Journal of Computational Physics 403 (2020) 109056

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

# Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks

Nicholas Geneva, Nicholas Zabaras *

*Center for Informatics and Computational Science, University of Notre Dame, 311 Cushing Hall, Notre Dame, IN 46556, USA*

ARTICLE INFO

*Article history:*
Received 13 June 2019
Received in revised form 17 September 2019
Accepted 22 October 2019
Available online 24 October 2019

*Keywords:*
Physics-informed machine learning
Auto-regressive model
Deep neural networks
Convolutional encoder-decoder
Uncertainty quantification
Dynamic partial differential equations

ABSTRACT

In recent years, deep learning has proven to be a viable methodology for surrogate modeling and uncertainty quantification for a vast number of physical systems. However, in their traditional form, such models can require a large amount of training data. This is of particular importance for various engineering and scientific applications where data may be extremely expensive to obtain. To overcome this shortcoming, physics-constrained deep learning provides a promising methodology as it only utilizes the governing equations. In this work, we propose a novel auto-regressive dense encoder-decoder convolutional neural network to solve and model non-linear dynamical systems without training data at a computational cost that is potentially magnitudes lower than standard numerical solvers. This model includes a Bayesian framework that allows for uncertainty quantification of the predicted quantities of interest at each time-step. We rigorously test this model on several non-linear transient partial differential equation systems including the turbulence of the Kuramoto-Sivashinsky equation, multi-shock formation and interaction with 1D Burgers' equation and 2D wave dynamics with coupled Burgers' equations. For each system, the predictive results and uncertainty are presented and discussed together with comparisons to the results obtained from traditional numerical analysis methods.



## 1. Introduction

In almost all scientific domains, simulating systems of partial differential equations (PDEs) is of great importance and research interest. Given that many physical phenomena including heat diffusion, fluid dynamics, and elasticity are formalized with PDEs, numerically or analytically solving these governing equations is a core foundation for a vast spectrum of scientific and engineering disciplines. In recent decades exponential growth in computational power has made such numerical methods for solving PDEs even more accessible. However, in most modern-day applications, obtaining the desired resolution or accuracy with such simulations is still computationally expensive. Hence, many seek to strike an ideal balance between predictive accuracy and computational efficiency. In many situations, such as optimization or inverse problems, a large number of repeated simulations are required prioritizing the computational efficiency of the numerical simulator. Often surrogate models are used to ease this computational burden by providing a fast approximate model that can imitate a standard numerical solver at a significantly reduced computational cost.

* Corresponding author.
*E-mail addresses:* ngeneva@nd.edu (N. Geneva), nzabaras@gmail.com (N. Zabaras).
*URL:* https://cics.nd.edu/ (N. Zabaras).

https://doi.org/10.1016/j.jcp.2019.109056
0021-9991/

In recent years, machine learning and deep learning have entered a renaissance in which groundbreaking findings have made deep learning models widely successful for a vast number of applications [1]. One such application is surrogate modeling in which a deep learning model can be used as a black box method to approximate a physical system. Among the most popular deep learning models is deep neural networks (DNNs), which have proven to be an extremely effective method for modeling a wide spectrum of physical systems such as flow through porous media [2–4], Navier-Stokes equations [5], turbulence modeling [6], molecular dynamics [7] and more. Traditional DNNs are not probabilistic in nature resulting in Bayesian extensions of these models [8,9] to quantify the underlying uncertainty in these black box algorithms. While DNNs have been proven to be both accurate and computationally efficient for modeling and uncertainty quantification (UQ), it is commonly known that training such models may require a significant amount of data. Depending on the system of interest, training data may either be sparse, extremely expensive to obtain or not available at all. Considering that the underlying governing equations are known, in this work, we are particularly interested in the surrogate modeling of physical systems using physics-constrained loss functions. Such loss functions allow a surrogate model to be trained in the *absence* of data (e.g. without having to solve the equations governing the system of interest).

The philosophy of learning ordinary or partial differential equations through constraint based loss functions is far from a new idea with related works reaching back over two decades ago [10–13]. These early works focused on solving initial/boundary value problems in which the solution is parameterized by a fully-connected network which allows for a fully differentiable and closed analytic form [13]. With the resurgence of interest in neural networks, such techniques have been rediscovered by multiple works in recent years where this core idea has been expanded upon. As discussed in the work of Lagaris et al. [13,14] and later revisited by Raissi et al. [15], the use of fully connected networks with physics-constrained learning allows for a mesh free solution that can be evaluated anywhere on the domain while being trained on only a few points. Additionally, Lagaris et al. [14] and more recently Berg and Nyström [16] showed fully connected networks can be used to learn PDE solutions on even complex domains. Recently, several investigators have examined the use of variational formulations of the governing equations as loss functions to solve various PDEs [3,17–19] which has been proven to be effective. Sirignano et al. [20] show that the use of a fully connected network can be used for efficiently solving PDEs of high dimensionality where traditional discretization techniques become unfeasible. Several have also investigated the use of fully connected networks to solve high-dimensional stochastic PDEs with good success [21,22]. While fully connected networks could be optimized to compute a single solution of a PDE, several challenges remain in extending these ideas to surrogate model construction. For example, if the initial condition, boundary conditions, material properties, etc. are changed the model must be retrained. This means that from a computational aspect, such methods are difficult to justify if a numerical simulator can be used that is computationally less expensive than training the fully connected network. Clearly, with decades of numerical analysis progress this issue is applicable to an overwhelmingly large amount of PDE systems.

To the authors best knowledge only the works of Zhu et al. [3] and Karumuri et al. [19] seek to build surrogate models using physics-constrained, data free learning. In [3], a deep convolutional neural network was used to formulate a surrogate model for an elliptic PDE with a stochastic, high-dimensional permeability field. Additionally, Zhu et al. proposed a probabilistic framework based on a conditional flow-based generative model [23] to quantify the potential error arising from the model itself. It was also found that the data-less physics-constrained learning yielded a model with much better generalization capabilities than traditional data-driven learning. Karumuri et al. [19] used a deep fully connected ResNet [24] to build a surrogate also for elliptic PDEs with reasonable success. Note that both of these aforementioned works have been focused entirely on PDEs which are not time-dependent in nature.

In this work, we generalize these physics-constrained deep learning surrogate models to dynamical PDEs. The novel contributions of this paper are as follows: (a) We propose a deep auto-regressive dense encoder-decoder for predicting transient PDEs and the physics-constrained training algorithm; (b) Extend this model to a Bayesian framework using the recently proposed stochastic weight averaging Gaussian algorithm to quantify both epistemic and aleatoric uncertainty; (c) Implement this model for a chaotic/turbulent system, a system with multiple shock wave interactions and a 2D system of coupled non-linear PDEs far surpassing the complexity of other test cases shown in past literature; (d) Present and discuss the accuracy of the predictions as well as the associated uncertainty for each of the previously discussed PDEs; and (e) Compare the computational efficiency of the proposed surrogate model against other state-of-the-art numerical methods.

This paper is organized as follows: First, in Section 2, we briefly define and discuss the problem of interest. Section 3 discusses the auto-regressive dense encoder-decoder model, its training and use as a surrogate model. In Section 4, we extend this deep learning model to the Bayesian paradigm where we discuss the formulation of the posterior as well as the approximation of the predictive distribution. Following, in Section 5, the proposed model is implemented for the chaotic Kuramoto-Sivashinsky system and a study is presented of the turbulent statistics that the model produces. In Section 6, we also explore the use of the auto-regressive model for the prediction of shocks in the 1D Burgers' equation. Later in Section 7, we further extend this to the 2D coupled Burgers' system. Lastly, conclusions and discussion can be found in Section 8. All code, trained models and data used in this work is open-sourced for full reproducibility.[1]

[1] Code available at: https://github.com/cics-nd/ar-pde-cnn.

## 2. Problem definition

In this work, we are interested in using deep learning architectures for developing surrogate models of non-linear dynamical systems that evolve in both space and time using physics-constrained learning. Specifically, we wish to use the governing equations to formulate loss functions for training surrogate models without the need of (output) training data. Our goal is to develop surrogate models for a class of arbitrary transient PDE systems with an unknown, variable or stochastic initial state. Consider a transient system of PDEs that models a physical system:

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{x},t)_t + F\left(\boldsymbol{x},\boldsymbol{u}\left(\boldsymbol{x},t\right)\right) &= 0, \quad \boldsymbol{x}\in\Omega,\ t\in[0,T], \\ \mathcal{B}(\boldsymbol{u}) &= b\left(\boldsymbol{x},t\right), \quad \boldsymbol{x}\in\Gamma, \\ \boldsymbol{u}(\boldsymbol{x},0) &\sim p\left(\boldsymbol{u}(\boldsymbol{x},0)\right), \end{aligned} \tag{1}$$

where we have denoted this $n$-dimensional PDE by the temporal derivative $\boldsymbol{u}(\boldsymbol{x},t)_t$ and the remaining terms by $F(\cdot)$ which includes spatial derivatives and non-linear terms. $\boldsymbol{u}(\boldsymbol{x},t)$ are the system's state variables in the domain $\Omega$ with a boundary $\Gamma$. $\mathcal{B}$ is the boundary operator that enforces the desired boundary conditions. Lastly, the initial state $\boldsymbol{u}(\boldsymbol{x},0)$ is a real valued random field with a probability density, $p\left(\boldsymbol{u}(\boldsymbol{x},0)\right)$, that may or may not be known.

Our goal is to expand on the work in [3] in which PDE solutions were represented as an optimization problem by either minimizing an energy functional or alternatively the square of the PDE residual [25]. The objective in [3] was to predict quantities of interest for an elliptic PDE (defining Darcy's flow) in an image-to-image regression approach using a convolutional encoder-decoder architecture with an input being a property field (permeability) and the output being the quantities of interest (pressure and velocity). The use of a convolutional neural network proved to have some significant benefits over the more commonly used fully-connected networks including faster convergence and better accuracy. While successful for elliptic PDEs, the strategies in this past work cannot directly generalize to a dynamical system.

If one were developing a numerical algorithm to solve a dynamical system, the first step would be to discretize the time derivative, which is commonly referred to as a time-stepping or time integration method [26]. For time integration there are a vast number of options including standard explicit or implicit methods, Runge-Kutta methods, linear multi-step methods, implicit-explicit methods and more. However, the goal of all these techniques is the same: evolve the system from time $t$ to time $t+\Delta t$. Using this philosophy of discrete time integration, we propose building a surrogate model that performs time integration at a specified $\Delta t$ in an image-to-image regression algorithm using a convolutional encoder-decoder neural network. Let us consider $\boldsymbol{u}^n$ as the solution of the PDE with $d_0$ state variables on a given structured Euclidean discretization of $\Omega$ at time-step $n$. Namely, given $\Omega$ discretized with $D_i$ points in the $i$-th dimension, $\boldsymbol{u}^n \in \mathbb{R}^{d_0\times D_1}$ for a 1D system, $\boldsymbol{u}^n \in \mathbb{R}^{d_0\times D_1\times D_2}$ for a 2D system and $\boldsymbol{u}^n \in \mathbb{R}^{d_0\times D_1\times D_2\times D_3}$ for a 3D system. Our convolutional encoder-decoder model for simulating time integration at time-step $n$ is parameterized as follows:

$$\boldsymbol{u}^{n+1} = f\left(\boldsymbol{\chi}^{n+1},\mathbf{w}\right), \quad \boldsymbol{\chi}^{n+1} \equiv \left\{\boldsymbol{u}^n,\boldsymbol{u}^{n-1},\ldots,\boldsymbol{u}^{n-k}\right\}, \tag{2}$$

where $f$ represents the function learned by the deep learning model, $\mathbf{w}$ are the learnable parameters in this convolutional neural network. $\boldsymbol{\chi}^{n+1}$ is the model's input, for the prediction $\boldsymbol{u}^{n+1}$, consisting of the $k+1$ previous states of the system. By this model definition, we are interested in learning the dynamics or evolution of the system invariant to the current time $t$. The use of a convolutional neural network allows for a light-weight model that can evolve the system of interest by a discrete time-step efficiently without any matrix inversions, iterative relaxations or multi-step processes. Similar to the convolutional model in Zhu et al. [3], this model can be used/extended for tasks such as solving PDEs, surrogate modeling and performing uncertainty quantification.

To predict a given system's response for $N$ time-steps, the convolutional neural network is executed as an auto-regressive model. Given just a discretized initial state of the system $\boldsymbol{u}_0$, one can predict the system response as:

$$\begin{aligned} \boldsymbol{u}^1 &= f\left(\boldsymbol{\chi}^1,\mathbf{w}\right), \quad \boldsymbol{\chi}^1 = \{\boldsymbol{u}_0,\boldsymbol{u}_0,\boldsymbol{u}_0,\ldots,\boldsymbol{u}_0\}, \\ \boldsymbol{u}^2 &= f\left(\boldsymbol{\chi}^2,\mathbf{w}\right), \quad \boldsymbol{\chi}^2 = \left\{\boldsymbol{u}^1,\boldsymbol{u}_0,\boldsymbol{u}_0,\ldots,\boldsymbol{u}_0\right\}, \\ \boldsymbol{u}^3 &= f\left(\boldsymbol{\chi}^3,\mathbf{w}\right), \quad \boldsymbol{\chi}^3 = \left\{\boldsymbol{u}^2,\boldsymbol{u}^1,\boldsymbol{u}_0,\ldots,\boldsymbol{u}_0\right\}, \\ &\ldots \\ \boldsymbol{u}^N &= f\left(\boldsymbol{\chi}^N,\mathbf{w}\right), \quad \boldsymbol{\chi}^N = \left\{\boldsymbol{u}^{N-1},\boldsymbol{u}^{N-2},\ldots,\boldsymbol{u}^{N-1-k}\right\}, \end{aligned} \tag{3}$$

where the model must be executed $N$ times to obtain the prediction of the system at time-step $N$. Note, that the initial input to the model $\boldsymbol{\chi}^1$ is comprised of just the initial state, which is an approximation needed to "kick-start" the time series. The prediction for a particular time-step can be formulated as a recursive function of the model:

$$\boldsymbol{u}^{n+1} = f\left(\left\{f\left(\boldsymbol{\chi}^n,\mathbf{w}\right),f\left(\boldsymbol{\chi}^{n-1},\mathbf{w}\right),\ldots,f\left(\boldsymbol{\chi}^{n-k},\mathbf{w}\right)\right\},\mathbf{w}\right), \tag{4}$$

where the input $\boldsymbol{\chi}^{n+1}$ is formulated in terms of the model itself, with inputs that can be described in a similar manner. Thus only the initial state is needed for predicting a systems' response up to an arbitrary number of time-steps. For the prediction of an entire time series $\left[\boldsymbol{u}^N, \boldsymbol{u}^{N-1}, \dots \boldsymbol{u}^1\right]$, the model can be represented as a set of functions $\hat{f}(\boldsymbol{u}_0, \mathbf{w}) = \left\{f\left(\boldsymbol{\chi}^N, \mathbf{w}\right), f\left(\boldsymbol{\chi}^{N-1}, \mathbf{w}\right), \dots, f\left(\boldsymbol{\chi}^1, \mathbf{w}\right)\right\}$ in which each can be expressed recursively as a function of the initial state.

As discussed previously, we would like to formulate a methodology of physics-constrained learning such that the model can be trained with only a set of initial states $\boldsymbol{u}_0$. Although the same model is used to predict every time-step of a given time series as shown in Eq. (3), the core building block for training this DNN model is learning how to predict the transition of states from $t$ to $t + \Delta t$ regardless of the reference time $t$. For physics-constrained learning, we will pose the optimization problem for a single time-step, $n \rightarrow n+1$, as the minimization of the discrepancy between the model's prediction and the prediction of a discrete numerical time integration method $T_{\Delta t}$ of time-step $\Delta t$ for the governing equation of interest:

$$\arg\min_{\mathbf{w}} \left\| f\left(\boldsymbol{\chi}^{n+1}, \mathbf{w}\right) - T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^{n+1}, F_{\Delta x}(\cdot)\right)\right\|_2^2, \tag{5}$$

in which the systems states $\boldsymbol{u}^{n+1}$ are parameterized by the DNN. The time-integrator predicts numerically the state at the next time-step given some system states, $\boldsymbol{\mathcal{U}}^{n+1}$, and a discretized form of the additional terms of the PDE $F_{\Delta x}$. The exact definition of $\boldsymbol{\mathcal{U}}^{n+1}$ depends on the time integration method used. For example, for the explicit forward Euler scheme, $\boldsymbol{\mathcal{U}}^{n+1} = \left\{\boldsymbol{u}^n\right\}$ depends only on the previous time-step state while for the Crank-Nicolson scheme, $\boldsymbol{\mathcal{U}}^{n+1} = \left\{\boldsymbol{u}^{n+1}, \boldsymbol{u}^n\right\}$, depends on *both* the model's current prediction as well as the state at the previous time-step. As it will be discussed in greater detail in the following sections, given that $T_{\Delta t}$ is consistent with the governing equation, this minimization can be interpreted as the minimization of the residual of the discretized PDE.

### 2.1. Surrogate modeling of dynamical systems

In the context of this work, we are focused on developing a surrogate model that can efficiently predict a dynamical system's response $\boldsymbol{y} = \left[\boldsymbol{u}^N, \boldsymbol{u}^{N-1}, \dots \boldsymbol{u}^1\right]$ for time-steps $[1, N]$ for a given initial state realization $\boldsymbol{u}_{0,i} \sim p(\boldsymbol{u}_0)$ and a set of boundary conditions. We pose the following definition for this surrogate model:

**Definition 2.1** *(Deterministic surrogate model).* For a transient PDE system with specified boundary conditions, as in Eq. (1), and a finite set of initial conditions $\mathcal{S} = \left\{\boldsymbol{u}_{0,i}\right\}_{i=1}^M$, $\boldsymbol{u}_{0,i} \sim p\left(\boldsymbol{u}_0\right)$, train a surrogate to predict the dynamical response $\boldsymbol{y} = \hat{f}\left(\boldsymbol{u}_0^*, \mathbf{w}\right)$ for any initial condition, $\boldsymbol{u}_0^* \sim p\left(\boldsymbol{u}_0\right)$, such that the predicted response is the solution of the governing PDEs for the respective initial state.

The true density of $p\left(\boldsymbol{u}_0\right)$ may not be known. For example, $p\left(\boldsymbol{u}_0\right)$ may represent a set of states collected from an experiment or simulation. When this density is not known or samples are not available, one may need to approximate it for the sake of assembling a set of initial states for training. In the context of this work, we will pose this initial condition as a random function from which we can sample from to illustrate the applicability of our model. As discussed in Zhu et al. [3], surrogate modeling using physics-constrained learning can be interpreted as unsupervised learning since training takes place without any labeled training data. Rather it is up to the model to discover the dynamics of the system.

In the majority of problems of interest, the number of (initial) training data used will only express a portion of all the inputs that can be drawn from the density function $p\left(\boldsymbol{u}_0\right)$. Thus the surrogate model will only be trained to predict a part of all potential responses of the dynamical system. To account for this limited expressibility of both the input data used for training as well as the trained model itself, we also wish to formulate a probabilistic surrogate than can produce distributions over possible solutions, rather than a single point estimate. Hence, we pose the following definition for the probabilistic extension of this dynamical surrogate model:

**Definition 2.2** *(Probabilistic surrogate model).* For a transient PDE system with specified boundary conditions, such as Eq. (1), and a finite set of initial conditions $\mathcal{S} = \left\{\boldsymbol{u}_{0,i}\right\}_{i=1}^M$, $\boldsymbol{u}_{0,i} \sim p\left(\boldsymbol{u}_0\right)$, train a surrogate to predict the dynamical response density, $p\left(\boldsymbol{y}|\boldsymbol{u}_0^*, \mathcal{S}\right)$, such that the samples drawn from the predictive distribution satisfy the governing PDEs for the respective initial state.

## 3. Auto-regressive dense encoder-decoder

The prediction of time series is a classical machine learning problem with many models specifically designed for such tasks most predominately seen in the field of neural language processing. Among the most classical methods are standard recurrent neural networks which tend to be difficult to train [27], as well as long-short term memory (LSTM) architectures [28]. Recent advances of modeling time series include hierarchical networks [29], attention networks [30] and transformer networks [31]. For modeling dynamical systems, we propose the following auto-regressive dense encoder-decoder model (AR-DenseED) illustrated in Fig. 1 which does not rely on any latent variable recurrent connections or specific

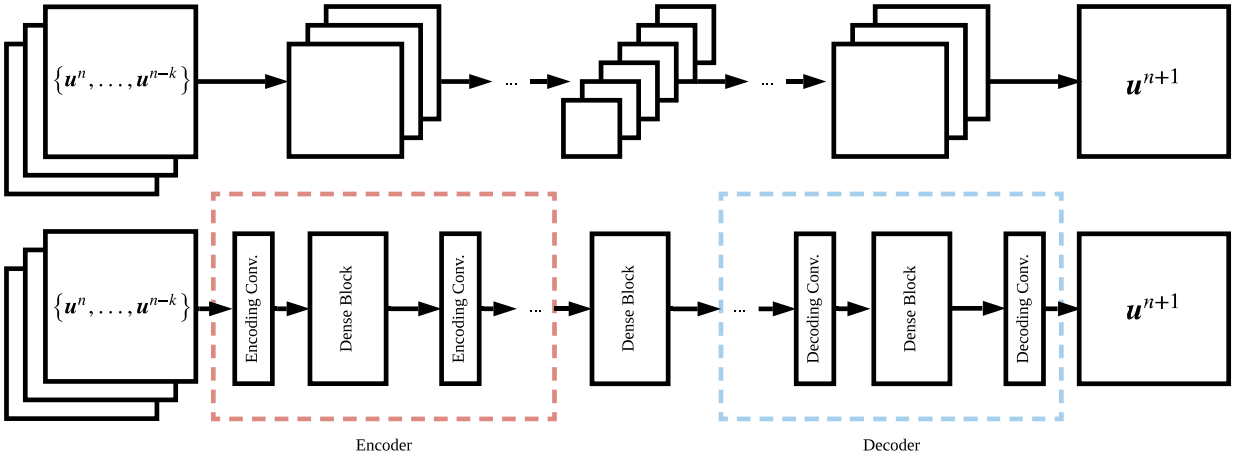



**Fig. 1.** Schematic of the auto-regressive dense encoder-decoder. The top shows the dimensionality of the data in the network, the bottom shows the model architecture. Using encoding convolutions lowers the dimensionality of the input feature map, while decoding convolutions increase the dimensionality. The encoding-decoding process is interleaved with dense blocks that contain multiple densely connected layers in which the dimensionality of the feature maps is held constant. Additional details on these components can be found in Appendix A and the work of Zhu and Zabaras [2].

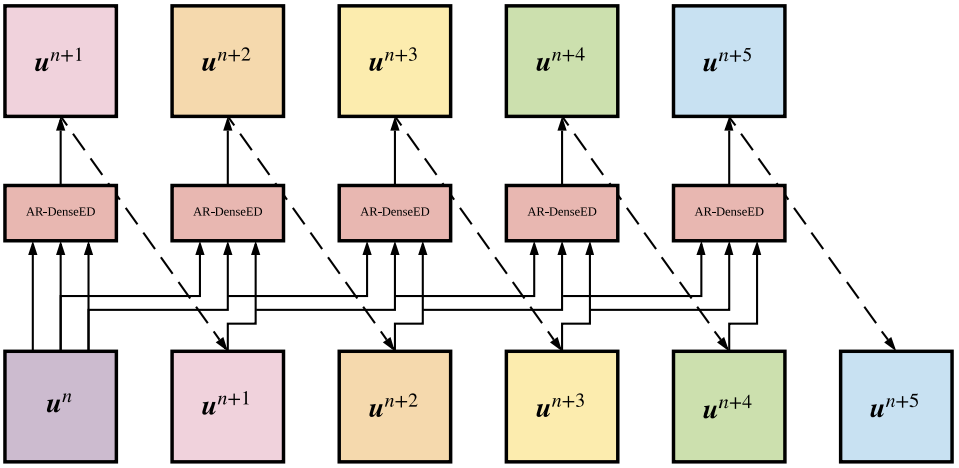



**Fig. 2.** AR-DenseED prediction, outlined in Algorithm 1, of the states at five uniformly spaced time-steps using the states at three previous time-steps as inputs. During prediction, the model used is identical for each time-step. At the beginning of the time sequence, all three inputs are the initial state $\boldsymbol{u}^n$ since no prior states are known. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

gate design as seen in LSTMs [28]. The key philosophy of AR-DenseED is to efficiently model time integration by learning how to evolve the system forward in time given $k+1$ previous states. In a true auto-regressive nature, the model predicts the dynamics of the system through sequential forward passes using the previous predictions as inputs as outlined in Algorithm 1. This is shown in Fig. 2 where an AR-DenseED using three previous time-steps as inputs predicts five time-steps into the future.

**Remark 1.** The number of previous time-steps used in the model's input, $\boldsymbol{\chi}^{n+1} \equiv \left\{\boldsymbol{u}^n, \boldsymbol{u}^{n-1}, \ldots, \boldsymbol{u}^{n-k}\right\}$, is a tunable hyper-parameter that can be adjusted depending on the system of interest. For all of the numerical examples tested, we found that including multiple time-steps in the input (i.e. $k \geq 1$) was essential for improving training stability. However, using too many past time-steps in $\boldsymbol{\chi}$ slows training and does not yield considerable predictive improvements.

AR-DenseED is built using successive layers of encoding/decoding convolutions and dense blocks originally proposed for solving elliptic systems in [2,3]. The convolutional encoder-decoder can be parameterized as the function composition:

$$f\left(\boldsymbol{\chi}^n, \mathbf{w}\right) = decoder \circ encoder(\boldsymbol{\chi}^n), \tag{6}$$

$$\boldsymbol{u}^n = decoder\left(\boldsymbol{z}, \mathbf{w}_d\right), \quad \boldsymbol{z} = encoder\left(\boldsymbol{\chi}^n, \mathbf{w}_e\right), \tag{7}$$

**Algorithm 1:** AR-DenseED Prediction.

**Input**: Trained neural network model: $f(\cdot, \mathbf{w})$; Test initial state: $\boldsymbol{u}_0$; Max number of time-steps to predict: $T_{max}$

$\boldsymbol{\chi}^1 \leftarrow \{\boldsymbol{u}_0, \boldsymbol{u}_0, \dots, \boldsymbol{u}_0\}$; $\quad \boldsymbol{u}_{out}[0] = \boldsymbol{u}_0$ ;

**for** $n = 1$ **to** $T_{max}$ **do**

$\quad \boldsymbol{u}^n \leftarrow f\left(\boldsymbol{\chi}^n, \mathbf{w}\right)$ ; ▷ Forward pass of model

$\quad \boldsymbol{u}_{out}[n] \leftarrow \boldsymbol{u}^n$;

$\quad \boldsymbol{\chi}^{n+1} \leftarrow \left\{\boldsymbol{u}^n, \boldsymbol{\chi}^n[0], \boldsymbol{\chi}^n[1], \dots, \boldsymbol{\chi}^n[k-1]\right\}$ ; ▷ Update input

**Output**: Predicted time series $\boldsymbol{u}_{out} = \left[\boldsymbol{u}_0, \boldsymbol{u}^1, \boldsymbol{u}^2, ..., \boldsymbol{u}^{T_{max}}\right]$;

where $\{\mathbf{w}_e, \mathbf{w}_d\} = \mathbf{w}$ are the encoder and decoder parameters, respectively. $\boldsymbol{z}$ are latent variables that are of lower dimensionality than $\boldsymbol{u}^n$. The input channel dimensionality of the convolutional model is the product of the number of state-variables $d_0$ and the number of previous time-steps used in $\boldsymbol{\chi}^n$. The number of output channels is equivalent to the number of state-variables that are predicted at a given time-step. While we use $f\left(\boldsymbol{\chi}^n, \mathbf{w}\right)$ to encapsulate this process, one can interpret this model as learning two processes: encoding data from previous time-steps to a latent variable $\boldsymbol{z}$ and the prediction of the next state as a function of $\boldsymbol{z}$. We choose to use convolutional neural networks largely because they have been shown to yield faster convergence and better predictive capability for physics-constrained training compared to fully connected networks [3]. Additionally, convolutional neural networks require significantly less learnable weights than fully connected networks due to parameter sharing which allows for faster predictions [32]. Computational efficiency during test time is imperative for surrogate modeling where prediction speed of the surrogate needs to outperform a numerical solver to justify its use. Extensions of convolutional neural networks to unstructured and non-Euclidean domains can be achieved through geometric deep learning [33], an emerging field that focuses on extending convolutional operators past structured data. This will clearly require approximating the spatial gradients of the PDE on unstructured grids.

### 3.1. Physics-constrained loss function

To train this model, we will be extending the previous work of Zhu et al. [3] where governing PDEs are used to formulate a loss function. The model is trained such that its predictions satisfy the governing equations of the system requiring only initial states, $\boldsymbol{u}_0$. For clarity, we will refer to these initial states as *training scenarios*. Unlike works that have used fully connected networks for learning PDEs solutions in a similar manner (e.g. [11,13,15,19]), our convolutional neural network surrogate requires the gradients in the governing PDEs to be numerically approximated. In past works regarding the surrogate modeling of elliptic PDEs [3], finite-difference based approximations were used to compute spatial gradients. These approximations were found to be very effective and computationally efficient. Thus a similar approach will be used for spatial derivatives in this work.

In dynamical systems this leaves one more critical component: the time-integrator or time-stepping method [26]. As previously outlined in Eq. (5), we will pose the optimization of this model for a given time-step in terms of minimizing the difference between the model's predictions and a discrete numerical time-integrator $T_{\Delta t}$ of the governing PDE. The standard $L_2$ loss for a series of $N$ time-steps and a mini-batch of $M$ training scenarios, $\mathcal{S} = \left\{\boldsymbol{u}_{0,i}\right\}_{i=1}^{M}$, can be posed as follows:

$$\mathcal{L} = \frac{1}{M}\sum_{j=1}^{M}\sum_{i=1}^{N}\left\|\hat{\boldsymbol{u}}_j^i - \boldsymbol{u}_j^i\right\|_2^2, \quad \hat{\boldsymbol{u}}_j^i = T_{\Delta t}\left(\boldsymbol{\mathcal{U}}_j^i, F_{\Delta x}\right), \tag{8}$$

where $\boldsymbol{u}^i$ is the prediction from the neural network $f\left(\boldsymbol{\chi}^i, \mathbf{w}\right)$ making the loss implicitly dependent on all states within $\boldsymbol{\chi}^i$ and $\hat{\boldsymbol{u}}^i$ is the "target" calculated using the numerical time-integrator. The $L_2$ norm corresponds to a finite integral over the entire domain $\Omega$. It is important to recognize that the discretization of both the time and spatial derivatives has introduced numerical truncation error into our loss function. These errors in the deterministic case will ultimately be neglected, however we will expand on the idea of numerical error in the Bayesian model in Section 4.

This formulation allows for any time integration algorithm to be used, making it very versatile. Thus one can select a numerical integrator that has the desired properties regarding stability, computational cost and accuracy. In this work, we are interested in predicting large time-steps or when the model represents a time-step, $\Delta t$, resulting in a Courant-Friedrichs-Lewy (CFL) number greater than one. In this regime, explicit methods are fundamentally unstable for hyperbolic PDEs thus one must resort to costly implicit methods which require expensive matrix inversions [26,34]. Rather than performing a matrix inversion as is the case for an implicit time integration scheme, the neural network's predictions for the next time-step are used when evaluating the spatial gradients in $T_{\Delta t}$. This allows for an implicit like time integration without the need for matrix inversions during optimization. Alternatively, $T_{\Delta t}$ could encapsulate multiple explicit calculations each at smaller time-steps. This strategy was not investigated in this work but could allow for greater accuracy at the cost of additional computation.

**Remark 2.** The use of discretization methods makes the model vulnerable to the same numerical issues that plague each numerical approximation. Specifically with regards to the time-integrator, while an implicit scheme may be stable for

very large time-steps this comes with the implications of reduced accuracy which should be considered. However, the parametrization of the solution as a neural network potentially relaxes the traditional numerical analysis constraints regarding stability and accuracy. Thus this model could allow for large time-step predictions at a greater precision than vanilla numerical methods.

A point that remains to be seen is if this optimization function will lead to the solution of the PDE. By substituting the time-integrator $T_{\Delta t}$ into the loss, one can arrive at the following proposition:

**Proposition 1.** *The $L_2$ minimization between the model's prediction and a* consistent *numerical time integration method is analogous to the $L_2$ minimization of the discretized PDE residual.*

Intuitively, this is a logical statement since time integration methods are formulated on discretizing the temporal derivative. We can easily show this with a simple example for a single time-step from $n \rightarrow n+1$. Consider the standard forward Euler time integration scheme:

$$\frac{\hat{\boldsymbol{u}}^{n+1} - \boldsymbol{u}^n}{\Delta t} = -F_{\Delta x}\left(\boldsymbol{x}, \boldsymbol{u}^n\right) \tag{9}$$

$$\hat{\boldsymbol{u}}^{n+1} = T_{\Delta t}\left(\mathcal{U}^{n+1}, F_{\Delta x}\right) = \boldsymbol{u}^n - \Delta t\, F_{\Delta x}\left(\boldsymbol{x}, \boldsymbol{u}^n\right), \tag{10}$$

where $\mathcal{U}^{n+1} = \left\{\boldsymbol{u}^n\right\}$. Substituting this into the loss function in Eq. (8):

$$\begin{aligned}
\mathcal{L} &= \frac{1}{M}\sum_{i=1}^{M}\left\|\boldsymbol{u}_i^n - \Delta t\, F_{\Delta x}\left(\boldsymbol{x}_i, \boldsymbol{u}_i^n\right) - \boldsymbol{u}_i^{n+1}\right\|_2^2, \\
&= \frac{1}{M}\sum_{i=1}^{M}\left\|\boldsymbol{u}_i^{n+1} - \boldsymbol{u}_i^n + \Delta t\, F_{\Delta x}\left(\boldsymbol{x}_i, \boldsymbol{u}_i^n\right)\right\|_2^2, \\
&= \frac{\Delta t^2}{M}\sum_{i=1}^{M}\left\|\frac{\boldsymbol{u}_i^{n+1} - \boldsymbol{u}_i^n}{\Delta t} + F_{\Delta x}\left(\boldsymbol{x}_i, \boldsymbol{u}_i^n\right)\right\|_2^2, \\
&= \frac{\Delta t^2}{M}\sum_{i=1}^{M}\left\|\mathcal{R}\left(\boldsymbol{u}_i^{n+1}\right)\right\|_2^2 = \frac{\Delta t^2}{M}\sum_{i=1}^{M}\left\|\mathcal{R}\left(f\left(\boldsymbol{\chi}_i^{n+1}, \mathbf{w}\right)\right)\right\|_2^2,
\end{aligned} \tag{11}$$

we arrive at the minimization of the residual, $\mathcal{R}$, for the discretized PDE given the neural network's prediction. The same result can be obtained for implicit time integration methods as well. For example, this can easily be seen for the implicit backward Euler scheme by replacing $F_{\Delta x}\left(\boldsymbol{x}_i, \boldsymbol{u}_i^n\right)$ in Eq. (11) with $F_{\Delta x}\left(\boldsymbol{x}_i, \boldsymbol{u}_i^{n+1}\right)$ where $\boldsymbol{u}_i^{n+1}$ is the model's prediction. Thus, if we use a time integration method that is consistent with the governing PDE, this optimization function minimizes the residual of the PDE across the entire domain leading to the solution of the discretized PDE. After all, the minimization of the residual is the foundation for many iterative numerical methods (e.g. [35,36]) and has been empirically shown to be very effective for learning PDEs with deep learning models in past works [3,13,15].

To enforce boundary conditions, one can introduce an additional loss term that enforces the desired response at the domain boundary. This has been successfully implemented in past work for elliptic PDEs [3] and can easily be generalized to dynamical problems. The systems in this work all have periodic boundaries which are enforced by using circular padding in PyTorch [37] for all model convolutions as well as when evaluating the loss. This is the equivalent to the use of "ghost nodes" in numerical simulations for enforcing periodicity. Additional loss terms can be added to impose other prior physical constraints or conservation laws (*e.g.* solenoidality or mass conservation). Alternatively, one can modify the network architecture directly to impose constraints such as positivity and invariances [6,38].

### *3.2. AR-DenseED training*

This model has several important advantages that we will leverage. The first is that time is not an explicit input, thus this model has no fundamental limitation on its predictive range or initial conditions making it easier to train and better at extrapolation. This is a key advantage of this model compared to the standard fully-connected networks that use time as a discrete input which fundamentally limits their predictive capabilities [13,15,39]. During training, the model is allowed to progressively explore the system and learn the dynamics by slowly "unrolling" the number of time-steps it predicts as training progresses. For example, at the beginning of training, the model only predicts a few time-steps from its initial state, however this may increase to hundreds of time-steps as training continues. This allows the model to discover and learn dynamics that may be absent from the provided initial states. Additionally, since the model's output prediction is then

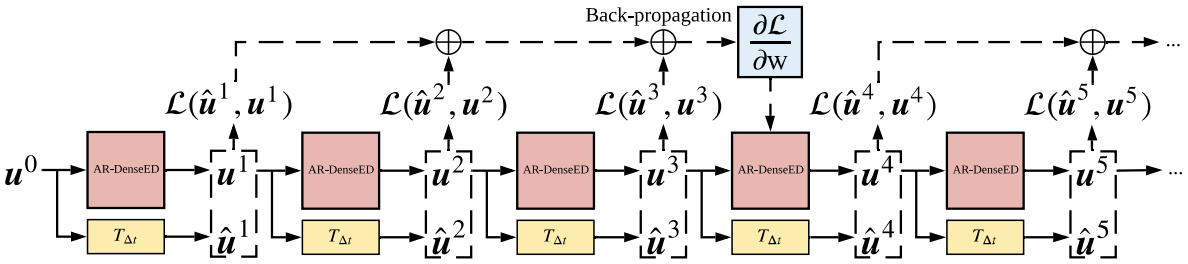



**Fig. 3.** Multi-time-step back-propagation of the AR-DenseED where $\mathbf{w}$ are the model's learnable weights, $\boldsymbol{u}^n$ is the model's prediction at time-step $n$, $\hat{\boldsymbol{u}}^n$ is the target value calculated using the numerical time-integrator $T_{\Delta t}$, and $\mathcal{L}$ is the physics-constrained loss at a single time-step. In this example, the computational graph evaluated during back-propagation spans all three predictions resulting in each contributing to the gradient descent update.

the input for the next time-step, we can back-propagate through multiple time-steps as illustrated in Fig. 3 without latent variable recurrent connections. Allowing the model to auto-regress itself forward in time and compute back-propagation through multiple time-steps promotes the learning of continuous time series. In practice, we only back-propagate through a small number of time-steps to avoid vanishing gradient issues. The training process is outlined in Algorithm 2.

**Algorithm 2:** Training AR-DenseED.

**Input**: Neural network model: $f(\cdot, \mathbf{w})$; Back-prop interval: $p$; Max number to time-steps to unroll: $T_{max}$; Number of epochs: $N$; Learning rate: $\eta$

$\text{tsteps} = \text{linspace}(p, T_{max}, N)$ ;
**for** $epoch = 1$ **to** $N$ **do**
  $\boldsymbol{\chi}^1 \leftarrow \{\boldsymbol{u}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{u}_0\}$ ;
  $T \leftarrow \text{tsteps[epoch]}$ ; ▷ Time-steps to unroll
  **for** $i = 1$ **to** $T$ **do**
    $\boldsymbol{u}^i \leftarrow f(\boldsymbol{\chi}^i, \mathbf{w})$ ; ▷ Forward pass of the model
    $\hat{\boldsymbol{u}}^i = T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right)$ ; ▷ Time integration
    $\mathcal{L}^i = \mathcal{L}^{i-1} + \text{MSE}\left(\hat{\boldsymbol{u}}^i, \boldsymbol{u}^i\right)$ ; ▷ Calculate Loss
    **if** *Mod(n,p)=0* **then**
      $\nabla \mathbf{w} \leftarrow \text{Backprop}(\mathcal{L}^i)$ ; ▷ Multi-step back-prop.
      $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathbf{w}$ ; ▷ Gradient Descent
      $\mathcal{L}^i = 0$ ; ▷ Zero loss
    $\boldsymbol{\chi}^{i+1} \leftarrow \{\boldsymbol{u}^i, \boldsymbol{\chi}^i[0], \boldsymbol{\chi}^i[1], \ldots, \boldsymbol{\chi}^i[k-1]\}$ ; ▷ Update input

**Output**: Trained auto-regressive model $f(\cdot, \mathbf{w})$;

## 4. Bayesian AR-DenseED

A challenge of physics-constrained learning with no output training data is developing a meaningful probabilistic framework. In past works, a probabilistic surrogate was proposed using the Boltzmann distribution as a reference density and minimizing the Kullback-Leibler divergence for a generative model [3,40]. While such methodologies represent some built in uncertainty in the model and can yield reasonable error bars, the true interpretation of the resulting uncertainty is much less concrete. Thus in this work, we propose a novel Bayesian framework for physics-constrained models that allow for interpretable uncertainty measures to be produced in the absence of training data.

To formulate the Bayesian AR-DenseED (BAR-DenseED) model, we wish to account for the two major uncertainty components: *aleatoric* uncertainty which quantifies noise in the observations and *epistemic* uncertainty which captures inherit uncertainty in the model [41]. Epistemic uncertainty is associated with the confidence of the model's predictions which is influenced by factors such as limited training scenarios, limited expressibility of the model, etc. For DNNs, epistemic uncertainty is most commonly captured by placing priors on the parameters of the model often being formulated as a Bayesian neural network [9]. Aleatoric uncertainty involves the noise that potentially exists in the data on which the model is trained on, and is often captured by placing a distribution over the model's outputs [42]. In a data-driven sense, aleatoric uncertainty arises from the simulators or sensors used to collect the training data. In the physics-constrained learning paradigm, we will interpret aleatoric uncertainty as the quantification of error associated with the truncation error introduced when formulating the physics-constrained loss function. As discussed in Section 3.1, the physics-constrained optimization of the auto-regressive model is posed as the minimization of the error between the model's predictions and a numerical time-integrator of the same time-step size. However, this numerical time-integrator introduces truncation error:

$$\boldsymbol{u}_i^{n+1} = T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^{n+1}, F_{\Delta x}\right) + E_{\Delta t} + E_{\Delta x}, \tag{12}$$

where $E_{\Delta t}$ and $E_{\Delta x}$ denote the error associated with the discretization of the temporal and spatial derivatives, respectively. In the deterministic case, such errors are neglected, however, depending on the resolution of the spatial discretization or the time-step size these errors can impact a numerical solver's accuracy. For most numerical solvers using explicit time integration schemes, the bulk of this error arises from the discretization of the spatial derivatives. Alternatively, when predicting large time-steps with implicit methods the discrete time integration can become the primary source of error due to numerical diffusion [43].

### 4.1. Posterior formulation

With an idea of the sources of uncertainty we wish to account for, let us start with defining a posterior over the model parameters. In a data-driven model, the likelihood captures the probability of the observations for a given model. For physics-constrained learning where no observations are available, we take as "target data" the prediction of the numerical time-integrator $\hat{\boldsymbol{u}}$. Similar to data-driven probabilistic models [2,6], we account for the potential discretization error that may arise from $T_{\Delta t}$ through additive output-wise noise, $\epsilon$, for a single arbitrary time-step:

$$\hat{\boldsymbol{u}}^i = f\left(\boldsymbol{\chi}^i, \mathbf{w}\right) + \epsilon, \quad p\left(\epsilon\right) = \mathcal{N}\left(\epsilon \,|0, \beta^{-1}\boldsymbol{I}_d\right), \tag{13}$$

where for mathematical convenience, we represent the discretized state variables $\boldsymbol{u}$ and $\hat{\boldsymbol{u}}$ as vectors in $\mathbb{R}^d$. $\boldsymbol{I}_d$ denotes the identity matrix in $\mathbb{R}^{d\times d}$. The additive noise is taken as Gaussian with a *learnable* precision $\beta$. Thus the likelihood for a single time-step, $i$, becomes the following:

$$\begin{aligned} p\left(\hat{\boldsymbol{u}}^i|\boldsymbol{\chi}^i, \mathbf{w}, \beta\right) &= \mathcal{N}\left(\hat{\boldsymbol{u}}^i|f\left(\boldsymbol{\chi}^i, \mathbf{w}\right), \beta^{-1}\boldsymbol{I}_d\right), \\ &= \mathcal{N}\left(T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right)|f\left(\boldsymbol{\chi}^i, \mathbf{w}\right), \beta^{-1}\boldsymbol{I}_d\right). \end{aligned} \tag{14}$$

Both the inputs to the model $\boldsymbol{\chi}^i$ and time-integrator $\boldsymbol{\mathcal{U}}^i$ are found from the deterministic evolution of the model until time-step $i$. Under the Markov assumption, we can formulate the likelihood of an entire time-sequence as the product of individual steps:

$$\begin{aligned} p\left(\hat{\boldsymbol{u}}^N, \ldots, \hat{\boldsymbol{u}}^1|\boldsymbol{u}_0, \mathbf{w}, \beta\right) &= \prod_{i=1}^{N} p\left(\hat{\boldsymbol{u}}^i|\boldsymbol{\chi}^i, \mathbf{w}, \beta\right), \\ &= \prod_{i=1}^{N} \mathcal{N}\left(T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right)|f\left(\boldsymbol{\chi}^i, \mathbf{w}\right), \beta^{-1}\boldsymbol{I}_d\right). \end{aligned} \tag{15}$$

Note that the number of past time-steps contained in the model's input, $\boldsymbol{\chi}^i$, is analogous to the order of the Markov chain. In this likelihood, the numerical time-integrator, $T_{\Delta t}$, can be interpreted as calculating the target $\hat{\boldsymbol{u}}^i$ on-the-fly. Thus the evaluation of this likelihood function is in fact still data-less.

**Remark 3.** To find the maximum likelihood estimate (MLE), minimization of the negative log likelihood is often taken as the optimization objective [44]. In this likelihood formulation, when minimizing the negative log likelihood in Eq. (15), one recovers the standard $L_2$ loss as previously shown in Eq. (8) for the deterministic model. Thus the MLE is equivalent to the minimization of the strong residual of the discretized PDE for both implicit and explicit time integration schemes indicating the appropriateness of the selected likelihood for our physics-constrained model.

A gamma prior is assigned to the noise precision $\beta$:

$$p(\beta) = \Gamma\left(\beta|a_1, b_1\right), \tag{16}$$

where $a_1$ and $b_1$ are the shape and rate parameters, respectively. The hyper-parameters of the $\beta$ prior are set based on the *a priori* estimate of the magnitude of the discretization error of $T_{\Delta t}$ and $F_{\Delta x}$. Since we intend to build large time-step surrogate models with $CFL > 1$, we will assume that the majority error arises from the time integration method. However, the following procedure can easily be extended to account for spatial discretization error as well.

Given an arbitrary system, we can express the temporal truncation error in the following formula which is standard in Richardson extrapolation [45]:

$$\begin{aligned} \hat{\boldsymbol{u}}^i &= T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right) + c_0\left(\Delta t\right)^{k_0} + c_1\left(\Delta t\right)^{k_1} + c_2\left(\Delta t\right)^{k_2} + \ldots \\ &= T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right) + c_0\left(\Delta t\right)^{k_0} + \mathcal{O}\left(\left(\Delta t\right)^{k_1}\right), \end{aligned} \tag{17}$$

where $c_i$ are unknown constants, and $k_i$ are constants denoting the "order" of the error term such that $(\Delta t)^{k_i} > (\Delta t)^{k_{i+1}}$. Under the assumption that higher-order terms are negligible, we take the expected value of our prior to be $\mathbb{E}\left(\beta^{-1}\right) = c_0 (\Delta t)^{k_0}$. Additionally this prior is given a large variance to reduce its strength. The parameters $c_0$ and $k_0$ can be estimated based on the magnitude of the state variables and the order of accuracy of the temporal discretization, respectively [46]. In this work, $c_0$ is set to be approximately 20% the maximum value of the quantity of interest and $k_0$ is set to three which over estimates the accuracy of the second-order accurate time integration method used. This encourages the model to be more accurate at the start of training rather than attributing initial prediction discrepancies to noise.

As is standard for Bayesian neural networks, the network's $K$ learnable parameters, $\mathbf{w}$, are treated as random variables. Due to the large number of weights in our model, we propose a fully factorizable zero mean Gaussian with a Gamma-distributed precision scalar $\alpha$:

$$p(\mathbf{w}|\alpha) = \mathcal{N}\left(\mathbf{w}|0, \alpha^{-1}\boldsymbol{I}_K\right), \quad p(\alpha) = Gamma(\alpha|a_0, b_0), \tag{18}$$

where the rate parameter $a_0$ and the shape parameter $b_0$ are 0.5 and 10, respectively. This results in a prior with a Student's $\mathcal{T}$ density centered at zero that has a wider support region than a standard Gaussian. In our past works [2,6], a narrow Student's $\mathcal{T}$-distribution was used to more strongly promote sparsity [47], however it was found that a narrow prior would damage the predictive capability of BAR-DenseED. Thus the sparsity requirement is relaxed while still regulating the magnitude of the model's weights. We note that when one uses an optimizer with momentum and weight decay, such as ADAM [48], an implicit prior on the weights is enforced which is largely ambiguous [49]. Since we will ultimately approximate the posterior, this prior does not need to be accounted for in the formulation of the joint posterior used for optimization. As a result for a batch of $M$ i.i.d. training scenarios, $\mathcal{S} = \left\{\boldsymbol{u}_{0,i}\right\}_{i=1}^{M}$, the posterior of the network is as follows:

$$\begin{aligned} p(\mathbf{w}, \beta|\mathcal{S}) &\sim \prod_{j=1}^{M} p\left(\hat{\boldsymbol{u}}_j^N, \ldots, \hat{\boldsymbol{u}}_j^1 | \boldsymbol{u}_{0,j}, \mathbf{w}, \beta\right) p(\mathbf{w})\, p(\beta), \\ &\sim \prod_{j=1}^{M}\prod_{i=1}^{N}\left[p\left(\hat{\boldsymbol{u}}_j^i | \boldsymbol{\chi}_j^i, \mathbf{w}, \beta\right)\right] p(\mathbf{w}|\alpha)\, p(\alpha|a_0, b_0)\, p(\beta|a_1, b_1), \\ &\sim \prod_{j=1}^{M}\prod_{i=1}^{N}\left[\mathcal{N}\left(T_{\Delta t}\left(\boldsymbol{\mathcal{U}}_j^i, F_{\Delta x}\right) | f\left(\boldsymbol{\chi}_j^i, \mathbf{w}\right), \beta^{-1}\boldsymbol{I}_d\right)\right] \mathcal{N}\left(\mathbf{w}|0, \alpha^{-1}\boldsymbol{I}_K\right) \\ &\qquad \Gamma(\alpha|a_0, b_0)\, \Gamma(\beta|a_1, b_1). \end{aligned} \tag{19}$$

**Remark 4.** The task of computing the maximum *a posteriori* probability (MAP) estimate is closely related to maximizing the likelihood with the addition of appropriate weight regularization that arises from the use of priors on the model's parameters [44]. Thus, the MAP estimation minimizes a regularized form of the previously considered $L_2$ deterministic loss function that was defined based on the discretized PDE residual.

### 4.2. Posterior approximation

The Bayesian paradigm seeks to represent model uncertainty by marginalizing out the model's parameters which results in the predictive distribution. This marginalization is often not analytically tractable and is usually approximated with Monte Carlo sampling of the posterior. In earlier works, Bayesian DNNs focused on the use of Monte Carlo or ensemble based methodologies [50,51,9]. However, with the number of parameters in such models growing exponentially larger over recent years, such traditional methods are computationally intractable. As a result, many recent Bayesian deep learning frameworks focus on variational methods that fit a proposal distribution over the true posterior of the model's parameters. Variational ideas have led to multiple developments including: Bayes by back-prop [52], Bayesian dropout approximation [53] and Stein variational gradient descent [54].

For sampling the posterior of BAR-DenseED, we will use a recently proposed Stochastic Weight Averaging Gaussian (SWAG) [55]. SWAG is an approximate Bayesian method that builds upon the Stochastic Weight Averaging (SWA), an optimization methodology where running averages of model parameters are kept during the stochastic gradient descent (SGD) procedure [56–58]. SWAG approximates the posterior in two phases:

1. The model of interest is first trained using traditional machine learning methods to minimize the negative log of the posterior defined in Eq. (19) (equivalent to solving for the MAP estimate). Specifically in this work, we optimize the model using the ADAM [48], an extension of SGD, with exponential learning rate decay.
2. Once the model has been trained, SGD is ran again at a constant learning rate. During this process, samples of the model's parameters are collected. The core idea is to use SGD to explore the local support region of the MAP estimate. These SGD iterations can provide useful information about the form of the posterior which can then be used to approximate the posterior density function for full Bayesian inference.

In SWAG, the posterior over the BAR-DenseED parameters is approximated as a Gaussian distribution with $S$ samples of the model's parameters:

$$p(\boldsymbol{\theta}|\mathcal{S}) \sim \mathcal{N}(\boldsymbol{\theta}_{SWA}, \boldsymbol{\Sigma}_{SWA}), \quad \boldsymbol{\theta} \equiv \{\mathbf{w}, ln(\beta)\}, \tag{20}$$

which is standard when using the Laplace approximation. We note that the noise precision, $\beta$, has a log-normal posterior approximation to ensure that it is positive. The mean and the covariance are approximated using the model parameters proposed by SGD:

$$\boldsymbol{\theta}_{SWA} = \frac{1}{S}\sum_{i=1}^{S}\boldsymbol{\theta}_i, \quad \boldsymbol{\Sigma}_{SWA} = \frac{1}{2}\left(\boldsymbol{\Sigma}_{Diag} + \boldsymbol{\Sigma}_{lr}\right), \tag{21}$$

$$\bar{\boldsymbol{\theta}^2} = \frac{1}{S}\sum_{i=1}^{S}\boldsymbol{\theta}_i^2, \quad \boldsymbol{\Sigma}_{Diag} = \text{Diag}\left(\bar{\boldsymbol{\theta}^2} - \boldsymbol{\theta}_{SWA}^2\right), \tag{22}$$

$$\boldsymbol{\Sigma}_{lr} = \frac{1}{K-1}\boldsymbol{D}\boldsymbol{D}^T, \quad \boldsymbol{D}_i = (\boldsymbol{\theta}_i - \boldsymbol{\theta}_{SWA}), \tag{23}$$

where $\boldsymbol{\theta}_i$ are the model parameters at epoch $i$, and $\boldsymbol{D} \in \mathbb{R}^{K\times H}$ is a deviation matrix consisting of the $H$ most recent parameter samples forming a low-rank approximation. $\boldsymbol{D}_i \in \mathbb{R}^K$ is a column of this deviation matrix. This results in a simple sampling method outlined in Algorithm 3 approximating the posterior of the model parameters for a time series prediction. While many other Bayesian approaches exist, we selected to use SWAG for three main reasons: its simplicity in both formulation and implementation, its non-invasive nature to the learning of the neural network and its low computational overhead compared to other Bayesian approaches. These factors are important for the learning of time series problems since learning generally becomes significantly more expensive and difficult.

**Algorithm 3:** Approximating the BAR-DenseED Posterior with SWAG [55].

**Input**: Pre-trained model parameters optimized for MAP: $\boldsymbol{\theta}_0$; Number epochs to run: $N$; Time series training length: $T$; Sample frequency: $p$; Size of low-rank approximation matrix: $H$; Learning rate: $\eta_{swag}$; Negative log posterior: $\mathcal{L}_p$

$\bar{\boldsymbol{\theta}} = \boldsymbol{\theta}_0, \quad \bar{\boldsymbol{\theta}^2} = \boldsymbol{\theta}_0^2$;
$n = 1$ ; ▷ Number of sampled models
**for** $i = 1$ **to** $N$ **do**
  **for** $j = 1$ **to** $T$ **do**
    $\boldsymbol{u}^j \leftarrow f\left(\boldsymbol{\chi}^j, \mathbf{w}\right)$ ; ▷ Forward pass of the model
    $\hat{\boldsymbol{u}}^j = T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^j, F_{\Delta x}\right)$ ; ▷ Time integration
    $\boldsymbol{\theta}_j = \boldsymbol{\theta}_{j-1} - \eta_{swag}\nabla_{\boldsymbol{\theta}}\mathcal{L}_p\left(\hat{\boldsymbol{u}}^j, \boldsymbol{u}^j\right)$ ; ▷ Multi-step back-prop.
    $\boldsymbol{\chi}^{j+1} \leftarrow \{\boldsymbol{u}^j, \boldsymbol{\chi}^j[0], \boldsymbol{\chi}^j[1], \ldots, \boldsymbol{\chi}^j[k-1]\}$ ; ▷ Update input
  **if** *Mod(i,p) = 0* **then**
    $\bar{\boldsymbol{\theta}} = \frac{n\bar{\boldsymbol{\theta}} + \boldsymbol{\theta}_i}{n+1}$ ; ▷ First moment update
    $\bar{\boldsymbol{\theta}^2} = \frac{n\bar{\boldsymbol{\theta}^2} + \boldsymbol{\theta}_i^2}{n+1}$; ▷ Second moment update
    $\hat{\boldsymbol{D}} = \text{concat}\left(\hat{\boldsymbol{D}}[:, -(H-1):], \boldsymbol{\theta}_i, \text{dim} = 1\right)$;
    $n = n + 1$;
$\boldsymbol{D} = \hat{\boldsymbol{D}} - \bar{\boldsymbol{\theta}}$; ▷ Low-rank deviation matrix
**Output**: $\boldsymbol{\theta}_{SWA} = \bar{\boldsymbol{\theta}}$; $\boldsymbol{\Sigma}_{Diag} = \bar{\boldsymbol{\theta}^2} - \boldsymbol{\theta}_{SWA}^2$; $\boldsymbol{D}$;

Of significant importance is the sampling learning rate, which is directly related to both the shape as well as the convergence rate of the posterior. As discussed in [55], this learning rate should be large enough to sufficiently explore the support region of the minima the model has converged to. However, $\eta_{swag}$ should not be too large such that the model potentially jumps to other local minima during the collection of samples to approximate the posterior. This is due to the use of a single mode Gaussian as the approximate density for which multimodal data cannot be handled robustly.

**Remark 5.** While vanilla stochastic gradient descent is shown here for simplicity of illustrating the SWAG algorithm, fundamentally, one can use other optimization methods as well to collect SWAG samples such as gradient descent with momentum [55]. Similar to the use of various Markov chain Monte Carlo methods for approximating density functions, different stochastic optimization methods can be used to approximate the density of the posterior of the neural network.

### 4.3. Predictive statistics

As previously discussed, predictive statistics in a Bayesian framework are obtained by marginalizing or integrating out the parameters of the model. Given the previous Markov assumption, the predictive distribution can be posed in terms of a single arbitrary time-step. We will approximate the marginalization of the model parameters, often referred to as Bayesian model averaging, using Monte Carlo with $P$ parameter samples:

$$\begin{aligned} p\left(\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathcal{S}\right) &= \int p\left(\hat{\boldsymbol{u}}^*|f\left(\boldsymbol{\chi}^*,\mathbf{w}\right),\beta^{-1}\boldsymbol{I}\right)p\left(\mathbf{w},\beta|\mathcal{S}\right)d\mathbf{w}d\beta, \\ &\approx \frac{1}{P}\sum_{i=1}^{P} p\left(\hat{\boldsymbol{u}}^*|f\left(\boldsymbol{\chi}^*,\mathbf{w}_i\right),\beta_i^{-1}\boldsymbol{I}\right), \quad \{\mathbf{w}_i,\beta_i\}\sim p\left(\boldsymbol{\theta}|\mathcal{S}\right), \end{aligned} \tag{24}$$

where $\boldsymbol{\chi}^*$ and $\hat{\boldsymbol{u}}^*$ are the predictive inputs and outputs, respectively. The posterior, $p(\boldsymbol{\theta}|\mathcal{S})\equiv p(\mathbf{w},ln(\beta)|\mathcal{S})$, has been approximate by SWAG and is easily sampled from. The predictive expectation can be obtained as follows:

$$\begin{aligned} \mathbb{E}\left[\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathcal{S}\right] &= \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{S})}\left[\mathbb{E}\left(\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathbf{w},\beta\right)\right], \\ &= \mathbb{E}_{p(\mathbf{w}|\mathcal{S})}\left[f\left(\boldsymbol{\chi}^*,\mathbf{w}\right)\right] \approx \frac{1}{P}\sum_{i=1}^{P} f\left(\boldsymbol{\chi}^*,\mathbf{w}_i\right), \end{aligned} \tag{25}$$

where the additive output noise is not present due to its zero-mean Gaussian density. The predictive conditional covariance can also be obtained in a similar fashion:

$$\begin{aligned} \mathrm{Cov}\left[\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathcal{S}\right] &= \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{S})}\left[\mathrm{Cov}\left(\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathbf{w},\beta\right)\right] + \mathrm{Cov}_{p(\boldsymbol{\theta}|\mathcal{S})}\left(\mathbb{E}\left[\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathbf{w},\beta\right]\right), \\ &= \mathbb{E}_{p(ln(\beta)|\mathcal{S})}\left[\beta^{-1}\boldsymbol{I}\right] + \mathrm{Cov}_{p(\mathbf{w}|\mathcal{S})}\left(f\left(\boldsymbol{\chi}^*,\mathbf{w}\right)\right), \\ &\approx \frac{1}{P}\sum_{i=1}^{P}\left[\beta_i^{-1}\boldsymbol{I} + f\left(\boldsymbol{\chi}^*,\mathbf{w}_i\right)f\left(\boldsymbol{\chi}^*,\mathbf{w}_i\right)^T\right] - \mathbb{E}\left[\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathcal{S}\right]\mathbb{E}\left[\hat{\boldsymbol{u}}^*|\boldsymbol{\chi}^*,\mathcal{S}\right]^T, \end{aligned} \tag{26}$$

where $\mathbb{E}\left[\boldsymbol{u}^*|\boldsymbol{\chi}^*,\mathcal{S}\right]$ has been defined in Eq. (25). To predict an entire time series, each model is sampled at the first time-step and auto-regressed forward in time independently. Each set of parameters sampled from the posterior can be interpreted as an individual particle that is propagated forward in time. Thus as we predict further in time, we should expect the predictive variance of the model to gradually increase.

## 5. Kuramoto-Sivashinsky equation

The first physical system that we are interested in is the 1D Kuramoto-Sivashinsky (K-S) equation which is a fourth-order, nonlinear partial differential equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^4 u}{\partial x^4} = 0, \tag{27}$$

$$u(0,t) = u(L,t), \quad x\in[0,L], \quad t\in[0,T], \tag{28}$$

where $\nu$ is referred to as the "hyper-viscosity" and is set to $\nu = 1$ for the remainder of this section. The K-S equation is widely known for its chaotic behavior when the size of the periodic domain is sufficiently large (generally $L \geq 50$) in which the system becomes a spatio-temporally chaotic attractor [59]. The K-S PDE has attracted great interest as it serves as a prototypical problem for studying complex dynamics with its chaotic regime being weakly turbulent (as opposed to strong turbulence seen in the Navier-Stokes equations) [60,61]. Several physical systems such as chemical phase turbulence, plasma ion instabilities and flame front instabilities have all seen the K-S equation arise within them [62–64]. For our problem of interest, we take the domain $L$ to be $[0, 22\pi]$ putting the system well within its chaotic regime. The domain is discretized by 96 uniform cells and time-step is $\Delta t = 0.1$. Two sample responses of the K-S equation for two different initial conditions are illustrated in Fig. 4. During training and testing, we will ignore the initial transient state thus our initial conditions will be already fully developed "turbulence" ($t \geq 100$).

Our goal is for AR-DenseED to predict the chaotic response of the system accurately thus illustrating the potential of this model to predict physical dynamics. In the past, others have attempted to model this system by machine learning methods. Recently, in Pathak et al. [66] reservoir computing was used to predict the K-S system, however the model is trained on the past history of a specific state. Thus the model learns only for a specific initial condition. The recent formulations of physics-informed neural networks in Raissi et al. [15,39] have been able to work for learning a specific initial condition without training data, however these models have yet to be shown effective as a predictive surrogate.

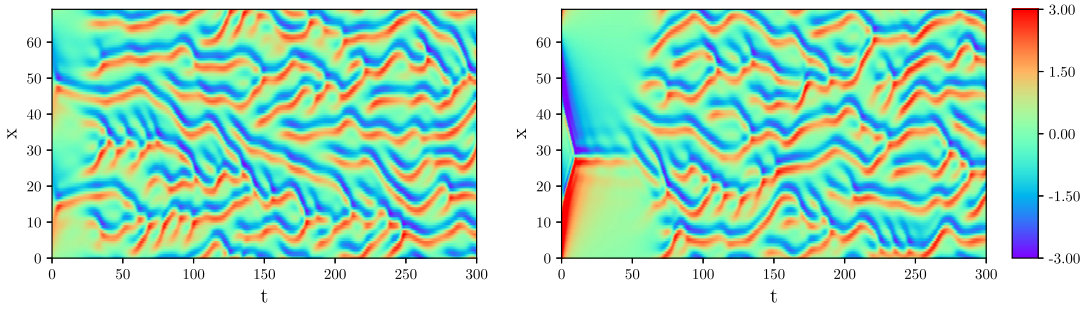



**Fig. 4.** The Kuramoto-Sivashinsky equation for two different initial states solved using the spectral ETDRK4 scheme [65].

The AR-DenseED used for the K-S equation consists of a single convolutional block, followed by a dense block, followed by a deconvolutional block resulting in a model with just over 4800 learnable weights. The two previous states of the system are used as inputs, $\boldsymbol{\chi}^{n+1} = \{\boldsymbol{u}^n, \boldsymbol{u}^{n-1}\}$. Similar to the numerical solver, the time-step size of the model is set to $\Delta t = 0.1$ with a spatial discretization of 96 points. As previously discussed, the negative log of the joint posterior in Eq. (19) is the loss function. For the physics-constrained loss function, the implicit Crank-Nicolson time integration is used and the remaining spatial gradients are discretized as follows:

$$\begin{aligned} T_{\Delta t}(\boldsymbol{\mathcal{U}}^n, F_{\Delta x}) &= \boldsymbol{u}^n + \Delta t\left[-0.5\left(F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^{n+1}) + F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^n)\right)\right], \\ F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^n) &= u^n u_x^n + u_{xx}^n + u_{xxxx}^n, \end{aligned} \tag{29}$$

$$uu_x = \frac{-u_{i+2}^2 + 8u_{i+1}^2 - 8u_{i-1}^2 + u_{i-2}^2}{24\Delta x}, \tag{30}$$

$$u_{xx} = \frac{-u_{i+2} + 16u_{i+1} - 30u_i + 16u_{i-1} - u_{i-2}}{12\Delta x^2}, \tag{31}$$

$$u_{xxxx} = \frac{-u_{i+3} + 12u_{i+2} - 39u_{i+1} + 56u_i - 39u_{i-1} + 12u_{i-2} - u_{i-3}}{6\Delta x^4}, \tag{32}$$

where the spatial gradients are approximated using fourth-order accurate finite difference discretizations that are implemented efficiently using convolutional operators. The model was trained for 100 epochs using 2560 training scenarios that were generated using a truncated Fourier series with random coefficients discussed in Appendix A.1. This Fourier series serves to approximate the physical turbulence of the system, and thus estimating the true distribution of the possible initial states $p(\boldsymbol{u}_0)$. During test time, we use 200 test cases of true turbulent initial states of the system obtained from a numerical simulator. This will demonstrate how one can use approximated training scenarios and physics-constrained learning to train a model that can be used on a true realization of the system. The training scenarios were mini-batched with a batch size of 256. During training the model was allowed to unroll itself in time up to 1000 time-steps to allow AR-DenseED to thoroughly explore the turbulent dynamics. Training on a single 1080Ti GPU took approximately 1.5 wall-clock hours. Additional details on the model and training parameters are discussed in Appendix A.

### *5.1. AR-DenseED deterministic predictions*

We start with the prediction of the deterministic AR-DenseED model. Predictive results are shown in Fig. 5 for three test initial conditions compared against a numerical solver. All predictions are obtained by only providing the initial state and evolving the system with 1000 consecutive iterations of the neural network. Overall the results are very impressive and are significant improvements compared to past literature despite only using a single initial state to predict. The model is able to maintain consistency with the numerical solver for between $t = [0, 30]$, but then diverges due to small prediction error that causes the model to shift its response as a result of the systems' chaotic nature. However, the predicted system remains qualitatively reasonable and stable for even extended times which is a significant advantage of our auto-regressive formulation.

For each test case, we calculate the spatial mean square error (MSE) at each time-step defined as:

$$\text{MSE}(t) = \frac{1}{N}\sum_{i=1}^{N}\left(u_i(t) - u_i^*(t)\right)^2, \tag{33}$$

where $N$, $u$ and $u^*$ are the total number of points used to discretize the domain, target value from the numerical simulator and the AR-DenseED prediction, respectively. The mean of this error value is shown in Fig. 6, where we can see the decay in the model accuracy at around $t = 20$ until AR-DenseED has fully diverged from the numerical simulator by

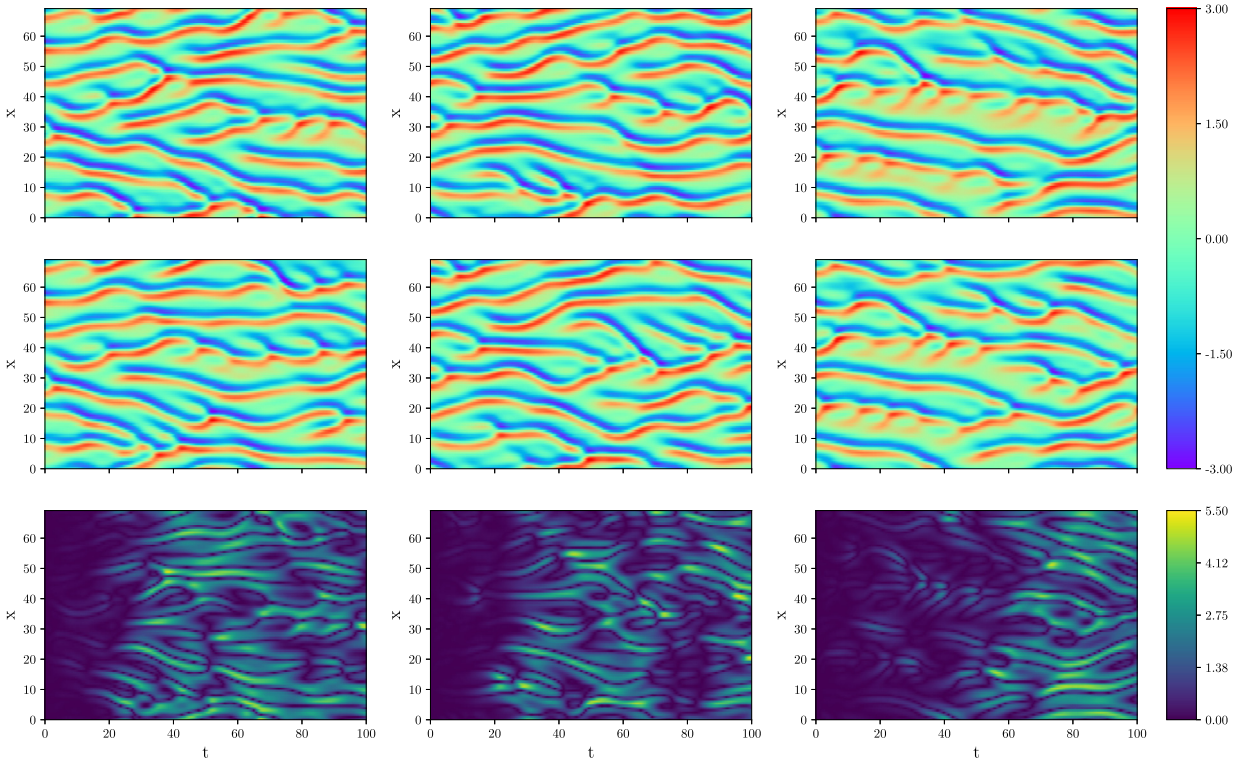



**Fig. 5.** Three test predictions of the Kuramoto-Sivashinsky equation using AR-DenseED. (Top to bottom) Target field solved system using the spectral ETDRK4 scheme, AR-DenseED prediction and finally the $L_1$ error.

**Table 1**
Wall-clock time for both spectral ETDRK4 scheme and AR-DenseED to simulate 5000 time-steps of the Kuramoto-Sivashinsky system. Wall-clock time estimates were obtained by averaging 10 independent simulation run times.

| | Hardware | Backend | $\Delta t$ | Wall-clock time (s) |
|---|---|---|---|---|
| Spectral | Intel Xeon E5-2680 | Matlab | 0.1 | 0.185 |
| AR-DenseED | Intel Xeon E5-2680 | PyTorch | 0.1 | 17.042 |
| AR-DenseED | GeForce GTX 1080 Ti | PyTorch | 0.1 | 12.225 |

$t > 40$. However, the predictions in Fig. 5 still appear to be physical despite not matching the numerical simulator. To illustrate that the model predicts physical turbulence, we compare the average energy spectral density in Fig. 7 for a randomly selected test case. Since this is a turbulent statistic, the energy density profile is the same regardless of the particular initial condition. This statistic was obtained by averaging time-steps between $t = [0, 500]$. This means our AR-DenseED is stable in its predictions for at least 5000 time-steps, far beyond its training time-range. This would not be possible with the traditional fully connected neural network approach for solving PDEs. The AR-DenseED is accurate with the simulation results for the larger wavelengths where the majority of the energy is concentrated. Additionally the AR-DenseED is able to correctly generate turbulence with the greatest energy at a similar wavelength as the simulation ($\text{Hz} \in [0.1, 0.15]$). While the model and simulation results begin to deviate from the numerical solution for smaller wavelengths, the energy decays at these higher frequencies meaning that the absolute error between the model and simulation is significantly less. Thus we confidently conclude that the AR-DenseED has truly learned to predict physical turbulence of the K-S equation.

The average prediction wall-clock time of both the spectral ETDRK4 scheme versus the AR-DenseED are given in Table 1. In this situation, the AR-DenseED fails to be a computational effective surrogate model compared to the highly-efficient spectral method which is able to take advantage of fast Fourier transform. However, the K-S system was able to provide an excellent illustration of how AR-DenseED can model complex, non-linear, chaotic systems. We will show in the following sections how AR-DenseED can be computationally more efficient than traditional numerical solvers.

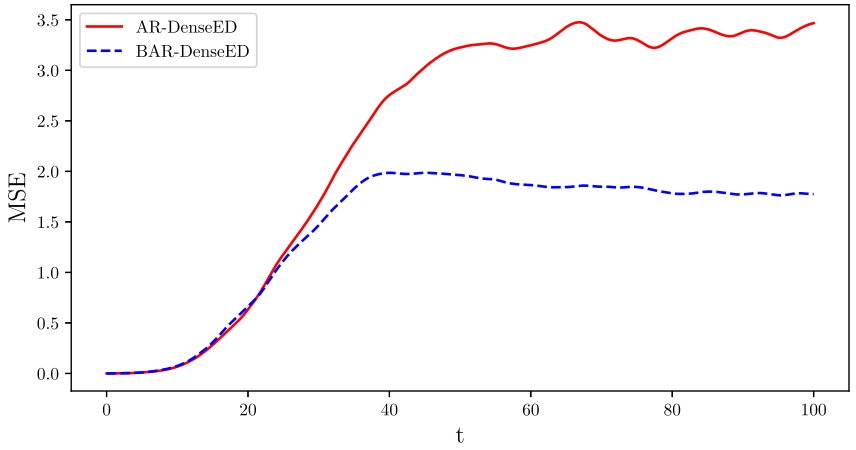



**Fig. 6.** The mean MSE as a function of time for a test set of 200 cases for the Kuramoto-Sivashinsky system. The error of BAR-DenseED is calculated using the expected value of the predictive distribution approximated using 30 samples of the posterior.

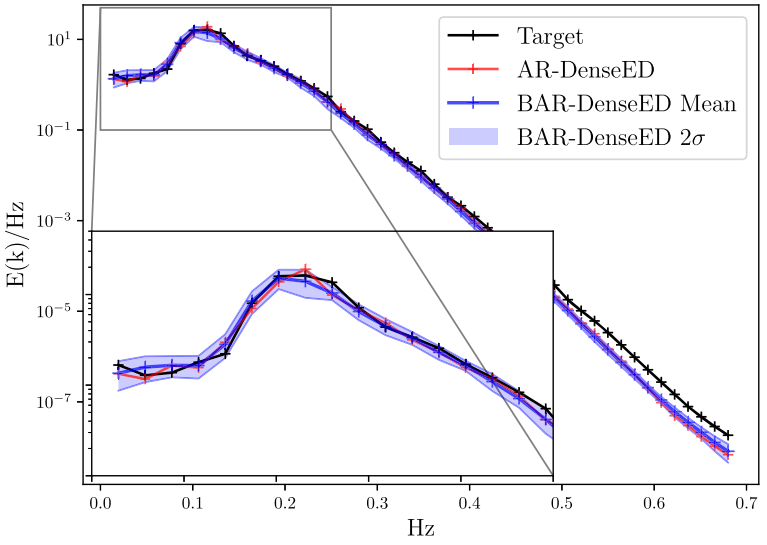



**Fig. 7.** The time-averaged spectral energy density of the simulated result using the spectral ETDRK4 scheme (target), AR-DenseED deterministic prediction and BAR-DenseED empirical mean and standard deviation calculated from 30 posterior samples. The averaged spectral energy density is the square of the modulus of the discrete Fourier transform over the domain, $x \in [0, 22\pi]$, time-averaged between $t \in [0, 500]$ [67].

### *5.2. BAR-DenseED probabilistic predictions*

To approximate the posterior with SWAG, 100 samples of the model's parameters were collected. This yielded reasonably diverse but accurate models and was found to be enough samples for $\boldsymbol{\theta}_{SWA}$ to converge. During this period the learning rate was lowered to 1e−10 for the neural network weights and 1e−6 for the additive output noise. While these learning rates may appear too small to sufficiently explore the local loss surface, for an auto-regressive model this was discovered to be a necessity as even very small changes to the parameters can have profound response changes during test time. Larger learning rates for SWAG sampling were found to produce models that were unstable.

We plot eight samples from the approximate posterior in Fig. 8 for a single test case with the target result in the top left. Due to the chaotic nature of the K-S system or the so called "butterfly effect", samples from the posterior start with a similar response up for $t < 40$ and deviate for larger time values producing completely unique responses. Similar to the deterministic case, we calculate the mean squared error defined in Eq. (33) using the expected predictive response using 30 model samples for 200 test cases and plot the mean error value at each time-step in Fig. 6. Although, it appears BAR-DenseED performs much better than AR-DenseED for later time-steps this is due to the field being averaged out to around zero. Thus the predictive performance between the deterministic and Bayesian model is essentially equivalent in this case. We can propagate this uncertainty to the averaged spectral energy density illustrated in Fig. 7 where 30 model samples are used to calculate the spectral density. At the largest wave lengths ($Hz < 0.3$), we can see that the model has

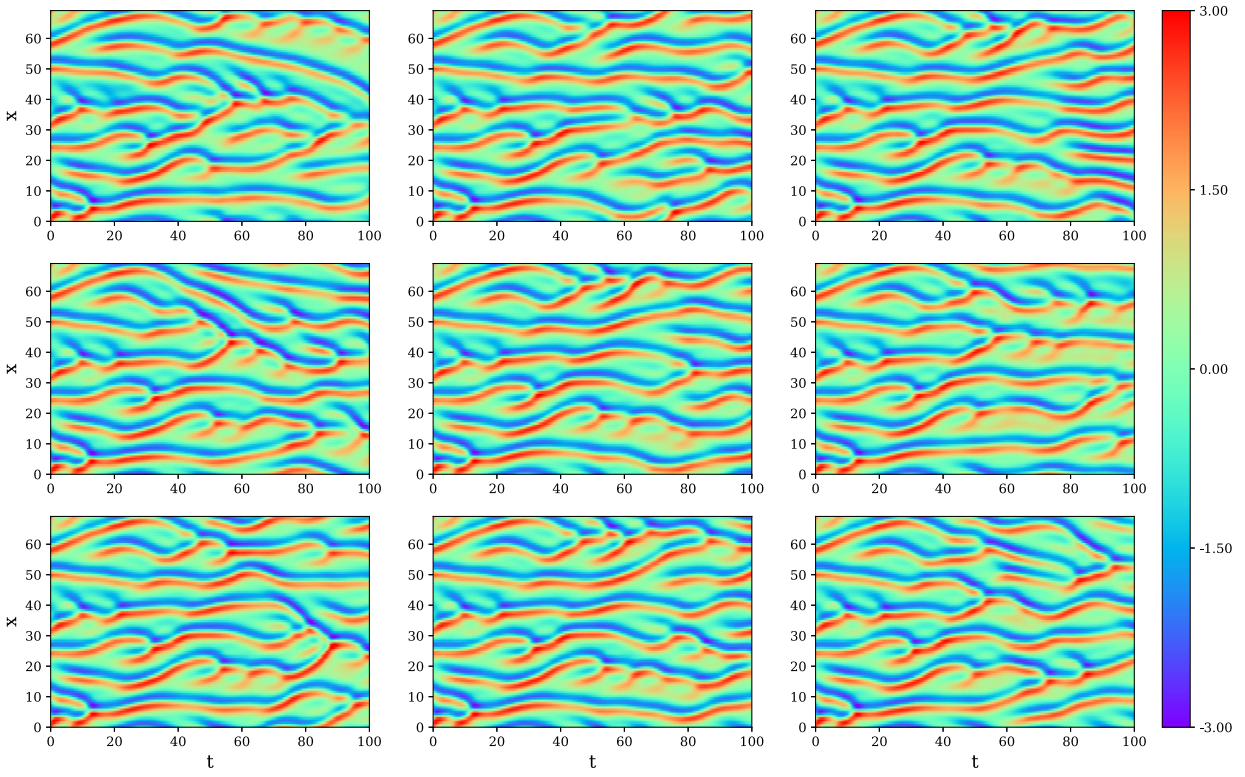



**Fig. 8.** Samples from the posterior of BAR-DenseED approximated using SWAG for the Kuramoto-Sivashinsky system. The top left is the simulated result using the spectral ETDRK4 scheme.

reasonable error bars that are able to capture the true solution. For smaller wave-lengths, the predicted energy density appears to be consistent between the samples.

## 6. 1D viscous Burgers' equation

Let us now consider the 1D viscous Burgers' equation in a periodic domain:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0, \tag{34}$$

$$u(0,t) = u(L,t), \quad x \in [0,L], \quad t \in [0,T], \tag{35}$$

where $u$ is the velocity and $\nu$ is the viscosity. The Burgers' equation is a fundamental PDE that arises in multiple areas ranging from fluid dynamics to traffic flow. It is most recognized for its characteristic shock formations [68]. While cases of the 1D Burgers' equation have been recovered by machine learning models in the past [15], ultimately these have been for relatively simple initial conditions consisting of a single shock. Here, we would like to model much more complex dynamics by having a variable initial condition that contains multiple waves. Consider a domain $x \in [0,1]$ with a constant viscosity of $\nu = 0.0025$ and the random initial condition given by a Fourier series with random coefficients:

$$\begin{aligned} w(x) &= a_0 + \sum_{l=1}^{L} a_l \sin(2l\pi x) + b_l \cos(2l\pi x), \\ u(x,0) &= \frac{2w(x)}{\max_x |w(x)|} + c, \end{aligned} \tag{36}$$

where $a_l, b_l \sim \mathcal{N}(0,1)$, $L = 4$ and $c \sim \mathcal{U}(-1,1)$. In [69], the authors modeled a simpler random initial condition for the 1D viscous Burgers' system using a LSTM based model with some success. However, this work reduced the complexity of the problem system by learning a reduced-order model rather than the true system. Simulated system responses for several initial conditions are shown in Fig. 9. We can see that the underlying dynamics of the system are fairly complex due to multiple shocks forming and then combining at later time-steps. Shocks that intersect then combine and move in a different trajectory, making this a difficult system to predict accurately.

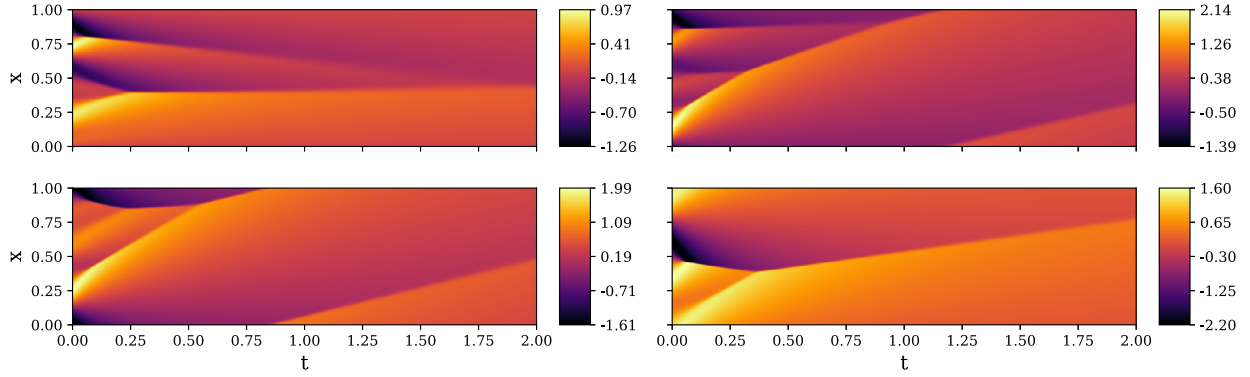



**Fig. 9.** 1D viscous Burgers' equation simulations for four various initial conditions solved using Fenics finite element package [70].

The auto-regressive model used for the 1D Burgers' system is similar to the one used for the K-S system in Section 5 with a few modifications. For this system the five previous time-steps are used as inputs, $\boldsymbol{\chi}^{n+1} = \{\boldsymbol{u}^n, \boldsymbol{u}^{n-1}, \ldots, \boldsymbol{u}^{n-4}\}$ resulting in about 13000 learnable weights. The time-step value of the model is $\Delta t = 0.005$ with a spatial discretization of 512 points. This places the CFL number of the model well above one based on the maximum potential velocity for the specified random initial condition. Again the negative log of the joint posterior in Eq. (19) is the loss function with the implicit Crank-Nicolson time integration. The remaining spatial gradients are discretized as follows:

$$\begin{aligned} T_{\Delta t}(\boldsymbol{\mathcal{U}}^n, F_{\Delta x}) &= \boldsymbol{u}^n + \Delta t\left[-0.5\left(F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^{n+1}) + F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^n)\right)\right], \\ F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^n) &= u^n u_x^n - \nu u_{xx}^n, \end{aligned} \tag{37}$$

$$uu_x = \frac{u_{i+1}^2 - u_{i-1}^2}{4\Delta x}, \quad u_{xx} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}, \tag{38}$$

where the spatial gradients are approximated using second-order accurate approximations that are implemented efficiently using convolutional operators. The model was trained for 100 epochs with 2560 training scenarios randomly sampled from Eq. (36) and allowed to unroll a maximum of 200 time-steps from its initial state. Another 200 samples from Eq. (36) are used as a test set for assessing the models performance. Additional details on the model and training can be found in Appendix B.

### 6.1. AR-DenseED deterministic predictions

During testing we use the trained AR-DenseED to predict 400 time-steps from its initial state. This means that half of its prediction ($t > 1.0$) is extrapolation beyond the time range used during training. Four test cases are plotted in Fig. 10, from which we can see that the AR-DenseED is able to predict this system accurately without any training data. The target response is a high-fidelity finite element method (FEM) simulation at time-step size $\Delta t = 0.001$, which is five times smaller than our surrogate model. Overall, the AR-DenseED is able to accurately predict the shock formations and intersections with very distinct shock discontinuities. In our past work [3], we have found that convolutional neural networks like AR-DenseED can predict very sharp features much better than fully-connected models. Additionally, we can see the reason why this system is difficult for surrogate modeling as one slight miscalculation in the shock intersection can result in compounding error. This is illustrated in Fig. 10 where in some of the test cases the model gives a good prediction but a slight miscalculation in the shock intersection results in a growing error. The excellent extrapolation capabilities of AR-DenseED are also shown for which the model is able to yield accurate predictions far beyond its initial training range.

Now we consider the full 200 test cases with target solutions provided by the high-fidelity FEM simulation. For each test case, we calculate the spatial mean square error (MSE) defined in Eq. (33). In Fig. 11, we plot the mean and median of the MSE for the entire test set. We can see an initial spike in the error during the initial shock formations/intersections that then decays as the system also decays. However, the MSE can be slightly misleading to the actual quality of the prediction for this system since a small deviation in shock trajectory can potentially yield a growing error. Thus, we also compute the energy square error (ESE) for a 1D domain:

$$\begin{aligned} \mathrm{ESE}(t) &= \left[\int_0^1 \frac{(u(x,t))^2}{2}dx - \int_0^1 \frac{(u^*(x,t))^2}{2}dx\right]^2, \\ &= \left[\frac{1}{N}\sum_{i=1}^{N}\frac{(u_i(t))^2}{2} - \frac{1}{N}\sum_{i=1}^{N}\frac{(u_i^*(t))^2}{2}\right]^2, \end{aligned} \tag{39}$$

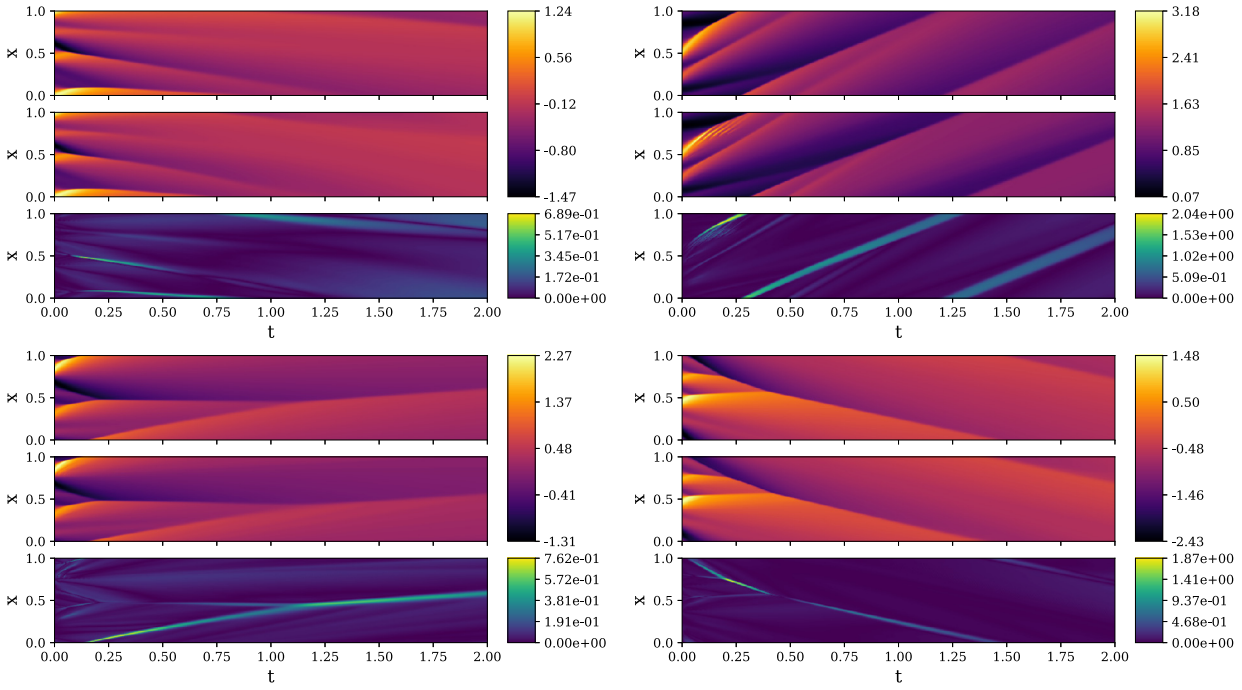



**Fig. 10.** AR-DenseED predictions for four test initial conditions of the 1D Burgers' system. (Top to bottom) FEM target solution, AR-DenseED prediction, and $L_1$ error.

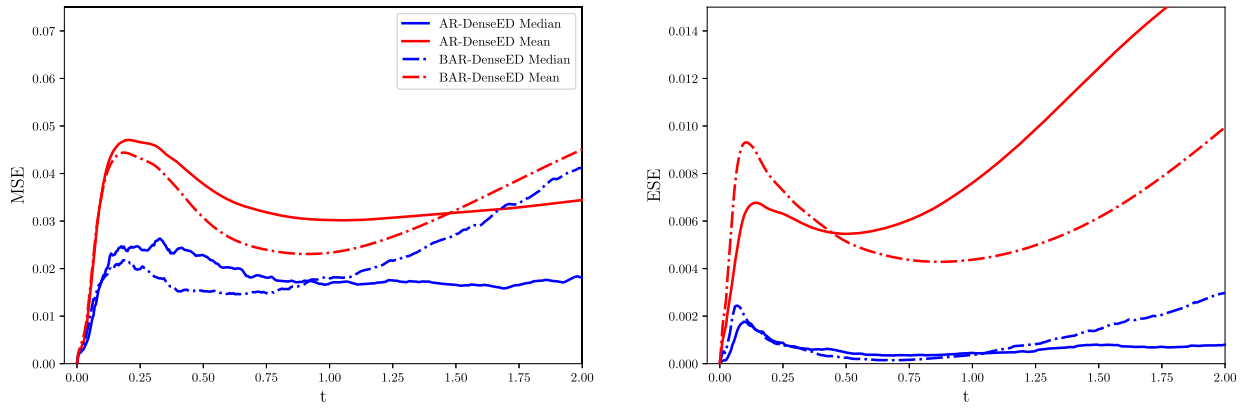



**Fig. 11.** (Left to right) The mean square error (MSE) and energy squared error (ESE) as a function of time for a test set of 200 cases for the 1D Burgers' system. The error of BAR-DenseED is calculated using the expected value of the predictive distribution approximating using 30 samples of the posterior.

which instead captures the discrepancy of the total energy, $u^2/2$, within the domain, making this metric invariant to shock location. Similarly, we plot the mean and median of the ESE for the 200 test cases in Fig. 11. For the ESE, we also see an initial spike followed by stable performance until the extrapolation region where the error begins to grow. This behavior is to be expected as the model moves further from the initial training range. Additionally, for both plots, we note that the mean error is consistently higher than the median which is a clear indication of outlier test cases that perform extremely poorly compared to the majority. Unfortunately, this is a core drawback of the auto-regressive approach; if the initial prediction is poor this error will only grow as time progresses. Finally, we compare the prediction computational cost of this surrogate model with both FEM and finite difference method (FDM) using fourth-order Runge-Kutta time integration in Table 2. AR-DenseED is significantly computationally cheaper than the traditional methods, making it a potentially useful surrogate.

### 6.2. BAR-DenseED probabilistic predictions

For the posterior approximation, 90 samples of the networks parameters were collected using a learning rate of $4e-8$. Several samples from the posterior are illustrated in Fig. 12 where slight differences in the predictions in earlier times

**Table 2**
Wall-clock time of finite element, finite difference and AR-DenseED to simulate 400 time-steps of the 1D-Burgers' system. Wall-clock time estimates were obtained by averaging 10 independent simulation run times.

| | Hardware | Backend | $\Delta t$ | Wall-clock (s) |
|---|---|---|---|---|
| Finite Element | Intel Xeon E5-2680 | Fenics | 0.0005 | 43.696 |
| Finite Element | Intel Xeon E5-2680 | Fenics | 0.001 | 22.645 |
| Finite Element | Intel Xeon E5-2680 | Fenics | 0.005 | 7.450 |
| Finite Difference | Intel Xeon E5-2680 | PyTorch | 0.0005 | 4.856 |
| Finite Difference | GeForce GTX 1080 Ti | PyTorch | 0.0005 | 12.8359 |
| Finite Difference | Intel Xeon E5-2680 | PyTorch | 0.001 | 2.862 |
| Finite Difference | GeForce GTX 1080 Ti | PyTorch | 0.001 | 6.264 |
| AR-DenseED | Intel Xeon E5-2680 | PyTorch | 0.005 | 1.286 |
| AR-DenseED | GeForce GTX 1080 Ti | PyTorch | 0.005 | 0.705 |

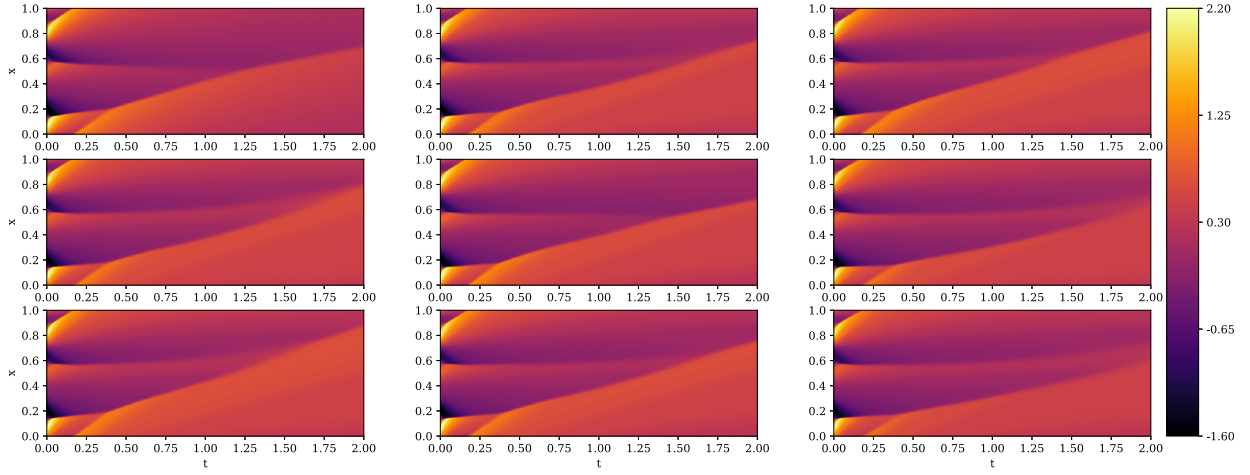



**Fig. 12.** Samples from the posterior of BAR-DenseED approximated using SWAG for the 1D Burgers' system. The top left is the simulated result using the finite element method.

change the final location of the wave at later times. The predictive expectation and variance computed using 30 model samples is plotted for four test cases in Fig. 13. We can clearly see that the bulk of the variance in the predictions occurs precisely where the shocks form/intersect as expected. Similar to the deterministic model, we also plot the mean squared error and energy squared error of a test set of 200 cases using the expected prediction of BAR-DenseED in Fig. 11. The Bayesian framework is able to generally out perform the deterministic case. One of the main reasons for this increase in accuracy is the reduction of outliers where AR-DenseED would yield an unsatisfactory prediction for only a small percentage of test cases. Due to the model averaging in the predictive expectation, BAR-DenseED is able to more robustly handle these test cases where some individual predictions may be very poor.

To better understand the uncertainty of BAR-DenseED and how it changes as the time series progresses, we plot several instantaneous profiles for two randomly selected test cases in Figs. 14 and 15. The first three profiles at $t = 0.10$, $t = 0.25$ and $t = 0.75$ are within the time-range that was used during training. The last profile at $t = 1.50$ is considered extrapolation as it lies outside the training time-range. In addition to the BAR-DenseED profile, the predicted solutions of the numerical solvers discussed in Table 2 are also shown. We note that there is a clear prediction discrepancy between the FEM and FDM solutions, largely due to different numerical discretizations. For this system, we hold the FEM solution as the higher accuracy method.

From both test cases, we can observe several important trends: the first is that for the earlier times the model compares well with the numerical solvers. Second, as the shocks form, we can notice large spikes in the standard deviation at these locations which corresponds to the behavior seen in Fig. 13. Finally, we see the range of the error bars increases as the model begins to extrapolate indicating the model is less confident as it moves farther from its training range. However, it is clear that in the extrapolated regions the model is still aware of the correct structure with the error bars capturing the true solution.

## 7. 2D coupled Burgers' equation

Lastly, we will consider the 2D coupled Burgers' system:

$$\boldsymbol{u}_t + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nu \Delta \boldsymbol{u} = 0, \tag{40}$$

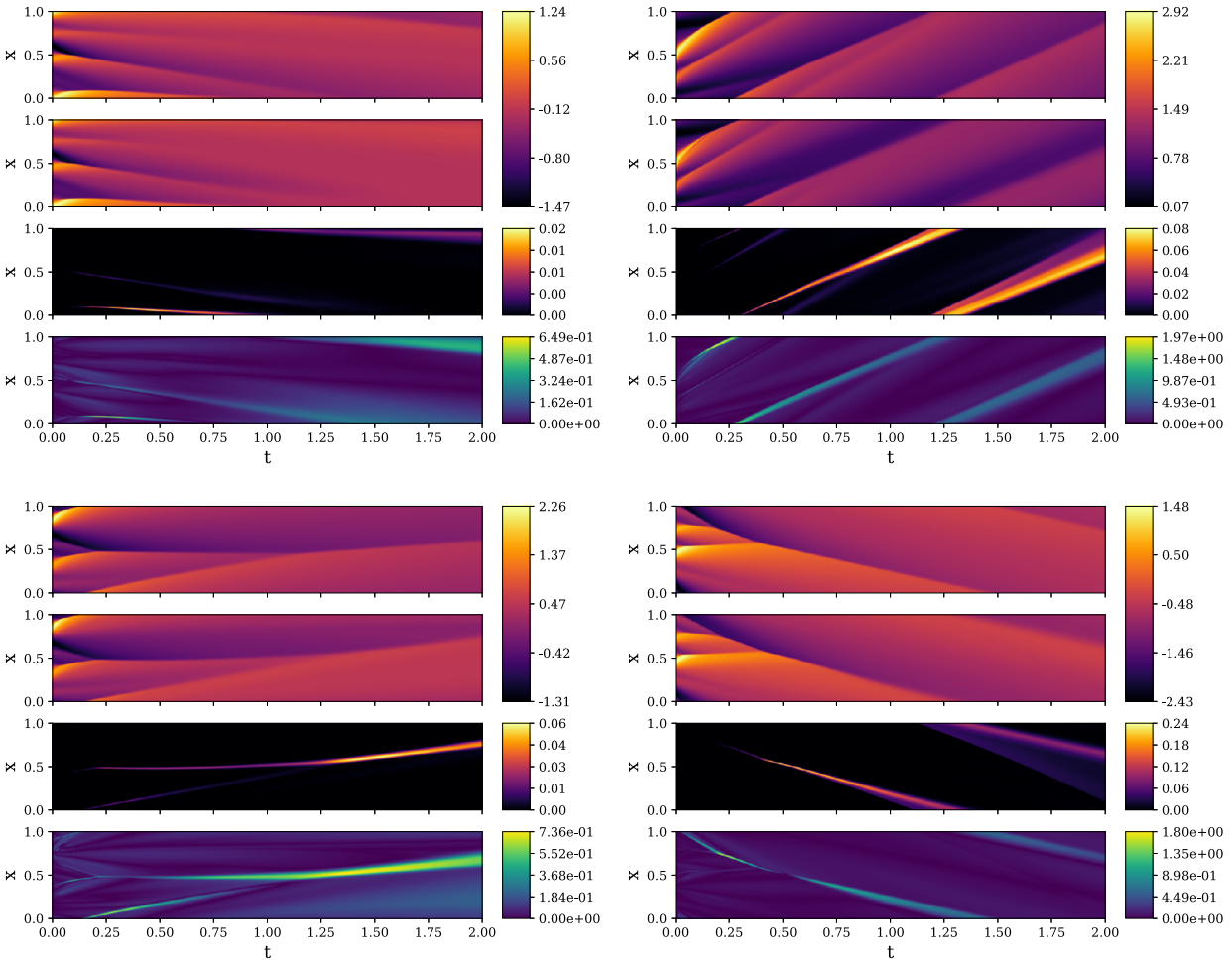



**Fig. 13.** BAR-DenseED predictions for four test initial conditions of the 1D Burgers' system. (Top to bottom) FEM target solution, BAR-DenseED expected response, BAR-DenseED variance and $L_1$ error between the target and expected values.

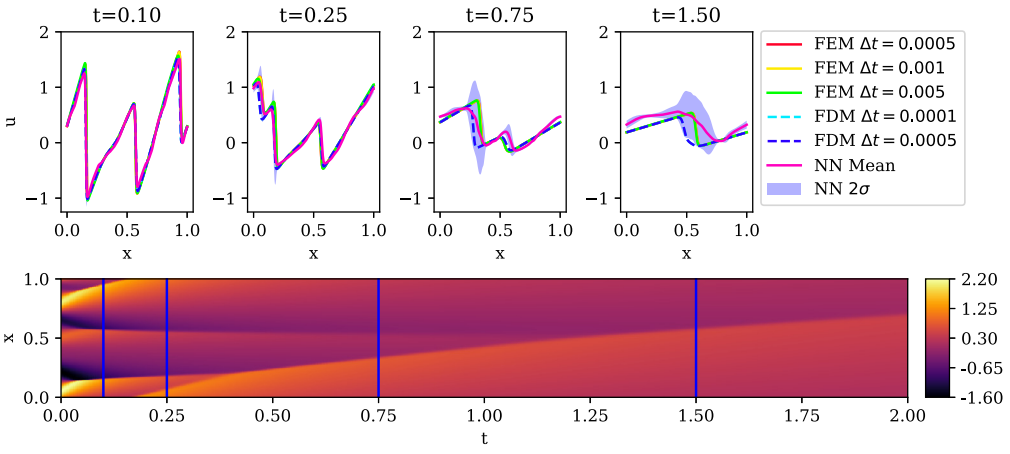



**Fig. 14.** Instantaneous profiles of both the finite element method (FEM) and finite difference method (FDM) numerical solvers along with BAR-DenseED neural network (NN) predictive expectation and standard deviation at four various times of a test case. The bottom contour is the ideal target calculated using FEM with a time-step size $\Delta t = 0.0005$. The blue lines mark each profile location.

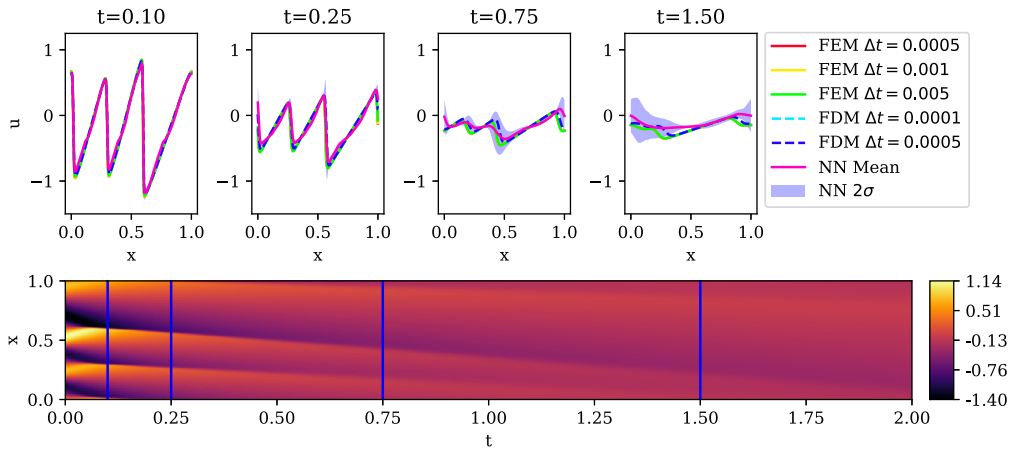



**Fig. 15.** Instantaneous profiles of both finite element method (FEM) and finite difference method (FDM) numerical solvers along with BAR-DenseED neural network (NN) predictive expectation and standard deviation at four various times of a test case. The bottom contour is the ideal target calculated using FEM with a time-step size $\Delta t = 0.0005$. The blue lines mark each profile location.

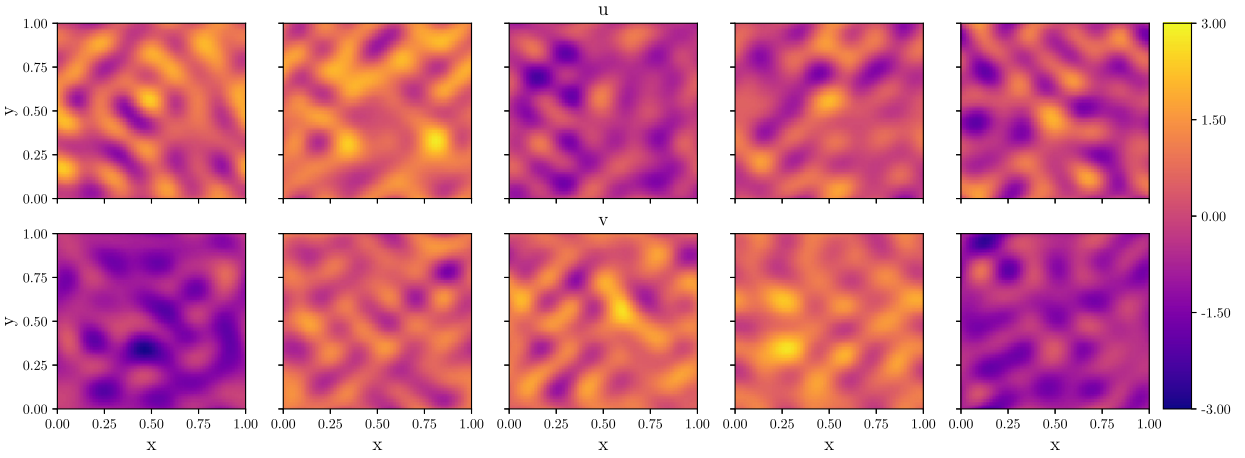



**Fig. 16.** Randomly generated initial conditions for the 2D coupled Burgers' system. (Top to bottom) The $x$-velocity and $y$-velocity components. (Left to right) Different samples of the random initial condition.

$$\boldsymbol{u}(0, y, t) = \boldsymbol{u}(L, y, t), \quad \boldsymbol{u}(x, 0, t) = \boldsymbol{u}(x, L, t), \tag{41}$$

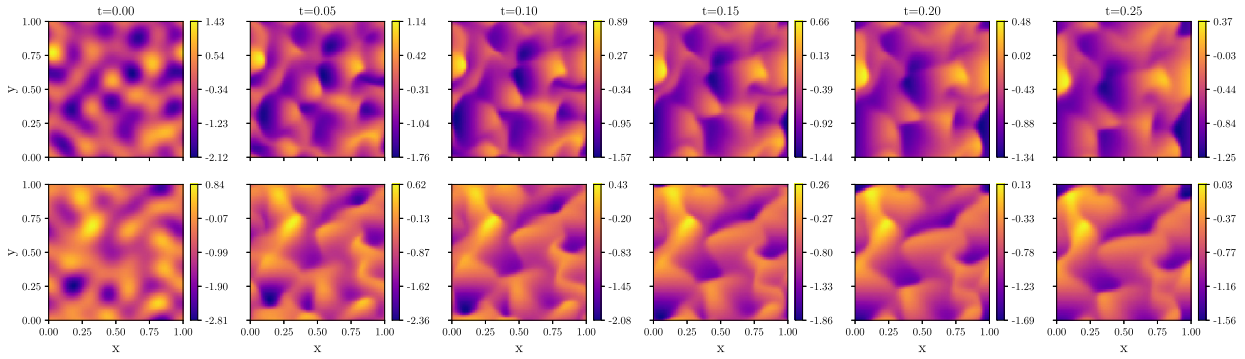
$$\{x, y\} \in [0, L], \quad t \in [0, T], \tag{42}$$

which when expanded into its components takes the following form:

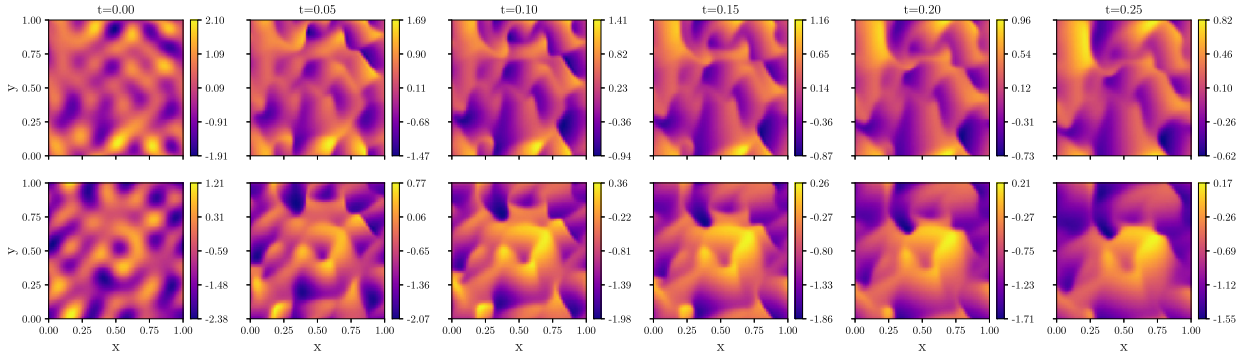
$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0, \\ &\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0, \end{aligned} \tag{43}$$

where $\nu$ is the viscosity of the system which will be held at $\nu = 0.005$ and the domain size set to $\{x, y\} \in [0, 1]$. $u$ and $v$ are the $x$ and $y$ velocity components, respectively. The 2D coupled Burgers' equation is an excellent benchmark PDE due to both its non-linear term as well as diffusion operator, making it much more complex than the standard advection or diffusion equations. The 2D coupled Burgers' belongs to a much broader class of PDEs that are related to various physical problems including shock wave propagation in viscous fluids, turbulence, super-sonic flows, acoustics, sedimentation and airfoil theory. Given its similar form, the coupled Burgers' equation is often regarded as an essential stepping-stone to the full Navier-Stokes equations [71,72].

As in our previous examples, we are interested in surrogate modeling for various initial conditions. We will initialize the field using a truncated Fourier series with random coefficients:




(a) Simulation 1




(b) Simulation 2

**Fig. 17.** 2D coupled Burgers' equation simulations for two various initial conditions solved using the Fenics finite element package [70]. (Top to bottom) $x$-velocity and $y$-velocity components.

$$\begin{aligned} \boldsymbol{w}(x,y) &= \sum_{i=-L}^{L}\sum_{j=-L}^{L} \boldsymbol{a}_{ij}\sin(2\pi\,(ix+jy)) + \boldsymbol{b}_{ij}\cos(2\pi\,(ix+jy)), \\ \boldsymbol{u}(x,y,0) &= \frac{2\boldsymbol{w}(x,y)}{\max_{\{x,y\}}|\boldsymbol{w}(x,y)|} + \boldsymbol{c}, \end{aligned} \tag{44}$$

where $\boldsymbol{a}_{ij}, \boldsymbol{b}_{ij} \sim \mathcal{N}(0, \boldsymbol{I}_2)$, $L = 4$ and $\boldsymbol{c} \sim \mathcal{U}(-1,1) \in \mathbb{R}^2$. Several of these randomly generated initial conditions are illustrated in Fig. 16. While these initial conditions may appear similar, the evolution of the systems results in the forming of many complex and unique structures. To illustrate this, we plot several time-steps two FEM simulations of different initial conditions in Fig. 17. As the system develops, we can see very distinct structures forming as waves form, mix, interact and dissipate with each other. This is why the 2D coupled Burgers' system is a difficult system to model and serves as an excellent benchmark for our proposed surrogate.

The AR-DenseED model used for the 2D coupled Burgers' equations is the largest of the examples shown in this work with about 72000 learnable parameters. However, in the scope of the deep learning field over the past several years this network is still light weight. Both the $x$ and $y$ velocity components are predicted by the same model in the form of two output channels. Similarly both velocity components from three previous time-steps are used as inputs, $\boldsymbol{\chi}^{n+1} = \{\boldsymbol{u}^n, \boldsymbol{v}^n, \boldsymbol{u}^{n-1}, \boldsymbol{v}^{n-1}, \boldsymbol{u}^{n-2}, \boldsymbol{v}^{n-2}\}$, resulting in six input channels. The time-step value of the model is $\Delta t = 0.005$ with a spatial discretization of $64 \times 64$ points. Again the negative log of the joint posterior in Eq. (19) is the objective function with the implicit Crank-Nicolson time-integrator and other spatial gradients of the physics-constrained loss being discretized as:

$$\begin{aligned} T_{\Delta t}(\boldsymbol{\mathcal{U}}^n, F_{\Delta x}) &= \boldsymbol{u}^n + \Delta t\left[-0.5\left(F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^{n+1}) + F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^n)\right)\right], \\ F_{\Delta x}(\boldsymbol{x}, \boldsymbol{u}^n) &= \boldsymbol{u}^n \cdot \nabla \boldsymbol{u}^n - \nu \Delta \boldsymbol{u}^n, \end{aligned} \tag{45}$$

$$\boldsymbol{u}_x = \frac{1}{8\Delta x}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \boldsymbol{u}, \quad \boldsymbol{u}_y = \frac{1}{8\Delta x}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \boldsymbol{u}, \tag{46}$$

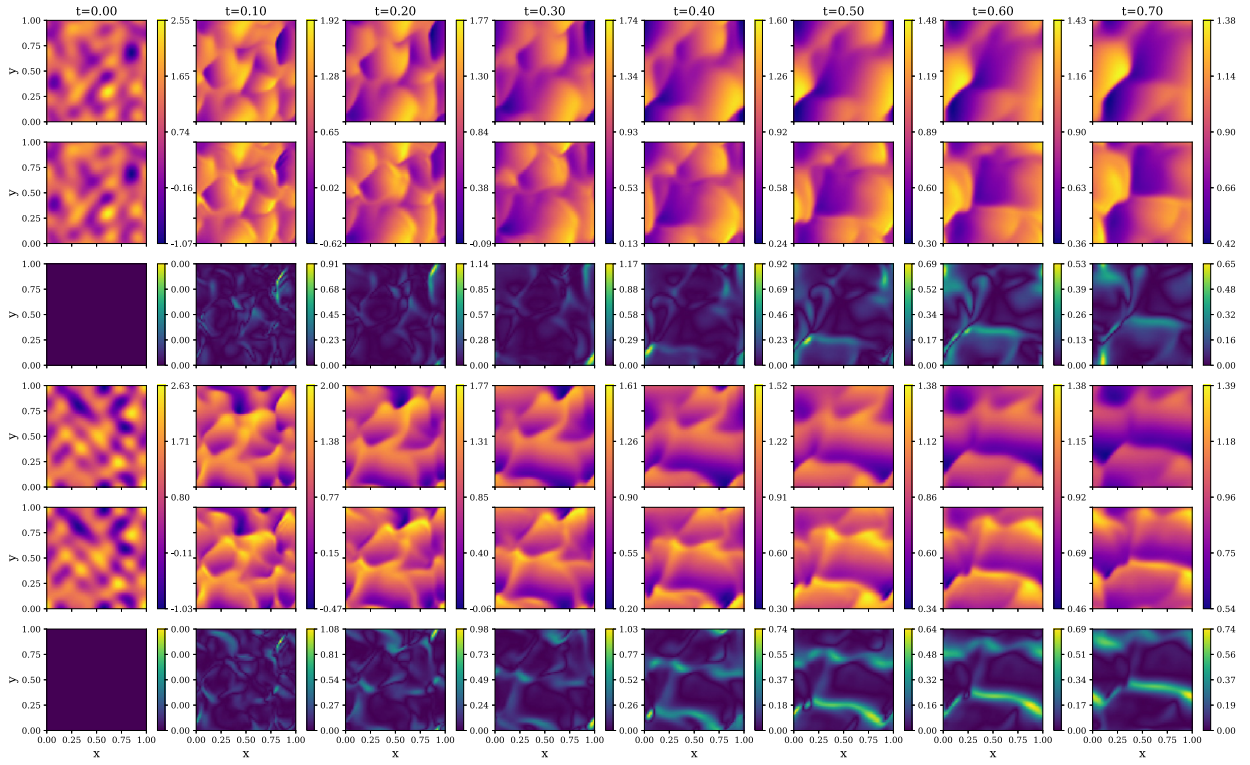



**Fig. 18.** AR-DenseED predictions of a 2D coupled Burgers' test case. (Top to bottom) $x$-velocity FEM target solution, $x$-velocity AR-DenseED prediction, $x$-velocity $L_1$ error, $y$-velocity FEM target solution, $y$-velocity AR-DenseED prediction and $y$-velocity $L_1$ error.

$$\Delta \boldsymbol{u} = \frac{1}{2\Delta x^2} \begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \boldsymbol{u}, \tag{47}$$

where the spatial gradients are approximated using Sobel filter 2D convolutions which are analogous to smoothed second-order accurate finite difference approximations [73]. Using the Sobel filter was found to increase the stability of training and significantly reduce spurious oscillations in the model's predictions. The model was trained on 5120 training scenarios sampled from Eq. (44) with a mini-batch size of 128. AR-DenseED was optimized for 100 epochs and allowed to unroll a maximum of 100 time-steps from its initial state. Training on a single 1080Ti GPU required 9 wall-clock hours. To assess the performance of the model, another set of 200 samples from Eq. (44) was used as testing scenarios. Additional details on the model and training can be found in Appendix C.

### *7.1. AR-DenseED deterministic predictions*

For the 2D couple Burgers' system, the trained AR-DenseED is used to predict 200 time-steps from the initial state at $t = 0$. Similar to the 1D Burgers' test case, this means that half of this predicted region ($t > 0.5$) is extrapolation beyond the time range that the model was trained on. The target response is a high-fidelity FEM simulation with a discretization of $128 \times 128$ interpolated to a $64 \times 64$ grid. The AR-DenseED's predictions are shown for two test cases in Figs. 18 and 19 in which several instantaneous time-steps are plotted. Overall, the model does a remarkable job accurately predicting the complex structures and discontinuities of this system even into the extrapolation region. As expected the bulk of the error is concentrated on shock interfaces and wave fronts, however, the model's extremely good predictive capability is clear.

The mean squared error defined in Eq. (33) as well as the energy squared error defined in Eq. (39), both generalized to two dimensions, are evaluated for each time-step for the 200 test scenarios. The mean and the median of these error values for the entire test set are plotted in Fig. 20. AR-DenseED is shown to have very stable prediction error within the training time region. When AR-DenseED performs extrapolatory predictions at $t > 0.5$, we can see that the mean error begins to grow faster than the median suggesting that several outlier test cases have very poor predictions. We also compared the prediction computational cost of AR-DenseED for this 2D system again a set of FEM solutions of various spatial discretizations in Table 3. Here, we can see that the AR-DenseED is able to far outperform the FEM simulations with a wall-clock time that is significantly less than all simulations by several orders of magnitude. When one takes full advantages of a GPU hardware accelerator, this speed up becomes even larger. Additionally this neglects the ability of the neural network to easily compute

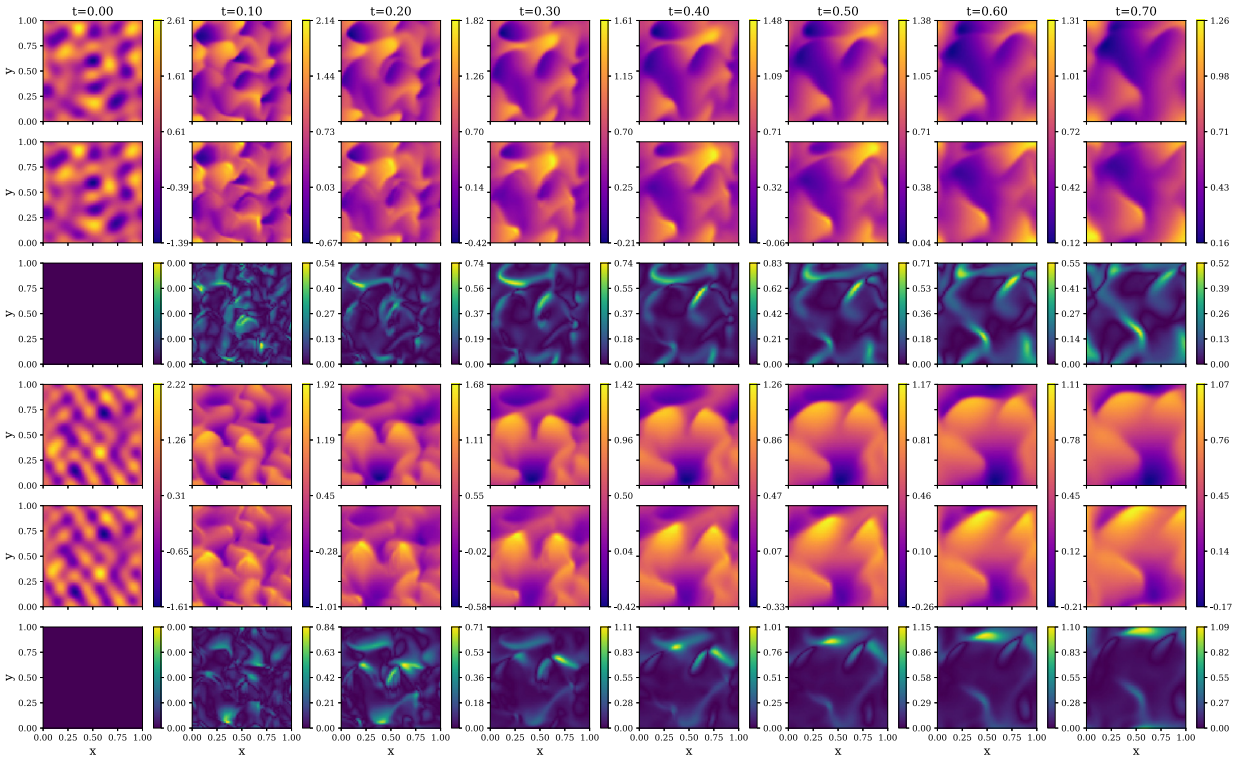



**Fig. 19.** AR-DenseED predictions of a 2D coupled Burgers' test case. (Top to bottom) $x$-velocity FEM target solution, $x$-velocity AR-DenseED prediction, $x$-velocity $L_1$ error, $y$-velocity FEM target solution, $y$-velocity AR-DenseED prediction and $y$-velocity $L_1$ error.

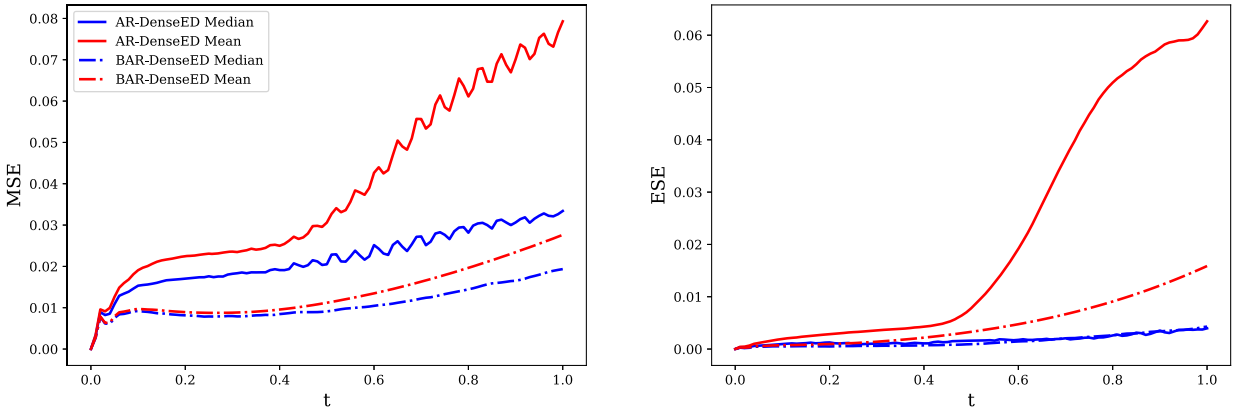



**Fig. 20.** (Left to right) The mean square error (MSE) and energy squared error (ESE) as a function of time for a test set of 200 cases for the 2D coupled Burgers' system. The error of BAR-DenseED is calculated using the expected value of the predictive distribution approximating using 30 samples of the posterior.

multiple simulations in a single batch, making its predictive efficiency even greater. This illustrates the true potential power of these deep convolutional surrogate models.

### *7.2. BAR-DenseED probabilistic predictions*

To approximate the posterior with SWAG, 90 samples of the network's parameters were collected with a learning rate of 3e−8 for the neural network parameters and 3e−5 for the output noise $\beta$. Predictions of several posterior samples at times $t = 0.1$ and $t = 0.5$ are shown in Figs. 21 and 22, respectively. As expected, the variance of the samples increases significantly as time progresses. This is also reflected in the BAR-DenseED prediction contours for two test cases in Figs. 23 and 24 in which the predictive expectation and variance computed using 30 model samples are shown for several time-steps. Again we see that the majority of the uncertainty is concentrated on the leading face of the shocks/waves which is precisely where

**Table 3**
Wall-clock time of finite element simulation and AR-DenseED to simulate 200 time-steps of the 2D coupled Burgers' system. Wall-clock time estimates were obtained by averaging 10 independent simulation run times.

| | Hardware | Backend | $\Delta t$ | $\Delta x$ | Wall-clock (s) |
|---|---|---|---|---|---|
| Finite Element | Intel Xeon E5-2680 | Fenics | 0.005 | 1/128 | 2955.38 |
| Finite Element | Intel Xeon E5-2680 | Fenics | 0.005 | 1/64 | 418.83 |
| Finite Element | Intel Xeon E5-2680 | Fenics | 0.005 | 1/32 | 133.65 |
| AR-DenseED | Intel Xeon E5-2680 | PyTorch | 0.005 | 1/64 | 4.691 |
| AR-DenseED | GeForce GTX 1080 Ti | PyTorch | 0.005 | 1/64 | 0.841 |

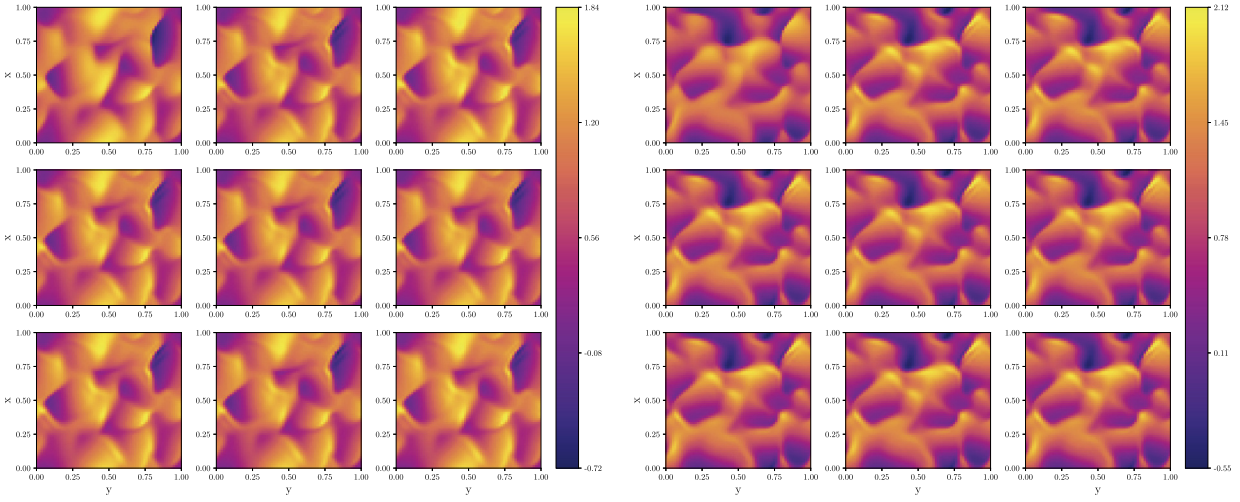


**Fig. 21.** (Left to right) Samples of the $x$-velocity and $y$-velocity component from the posterior of BAR-DenseED approximated using SWAG at $t = 0.1$ for the 2D coupled Burgers' system. The top left in each grid is the simulated result using FEM.

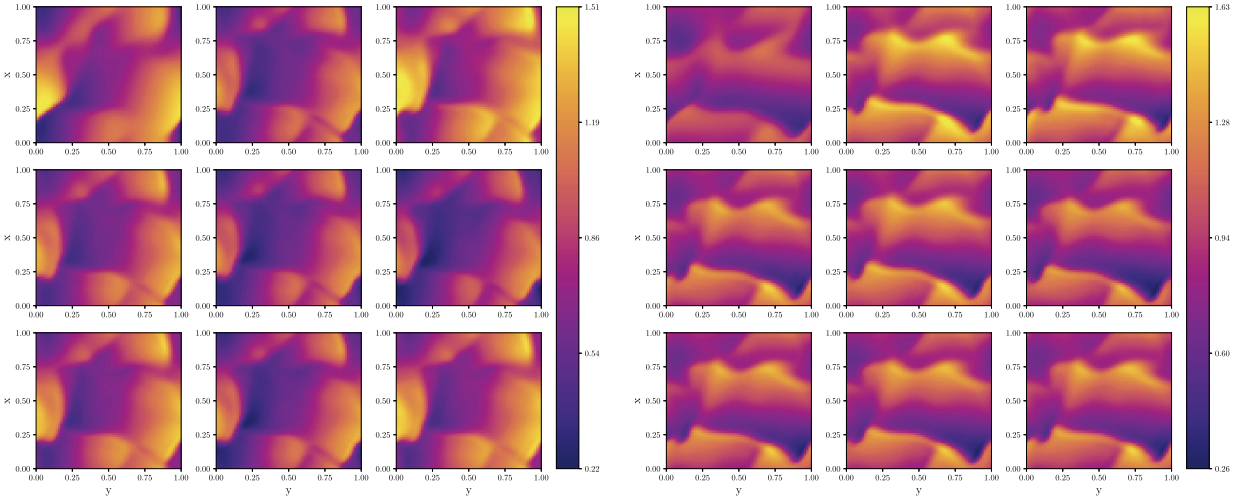


**Fig. 22.** (Left to right) Samples of the $x$-velocity and $y$-velocity component from the posterior of BAR-DenseED approximated using SWAG at $t = 0.5$ for the 2D coupled Burgers' system. The top left in each grid is the simulated result using FEM.

we would expect it for a well calibrated model. To more clearly illustrate the uncertainty estimates of the probabilistic model, velocity profiles of both velocity components are plotted in Fig. 25 for a randomly selected test case. Overall, we can see that the predictive standard deviation is able to capture the true solution for almost all times. Finally, the mean squared error and energy squared error are also plotted using the predictive expectation of BAR-DenseED in Fig. 20. The Bayesian model is able to have smaller and more stable prediction error even when extrapolating beyond the training time range.

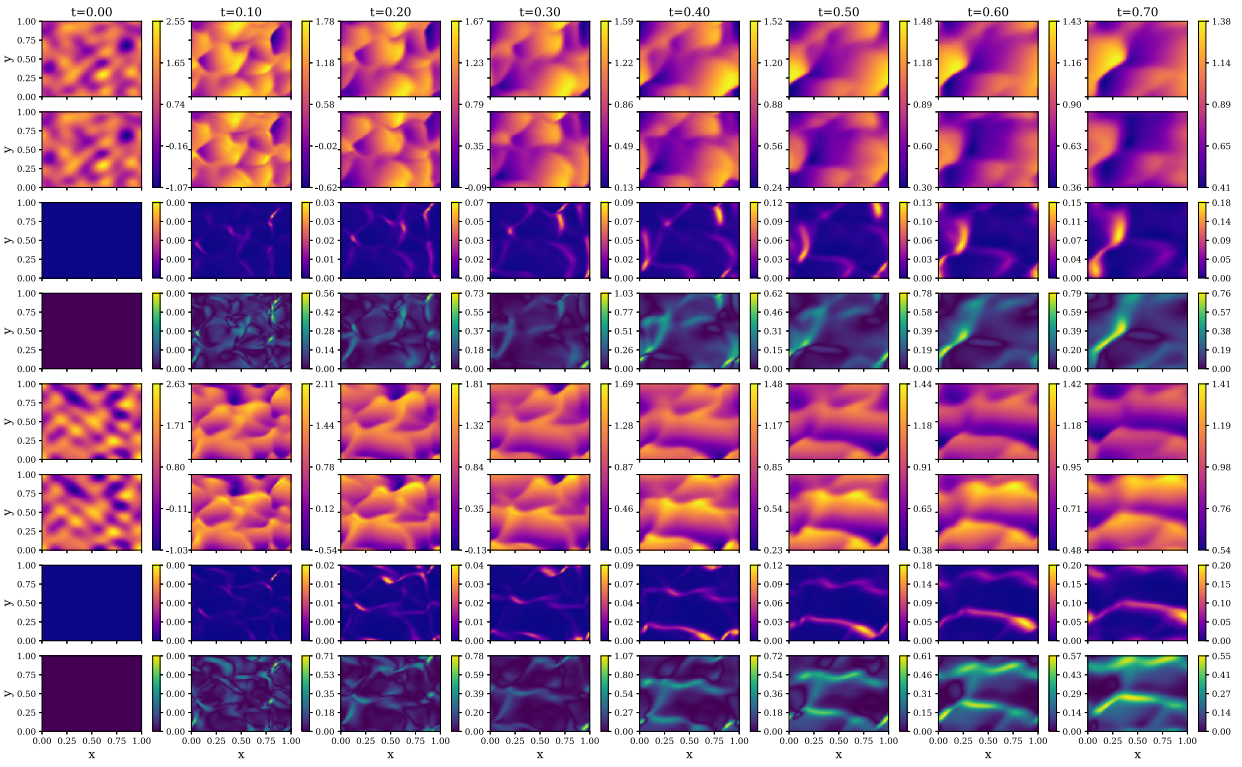



**Fig. 23.** BAR-DenseED predictions for a 2D coupled Burgers' test case. (Top to bottom) $x$-velocity FEM target solution, BAR-DenseED expected response, BAR-DenseED variance, $L_1$ error between the target and expected values and similarly followed by the $y$-velocity component.

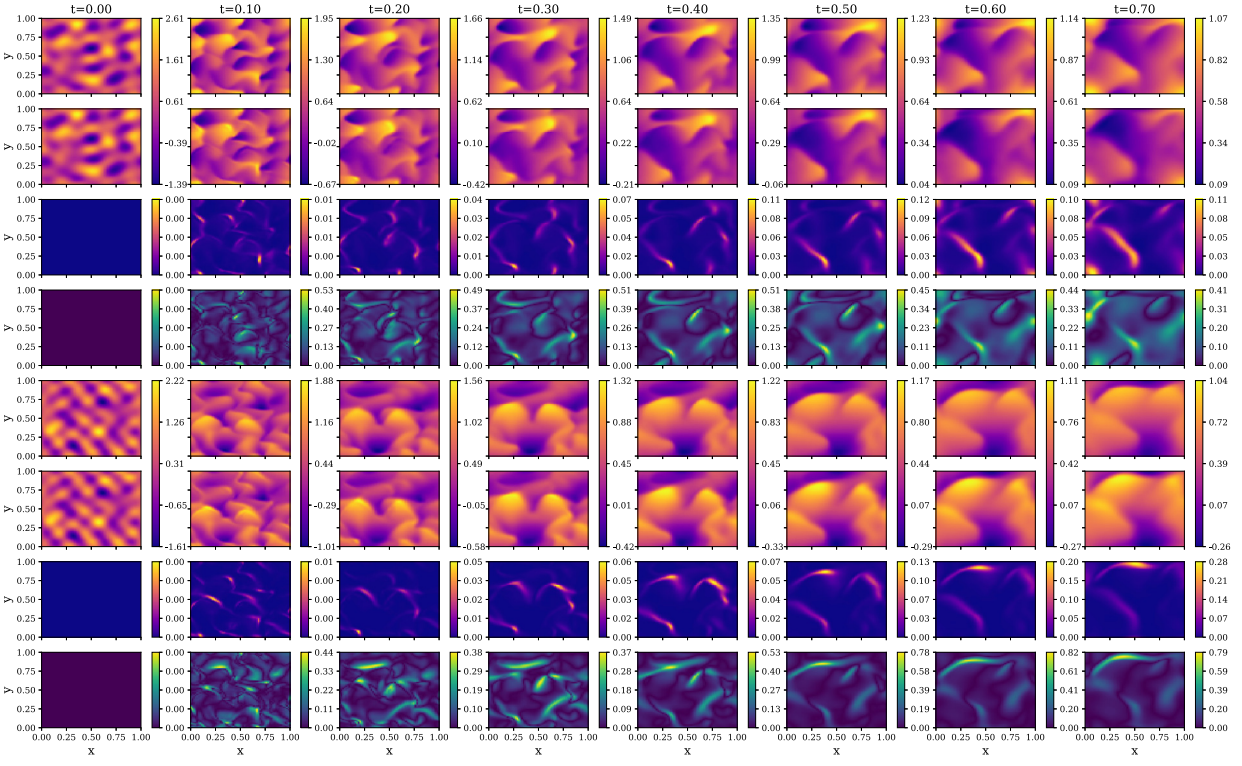



**Fig. 24.** BAR-DenseED predictions for a 2D coupled Burgers' test case. (Top to bottom) $x$-velocity FEM target solution, BAR-DenseED expected response, BAR-DenseED variance, $L_1$ error between the target and expected values and similarly followed by the $y$-velocity component.

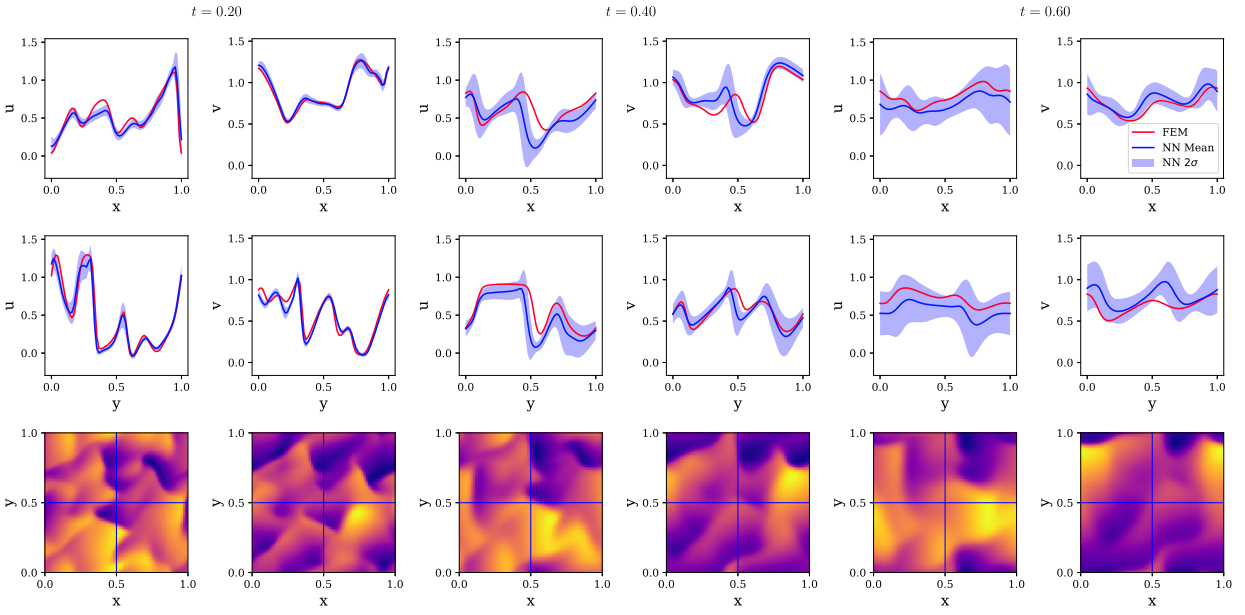



**Fig. 25.** Instantaneous profiles of the finite element method (FEM) solver and BAR-DenseED (NN) predictive expectation and standard deviation at three various times of a test case. (Top to bottom) Horizontal profile at $y = 0.5$, vertical profile at $x = 0.5$ and target FEM contour with blue lines to show the profile locations. (Left to right) $x$-velocity and $y$-velocity profiles at $t = 0.2$, 0.4 and 0.6.

## 8. Conclusion

In this work, we have presented a deep auto-regressive convolutional neural network model that can be used to learn and surrogate model the dynamics of transient PDEs. To train this model, physics-constrained deep learning is used where the governing equations of the system of interest are used to formulate a loss function. This allows the model to be trained with zero (output) training data. Additionally, we proposed a Bayesian probabilistic framework built on top of this deep learning model to allow for uncertainty quantification (including both epistemic and aleatoric uncertainty). This model was implemented for three PDE systems: the first is the chaotic Kuramoto-Sivashinsky equation for which the model was used to accurately reproduce physical turbulent statistics. The second is the 1D Burgers' equation at a low viscosity where the model was able to successfully predict multi-shock wave formation and intersections. At last is the 2D coupled Burgers' equations for which the model was able to accurately predict the complex wave dynamics of this system. Overall, the proposed model showed exceptional predictive accuracy and was able to successfully extrapolate to predict outside the time-range used when training.

Although fully connected networks are frequently used to solve PDE systems due to their analytical and mesh-less benefits, the performance of convolutional neural networks for solving and surrogate modeling of PDEs is exceptional. In this work, we have further shown that convolutional neural networks can be used effectively in physics-constrained learning and build surrogate models that are order of magnitudes faster than state-of-the-art numerical solvers. A particular drawback of convolutional neural networks is the requirement that both spatial and temporal derivatives be discretized, which opens the model up to the challenges that are faced in traditional numerical algorithms such as truncation error, oscillations, convergence criterion and more. However, one can also use the deep repository of techniques and tricks developed by the numerical analysis community to address these potential issues. This would be an interesting avenue to investigate as one could incorporate methods such as flux limiters or non-oscillatory schemes to yield predictions that have similar numerical benefits. One could also consider higher-order derivatives and their impact on accuracy versus training stability.

The most obvious path to further develop this model is to implement it for more complex and larger systems. This could include systems such as the Navier-Stokes equations, coupled transport through porous media, combustion and more. However, there are still significant challenges that will need to be addressed. The most important is training cost; with any time series problem training a deep learning model becomes exponentially more difficult and more costly. Although our model is able to be trained in a very reasonable amount of time given the complexity of the physical systems modeled as well as the hardware used, improving the training of the model will still be an important area of study. This may involve the use of network architectures considered in recent neural language processing literature such as self-attention mechanisms. Another potential extension is the incorporation of data and physics-constrained learning to create this hybrid learning framework. Specifically for time series, one may not have the system state at every time interval that is desired.

Physics-constrained learning could be an answer to help bridge this challenge of predicting at fine resolutions with sparse data.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

The authors acknowledge support from the Defense Advanced Research Projects Agency (DARPA) under the Physics of Artificial Intelligence (PAI) program (contract HR00111890034). The work of NG is also supported by the National Science Foundation (NSF) Graduate Research Fellowship Program grant No. DGE-1313583. Additional computing resources were provided by the University of Notre Dame's Center for Research Computing (CRC), NSF supported "Extreme Science and Engineering Discovery Environment" (XSEDE) on the Bridges and Bridges-GPU cluster through research allocation No. TG-CTS180038 and by the AFOSR Office of Scientific Research through the DURIP program.

## Appendix A. Kuramoto-Sivashinsky

The following appendix discusses details related to the model used to predict in the Kuramoto-Sivashinsky equation in Section 5. For this system, a small dense encoder-decoder model was used as depicted in Fig. A.26. The three components of this model are the encoding convolution, the dense block and decoding block. The resulting model encodes a given 1D input $\left\{\boldsymbol{u}^n, \boldsymbol{u}^{n-1}\right\} \in \mathbb{R}^d$ to a set of latent variables that are of dimensionality $\boldsymbol{z}_i \in \mathbb{R}^{d/2}$. These latent variables are then decoded to the prediction $\boldsymbol{u}^{n+1} \in \mathbb{R}^d$. Examples of a dense block and decoding block are shown in Fig. A.27 and Fig. A.28 originally proposed in Zhu and Zabaras [2].

While this model is relatively small, we found that smaller models were more stable in training and had the additional benefit of being faster. To help with learning periodic boundaries, circular padding was used for all convolutions within the model. The model was optimized with ADAM [48] with an initial learning rate of 1e−3. It was found that decaying the learning rate exponentially yielded the most stable, consistent and accurate results. Additional model training parameters can be reference in Table A.4.

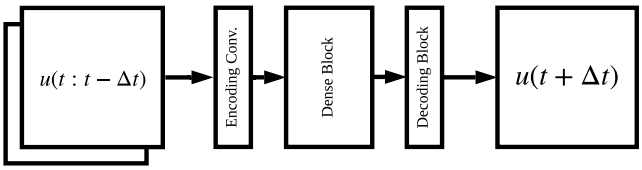



**Fig. A.26.** The AR-DenseED model with 4821 learnable parameters used for the Kuramoto-Sivashinsky equation. This model consists of an encoding convolution, single dense block with a growth rate of 4 and a length of 4 followed by a decoding block. The two previous time-steps are used as inputs.

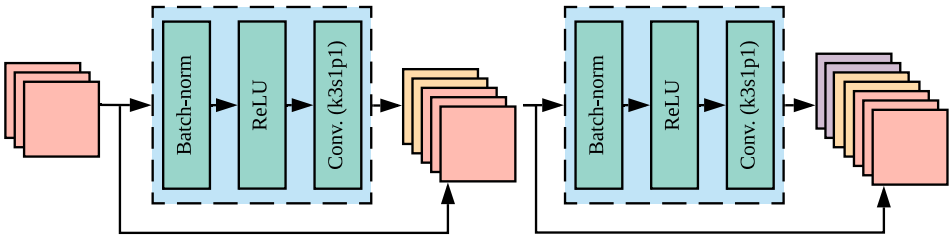



**Fig. A.27.** Schematic of a dense block with a growth rate of 2 and length of 2 consisting of batch-normalization [74], Rectified Linear Unit (ReLU) activation functions [75] and convolutions. The key feature is the residual connection that stacks the output of each convolution increasing the number of feature channels substantially. Convolutions are denoted by the kernel size $k$, stride $s$ and padding $p$.

### *A.1. Training initial states*

To train the model, we need to provide it a set of initial states or training scenarios to start at. Simulator data could be used for this, however to stay consistent with the zero training data philosophy, we chose to use a truncated Fourier series with random coefficients. We propose the use of:

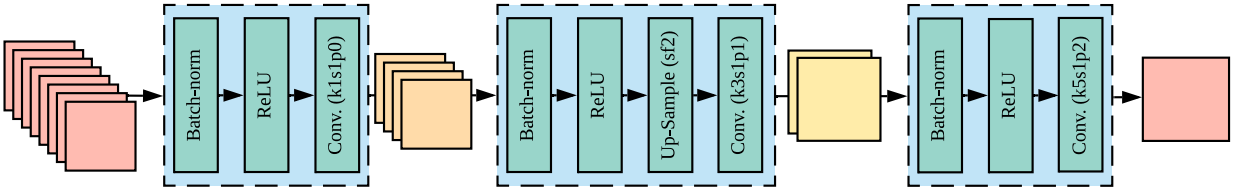



**Fig. A.28.** Schematic of a decoding block which consists of several sequential layers of batch-normalization [74], Rectified Linear Unit (ReLU) activation function [75] and convolutions. Nearest neighbor up-sampling is used to increase the size of the data to the desired dimensions. Convolutions are denoted by the kernel size $k$, stride $s$ and padding $p$.

**Table A.4**
AR-DenseED and BAR-DenseED training parameters used for the Kuramoto-Sivashinsky system.

| Training parameters | | SWAG [55] parameters | |
|---|---|---|---|
| Optimizer | ADAM [48] | Optimizer | ADAM [48] |
| Weight Decay | 0 | Weight Decay | 0 |
| Learning Rate | 1e−3 | Learning Rate | 1e−10 |
| $\beta$ Learning Rate | 1e−3 | $\beta$ Learning Rate | 1e−6 |
| Exponential Decay Rate | 0.995 | Collection Rate | 1 epoch |
| Training Epochs | 100 | Models Collected | 100 |
| Training Scenarios | 2560 | Deviation Matrix $H$ | 10 |
| Mini-batch Size | 256 | | |

$$u(x,0) = 2a\frac{w(x) - \min_x w(x)}{\max_x w(x) - \min_x w(x)} - a, \quad w(x) = \sum_{n=1}^{3} \frac{\lambda_n}{n} \sin\left(\frac{n\pi x}{l} + c\right),$$

$$\lambda_n = [1, \mathcal{N}(0,2), 1], \quad c = 2\pi\, \mathcal{U}[0,1], \quad a = \mathcal{N}(0, 0.5) + a_0,$$

$$l = L/(2k_0), \quad k_0 = \lfloor L/(2\pi\sqrt{2}) + 0.5 \rfloor,$$

where $a_0$ is the *a priori* mean amplitude estimate (set to 2.5), $L$ is the domain size and $k_0$ is the number of unstable modes which can be estimated given the domain length such that $k_0 = L/(2\sqrt{2}\pi)$ [76]. This function is designed to provide a physically realizable initial condition for AR-DenseED to explore the physics of the K-S system.

## Appendix B. 1D viscous Burgers' system

The following appendix discusses details related to the model used to predict the 1D Burgers' equation in Section 6. Of the three systems present in this paper, the 1D viscous Burgers' system was found to be the most difficult to train. This is most likely due to the sharp gradients in this system which carries over to the loss function resulting in exploding gradients during optimization. The model for this system, depicted in Fig. B.29, is approximately three times as large as the model used for the K-S system. Similar to the K-S model, the 1D-Burgers model encodes a given 1D input $\{\boldsymbol{u}^n, \boldsymbol{u}^{n-1}..., \boldsymbol{u}^{n-4}\} \in \mathbb{R}^d$ to a set of latent variables that are of dimensionality $\boldsymbol{z}_i \in \mathbb{R}^{d/2}$. These latent variables are then decoded to the prediction $\boldsymbol{u}^{n+1} \in \mathbb{R}^d$. More time-steps were used in the model input compared to the K-S system to increase learning stability. Interestingly, if the model was too large, SWAG would fail to approximate a good posterior regardless of the learning rate resulting in sampled models being unstable during prediction. Additional training parameters are listed in Table B.5.

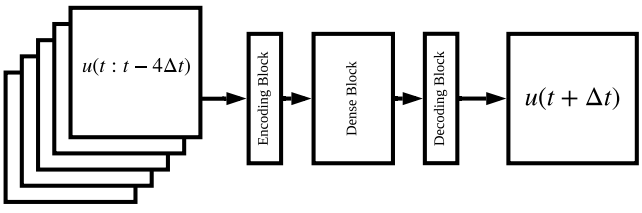



**Fig. B.29.** The AR-DenseED model with 13442 learnable parameters used for the 1D Burgers' equation. This model consists of an encoding convolutional block, single dense block with a growth rate of 4 and a length of 1 followed by a decoding block. The five previous time-steps are used as inputs.

**Table B.5**
AR-DenseED and BAR-DenseED training parameters used for the 1D Burgers' system.

| Training parameters | | SWAG [55] parameters | |
|---|---|---|---|
| Optimizer | ADAM [48] | Optimizer | ADAM [48] |
| Weight Decay | 0 | Weight Decay | 0 |
| Learning Rate | 1e−3 | Learning Rate | 4e−8 |
| $\beta$ Learning Rate | 1e−3 | $\beta$ Learning Rate | 5e−6 |
| Exponential Decay Rate | 0.99 | Collection Rate | 1 epoch |
| Training Epochs | 100 | Models Collected | 90 |
| Training Scenarios | 2560 | Deviation Matrix $H$ | 30 |
| Mini-batch Size | 256 | | |

## Appendix C. 2D coupled Burgers' system

The following appendix discusses details related to the model used to predict the 2D coupled Burgers' equation in Section 7. As shown in Fig. C.30, the model used for the 2D Burgers' system is very similar to the other two test cases. The key difference is that the model now predicts two values: the $x$ and $y$ velocity components. Similarly the model uses both velocity components as inputs. Thus when three previous time-steps are used in $\boldsymbol{\chi}^{n+1}$ the model has six input channels. This model encodes a given 2D input $\left\{\boldsymbol{u}^n, \boldsymbol{v}^n, \boldsymbol{u}^{n-1}, \boldsymbol{v}^{n-1}, \boldsymbol{u}^{n-2}, \boldsymbol{v}^{n-2}\right\} \in \mathbb{R}^{d\times d}$ to a set of latent variables that are of dimensionality $\boldsymbol{z}_i \in \mathbb{R}^{d/2\times d/2}$. These latent variables are then decoded to the prediction $\left\{\boldsymbol{u}^{n+1}, \boldsymbol{v}^{n+1}\right\} \in \mathbb{R}^{d\times d}$. Although this system has some of the most complex dynamics, to our surprise we found that it was the easiest to train with significantly better stability and consistency than the other 1D systems. We largely attribute this to the use of Sobel filters to approximate the gradients which help dampen higher frequencies preventing oscillations. Additional training parameters are listed in Table C.6.

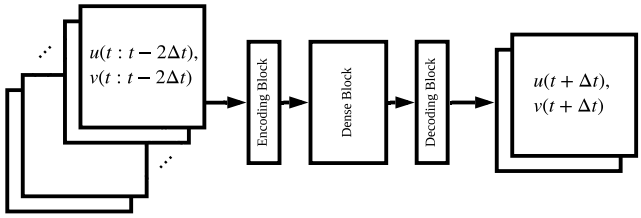



**Fig. C.30.** The AR-DenseED model with 71953 learnable parameters used for the 2D coupled Burgers' equation. This model consists of an encoding convolutional block, single dense block with a growth rate of 4 and a length of 4 followed by a decoding block. The five previous time-steps are used as inputs.

**Table C.6**
AR-DenseED and BAR-DenseED training parameters used for the 2D coupled Burgers' system.

| Training parameters | | SWAG [55] parameters | |
|---|---|---|---|
| Optimizer | ADAM [48] | Optimizer | ADAM [48] |
| Weight Decay | 0 | Weight Decay | 0 |
| Learning Rate | 1e−3 | Learning Rate | 3e−8 |
| $\beta$ Learning Rate | 1e−3 | $\beta$ Learning Rate | 3e−5 |
| Exponential Decay Rate | 0.995 | Collection Rate | 1 epoch |
| Training Epochs | 100 | Models Collected | 90 |
| Training Scenarios | 5120 | Deviation Matrix $H$ | 30 |
| Mini-batch Size | 128 | | |

## References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436, https://doi.org/10.1038/nature14539.
[2] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447, https://doi.org/10.1016/j.jcp.2018.04.018, http://www.sciencedirect.com/science/article/pii/S0021999118302341.
[3] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81, https://doi.org/10.1016/j.jcp.2019.05.024, http://www.sciencedirect.com/science/article/pii/S0021999119303559.
[4] R.K. Tripathy, I. Bilionis, Deep UQ: learning deep neural network surrogate models for high dimensional uncertainty quantification, J. Comput. Phys. 375 (2018) 565–588, https://doi.org/10.1016/j.jcp.2018.08.036, http://www.sciencedirect.com/science/article/pii/S0021999118305655.
[5] C. Yang, X. Yang, X. Xiao, Data-driven projection method in fluid simulation, Comput. Animat. Virtual Worlds 27 (3–4) (2016) 415–424, https://doi.org/10.1002/cav.1695, https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1695.

[6] N. Geneva, N. Zabaras, Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks, J. Comput. Phys. 383 (2019) 125–147, https://doi.org/10.1016/j.jcp.2019.01.021, http://www.sciencedirect.com/science/article/pii/S0021999119300464.

[7] M. Schöberl, N. Zabaras, P.-S. Koutsourelakis, Predictive collective variable discovery with deep Bayesian models, J. Chem. Phys. 150 (2) (2019) 024109, https://doi.org/10.1063/1.5058063.

[8] D.J. MacKay, A practical Bayesian framework for backpropagation networks, Neural Comput. 4 (3) (1992) 448–472, https://doi.org/10.1162/neco.1992.4.3.448.

[9] R.M. Neal, Bayesian Learning for Neural Networks, vol. 118, Springer Science & Business Media, 2012, https://www.springer.com/us/book/9780387947242.

[10] D.C. Psichogios, L.H. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE J. 38 (10) (1992) 1499–1511, https://doi.org/10.1002/aic.690381003, https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690381003.

[11] A.J. Meade Jr, A.A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Model. 19 (12) (1994) 1–25, https://doi.org/10.1016/0895-7177(94)90095-7, http://www.sciencedirect.com/science/article/pii/0895717794900957.

[12] A. Meade, A. Fernandez, Solution of nonlinear ordinary differential equations by feedforward neural networks, Math. Comput. Model. 20 (9) (1994) 19–44, https://doi.org/10.1016/0895-7177(94)00160-X, http://www.sciencedirect.com/science/article/pii/089571779400160X.

[13] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000, https://doi.org/10.1109/72.712178.

[14] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049, https://doi.org/10.1109/72.870037.

[15] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, https://doi.org/10.1016/j.jcp.2018.10.045, http://www.sciencedirect.com/science/article/pii/S0021999118307125.

[16] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41, https://doi.org/10.1016/j.neucom.2018.06.056, http://www.sciencedirect.com/science/article/pii/S092523121830794X.

[17] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12, https://doi.org/10.1007/s40304-018-0127-z.

[18] M.A. Nabian, H. Meidani, A deep neural network surrogate for high-dimensional random partial differential equations, CoRR abs/1806.02957, arXiv:1806.02957, http://arxiv.org/abs/1806.02957.

[19] S. Karumuri, R. Tripathy, I. Bilionis, J. Panchal, Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks, preprint arXiv:1902.05200.

[20] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029, http://www.sciencedirect.com/science/article/pii/S0021999118305527.

[21] P. Grohs, F. Hornung, A. Jentzen, P. Von Wurstemberger, A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations, preprint arXiv:1809.02362.

[22] Y. Khoo, J. Lu, L. Ying, Solving for high-dimensional committor functions using artificial neural networks, Res. Math. Sci. 6 (1) (2018) 1, https://doi.org/10.1007/s40687-018-0160-2.

[23] L. Dinh, J. Sohl-Dickstein, S. Bengio, Density estimation using real NVP, CoRR, arXiv:1605.08803.

[24] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, abs/1512.03385, arXiv:1512.03385, 2015.

[25] V.M. Filippov, V.M. Savchin, S.G. Shorokhov, Variational principles for nonpotential operators, J. Math. Sci. 68 (1994) 275–398, https://doi.org/10.1007/BF0125231, https://doi.org/10.1007/BF01252319.

[26] A. Ralston, P. Rabinowitz, A First Course in Numerical Analysis, Courier Corporation, 2001.

[27] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: Proceedings of the 30th International Conference on International Conference on Machine Learning, vol. 28, ICML'13, JMLR.org, 2013, pp. 1310–1318, http://dl.acm.org/citation.cfm?id=3042817.3043083.

[28] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.

[29] Yong Du, W. Wang, L. Wang, Hierarchical recurrent neural network for skeleton based action recognition, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2015, pp. 1110–1118.

[30] Y. Kim, C. Denton, L. Hoang, A.M. Rush, Structured attention networks, CoRR arXiv:1702.00887, http://arxiv.org/abs/1702.00887.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017, pp. 5998–6008, http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf, 2017.

[32] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[33] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond Euclidean data, IEEE Signal Process. Mag. 34 (4) (2017) 18–42, https://doi.org/10.1109/MSP.2017.2693418.

[34] R.J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems, vol. 98, SIAM, 2007.

[35] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869, https://doi.org/10.1137/0907058.

[36] S.V. Patankar, D.B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, in: Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion, Pergamon, 1983, pp. 54–73, http://www.sciencedirect.com/science/article/pii/B9780080309378500131.

[37] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch, in: NIPS Autodiff Workshop, 2017.

[38] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166, https://doi.org/10.1017/jfm.2016.615.

[39] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (1) (2018) 932–955, http://dl.acm.org/citation.cfm?id=3291125.3291150.

[40] Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, J. Comput. Phys. 394 (2019) 136–152, https://doi.org/10.1016/j.jcp.2019.05.027, http://www.sciencedirect.com/science/article/pii/S0021999119303584.

[41] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017, pp. 5574–5584, http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.pdf.

[42] Y. Gal, Uncertainty in Deep Learning, Ph.D. thesis, University of Cambridge, 2016, http://mlg.eng.cam.ac.uk/yarin/thesis/thesis.pdf.

[43] A. Iserles, A First Course in the Numerical Analysis of Differential Equations, 2nd edition, Cambridge Texts in Applied Mathematics, vol. 44, Cambridge University Press, 2008.

[44] C.M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[45] L.F. Richardson, J.A. Gaunt, VIII. The deferred approach to the limit, Philos. Trans. R. Soc. Lond. Ser. A 226 (636–646) (1927) 299–361, https://doi.org/10.1098/rsta.1927.0008, containing papers of a mathematical or physical character.
[46] T.A. Oliver, N. Malaya, R. Ulerich, R.D. Moser, Estimating uncertainties in statistics computed from direct numerical simulation, Phys. Fluids 26 (3) (2014) 035101, https://doi.org/10.1063/1.4866813.
[47] M.E. Tipping, Sparse Bayesian learning and the relevance vector machine, J. Mach. Learn. Res. 1 (June 2001) 211–244, http://www.jmlr.org/papers/v1/tipping01a.html.
[48] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, preprint arXiv:1412.6980.
[49] I. Loshchilov, F. Hutter, Fixing weight decay regularization in Adam, CoRR abs/1711.05101, arXiv:1711.05101.
[50] M.D. Richard, R.P. Lippmann, Neural network classifiers estimate Bayesian a posteriori probabilities, Neural Comput. 3 (4) (1991) 461–483, https://doi.org/10.1162/neco.1991.3.4.461.
[51] D. Barber, C.M. Bishop, Ensemble Learning in Bayesian Neural Networks, NATO ASI Series of Computer and Systems Sciences, vol. 168, 1998, pp. 215–238, https://www.microsoft.com/en-us/research/publication/ensemble-learning-in-bayesian-neural-networks/.
[52] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, preprint arXiv:1505.05424.
[53] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: representing model uncertainty in deep learning, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning, vol. 48, ICML'16, JMLR.org, 2016, pp. 1050–1059, http://dl.acm.org/citation.cfm?id=3045390.3045502.
[54] Q. Liu, D. Wang, Stein variational gradient descent: a general purpose Bayesian inference algorithm, in: D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016, pp. 2378–2386, http://papers.nips.cc/paper/6338-stein-variational-gradient-descent-a-general-purpose-bayesian-inference-algorithm.pdf.
[55] W. Maddox, T. Garipov, P. Izmailov, D.P. Vetrov, A.G. Wilson, A simple baseline for Bayesian uncertainty in deep learning, CoRR abs/1902.02476, arXiv:1902.02476.
[56] D. Ruppert, Efficient Estimations From a Slowly Convergent Robbins-Monro Process, Tech. rep., Cornell University Operations Research and Industrial Engineering, 1988, https://hdl.handle.net/1813/8664.
[57] B. Polyak, A. Juditsky, Acceleration of stochastic approximation by averaging, SIAM J. Control Optim. 30 (4) (1992) 838–855, https://doi.org/10.1137/0330046.
[58] P. Izmailov, D. Podoprikhin, T. Garipov, D.P. Vetrov, A.G. Wilson, Averaging weights leads to wider optima and better generalization, CoRR abs/1803.05407, arXiv:1803.05407.
[59] J.M. Hyman, B. Nicolaenko, The Kuramoto–Sivashinsky equation: a bridge between PDE's and dynamical systems, Physica D: Nonlinear Phenom. 18 (1) (1986) 113–126, https://doi.org/10.1016/0167-2789(86)90166-1, http://www.sciencedirect.com/science/article/pii/0167278986901661.
[60] J.M. Hyman, B. Nicolaenko, S. Zaleski, Order and complexity in the Kuramoto-Sivashinsky model of weakly turbulent interfaces, Physica D: Nonlinear Phenom. 23 (1) (1986) 265–292, https://doi.org/10.1016/0167-2789(86)90136-3, http://www.sciencedirect.com/science/article/pii/0167278986901363.
[61] R.W. Wittenberg, P. Holmes, Scale and space localization in the Kuramoto–Sivashinsky equation, Chaos, Interdisc. J. Nonlinear Sci. 9 (2) (1999) 452–465, https://doi.org/10.1063/1.166419.
[62] R.E. LaQuey, S.M. Mahajan, P.H. Rutherford, W.M. Tang, Nonlinear saturation of the trapped-ion mode, Phys. Rev. Lett. 34 (1975) 391–394, https://doi.org/10.1103/PhysRevLett.34.391, https://link.aps.org/doi/10.1103/PhysRevLett.34.391.
[63] Y. Kuramoto, T. Tsuzuki, Persistent propagation of concentration waves in dissipative media far from thermal equilibrium, Prog. Theor. Phys. 55 (2) (1976) 356–369, https://doi.org/10.1143/PTP.55.356.
[64] D. Michelson, G. Sivashinsky, Nonlinear analysis of hydrodynamic instability in laminar flames, II: numerical experiments, Acta Astronaut. 4 (11) (1977) 1207–1221, https://doi.org/10.1016/0094-5765(77)90097-2, http://www.sciencedirect.com/science/article/pii/0094576577900972.
[65] S. Cox, P. Matthews, Exponential time differencing for stiff systems, J. Comput. Phys. 176 (2) (2002) 430–455, https://doi.org/10.1006/jcph.2002.6995, http://www.sciencedirect.com/science/article/pii/S0021999102969950.
[66] J. Pathak, Z. Lu, B.R. Hunt, M. Girvan, E. Ott, Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data, Chaos: an interdisciplinary, Int. J. Nonlinear Sci. 27 (12) (2017) 121102, https://doi.org/10.1063/1.5010300.
[67] C.D. Brummitt, J. Sprott, A search for the simplest chaotic partial differential equation, Phys. Lett. A 373 (31) (2009) 2717–2721, https://doi.org/10.1016/j.physleta.2009.05.050, http://www.sciencedirect.com/science/article/pii/S0375960109006409.
[68] G.B. Whitham, Linear and Nonlinear Waves, vol. 42, John Wiley & Sons, 2011.
[69] S. Pan, K. Duraisamy, Long-time predictive modeling of nonlinear dynamical systems using neural networks, Complexity (2018), https://doi.org/10.1155/2018/4801012.
[70] M.S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The FEniCS project version 1.5, Arch. Numer. Softw. 3 (100) (2015) 9–23, http://publications.lib.chalmers.se/records/fulltext/228672/local_228672.pdf.
[71] A. Ali, S. ul Islam, S. Haq, A computational meshfree technique for the numerical solution of the two-dimensional coupled Burgers' equations, Int. J. Comput. Methods Eng. Sci. Mech. 10 (5) (2009) 406–422, https://doi.org/10.1080/15502280903108016.
[72] J. Nee, J. Duan, Limit set of trajectories of the coupled viscous Burgers' equations, Appl. Math. Lett. 11 (1) (1998) 57–61, https://doi.org/10.1016/S0893-9659(97)00133-X, http://www.sciencedirect.com/science/article/pii/S089396599700133X.
[73] I. Sobel, G. Feldman, A $3 \times 3$ isotropic gradient operator for image processing, 1968, presented at a talk at the Stanford Artificial Intelligence Project, pp. 271–272, 1968.
[74] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, preprint, arXiv:1502.03167.
[75] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: G. Gordon, D. Dunson, M. Dudík (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 15, PMLR, Fort Lauderdale, FL, USA, 2011, pp. 315–323, http://proceedings.mlr.press/v15/glorot11a.html.
[76] P. Cvitanović, R. Davidchack, E. Siminos, On the state space geometry of the Kuramoto—Sivashinsky flow in a periodic domain, SIAM J. Appl. Dyn. Syst. 9 (1) (2010) 1–33, https://doi.org/10.1137/070705623.