i=1}^N = \{\phi_{\text{dec}}(z_n^i)\}_{i=1}^N$;

  Compute weights, $\{\tilde{w}_n^i\}_{i=1}^N = \{\rho_{\eta_n}(y_n - h(u_n^i))w_{n-1}^i\}_{i=1}^N$;

  Normalize weights = $\{w_n^i\}_{i=1}^N = \left\{\tilde{w}_n^i / \sum_{j=0}^N \tilde{w}_n^j\right\}_{i=1}^N$;

  if $1/\sum_{i=0}^N (\tilde{w}_n^i)^2 < \lambda_{ESS}$ **then**

    Resample latent states, $\{z_n^i\}_{i=1}^N$ with computed weights

  end if

end while

Output: Assimilated latent ensemble: $\{z_{0:n}^i\}_{i=1}^N$, decoded ensemble: $\{u_{0:n}^i\}_{i=1}^N = \{\phi_{\text{dec}}(z_{0:n}^i)\}_{i=1}^N$

Algorithm 1. D-LSPF (based on the Boostrap particle filter)

Note that the speed-up in running the particle filter in the latent space comes at a cost of a training stage and the cost of encoding and decoding (which is fast due to parallel computations on a GPU). This is, however, not a significant drawback as the training takes place offline and the AE and time stepping network can be used numerous times after training. If the system under consideration appears to be prohibitively large, the necessary training time and amount of training data may make the problem intractable for the proposed method. Furthermore, the method requires the existence of a low-dimensional solution manifold, which makes its use for chaotic systems troublesome.

Latent space regularized autoencoder

For the D-LSPF to function efficiently, the latent space needs to satisfy certain properties. Firstly, the latent space must be smooth enough: to ensure that the latent space perturbations are meaningful, two states that are close to each other in the high-fidelity space must also be close in the latent space. With smoothness thus defined, we enforce this property using a prior distribution on the learned latent space in the form of a Wasserstein autoencoder (WAE)[17] using the maximum mean discrepancy (MMD) loss term—the variational autoencoder (VAE)[16] could serve the same purpose; however, the VAE tends to also smoothen the reconstructions which is undesirable in our settings. For a training dataset, $D_q = \{q_1, \ldots, q_N\}$, of high-fidelity states, the MMD loss term is given by

$$\text{MMD}\big(\phi_{\text{enc}}; D_q\big) = \frac{1}{N(N-1)} \sum_{l \neq j}^{N} \big[k(z_l, z_j) + k(\phi_{\text{enc}}(q_l), \phi_{\text{enc}}(q_j))\big] + \frac{2}{N^2} \sum_{l,j}^{N} k(z_l, \phi_{\text{enc}}(q_j)), \tag{8}$$

where $z_i \sim N(0,1)$, for $i = 1, \ldots, N$, are sampled in each batch, and k is a kernel function chosen to be the following in this work[17]:

$$k(z_l, z_j) = \sum_{s \in S} \frac{s^2}{s^2 + ||z_l - z_j||_2^2}, \quad S = \{0.2, 0.5, 0.9, 1.3\} \tag{9}$$

Secondly, the autoencoder should ensure that latent space trajectories are simple and easy to learn by a time stepping neural network. We achieve this by adding a consistency regularization term, as in Wan et al. (2023)[18], which ensures that the time evolution of the latent state can be modeled by means of an ODE system and thus promotes differentiability, and thereby smoothness, of the time evolution map. The consistency regularization term is given by:

$$C(\phi_{\text{enc}}, \phi_{\text{dec}}) = \sum_{i=1}^{N} |z_i - \phi_{\text{enc}}(\phi_{\text{dec}}(z_i))|^2. \tag{10}$$

As in Eq. (8), z_i for $i = 1, \ldots, N$, are sampled from a standard normal distribution. In summary, the complete loss function is given by:

$$L_{\text{WAE}}(\phi_{\text{enc}}, \phi_{\text{dec}}; D_q) = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\big(\boldsymbol{q}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{q}_i))\big)^2}_{\text{Reconstruction}} + \underbrace{\alpha R_{AE}(\phi_{\text{enc}}, \phi_{\text{dec}})}_{\text{Weight regularization}} + \underbrace{\beta \text{MMD}\big(\phi_{\text{enc}}; D_q\big)}_{\text{Divergence}} + \underbrace{\lambda C(\phi_{\text{enc}}, \phi_{\text{dec}})}_{\text{Consistency}}. \tag{11}$$

The regularization, $R_{AE}(\phi_{\text{enc}}, \phi_{\text{dec}})$, is a weight regularization term that aims to ensure generalization beyond the training set. In this work, we choose the l^2 norm of the neural network weights. The parameters α, β, and λ, are considered hyperparameters and are determined through hyperparameter tuning. Furthermore, the dimension of the latent space is problem dependent and is chosen via hyperparameter tuning.

Remark There are several alternative approaches to achieve similar effects as the divergence and consistency regularization terms. For example, in Geneva & Zabaras (2022)[12] a Koopman-based embedding is utilized so that the latent dynamics become suitable for time stepping. However, despite extensive experimentation, we obtained results of significantly lower quality than with the above mentioned approaches.

Transformer-based dimensionality reduction

In our approach, the loss function ensures that the autoencoder, and thereby the latent space, has certain desirable properties. To ensure that the autoencoder can learn a low-dimensional representation and reconstruct it accurately, the architecture also has to be able to handle a multitude of possible high-fidelity states.

The arguably most common layers for AEs are convolutional and pooling layers[12,14,18]. Convolutional layers however tend to have inherent inductive biases and struggle with discontinuous signals, resulting in spurious oscillations. Transformers, originally developed for text processing, have proven effective for image processing in the form of vision transformers (ViT)[36]. These transformers divide images into patches and apply the attention mechanism between each set of patches. Yet, since there is no natural way of reducing or expanding the dimensionality of the data, the ViT has been used to dimensionality reduction tasks only to a limited degree[37–40]. Importantly, current ViTs have not been integrated with increasing numbers of channels in convolutional layers to represent increasingly complicated features. In this section, we extend the vision transformer layer to combine the advantages of convolutional layers (i.e., dimensionality reduction/expansion and increasing/decreasing number of channels) and ViTs (global information, patch processing).

Before presenting the proposed layer, we briefly describe the standard transformer layer. The main component of which is the so-called scaled dot-product attention. For a matrix, $X \in \mathbb{R}^{c\times d}$, the scaled dot-product attention is computed by:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V, \quad K = F_k(X) \in \mathbb{R}^{c\times d}, \quad Q = F_q(X) \in \mathbb{R}^{c\times d}, \quad V = F_v(X) \in \mathbb{R}^{c\times d}, \tag{12}$$

where K is denoted the keys, Q the queries, and V the values. F_k, F_q, and F_v are shallow neural networks to be trained. c is the so-called context length and d the embedding dimension. The context length refers to the number of elements in the input sequence, e.g., words in a sentence or pixels in an image. By defining multiple F_k, F_q, and F_v networks, we can compute several attention maps between the different embeddings in parallel and then concatenate the attention maps along the dth dimension. Each of these attention computations is called a head. The number of heads is typically considered a hyperparameter. By connecting the attention layer to a residual connection, a normalization layer, a dense neural network, another residual connection and another normalization, we form the transformer encoder module, see Fig. 2 for a visualization. We should keep in mind that the term transformer encoder refers to a naming convention of this layer and should not be confused with the AE encoder. The transformer layer is unaware of the relative positions of the individual nodes. Therefore, a positional encoding is added to the features before passing them through the transformer.

The attention mechanism scales quadratically with the context length. Therefore, it is infeasible for large images or when real-time inference is of importance. The vision transformer layer overcomes this by dividing the data into a number of patches, where each patch is flattened and embedded, after which the attention between the patches is computed.

In the proposed layer, the expansion/reduction of channels and dimensions is done on each individual patch. It can be interpreted as a type of domain decomposition, where the reduction/expansion is performed on each subdomain and the communication between subdomains is handled through the attention mechanism. Formally,

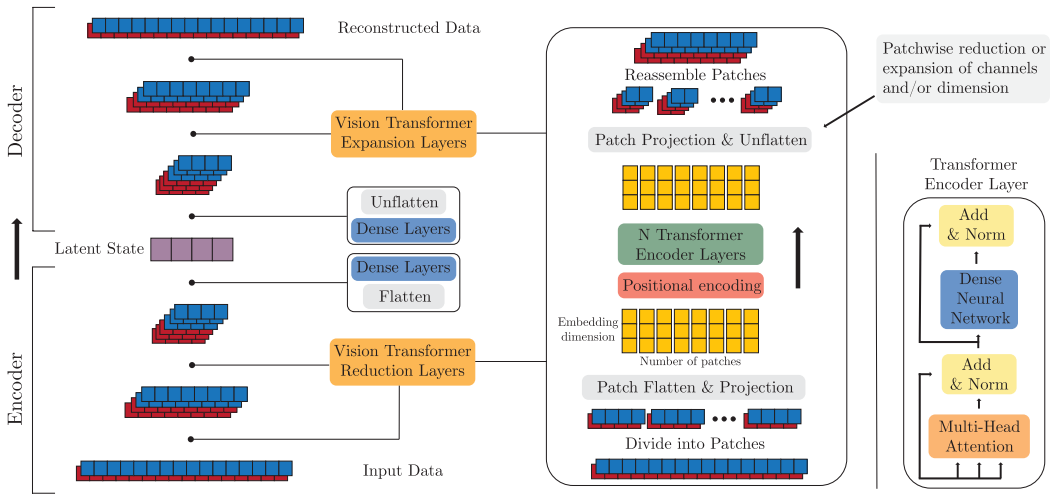

Figure 2. Visualization of the ViT dimensionality reduction/expansion layer.

let the superscript, l, denote the l'th layer. We divide an input $x^l \in \mathbb{R}^{N_c^l \times N_x^l}$ (N_c^l channels and a spatial dimension of size N_x^l), into p patches, $x_1^l, x_2^l, \ldots, x_p^l$ of size N_p^l. That is, $x_i^l \in \mathbb{R}^{N_c^l \times N_p^l}$, for all i. Then, each patch is flattened and projected onto an N_e^l-dimensional embedding space, $e = (e_1, e_2, \ldots, e_p) \in \mathbb{R}^{N_e \times N_p}$. Positional encodings are then added after which the embeddings are passed through a standard transformer encoder layer. Each embedded vector, e_i, is projected onto a new dimension of size $N_c^{l+1} N_p^{l+1}$, unflattened, $x_i^{l+1} \in \mathbb{R}^{N_c^{l+1} \times N_p^{l+1}}$ and recombined, $x^{l+1} \in \mathbb{R}^{N_c^{l+1} \times N_x^{l+1}}$. The process is visualized in Fig. 2.

Time stepping

Once the AE is trained, we can transform high-fidelity trajectories into latent trajectories. To compute the latent trajectories, we next need to perform time stepping in the latent space. For this, we make use of transformers[41] as they are well suited for modeling physical systems, being able to mimic the structure of multistep time-marching methods[12].

Time stepping in the latent space is done by means of a map, f, that advances a latent state in time. Adopting the concept of multistep time integrators, we use several previous time steps to predict the next latent state:

$$\boldsymbol{z}_{n+1} = f(\boldsymbol{z}_{n-k:n}; \boldsymbol{m}_n), \tag{13}$$

where k is referred to as the memory. For multiple time steps, we apply the transformer model recursively. Training is done by minimizing the loss function:

$$L(f; D_z) = \sum_{n=k}^{N_t - s} \sum_{i=1}^{s} ||f^i(\boldsymbol{z}_{n-k:n}; \boldsymbol{m}_n) - \boldsymbol{z}_{n+1:n+1+i}||_2^2 + \alpha R_f(f),$$

where $D_z = \{z_1, \ldots, z_{N_t}\}$ is the training dataset of latent states, R_f is a weight regularization term similar to R_{AE} in Eq. (11), f^i means applying f i times, recursively, on the output, and s is the output sequence length. In this paper, we choose R_f to be the l^2 norm of the neural network weights. After trajectories are computed in the latent space, high-fidelity trajectories are recovered through the decoder.

In the high-fidelity space, dynamics are not only dependent on the previous state but also on a set of parameters, and the same applies to the latent dynamics. Including the parameters of interest in the latent space time stepping model can be done in several ways, depending on the specific choice of neural network architecture. We adopt the approach presented in[42], where the parameters are encoded and added to the sequence of states as the first entry:

$$\{g(\boldsymbol{m}_n), \boldsymbol{z}_{n-k}, \boldsymbol{z}_{n-k+1}, \ldots, \boldsymbol{z}_n\} \to f(\{g(\boldsymbol{m}_n), \boldsymbol{z}_{n-k}, \boldsymbol{z}_{n-k+1}, \ldots, \boldsymbol{z}_n\}) = \boldsymbol{z}_{n+1}, \tag{14}$$

where g is a parameter encoder that lifts a vector of parameters to the same dimension as the latent state. This efficiently allows attention to be computed between the parameters and the sequence of states. Figure 3 visualizes the time stepping transformer model.

Results and discussion

We demonstrate the potential and strength of the D-LSPF for a variety of numerical experiments. The first test case serves as a simple benchmark problem. The second test case uses real-world data from an experimental setting, and shows that the D-LSPF can be applied to real-world situations even when trained on simulation data. The last test case is a realistic engineering setting and is used as an ablation study to emphasize the performance of the architectural choices. An overview of the test cases can be found in Table 1.

For all neural networks, hyperparameter tuning was performed to find the optimal settings. For both the AE and the time stepping neural network, the performance was measured with a validation dataset that was different from both the training and test datasets. For the WAE we measured the reconstruction error with the MSE, and for the time stepping neural network we measured the MSE of the predicted latent states. The regularization parameters α, β, and γ, as well as the number of channels in each layer, learning rate, and latent space dimension were considered hyperparameters for the WAE. For the time stepping neural network, the number of previous time steps, regularization parameter α, the number of transformer layers, and the transformer embedding dimension were considered hyperparamters. In general, we found that the time stepping network was not sensitive to

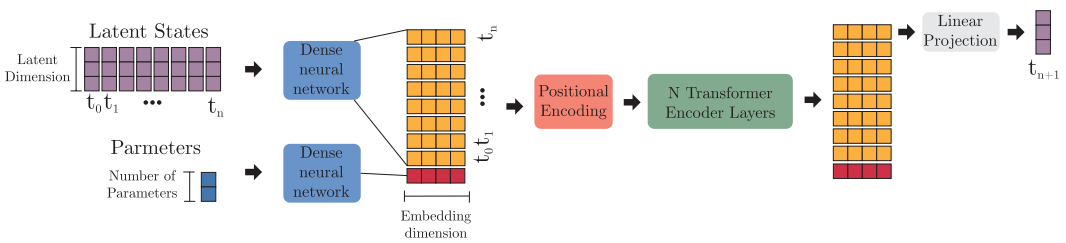

Figure 3. Illustration of the transformer model for parameterized time stepping.

	Burgers	Multi-phase pipeflow	Waves over submerged bar
Num train samples	1024	5000	210
Num test samples	20	8	1
Parametric	No	Yes	Yes
Simulated observations	Yes	Yes	No
Noise variance	0.01	10^{-5} (pressure^2 [bar^2])	–
Num sensors	8	9	8
Num time steps between obs	30	400	[25, 75]
State degrees of freedom	256	1536	1024
Num states	1	3	2
Num states observed	1	1	1
Likelihood variance	0.01	10^{-5} (pressure^2 [bar^2])	7.5×10^{-5} (wave height^2 [m^2])
Latent model error, $\hat{\xi}$, variance	10^{-8}	10^{-5}	10^{-5}
Parameter perturbation, ζ, variance	10^{-10}	0.0	10^{-6}
Parameter prior distribution	-	$C_d \sim U[1,2]$, $x_l \sim U[10, 5990]$	$U[0.07, 0.35]$

Table 1. Overview of test cases.

the choice of hyperparameters, while the WAE required more tuning. Around 20 runs were used to determine the final choice of hyperparameters for the WAE for all test cases and for the time stepping network around 5 runs were used. While this is, in general, considered a relatively small number of runs, it appeared sufficient to get the desired performance. For the alternative methods we compare with, we adopt hyperparameters as chosen in the respective papers when applicable, and performed hyperparameter tuning when not.

All neural networks were implemented using PyTorch[43]. The modified ViT layers are implemented by modifying the code from https://github.com/lucidrains/vit-pytorch. Training and testing were performed using an Nvidia RTX 3090 GPU and 32 core AMD Ryzen 9 3950X CPU. All models are trained with the Adam optimizer[44] and a warm-up cosine annealing learning rate scheduler. Gradient clipping was applied when training the transformers. The states and parameters were transformed to be between 0 and 1 before being passed to the autoencoder. The time stepping networks are trained without teacher forcing, and with a limited unrolling.

Training of the WAE costs approximately 1 hour for the viscous Burgers equations, 16 hours for the harmonic wave generation over a submerged bar test case, and 48 hours for the multi-phase leak localization test case. Similar training time was spent for the time stepping neural networks. All the training was performed using 1 GPU.

Importantly, we focus on the performance of the online data assimilation by the dimensionality reduction and time stepping. Therefore, we do not go into great detail about the stability and generalization aspects of the methodology. These topics were however discussed in detail in e.g. Mücke (2021)[14] and Geneva and Zabaras (2022)[12]. Stability and generalization were tested using validation datasets, where we observed that the neural networks generalized beyond the training data and the time stepping appeared sufficiently stable to predict an entire time horizon of interest without assimilating data. This suggests that the surrogate models are suitable for the test cases considered even when observations are sparse in time. Testing how the surrogate models would perform beyond the training horizon is beyond the scope of this work. Furthermore, in many applications, the possible outcomes are relatively well-understood and can be represented well in the training data and we found no reason to assume that the trained models would deteriorate if the training horizon were expanded further. It should be noted that the neural networks were trained with relatively large datasets and sufficient training time. Analyzing whether similar results can be achieved with less data and training time is subject to future work.

We compare the D-LSPF with a high-fidelity particle filter and the Reduced-Order Autodifferentiable Ensemble Kalman Filter (ROAD-EnKF) method[33]. We adopt the same architectures as presented in the original paper, namely a Fourier decoder network for decoding the latent state and dense neural network for time stepping. The ROAD-EnKF has already been compared with alternative methods and has shown superior performance in the original paper. Hence, it serves as a suitable representation of the state-of-the-art. Furthermore, with this comparison we show the increased accuracy one can achieve by using particle filters rather than ensemble Kalman filters, even when the ensemble Kalman filter is used in conjunction with neural networks. In particular, this increased accuracy can be achieved without sacrificing the real-time constraints due to the low-dimensional latent space and surrogate model of the D-LSPF.

Viscous Burgers equation

The first test case is the viscous Burgers equation:

$$\begin{aligned} &\partial_t q(x,t) = \nu \partial_{xx} q(x,t) - q(x,t)\partial_x q(x,t), \\ &q(0,t) = q(L,0) = 0, \\ &q(x,0) = Q \sin\left(\frac{2\pi x}{L}\right), \end{aligned} \tag{15}$$

with $x \in [0, L]$, $L = 2$, $\nu = 1/150$, and $Q \sim \mathrm{U}[0.5, 1.5]$. We only perform state estimation so neither the AE nor the time stepping NN receives any parameters as input. The observations used for the data assimilation are simulated, as well as the training data. We add normally distributed noise with a standard deviation of 0.1. equation (15) is discretized using a second-order finite difference scheme in space and a Runge-Kutta 45 method in time. The observations are simulated using the same discretization as the training data. We consider $t \in [0, 0.3]$ with a step size of 0.001, resulting in 300 time steps. For the data assimilation, we test on a case where the state is observed at 8 spatial locations, $x = (0.0, 0.286, 0.571, 0.857, 1.143, 1.429, 1.714, 2.0)$, $N_y = 8$, at every 10th time step. The latent dimension in the D-LSPF is chosen to be 16.

We compare the D-LSPF with 100 and 1000 particles with the ROAD-EnKF with a latent dimension of 40. The hyperparameters are taken from the original paper as they also consider the viscous Burgers equation. The ROAD-EnKF is trained on the same training data as the D-LSPF with full access to the entire states in space and time. All methods are evaluated on 20 different simulated solutions, with $Q \sim \mathrm{U}[0.5, 1.5]$. We compare the performance by computing the root-mean-square error (RMSE) and the averaged RMSE of the 2nd, 3rd, and 4th moment of the state ensemble, referred to as the average moment RMSE (AMRMSE). The AMRMSE measures how accurately the distributional information of the posterior is approximated and therefore how accurately uncertainty is quantified. Moreover, we also present the negative log-likelihood (NLL) with respect to the high-fidelity posterior.

In Table 2, the results for the test case are shown. The D-LSPF shows superior performance with respect to AMRMSE and NLL by one order of magnitude, suggesting that the D-LSPF quantifies the uncertainty in a more accurate way compared to ROAD-EnKF for this case. For the mean state estimation, the D-LSPF also performs 3.75 times better. It is worth noting that the D-LSPF with 100 and 1000 particles exhibit very similar performance under all metrics. For the RMSE, they are the same up to $O(10^{-2})$. Regarding the NLL, we see that the D-LSPF with 100 particles performs a bit better than with 1000 particles, which may be due to random effects. In general, the results suggest that 100 particles are sufficient. Regarding timing, the D-LSPF with 100 particles and the ROAD-EnKF are comparable using GPUs, while the ROAD-EnKF is slightly faster using a CPU.

Lastly, in Fig. 4, we show state estimation results at three different time points. It is clear that the uncertainty bands shrink as time passes and more observations become available as expected. The ROAD-EnKF state estimation is visually slightly worse than the D-LSPF and the high-fidelity particle filter.

Harmonic wave generation over a submerged bar

In this test case, the data comes from a real-world experiment[45]. The setting is a 25m long and 0.4m tall wave tank, with waves being generated from the left side, traveling to the right. At the seabed of the tank, a 0.3m tall submerged bar is placed, see Fig. 5. Eight sensors measure the surface height of the water at

	RMSE ↓	AMRMSE ↓	NLL ↓	GPU ↓	CPU ↓
HF(1000)	$\mathbf{9.0 \times 10^{-3}}$	–		–	**22.7 s**
D-LSPF(100)	$\mathbf{1.6 \times 10^{-2}}$	1.3×10^{-4}	**0.38**	**0.6 s**	1.5 s
D-LSPF(1000)	$\mathbf{1.6 \times 10^{-2}}$	$\mathbf{1.1 \times 10^{-4}}$	0.42	1.2 s	13.4 s
ROAD-EnKF (100)	6.0×10^{-2}	1.2×10^{-3}	4.73	**0.6 s**	**0.8 s**

Table 2. Results for the viscous Burgers equation, with in parenthesis the number of particles. For the high-fidelity (HF) particle filter, 30 CPUs were used in parallel. For the other methods a single CPU or GPU was used. The downward pointing arrow means lower values are better. RMSE is the root mean squared error, the AMRMSE is the average moment RMSE with respect to the high-fidelity particle posterior, and NLL is the negative log-likelihood with respect to the high-fidelity particle posterior. The AMRMSE is computed by comparing the surrogate model ensembles with the HF particle filter solution. Best values are in bold.

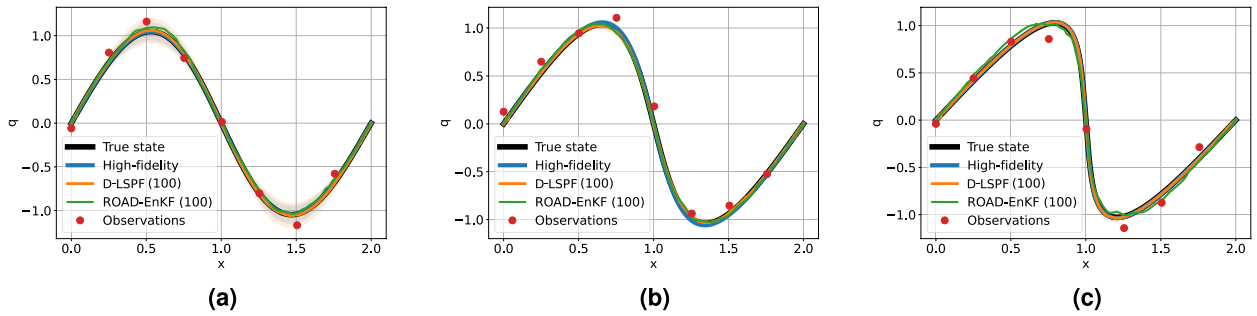

Figure 4. State estimation for the viscous Burgers equations with observations every 10 time step using the high-fidelity particle filter, the D-LSPF and the ROAD-EnKF. The high-fidelity particle filter was run with 1000 particles and the D-LSPF and ROAD-EnKF were run with 100 particles. (**a**) $t = 0.03$. (**b**) $t = 0.15$. (**c**) $t = 0.29$. It is clear that all methods approximates the state well. The ROAD-EnKF approximation is slightly off in (**c**). This difference is, however, negligible. .

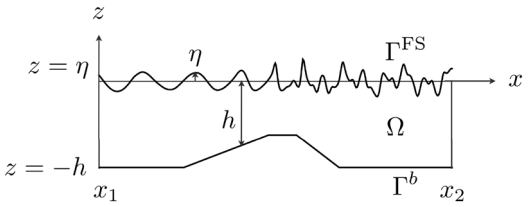

Figure 5. Wave tank setup and physical variables. Figure comes from[47].

$x = (4, 10.5, 13.5, 14.5, 15.7, 17.3, 19.0, 21.0)$. For the state and parameter estimation, we aim to reconstruct the surface elevation, the velocity potential, and the height of the submerged bar. A similar study was conducted in[46], where the uncertainty of the water wave height was quantified given random perturbations on the seabed; in[46] however only uncertainty of the forward problem was considered, whereas we solve the inverse problem with uncertainty quantification, given the observations. For the neural network surrogate model, we only consider the surface elevation, η, and the surface velocity potential, $\tilde{\phi}$, as the state and the height of the submerged bar is the parameter of interest. This highlights an important advantage of using a non-intrusive surrogate model, as it becomes possible to only model the relevant quantities of interest, bypassing the computations of the velocity potential in the vertical direction.

To generate the training data, we model the setup using the fully nonlinear water wave model for deep fluids as described in[47]. The problem is modeled in 2D by means of a set of 1D PDEs for the free surface boundary conditions, together with a 2D Laplace problem in the full domain. Let x be the horizontal component and z the vertical component, see Fig. 5. The velocity potential, $\phi : (x, z, t) \mapsto \phi(x, z, t)$, is the scalar function defined on the whole 2D domain, and the free surface elevation, $\eta : (x, t) \mapsto \eta(x, t)$, is defined only on the 1D surface. The free surface boundary conditions can be expressed in the so-called Zakharov form[48], modeled by two 1D PDEs—the wave height, η, and the velocity potential, $\tilde{\phi}$:

$$\begin{aligned} \frac{\partial \eta}{\partial t} &= -\nabla\eta \cdot \nabla\tilde{\phi} + \tilde{w}(1 + \nabla\eta \cdot \nabla\eta), \\ \frac{\partial \tilde{\phi}}{\partial t} &= -g\eta - \frac{1}{2}\Big(\nabla\tilde{\phi} \cdot \nabla\tilde{\phi} - \tilde{w}^2(1 + \nabla\eta \cdot \nabla\eta)\Big). \end{aligned} \tag{16}$$

Equation (16) is defined on the surface part of the domain, Γ^{FS}. $\tilde{w} = \partial_z\phi|_{z=\eta}$ and $\tilde{\phi} = \phi|_{z=\eta}$ are the surface parts of the functions that are defined on the 2D domain and Γ^{FS} is the free surface, as shown in Fig. 5. The velocity potential on the domain is modeled by the 2D Laplace problem, via the σ-transform:

$$\begin{aligned} \nabla^\sigma(K(x;t)\nabla^\sigma\phi) = 0, \quad &\text{in} \quad \Omega^c, \\ \phi = \tilde{\phi}, \quad z = \eta \quad &\text{on} \quad \Gamma^{FS}, \\ \mathbf{n} \cdot \nabla\phi = 0, \quad z = -h(x,y) \quad &\text{on} \quad \Gamma^b, \end{aligned} \tag{17}$$

where $\sigma = (z + h(x))d(x,t)^{-1}$, $\Omega^c = \{(x,\sigma)|0 \le \sigma \le 1\}$, and

$$K(x,t) = \begin{bmatrix} d & -\sigma\partial_x\eta \\ -\sigma\partial_x\eta & \frac{1+(\sigma\partial_x\eta)^2}{d} \end{bmatrix}. \tag{18}$$

We use the spectral element method, as described in[47], for the discretization of the equations. We use 103 elements in the horizontal direction and 1 element in the vertical direction, both with 6th order polynomials, to generate the training data. The equations are solved with a step size of 0.03535 with $t \in [0, 42.42]$, resulting in 1200 time steps, and the bar height is uniformly sampled between 0.1 and 0.325. The states are interpolated onto a regular grid of 512 points.

Sensor observations from the experiment are available at a time frequency of 0.03535s, which was also chosen as the step size for the simulations. To demonstrate how the D-LSPF performs with varying time intervals between the observations, we show the results for sensor observations at every 25th and 75th time step, corresponding to every 0.884s and 2.651s, respectively. We refer to these two settings as case 1 and 2. We only observe the wave height and not the velocity potential. Since we deal with real-world data, the true full state is not available. Therefore, we measure the accuracy against the full time series of observations, showcasing that the D-LSPF can accurately estimate the state between observations. We do, however, also compare the results with a high-fidelity simulation with the true bar height. Furthermore, we present the accuracy of the bar height estimates.

We compare the D-LSPF with the ROAD-EnKF[33], where we train the ROAD-EnKF model on the same simulated data as the D-LSPF with full access to the states in space and time. To deal with the multiple states, wave height and velocity potential, we introduce a slight modification in the decoder network in the ROAD-EnKF compared to the original paper[33], by ensuring that the Fourier decoder networks outputs data with two channels. The latent dimension is chosen to be 8 for the D-LSPF and 40 for the ROAD-EnKF. The neural network architectures for the modified ROAD-EnKF model have been chosen through hyperparameter tuning. Note that the ROAD-EnKF method is not able to perform parameter estimation.

	Case 1—every 25 time step				Case 2—every 75 time step			
	S-RRMSE ↓	S-PICP ↑	P-RRMSE ↓	Time ↓	S-RRMSE ↓	S-PICP ↑	P-RRMSE ↓	Time ↓
D-LSPF(100)	$\mathbf{3.6 \times 10^{-1}}$	4.4×10^{-1}	$\mathbf{4.4 \times 10^{-3}}$	**1.5 s**	4.3×10^{-1}	4.9×10^{-1}	5.3×10^{-3}	**1.4 s**
D-LSPF(1000)	3.8×10^{-1}	3.9×10^{-1}	4.6×10^{-3}	10.5 s	$\mathbf{4.0 \times 10^{-1}}$	5.6×10^{-1}	$\mathbf{4.9 \times 10^{-3}}$	10.0 s
ROAD-EnKF(100)	1.1	5.0×10^{-1}	–	5.4 s	1.0	$\mathbf{5.8 \times 10^{-1}}$	–	5.1 s
ROAD-EnKF(1000)	1.0	$\mathbf{5.5 \times 10^{-1}}$	–	31.3 s	1.0	$\mathbf{5.8 \times 10^{-1}}$	–	29.9 s

Table 3. Results for the harmonic wave generation test case using the D-LSPF and the ROAD-EnKF with 100 and 1000 particles. Timings are measured using a single GPU. The PICP is computed using the 2.5th and the 97.5th percentile. An upward pointing arrow means larger values are better and a downward pointing arrow means lower values are better. "S-" and "P-" refer to the state and parameters, respectively. RRMSE is the relative root mean squared error and PICP is the probability interval coverage percentage for the 95th percentile. Best values are in bold.

Table 3 contains the Relative RMSE (RRMSE), probability interval coverage percentage (PICP), and timings for the D-LSPF and the ROAD-EnKF in both variations of the test case using 100 and 1000 particles. The D-LSPF clearly performs best with respect to the state estimation with an improvement of one order of magnitude, for both 100 and 1000 particles. For the PICP, the ROAD-EnKF does slightly better, however, inspecting Figs. 6a, b, we see that the ROAD-EnKF has large uncertainty intervals while being quite inaccurate on average compared to the D-LSPF. In general, the PICP is less relevant when the RRMSE is bad. The large uncertainty intervals for the ROAD-EnKF state estimation are found because the problem is highly nonlinear and EnKF type methods are known to perform insufficiently well in such cases. Moreover, the lacking parameter estimation in the method also contributes to the large uncertainties. These results are further highlighted in Fig. 7a, b, plotting the windowed RRMSE versus time. The windowed RRMSE measures the RRMSE in a time window in order to show how the state estimation improves when more observations become available. The D-LSPF converges and even

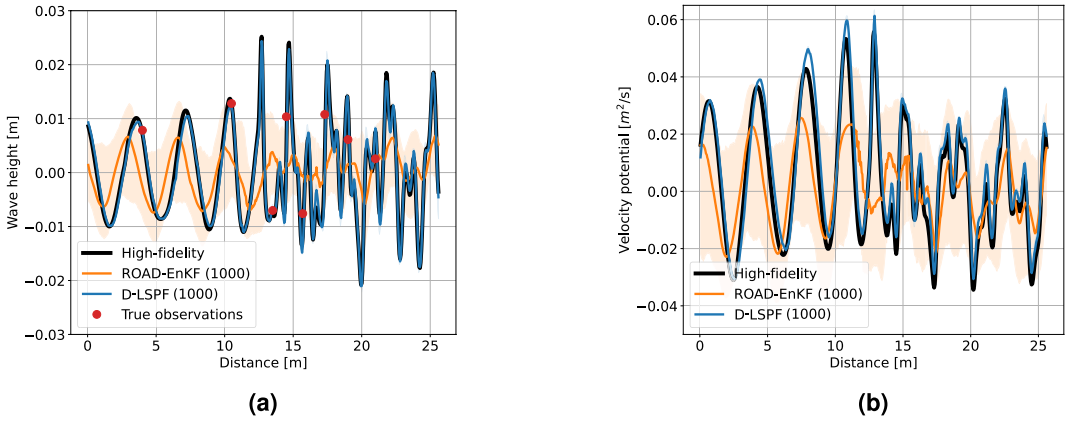

Figure 6. State estimation at t = 40 s for the harmonic wave test case with observations every 75 time steps. Note that only wave height is observed and not velocity potential. (**a**) Wave height estimation, $\eta \pm 2\sigma$. (**b**) Velocity potential estimation, $\tilde{\phi} \pm 2\sigma$.

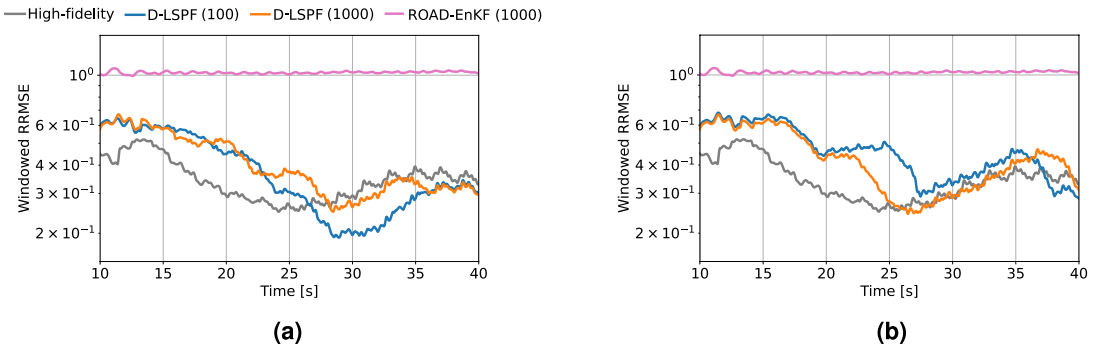

Figure 7. Windowed RRMSE for the harmonic wave test case with a window size of 75 time steps. (**a**) Observations every 25 time step. (**b**) Observations every 75 time step.

surpasses the high-fidelity simulation, showcasing how assimilating observations impactfully improves accuracy. We also notice for both the D-LSPF and ROAD-EnKF that there are only minor differences between using 100 and 1000 particles. Furthermore, the RRMSE only varies slightly between the two cases, suggesting that both methods are stable with respect to observation frequency.

Besides the accuracy, Table 3 also notes the computation time. In both cases, both the D-LSPF and the ROAD-EnKF are faster than real-time, as the data assimilation takes place over 40s in physical time and the D-LSPF and ROAD-EnKF computation times vary between 1.4 and 29.9 s. In general, the D-LSPF is between 3 and 4 times faster than the ROAD-EnKF. For comparison, a single high-fidelity model simulation takes on average some 1062s. Hence, running the particle filter using the high-fidelity model on 30 CPU cores, assuming no overhead associated with the parallelization, would take approximately 3542 s with 100 particles and 35,420 s for 1000 particles, yielding a speed-up of 2345 for 100 particles and 3376 for 1000 particles on a GPU when using the D-LSPF. When deploying the D-LSPF and running it as the data comes in, it is not possible to perform the data assimilation task faster than the arrival of observations. However, the timings show that the method assimilates the data without any delay.

Figure 8 shows the quality of the D-LSPF estimates of the state and the sensor locations between observations. Furthermore, the difference between using 100 and 1000 particles is negligible for the state estimates. However, when zooming in, it becomes clear that using more particles result in better uncertainty intervals. Lastly, in Fig. 9, the posterior distributions of the bar height for varying numbers of particles and observation frequency are shown. While the average bar height estimates are very similar for the 100 and 1000 particles, the distributions change from being multimodal to unimodal.

Multi-phase leak localization

Here, we consider leak localization in a two-phase pipe flow for liquid (water) and air, and we assume that they are mixed. We consider a case where only a few sensors are placed along a 5km long pipe, measuring pressure.

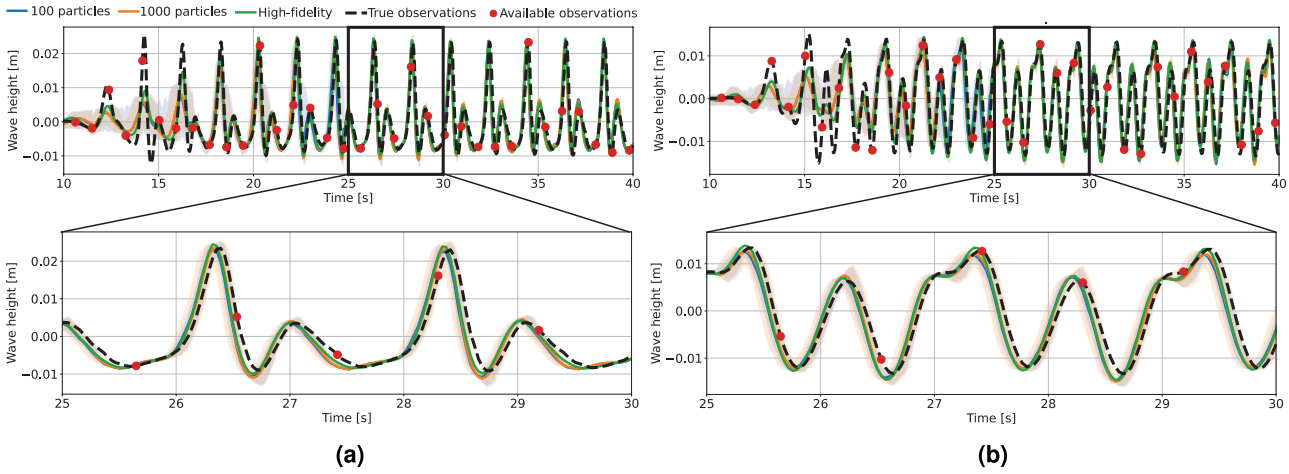

Figure 8. State estimation at two sensor locations computed with the D-LSPF with 100 and 100 particles compared with a high-fidelity simulation with the true bar height and the true sensor data. (**a**) Sensor located at 13.5m. (**b**) Sensor located at 14.5m.

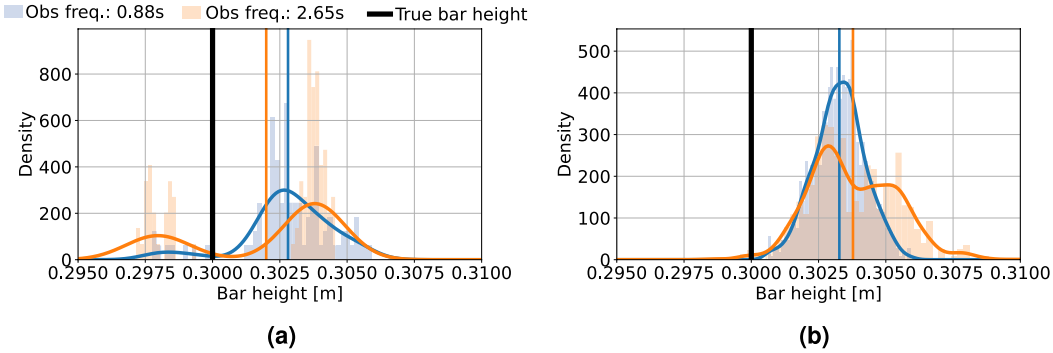

Figure 9. Posterior distribution over the bar height at $t = 40s$ for varying observation frequency and number of particles. The density outlines are computed using kernel density estimation based on the posterior particles. Vertical lines represent the mean of the distributions. (**a**) D-LSPF with 100 particles. (**b**) D-LSPF with 1000 particles.

The system is in a steady state when a leak occurs, where we then use the particle filter to compute a distribution over the leak location, leak size, liquid holdup, mixture velocity, and pressure. This problem is similar to the setup in[6], where single-phase CO_2 was considered, but state estimation was only performed in a non-leakage case.

The state for data assimilation and our surrogate model is the primitive state, consisting of the liquid holdup, pressure, and mixture velocity, $q = (\alpha_l, p, u_m)$. Both training, testing, and observations data are simulated and noise is artificially added to the observations with a standard deviation of 0.01. We use the homogeneous equilibrium model (HEM)[49] for the simulations. The HEM is a set of nonlinear, hyperbolic one-dimensional PDEs:

$$\begin{aligned} &\partial_t(A_g\rho_g) + \partial_x(A_g\rho_g u_m) = -\alpha_g L(\rho_m)\delta(x - x_l),\\ &\partial_t(A_l\rho_l) + \partial_x(A_l\rho_l u_m) = -\alpha_l L(\rho_m)\delta(x - x_l),\\ &\partial_t(A\rho_m u_m) + \partial_x\left(\rho_m u_m^2 A + p(\rho_g)A\right) = -\frac{\rho u_m|u_m|}{2} f_f(\rho, u_m), \end{aligned} \tag{19}$$

with boundary conditions $\rho_g u_m(0,t) = (\rho_g u_m)_0$ and $\rho_l u_m(0,t) = (\rho_l u_m)_0$ at the pipe inlet on the left side, and $p(L,t) = p_L$ at the outlet on the right side. f_f is the wall friction, Re is the Reynolds number, and L is the leak size. δ is the Dirac delta function, ensuring that the leak is only active at $x = x_l$. A is the cross section area of the pipe, A_g is the area occupied by gas and A_l the area occupied by liquid. Hence, for $\alpha_g \in [0,1]$ and $\alpha_l \in [0,1]$ being the fraction of gas and liquid, respectively, we have:

$$A_g = \alpha_g A, \quad A_l = \alpha_l A, \quad A = A_g + A_l, \quad \alpha_g + \alpha_g = 1. \tag{20}$$

ρ_l is the liquid density assumed to be constant, ρ_g is the gas density, and ρ_m is the mixture density (not to be confused with the probability density functions, ρ),

$$\rho_m = \alpha_g\rho_g + \alpha_l\rho_l. \tag{21}$$

The wall friction, f_f, is given by,

$$f_f = 2\left(\left(\frac{8}{Re}\right)^{12} + (a+b)^{-1.5}\right)^{1/12}, \quad a = \left(-2.457\ln\left(\frac{7}{Re}\right)^{0.9} + 0.27\frac{\varepsilon}{2r}\right)^{16}, \quad b = \left(\frac{37530}{Re}\right)^{16}, \tag{22}$$

the Reynolds number, Re, is given by

$$Re = \frac{2r\rho_m u_m}{\mu_m}, \quad \mu_m = \alpha_g\mu_g + \alpha_l\mu_l, \tag{23}$$

and the leak size, L, is given by,

$$L(\rho_m) = C_d\sqrt{\rho_m(p(\rho_g) - p_{\text{amb}})}. \tag{24}$$

For specific values of all constants in Eq. (19), see Table 4. We discretize the PDEs using the nodal discontinuous Galerkin method[50] with Legendre polynomials for the modal representation and Lagrange polynomials for the

Physical quantity	Constant	Value	Unit
Pipe length	L	5000	m
Diameter	d	0.2	m
Radius	r	0.1	m
Cross-sectional area	A	0.0314	m^2
Speed of sound in gas	c	308	m/s
Ambient pressure	p_{amb}	1.01325	bar
Reference pressure	p_{ref}	1.0	bar
Reference density (gas)	ρ_g	1.26	kg/m^3
Reference density (liquid)	ρ_l	1003	kg/m^3
Inflow velocity	v_0	4.0	m/s
Outflow pressure	p_L	10.0	bar
Pipe roughness	ε	10^{-8}	m
Fluid viscosity (gas)	μ_g	1.8×10^{-5}	$N\,s/m^2$
Fluid viscosity (liquid)	μ_l	1.516×10^{-5}	$N\,s/m^2$
Temperature	T	278	Kelvin
Discharge coefficient	C_d	[1, 2]	m
Leakage location	x_l	[10, 5990]	m

Table 4. Parameters for the multi phase pipe flow equations (19). Note that the discharge coefficient and the leakage location have values denoted by intervals, as they are the parameters to determine.

nodal degrees of freedom. We use the Lax-Friedrichs discretization for the numerical flux and BDF2 for time stepping. The nonlinear equations coming from the implicit time stepping are solved using Newton's method, where the resulting linear systems are solved via an LU factorization. Furthermore, the Jacobian matrix is only updated every 500 time-steps to speed up the computations.

The true state and observations used for the data assimilation are simulated using 3000 elements and third-order polynomials. The states are then evaluated on regular grid of 512 points. The training data is simulated using 2000 elements and second-order polynomials. Thereafter, it is evaluated on a regular grid consisting of 512 grid points. The equations are solved with a time-step size of 0.01, for $t = [0, 120]$ s. Hence, there are 12,000 time steps. The surrogate model is trained to take steps 10 times larger than the high-fidelity model.

For testing the D-LSPF, we compute a high-fidelity particle filter solution with 5000 particles as a baseline to measure if we estimate the distributional information accurately. The high-fidelity solver uses 700 elements and 3rd order polynomials. The available observations are located at eight spatial locations: x=(489.24, 978.47, 1467.71, 1956.95, 2446.18, 2935.42, 3424.66, 3913.89, 4403.13). Only pressure is observed. The observations arrive with a time frequency of 4 s, corresponding to every 400 time steps of the PDE model. To ensure consistent performance across various configurations, we compute the metrics over eight different test trajectories with varying leak locations and sizes and take the average. We compute the state and parameter RRMSE against the true solution, the state AMRMSE against a HF particle filter solution, the state and parameter NLL against the HF particle filter solution, as well as the Wasserstein-1 distance of the posterior parameter distribution against the HF posterior.

We use this test case as an ablation study: we use the same particle filter setting for all approaches and only replace the chosen architectures. We compare the presented architectures with three alternatives: One where the transformer-based AE layers are replaced with convolutional ResNet layers. The dimensionality reduction in this network is performed through strided convolutions and the dimensionality expansion is performed using transposed convolutions. The second alternative is using the proposed ViT architecture but without the Wasserstein distance and consistency regularization in the latent space for the autoencoder. Lastly, we compare with a setup that still uses the proposed ViT AE with regularization, but replaces the transformer based time stepping with a neural ODE (NODE)[51]. In all cases, a latent dimension of 8 is chosen.

Remark We also trained a Fourier Neural Operator (FNO)[52] as a surrogate model, but without success. In all attempts, the solutions exploded after a certain number of time steps. This may be due to the fact that there is a clear discontinuity which is located at the parameter, x_l. Fourier series may not be good approximators for such tasks. For this reason, FNO results are omitted from this work.

The results are summarized in Table 5. Our chosen architecture demonstrates superior performance in most metrics. Note in particular that the results using the convolutional AE are significantly worse compared to the ViT AE, emphasizing the advantages of the transformer-based architecture across different combinations. Furthermore, the D-LSPF outperforms the high-fidelity particle filter in estimating the leak location and size as well as the state. This can be explained from the fact that the D-LSPF is trained on higher resolution trajectories. As this came at a higher cost only at the training stage, it does not add to the computation time when running the particle filter.

In Fig. 10, the true velocity at $t = 40$ and $t = 120$ are compared with estimates using a high-fidelity model, the D-LSPF with the ViT AE, and the D-LSPF with convolutional AE. The convolutional AE clearly fails to reconstruct the state in any meaningful way, while the D-LSPF with the ViT AE accurately estimates the velocity

	HF	Reg-ViT-Trans	NoReg-ViT-Trans	Reg-Conv-Trans	Reg-ViT-NODE
P-RRMSE ↓	5.2×10^{-1}	$\mathbf{9.6 \times 10^{-3}}$	1.1×10^{-2}	7.4×10^{-1}	1.5×10^{-2}
P-Wasserstein-1 ↓	–	169.9	**168.4**	875.7	172.7
S-RRMSE ↓	7.9×10^{-2}	$\mathbf{2.5 \times 10^{-2}}$	2.9×10^{-2}	8.3×10^{-2}	3.1×10^{-2}
S-AMRMSE ↓	–	$\mathbf{4.3 \times 10^{-3}}$	4.4×10^{-3}	4.5×10^{-3}	4.5×10^{-3}
P-NLL ↓	–	**5.09**	5.70	6.01	12.54
S-NLL ↓	–	16.87	**15.37**	18.37	17.32
Time (GPU) ↓	–	24.89 s	24.90 s	23.58 s	**6.52**
Speed-up (GPU) ↓	–	1807.95	1807.23	1908.40	**6901.84**
Time (CPU) ↓	45,000 s	332.3 s	317.0 s	328.8 s	**45.9 s**
Speed-up (CPU) ↓	–	135.4	142.0	136.9	**980.8**

Table 5. Computation times using a GPU for the D-LSPF applied to the multi phase leak localization test case. All timings are computed with 5000 particles. The high-fidelity solver makes use of 100 CPU cores and the neural network uses one GPU. "P-" refers to parameter estimation and "S-" refers to state estimation. RRMSE is the relative root mean squared error, the AMRMSE is the average moment RMSE with respect to the high-fidelity particle posterior, and NLL is the negative log-likelihood with respect to the high-fidelity particle posterior. Best values are in bold.

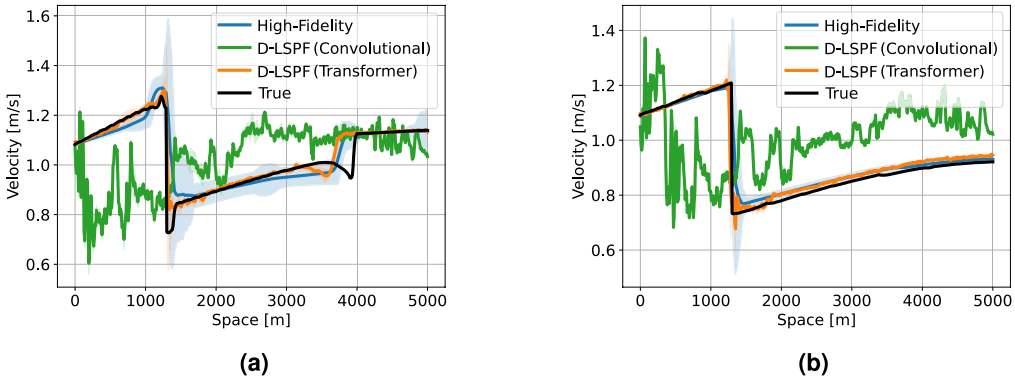

Figure 10. Velocity estimation results for the multi phase pipeflow test case. (**a**) Velocity estimation at $t = 40$. (**b**) Velocity estimation at $t = 120s$.

with the uncertainty concentrated around the leak location as expected. It is, however, worth noticing that the uncertainty is significantly smaller for the D-LSPF solution, suggesting that the particles may have degenerated. In Fig. 11, we show the convergence accuracy of the leak location estimation for all the neural network setups. We see in Fig. 11a that the proposed setup is superior than the alternatives with respect to the RRMSE for all number of particles. Similarly, we reach the same conclusion with respect to the Wasserstein-1 distance of the posterior distribution of the leak location.

Conclusion

We presented a novel particle filter, the D-LSPF, for fast and accurate data assimilation with uncertainty quantification. The D-LSPF was based on a surrogate model utilizing dimensionality reduction and latent space time stepping. The use of particle filters provided estimates for the state and parameters as well as for the associated uncertainties. For the AE, we made use of a novel extension of the vision transformer for dimensionality reduction and reconstruction. Furthermore, we discussed a number of regularization techniques to improve the performance of the D-LSPF.

We demonstrated the D-LSPF on three different test cases with varying characteristics and complexities. In the first test, we compared with alternative deep learning-based data assimilation methods for the viscous Burgers equation. The D-LSPF showed superior performance by an order of magnitude regarding uncertainty estimation as well as almost 4 times better mean reconstruction. In the second test case, we performed state and parameter estimation on a wave tank experiment with few observations available in time and space. The D-LSPF performed up to 3 times better than alternative approaches while also being faster and successfully estimated both the state and parameter. Lastly, we applied the D-LSPF on a leak localization problem for multi-phase flow in a long pipe. We showed how the ViT AE and the regularization techniques drastically improved the state and parameter estimation: the D-LSPF provided speed-ups of 3 orders of magnitude compared with a high-fidelity particle filter while also being more accurate.

Several aspects of our work could benefit from future investigations. In particular, we made use of the bootstrap particle filter. While this particular version of the particle filter has many advantages such as relative ease of implementation and convergence, there are alternatives. It might be fruitful to analyze the quality of filters that utilize the differentiability of neural networks, such as the nudging particle filter with gradient nudging[53] or particle filters based on Stein variational gradient descent[54,55].

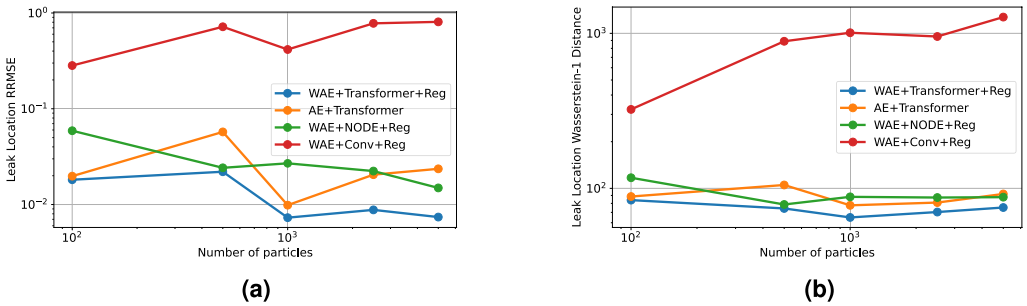

Figure 11. Results for the leak size and location estimation at time 120 s, for varying neural network specifications and number of particles. (**a**) Leak location and size Relative RMSE. (**b**) Wasserstein distance for the leak location.

In all, the D-LSPF significantly sped up complex data assimilation tasks without sacrificing accuracy, thus enabling fusing of increasingly complex models with data in real-time. It is important to note that the D-LSPF is not exclusively for use in fluid dynamics. The only prerequisites for the method are that a low-dimensional solution manifold must exist, and a high-fidelity model that approximates the system at hand sufficiently well is available, such that training data can be simulated. Hence, the D-LSPF can potentially be used in other fields where real-time data assimilation is of importance, such as e.g. healthcare and climate modeling.

Data availibility

The code for setting up and training the neural networks, including hyperparameter settings, for the test cases can be found in the GitHub repository https://github.com/nmucke/latent-time-stepping. The code for simulating training data for Burgers equations and the multi-phase leak location problem can be found in the same repository. The code for simulating the training data for the harmonic wave generation over a submerged bar test case can be made available upon request and in consultation with Associate professor Allan Peter Engsig-Karup. The code for the particle filter implementations as well as the test data can be found in the GitHub repository https://github.com/nmucke/data-assimilation.

Received: 5 June 2024; Accepted: 9 August 2024
Published online: 21 August 2024

References

1. Rasheed, A., San, O. & Kvamsdal, T. Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access* **8**, 21980–22012 (2020).
2. Asch, M. *A Toolbox for Digital Twins: From Model-Based to Data-Driven* (SIAM, 2022).
3. Kapteyn, M. G., Pretorius, J. V. & Willcox, K. E. A probabilistic graphical model foundation for enabling predictive digital twins at scale. *Nat. Comput. Sci.* **1**, 337–347 (2021).
4. Houtekamer, P. L. & Mitchell, H. L. Data assimilation using an ensemble Kalman filter technique. *Mon. Weather Rev.* **126**, 796–811 (1998).
5. Evensen, G., Vossepoel, F.C. & van Leeuwen, P. J. *Data Assimilation Fundamentals: A Unified Formulation of the State and Parameter Estimation Problem* (Springer, 2022).
6. Uilhoorn, F. E. A particle filter-based framework for real-time state estimation of a non-linear hyperbolic PDE system describing transient flows in CO2 pipelines. *Comput. Math. Appl.* **68**, 1991–2004 (2014).
7. Albarakati, A. *et al.* Model and data reduction for data assimilation: Particle filters employing projected forecasts and data with application to a shallow water model. *Comput. Math. Appl.* **116**, 194–211 (2022).
8. Chen, Y. & Oliver, D. S. Levenberg–Marquardt forms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Comput. Geosci.* **17**, 689–703 (2013).
9. Kitagawa, G. Monte Carlo filter and smoother for non-gaussian nonlinear state space models. *J. Comput. Graph. Stat.* 1–25 (1996).
10. Fearnhead, P. & Künsch, H. R. Particle filters and data assimilation. *Annu. Rev. Stat. Appl.* **5**, 421–449 (2018).
11. Lee, K. & Carlberg, K. T. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *J. Comput. Phys.* **404**, 108973 (2020).
12. Geneva, N. & Zabaras, N. Transformers for modeling physical systems. *Neural Netw.* **146**, 272–289 (2022).
13. Champion, K., Lusch, B., Kutz, J. N. & Brunton, S. L. Data-driven discovery of coordinates and governing equations. *Proc. Natl. Acad. Sci.* **116**, 22445–22451 (2019).
14. Mücke, N. T., Bohté, S. M. & Oosterlee, C. W. Reduced order modeling for parameterized time-dependent PDEs using spatially and memory aware deep learning. *J. Comput. Sci.* **53**, 101408 (2021).
15. Ballard, D. H. Modular learning in neural networks. *Proc. Sixth Natl. Conf. Artif. Intell.-Vol.* **1**, 279–284 (1987).
16. Kingma, D. P. & Welling, M. Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114 (2013).
17. Tolstikhin, I., Bousquet, O., Gelly, S. & Schoelkopf, B. Wasserstein auto-encoders. arXiv preprint arXiv:1711.01558 (2017).
18. Wan, Z. Y., Zepeda-Núñez, L., Boral, A. & Sha, F. Evolve smoothly, fit consistently: Learning smooth latent dynamics for advection-dominated systems. arXiv preprint arXiv:2301.10391 (2023).
19. Cheng, S. *et al.* Machine learning with data assimilation and uncertainty quantification for dynamical systems: A review. *IEEE/CAA J. Autom. Sin.* **10**, 1361–1387 (2023).
20. Patel, D. V., Ray, D., Ramaswamy, H. & Oberai, A. Bayesian inference in physics-driven problems with adversarial priors. In *NeurIPS 2020 Workshop on Deep Learning and Inverse Problems* (2020).
21. Xia, Y. & Zabaras, N. Bayesian multiscale deep generative model for the solution of high-dimensional inverse problems. *J. Comput. Phys.* **455**, 111008 (2022).
22. Mücke, N. T., Sanderse, B., Bohté, S. M. & Oosterlee, C. W. Markov chain generative adversarial neural networks for solving Bayesian inverse problems in physics applications. *Comput. Math. Appl.* **147**, 278–299 (2023).
23. Seabra, G., Mücke, N., Silva, V., Voskov, D. & Vossepoel, F. Ai enhanced data assimilation and uncertainty quantification applied to geological carbon storage. arXiv preprint arXiv:2402.06110 (2024).
24. Brunton, S. L., Proctor, J. L. & Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl. Acad. Sci.* **113**, 3932–3937 (2016).
25. Maddison, C. J. *et al.* Filtering variational objectives. *Adv. Neural Inf. Process. Syst.* **30** (2017).
26. Moretti, A. K., Wang, Z., Wu, L., Drori, I. & Pe'er, I. Particle smoothing variational objectives. arXiv preprint arXiv:1909.09734 (2019).
27. Silva, V. L. S., Heaney, C. E. & Pain, C. C. Generative network-based reduced-order model for prediction, data assimilation and uncertainty quantification. In *LatinX in AI Workshop at ICML 2023 (Regular Deadline)* (2023).
28. Gonczarek, A. & Tomczak, J. M. Articulated tracking with manifold regularized particle filter. *Mach. Vis. Appl.* **27**, 275–286 (2016).
29. Yang, Y., Stork, J. A. & Stoyanov, T. Particle filters in latent space for robust deformable linear object tracking. *IEEE Robot. Autom. Lett.* **7**, 12577–12584 (2022).
30. Cheng, S. *et al.* Generalised latent assimilation in heterogeneous reduced spaces with machine learning surrogate models. *J. Sci. Comput.* **94**, 11 (2023).
31. Zhang, C., Cheng, S., Kasoar, M. & Arcucci, R. Reduced order digital twin and latent data assimilation for global wildfire prediction. *EGUsphere* **2022**, 1–24 (2022).
32. Peyron, M. *et al.* Latent space data assimilation by using deep learning. *Q. J. R. Meteorol. Soc.* **147**, 3759–3777 (2021).
33. Chen, Y., Sanz-Alonso, D. & Willett, R. Reduced-order autodifferentiable ensemble kalman filters. arXiv preprint arXiv:2301.11961 (2023).
34. Reich, S. & Cotter, C. *Probabilistic Forecasting and Bayesian Data Assimilation* (Cambridge University Press, 2015).

35. Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I. & Frey, B. Adversarial autoencoders. arXiv preprint arXiv:1511.05644 (2015).
36. Dosovitskiy, A. *et al.* An image is worth 16 x 16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020).
37. Ovadia, O., Kahana, A., Stinis, P., Turkel, E. & Karniadakis, G. E. Vito: Vision transformer-operator. arXiv preprint arXiv:2303.08891 (2023).
38. Li, G., Jin, D., Yu, Q. & Qi, M. Ib-transunet: Combining information bottleneck and transformer for medical image segmentation. *J. King Saud Univ.-Comput. Inf. Sci.* **35**, 249–258 (2023).
39. Ran, R., Gao, T. & Fang, B. Transformer-based dimensionality reduction. arXiv preprint arXiv:2210.08288 (2022).
40. Heo, B. *et al.* Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 11936–11945 (2021).
41. Vaswani, A. *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems*. Vol. 30 (2017).
42. Han, X., Gao, H., Pfaff, T., Wang, J.-X. & Liu, L.-P. Predicting physics in mesh-reduced space with temporal attention. arXiv preprint arXiv:2201.09113 (2022).
43. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. Vol. 32 (2019).
44. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
45. Beji, S. & Battjes, J. A. Numerical simulation of nonlinear wave propagation over a bar. *Coastal Eng.* **23**, 1–16 (1994).
46. Bigoni, D., Engsig-Karup, A. P. & Eskilsson, C. Efficient uncertainty quantification of a fully nonlinear and dispersive water wave model with random inputs. *J. Eng. Math.* **101**, 87–113 (2016).
47. Engsig-Karup, A. P., Eskilsson, C. & Bigoni, D. A stabilised nodal spectral element method for fully nonlinear water waves. *J. Comput. Phys.* **318**, 1–21 (2016).
48. Zakharov, V. E. Stability of periodic waves of finite amplitude on the surface of a deep fluid. *J. Appl. Mech. Tech. Phys.* **9**, 190–194 (1968).
49. Omgba-Essama, C. *Numerical Modelling of Transient Gas-liquid Flows (Application to Stratified & Slug Flow Regimes)*. Ph.D. Thesis, Cranfield University, CERES (2004).
50. Hesthaven, J. S. & Warburton, T. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications* (Springer, 2007).
51. Chen, R. T., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.* **31** (2018).
52. Li, Z. *et al.* Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895 (2020).
53. Akyildiz, Ö. D. & Míguez, J. Nudging the particle filter. *Stat. Comput.* **30**, 305–330 (2020).
54. Fan, J., Taghvaei, A. & Chen, Y. Stein particle filtering. arXiv preprint arXiv:2106.10568 (2021).
55. Maken, F. A., Ramos, F. & Ott, L. Stein particle filter for nonlinear, non-gaussian state estimation. *IEEE Robot. Autom. Lett.* **7**, 5421–5428 (2022).

Acknowledgements

This work is supported by the Dutch National Science Foundation NWO under the grant number 629.002.213. The authors also acknowledge Oracle for providing compute credits for their cloud platform, Oracle Cloud Infrastructure. The authors furthermore acknowledge the help and code provided by Associate professor Allan Peter Engsig-Karup for the results related to the harmonic wave generation over a submerged bar test case.

Author contributions

N. Mücke: Conceptualization, methodology, software, formal analysis, writing—original draft. S. Bohté: Funding acquisition, writing—review & editing, supervision, project administration. C. Oosterlee: Funding acquisition, formal analysis, writing—review & editing, supervision, project administration.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to N.T.M.