Modeling the Dynamics of PDE Systems with Physics-Constrained Deep Auto-Regressive Networks

Nicholas Genevaa, Nicholas Zabaras

University of Notre Dame

ngeneva@nd.edu *nzabaras@nd.edu* [1]

August 10, 2021

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

Overview

1. Introduction
2. Problem Definition
   - Surrogate Modeling of Dynamical Systems
3. Auto-regressive Dense Encoder-Decoder
   - AR-DenseED
   - Bayesian AR-DenseED
4. Kuramoto-Sivashinsky Equation
   - AR-DenseED Deterministic Predictions
   - BAR-DenseED Probabilistic Predictions
5. 1D Viscous Burgers' System
   - AR-DenseED Deterministic Predictions
   - BAR-DenseED Probabilistic Predictions
6. 2D Coupled Burgers' System
   - AR-DenseED Deterministic Predictions
   - BAR-DenseED Probabilistic Predictions
7. Conclusion

Overview

- An AutoRegressive
- Dense Encoder-Decoder
- CNN
- Non-linear dynamic systems
- Without Training (training data)
- No more Computational Cost than Standard Numerical Methods

General PDE Equation

We are interested in surrogate models of non-linear dynamical systems in a space-time domain.

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

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

Time Equation and Recursion

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

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

Loss Functions

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

Deterministic Surrogate Model

We pose the following definition for this surrogate model:

Definition

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

Probabilistic Surrogate Model

Definition

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

AR-DenseED Schematic

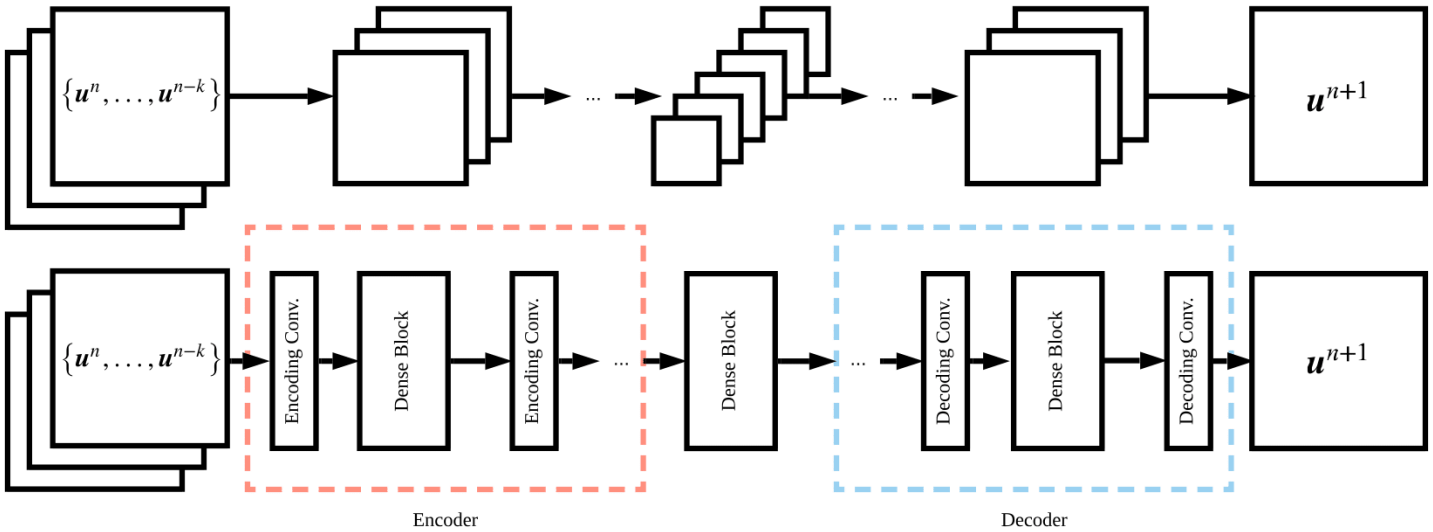


Figure: Schematic of the auto-regressive dense encoder-decoder. The top shows the dimensionality of the data in the network, the bottom shows the model architecture. Using encoding convolutions lowers the dimensionality of the input feature map, while decoding convolutions increase the dimensionality. 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.

AR-DenseED Training

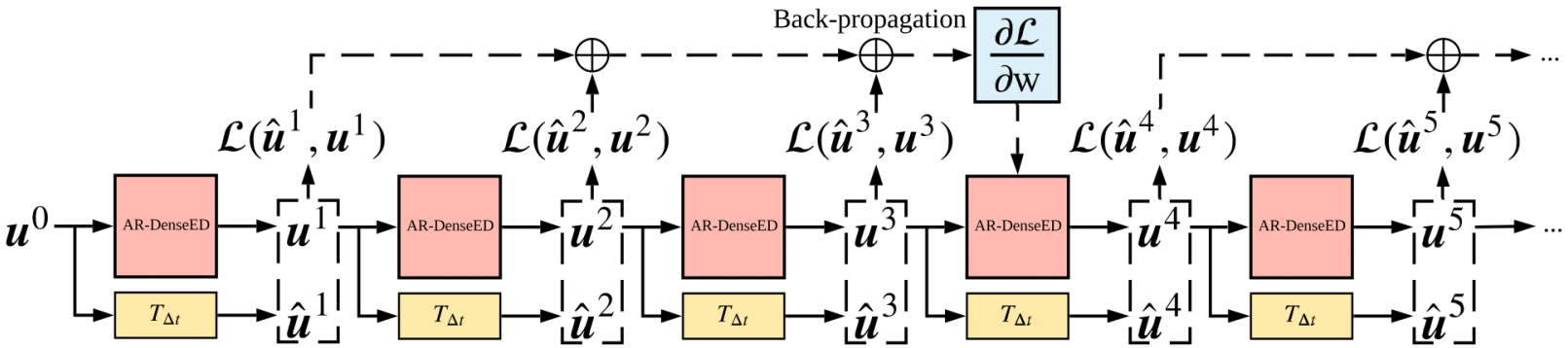


Figure: Multi-time-step back-propagation of the AR-DenseED where **w** are the model's learnable weights, u^n is the model's prediction at time-step n, $\hat{u}^n$ is the target value calculated using the numerical time-integrator $T_{\Delta t}$, and $\mathcal{L}$ is the physics-constrained loss at a 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.

Algorithm 1: AR-DenseED Prediction

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}

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

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

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

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

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

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

Algorithm 2: Training AR-DenseED

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: η

$\text{tsteps} = \text{linspace}(p, T_{max}, N)$;

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

    $\hat{\boldsymbol{u}}^i = T_{\Delta t}(\boldsymbol{\mathcal{U}}^i, F_{\Delta x})$; ▷ Time integration

    $\mathcal{L}^i = \mathcal{L}^{i-1} + \text{MSE}(\hat{\boldsymbol{u}}^i, \boldsymbol{u}^i)$; ▷ 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 $f(\cdot, \mathbf{w})$;

Bayesian AR-DenseED - Motivation

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.
Account for two major uncertainity components

- *aleotoric* which quantifies noise in the observations
- *epistemic* which quantifies inherent uncertainty in the model

However, The numerical time-integrator in the Loss Function introduces truncation error:

$$\mathrm{u}_i^{n+1} = T_{\Delta t}\left(\mathcal{U}^{n+1}, F_{\Delta x}\right) + E_{\Delta t} + E_{\Delta x}, \tag{6}$$

where $E_{\Delta t}$ and $E_{\Delta x}$ denote the error associated with the discretization of the temporal and spatial derivatives, respectively.

Bayesian AR-DenseED - Posterior Formulation

We account for the potential discretization error that may arise from $T_{\Delta t}$ through additive output-wise noise, ϵ, for a single arbitrary time-step:

$$\hat{\mathrm{u}}^i = f\left(\boldsymbol{\chi}^i, \mathbf{w}\right) + \epsilon, \quad p\left(\epsilon\right) = \mathcal{N}\left(\epsilon \,|0, \beta^{-1}\mathrm{I}_d\right), \tag{7}$$

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

$$\begin{aligned} p\left(\hat{\mathrm{u}}^i|\boldsymbol{\chi}^i, \mathbf{w}, \beta\right) &= \mathcal{N}\left(\hat{\mathrm{u}}^i|f\left(\boldsymbol{\chi}^i, \mathbf{w}\right), \beta^{-1}\mathrm{I}_d\right), \\ &= \mathcal{N}\left(T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right)|f\left(\boldsymbol{\chi}^i, \mathbf{w}\right), \beta^{-1}\mathrm{I}_d\right). \end{aligned} \tag{8}$$

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.

Bayesian AR-DenseED - Posterior Formulation

Under the Markov assumption, we can formulate the likelihood of an entire time-sequence as the product of individual steps:

$$p\left(\hat{\mathrm{u}}^N, \ldots, \hat{\mathrm{u}}^1 | \mathrm{u}_0, \mathbf{w}, \beta\right) = \prod_{i=1}^{N} p\left(\hat{\mathrm{u}}^i | \boldsymbol{\chi}^i, \mathbf{w}, \beta\right),$$

$$= \prod_{i=1}^{N} \mathcal{N}\left(T_{\Delta t}\left(\boldsymbol{\mathcal{U}}^i, F_{\Delta x}\right) | f\left(\boldsymbol{\chi}^i, \mathbf{w}\right), \beta^{-1} \mathbf{I}_d\right). \quad (9)$$

Bayesian AR-DenseED - Posterior Formulation

A gamma prior is assigned to the noise precision β:

$$p(\beta) = \Gamma\left(\beta | a_1, b_1\right), \tag{10}$$

where a_1 and b_1 are the shape and rate parameters, respectively. The hyper-parameters of the β prior are set based on the *a priori* estimate of the magnitude of the discretization error of $T_{\Delta t}$ and $F_{\Delta x}$.

Bayesian AR-DenseED - Posterior Formulation

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

$$\begin{aligned}\hat{\mathrm{u}}^i &= T_{\Delta t}\left(\mathcal{U}^i, F_{\Delta x}\right) + c_0\left(\Delta t\right)^{k_0} + c_1\left(\Delta t\right)^{k_1} + c_2\left(\Delta t\right)^{k_2} + \ldots \\ &= T_{\Delta t}\left(\mathcal{U}^i, F_{\Delta x}\right) + c_0\left(\Delta t\right)^{k_0} + \mathcal{O}\left(\left(\Delta t\right)^{k_1}\right), \end{aligned} \tag{11}$$

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 $\mathbb{E}\left(\beta^{-1}\right) = c_0\left(\Delta t\right)^{k_0}$. Additionally this prior is given a large variance to reduce its strength.

Bayesian AR-DenseED - Posterior Formulation

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 α:

$$p(\mathrm{w}|\alpha) = \mathcal{N}\left(\mathrm{w}|0, \alpha^{-1}\mathrm{I}_K\right), \quad p(\alpha) = \mathit{Gamma}(\alpha|a_0, b_0), \tag{12}$$

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.

Bayesian AR-DenseED - Posterior Formulation

For a batch of M i.i.d. training scenarios, $\mathcal{S} = \{\mathrm{u}_{0,i}\}_{i=1}^{M}$, the joint posterior of the network discounts the prior as follows:

$$
\begin{aligned}
p\left(\mathbf{w}, \beta | \mathcal{S}\right) &\sim \prod_{j=1}^{M} p\left(\hat{\mathrm{u}}_j^N, \ldots, \hat{\mathrm{u}}_j^1 | \mathrm{u}_{0,j}, \mathbf{w}, \beta\right) p\left(\mathbf{w}\right) p\left(\beta\right), \\
&\sim \prod_{j=1}^{M} \prod_{i=1}^{N} \left[p\left(\hat{\mathrm{u}}_j^i | \boldsymbol{\chi}_j^i, \mathbf{w}, \beta\right)\right] p\left(\mathbf{w} | \alpha\right) p\left(\alpha | a_0, b_0\right) p\left(\beta | a_1, b_1\right), \\
&\sim \prod_{j=1}^{M} \prod_{i=1}^{N} \left[\mathcal{N}\left(T_{\Delta t}\left(\boldsymbol{\mathcal{U}}_j^i, F_{\Delta x}\right) | f\left(\boldsymbol{\chi}_j^i, \mathbf{w}\right), \beta^{-1} \mathrm{I}_d\right)\right] \mathcal{N}\left(\mathrm{w} | 0, \alpha^{-1} \mathrm{I}_K\right) \\
&\qquad \Gamma\left(\alpha | a_0, b_0\right) \Gamma\left(\beta | a_1, b_1\right)
\end{aligned}
\tag{13}
$$

Maximum A posteriori Probability (MAP)

For sampling the posterior of BAR-DenseED, we will use a recently proposed Stochastic Weight Averaging Gaussian (SWAG) [**?**]. SWAG approximates the posterior in two phases:

1. We optimize the model using the ADAM [**?**], an extension of SGD, with exponential learning rate decay.
2. Once the model has been trained, Stochastic Gradient Descent (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.

BAR-DenseED Posterior with SWAG

A Gaussian distribution with S samples of the model's parameters:

$$p(\boldsymbol{\theta}|\mathcal{S}) \sim \mathcal{N}\left(\boldsymbol{\theta}_{SWA}, \boldsymbol{\Sigma}_{SWA}\right), \quad \boldsymbol{\theta} \equiv \{\mathbf{w}, ln(\beta)\}, \tag{14}$$

$$\boldsymbol{\theta}_{SWA} = \frac{1}{S}\sum_{i=1}^{S}\boldsymbol{\theta}_i, \quad \boldsymbol{\Sigma}_{SWA} = \frac{1}{2}\left(\boldsymbol{\Sigma}_{Diag} + \boldsymbol{\Sigma}_{lr}\right), \tag{15}$$

$$\bar{\boldsymbol{\theta}}^2 = \frac{1}{S}\sum_{i=1}^{S}\boldsymbol{\theta}_i^2, \quad \boldsymbol{\Sigma}_{Diag} = \mathrm{Diag}\left(\bar{\boldsymbol{\theta}}^2 - \boldsymbol{\theta}_{SWA}^2\right), \tag{16}$$

$$\boldsymbol{\Sigma}_{lr} = \frac{1}{K-1}\mathrm{D}\mathrm{D}^T, \quad \mathrm{D}_i = \left(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{SWA}\right), \tag{17}$$

where $\boldsymbol{\theta}_i$ are the model parameters at epoch i, and
$\mathrm{D} \in \mathbb{R}^{K\times H}$ is a deviation matrix consisting of the H most recent parameter samples forming a low-rank approximation.
$\mathrm{D}_i \in \mathbb{R}^K$ is a column of this deviation matrix.

Approximating the BAR-DenseED Posterior with SWAG

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: η_{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: $\boldsymbol{\theta}_{SWA} = \bar{\boldsymbol{\theta}}$; $\boldsymbol{\Sigma}_{Diag} = \bar{\boldsymbol{\theta}^2} - \boldsymbol{\theta}_{SWA}^2$; $\boldsymbol{D}$;

Predictive Statistics

We will approximate the marginalization of the model parameters, often referred to as Bayesian model averaging, using Monte Carlo with P parameter samples:

$$p\left(\hat{\mathrm{u}}^*|\boldsymbol{\chi}^*,\mathcal{S}\right) = \int p\left(\hat{\mathrm{u}}^*|f\left(\boldsymbol{\chi}^*,\mathbf{w}\right),\beta^{-1}\mathbf{I}\right) p\left(\mathbf{w},\beta|\mathcal{S}\right) d\mathbf{w}d\beta,$$

$$\approx \frac{1}{P}\sum_{i=1}^{P} p\left(\hat{\mathrm{u}}^*|f\left(\boldsymbol{\chi}^*,\mathbf{w}_i\right),\beta_i^{-1}\mathbf{I}\right), \quad \{\mathbf{w}_i,\beta_i\} \sim p\left(\boldsymbol{\theta}|\mathcal{S}\right), \quad (18)$$

where $\boldsymbol{\chi}^*$ and $\hat{\mathrm{u}}^*$ are the predictive inputs and outputs, respectively.

Predictive Statistics

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:

$$\begin{aligned}\mathbb{E}\left[\hat{\mathrm{u}}^*|\boldsymbol{\chi}^*, \mathcal{S}\right] &= \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{S})}\left[\mathbb{E}\left(\hat{\mathrm{u}}^*|\boldsymbol{\chi}^*, \mathbf{w}, \beta\right)\right], \\ &= \mathbb{E}_{p(\mathbf{w}|\mathcal{S})}\left[f\left(\boldsymbol{\chi}^*, \mathbf{w}\right)\right] \approx \frac{1}{P}\sum_{i=1}^{P} f\left(\boldsymbol{\chi}^*, \mathbf{w}_i\right), \end{aligned} \tag{19}$$

where the additive output noise is not present due to its zero-mean Gaussian density.

Predictive Statistics

The predictive conditional covariance can also be obtained in a similar fashion:

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

where $\mathbb{E}\left[\mathrm{u}^* | \boldsymbol{\chi}^*, \mathcal{S}\right]$ has been defined in Eq. (19). 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.

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:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^4 u}{\partial x^4} = 0, \tag{20}$$

$$u(0,t) = u(L,t), \quad x \in [0,L], \quad t \in [0,T], \tag{21}$$

where ν 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 [**?**].

Kuramoto-Sivashinsky Equation

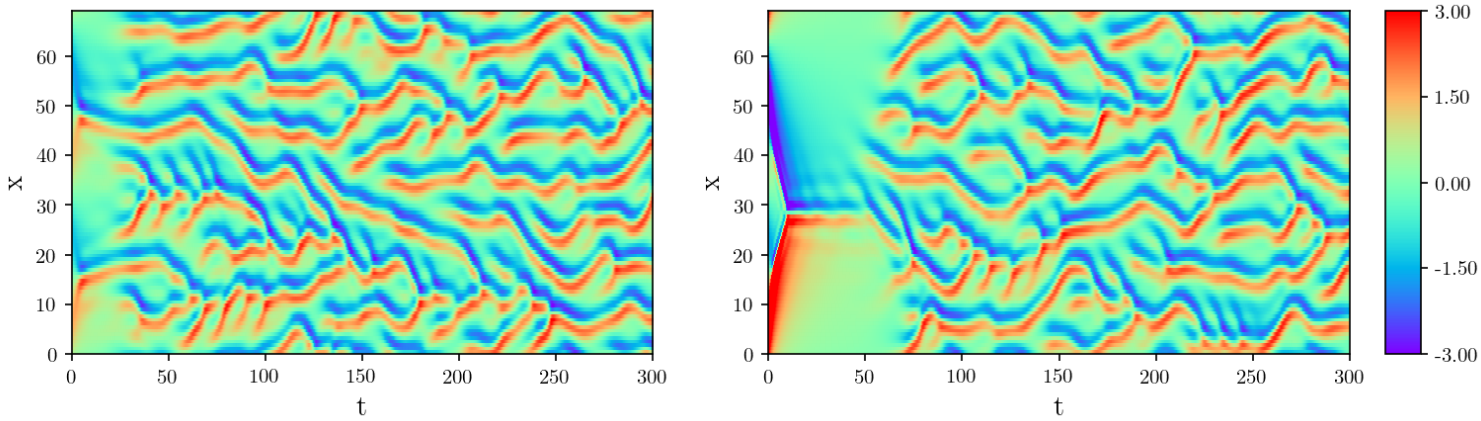


Figure: The Kuramoto-Sivashinsky equation for two different initial states solved using the spectral ETDRK4 scheme [**?**].

Kuramoto-Sivashinsky Equation

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, $\chi^{n+1} = \left\{u^n, u^{n-1}\right\}$. 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.

Kuramoto-Sivashinsky Equation

For the physics-constrained loss function, the implicit Crank-Nicolson time integration is used and the remaining spatial gradients are discretized as follows:

$$T_{\Delta t}(\mathcal{U}^n, F_{\Delta x}) = \mathrm{u}^n + \Delta t \left[-0.5 \left(F_{\Delta x}(\mathrm{x}, \mathrm{u}^{n+1}) + F_{\Delta x}(\mathrm{x}, \mathrm{u}^n)\right)\right], \tag{22}$$

$$F_{\Delta x}(\mathrm{x}, \mathrm{u}^n) = u^n u_x^n + u_{xx}^n + u_{xxxx}^n,$$

$$u u_x = \frac{-u_{i+2}^2 + 8u_{i+1}^2 - 8u_{i-1}^2 + u_{i-2}^2}{24\Delta x}, \tag{23}$$

$$u_{xx} = \frac{-u_{i+2} + 16u_{i+1} - 30u_i + 16u_{i-1} - u_{i-2}}{12\Delta x^2}, \tag{24}$$

$$u_{xxxx} = \frac{-u_{i+3} + 12u_{i+2} - 39u_{i+1} + 56u_i - 39u_{i-1} + 12u_{i-2} - u_{i-3}}{6\Delta x^4}, \tag{25}$$

Kuramoto-Sivashinsky Equation

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 **??**.
- This Fourier series serves to approximate the physical turbulence of the system, and thus estimating the true distribution of the possible initial states $p(u_0)$.
- During test time, we use 200 test cases of true turbulent initial states of the system obtained from a numerical simulator.

AR-DenseED Deterministic Predictions

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

$$\mathrm{MSE}(t) = \frac{1}{N}\sum_{i=1}^{N}\left(u_i(t) - u_i^*(t)\right)^2, \tag{26}$$

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.

AR-DenseED Deterministic Predictions

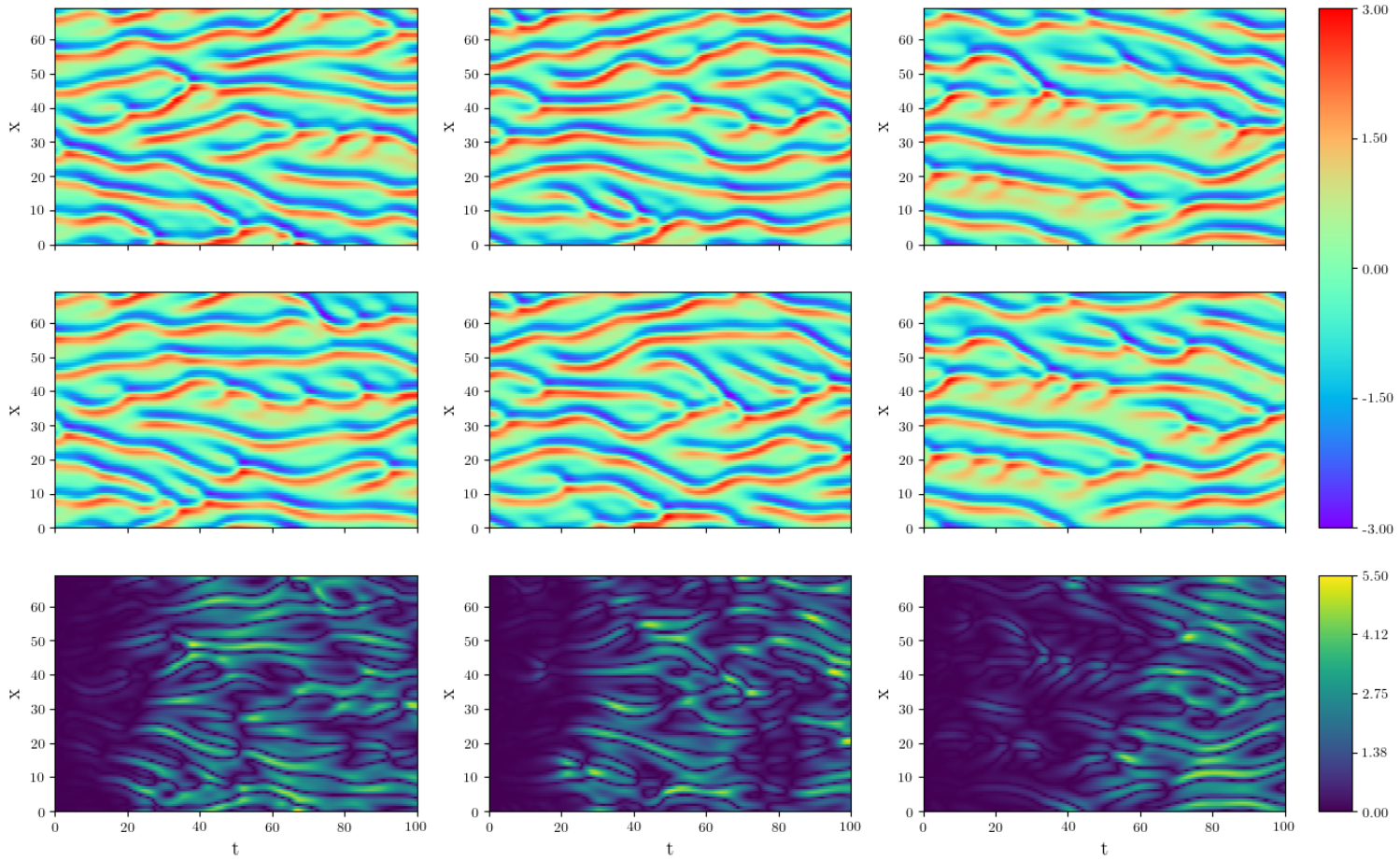


Figure: Three test predictions of the K-S equation using AR-DenseED. (Top to bottom) Target field solved system using the spectral ETDRK4 scheme, AR-DenseED prediction and finally the L_1 error.

AR-DenseED Deterministic Predictions

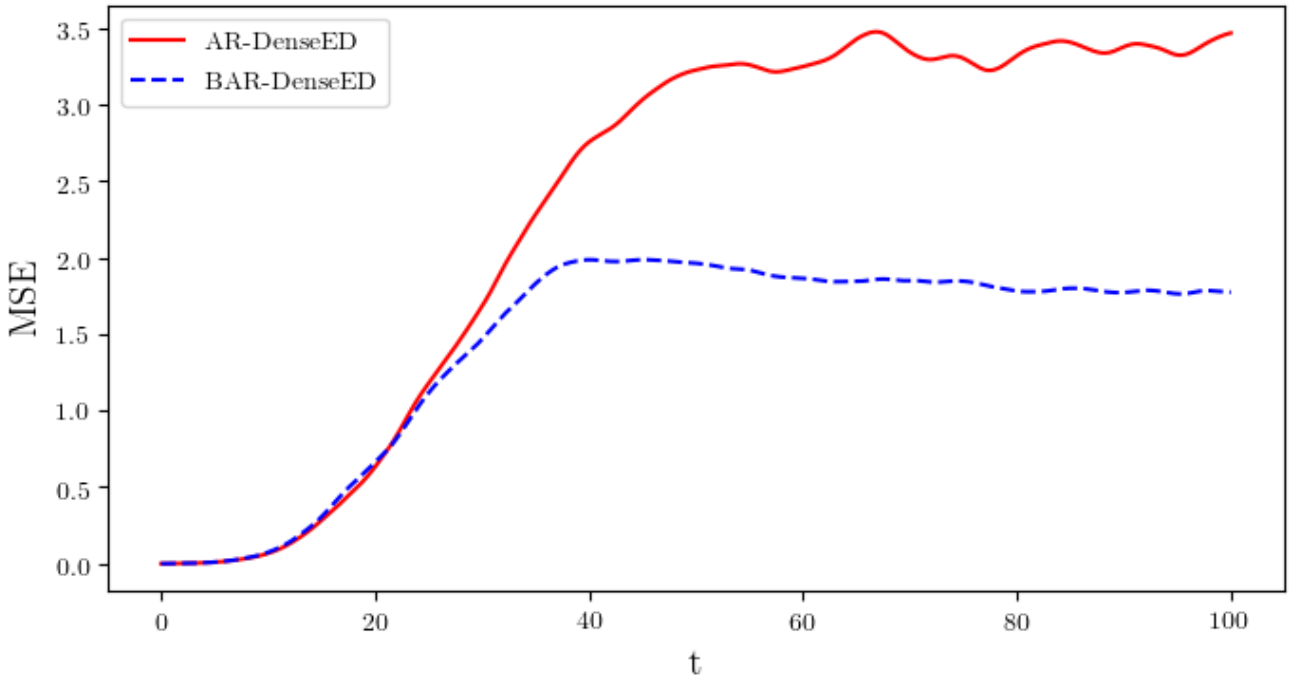


Figure: The mean MSE as a function of time for a test set of 200 cases for the Kuramoto-Sivashinsky system. The error of BAR-DenseED is calculated using the expected value of the predictive distribution approximated using 30 samples of the posterior.

AR-DenseED Deterministic Predictions

Table: 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	Δt	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

BAR-DenseED Probabilistic Predictions

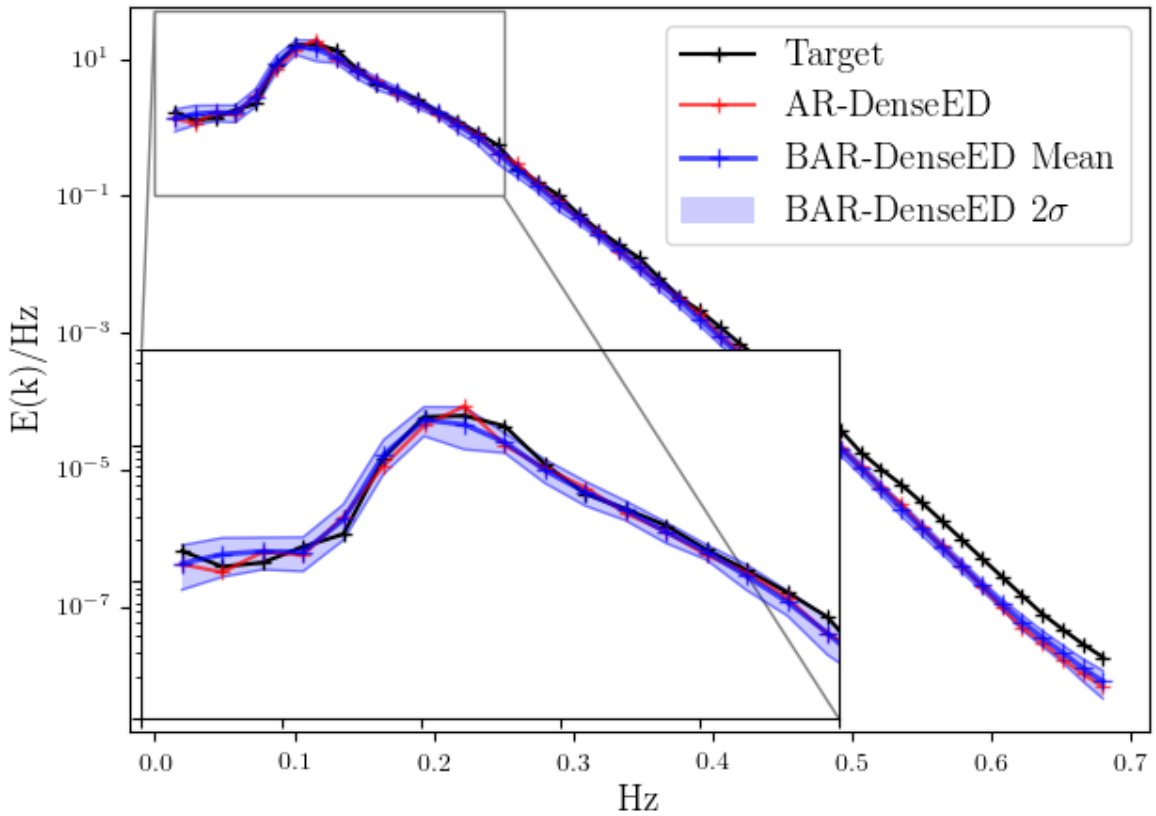


Figure: The time-averaged spectral energy density of the simulated result using the spectral ETDRK4 scheme (target), AR-DenseED deterministic prediction and BAR-DenseED empirical mean and standard deviation calculated from 30

BAR-DenseED Probabilistic Predictions

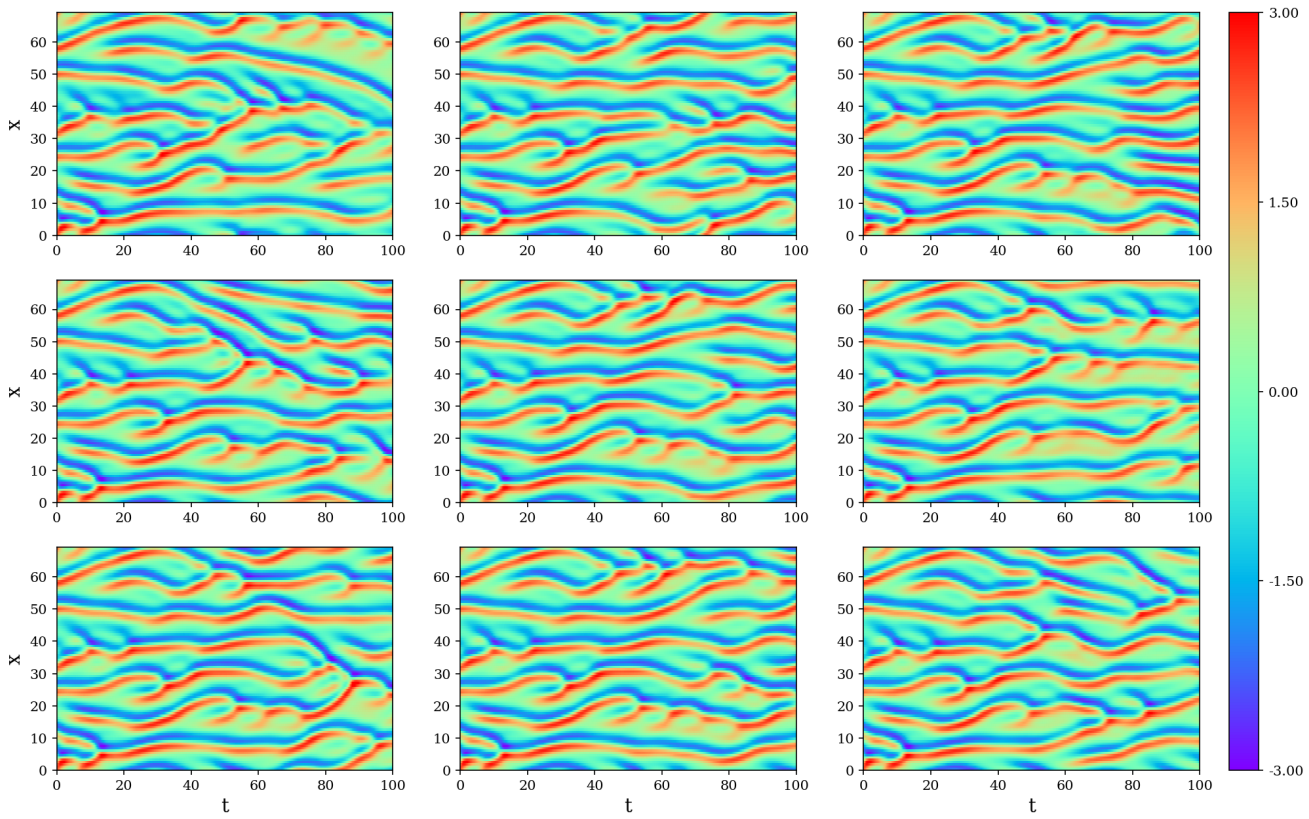


Figure: Samples from the posterior of BAR-DenseED approximated using SWAG for the Kuramoto-Sivashinsky system. The top left is the simulated result using the spectral ETDRK4 scheme.

1D Viscous Burgers' System

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

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

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

where u is the velocity and ν 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 [**?**].

1D Viscous Burgers' System

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:

$$w(x) = a_0 + \sum_{l=1}^{L} a_l \sin(2l\pi x) + b_l \cos(2l\pi x),$$
$$u(x, 0) = \frac{2w(x)}{\max_x |w(x)|} + c, \tag{29}$$

where $a_l, b_l \sim \mathcal{N}(0, 1)$, $L = 4$ and $c \sim \mathcal{U}(-1, 1)$.

1D Viscous Burgers' System

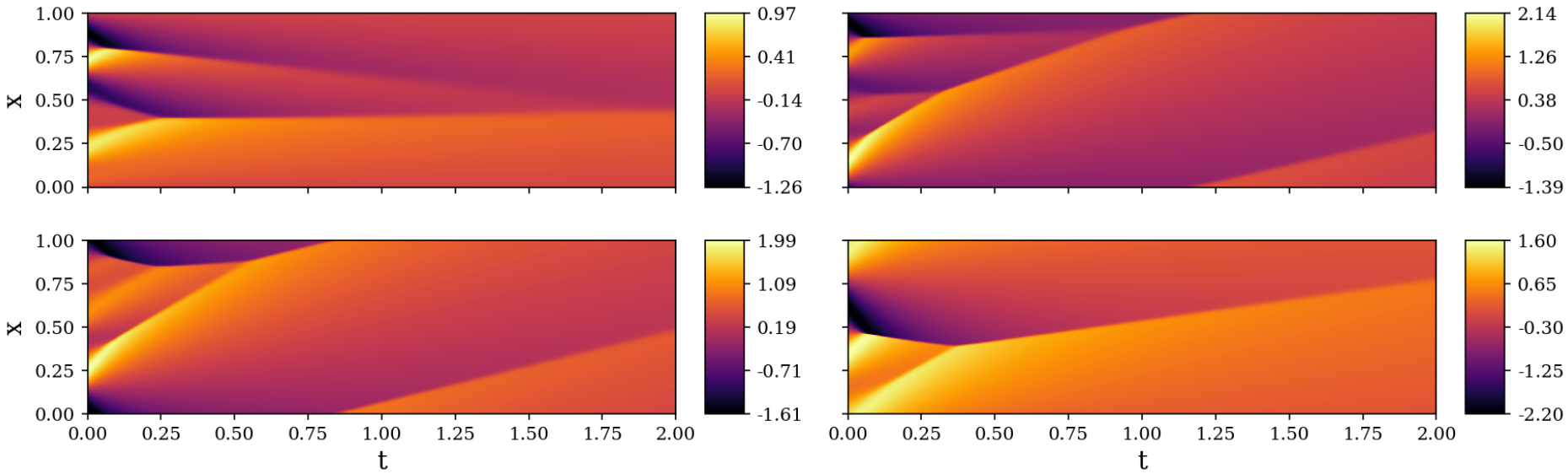


Figure: 1D viscous Burgers' equation simulations for four various initial conditions solved using Fenics finite element package [**?**].

1D Viscous Burgers' System

Again the negative log of the joint posterior in Eq. (13) is the loss function with the implicit Crank-Nicolson time integration. The remaining spatial gradients are discretized as follows:

$$\begin{aligned} T_{\Delta t}(\mathcal{U}^n, F_{\Delta x}) &= \mathrm{u}^n + \Delta t \left[-0.5 \left(F_{\Delta x}(\mathrm{x}, \mathrm{u}^{n+1}) + F_{\Delta x}(\mathrm{x}, \mathrm{u}^n) \right) \right], \\ F_{\Delta x}(\mathrm{x}, \mathrm{u}^n) &= u^n u_x^n - \nu u_{xx}^n, \end{aligned} \tag{30}$$

$$u u_x = \frac{u_{i+1}^2 - u_{i-1}^2}{4\Delta x}, \quad u_{xx} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}, \tag{31}$$

where the spatial gradients are approximated using second-order accurate approximations that are implemented efficiently using convolutional operators.

AR-DenseED Deterministic Predictions

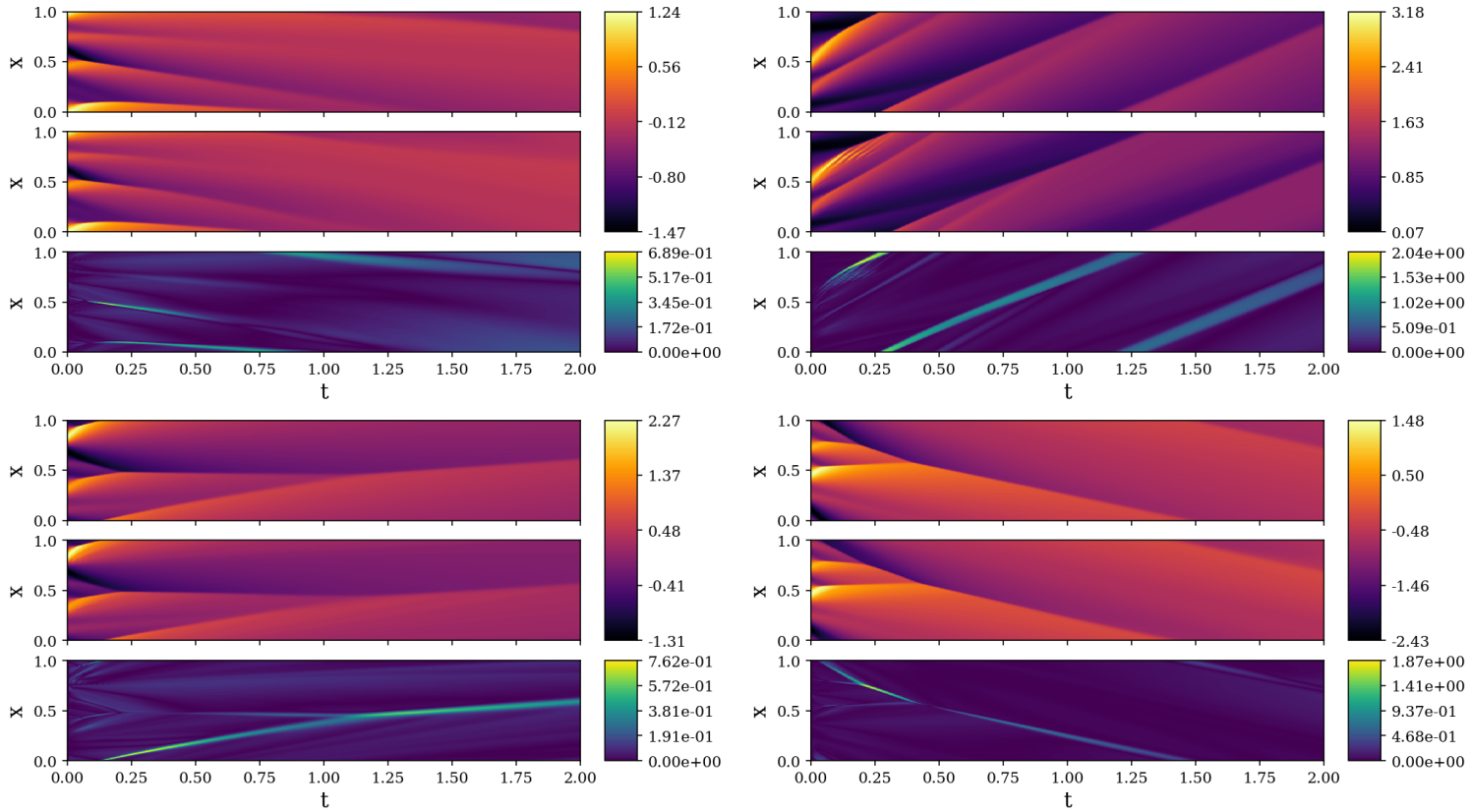


Figure: AR-DenseED predictions for four test initial conditions of the 1D Burgers' system. (Top to bottom) FEM target solution, AR-DenseED prediction, and L_1 error.

AR-DenseED Deterministic Predictions

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:

$$\begin{aligned} \mathrm{ESE}(t) &= \left[\int_0^1 \frac{(u(x,t))^2}{2}dx - \int_0^1 \frac{(u^*(x,t))^2}{2}dx\right]^2, \\ &= \left[\frac{1}{N}\sum_{i=1}^{N}\frac{(u_i(t))^2}{2} - \frac{1}{N}\sum_{i=1}^{N}\frac{(u_i^*(t))^2}{2}\right]^2, \end{aligned} \tag{32}$$

which instead captures the discrepancy of the total energy, $u^2/2$, within the domain, making this metric invariant to shock location.

AR-DenseED Deterministic Predictions

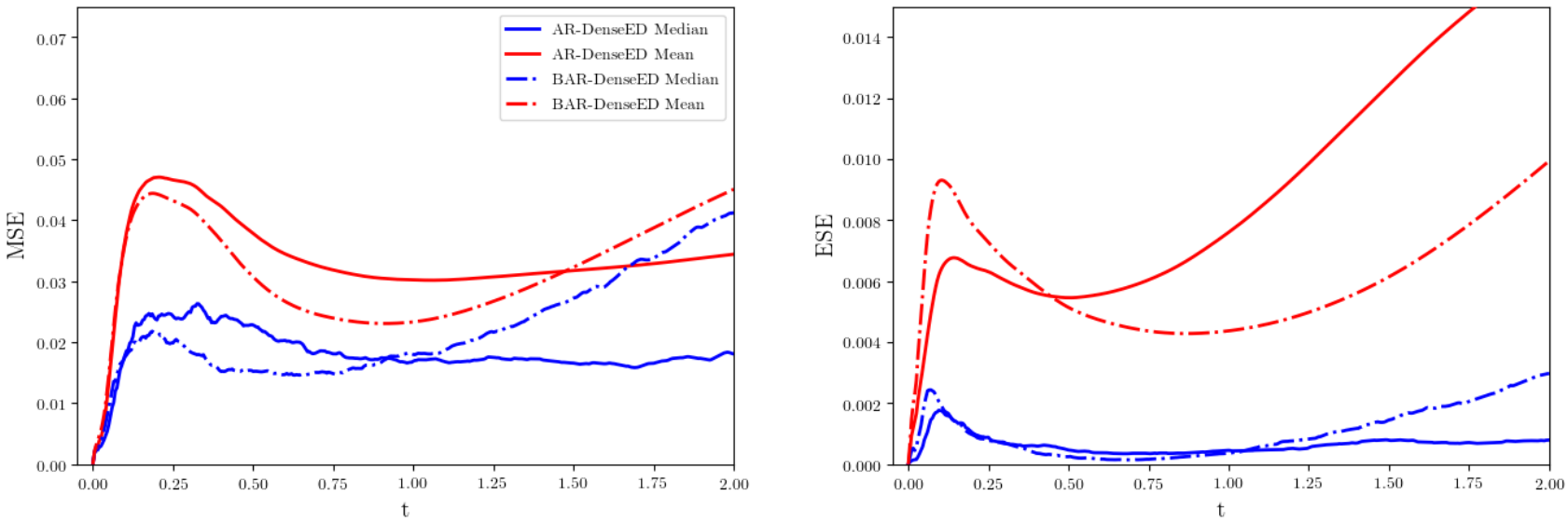


Figure: (Left to right) The mean square error (MSE) and energy squared error (ESE) as a function of time for a test set of 200 cases for the 1D Burgers' system. The error of BAR-DenseED is calculated using the expected value of the predictive distribution approximating using 30 samples of the posterior.

AR-DenseED Deterministic Predictions

Table: 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	Δt	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

BAR-DenseED Probabilistic Predictions

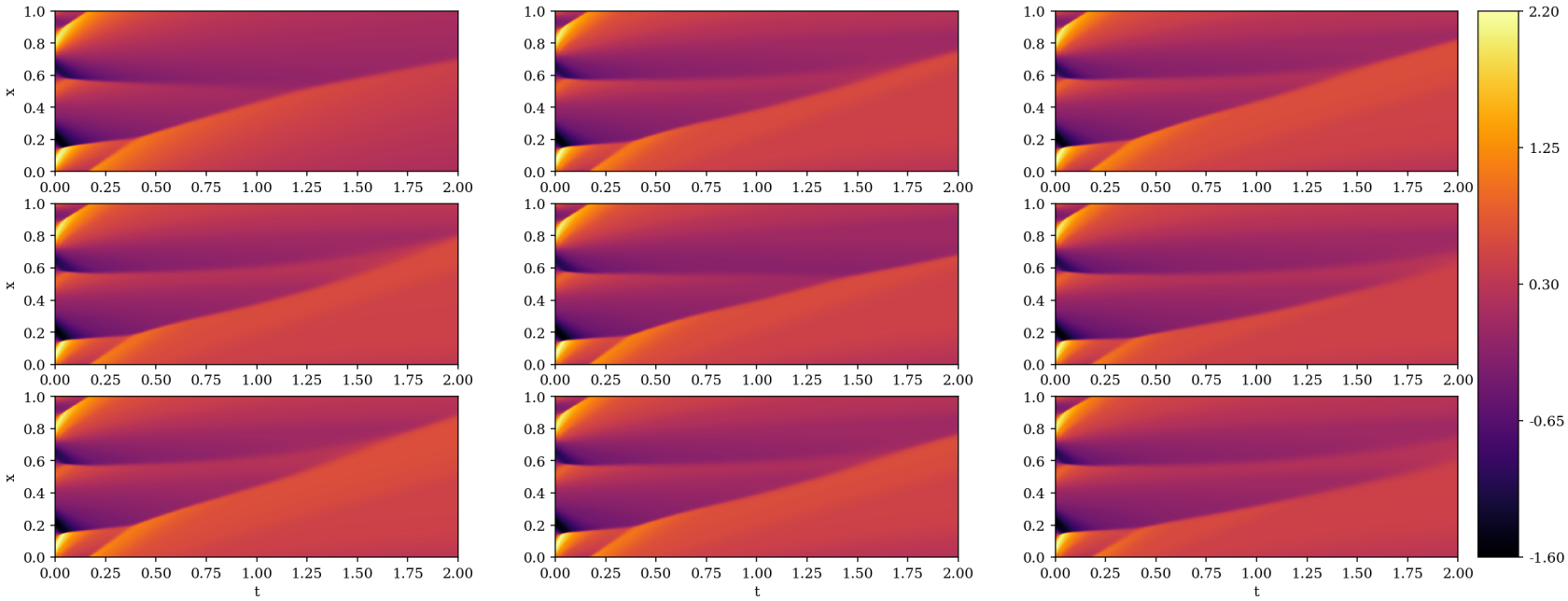


Figure: Samples from the posterior of BAR-DenseED approximated using SWAG for the 1D Burgers' system. The top left is the simulated result using the finite element method.

BAR-DenseED Probabilistic Predictions

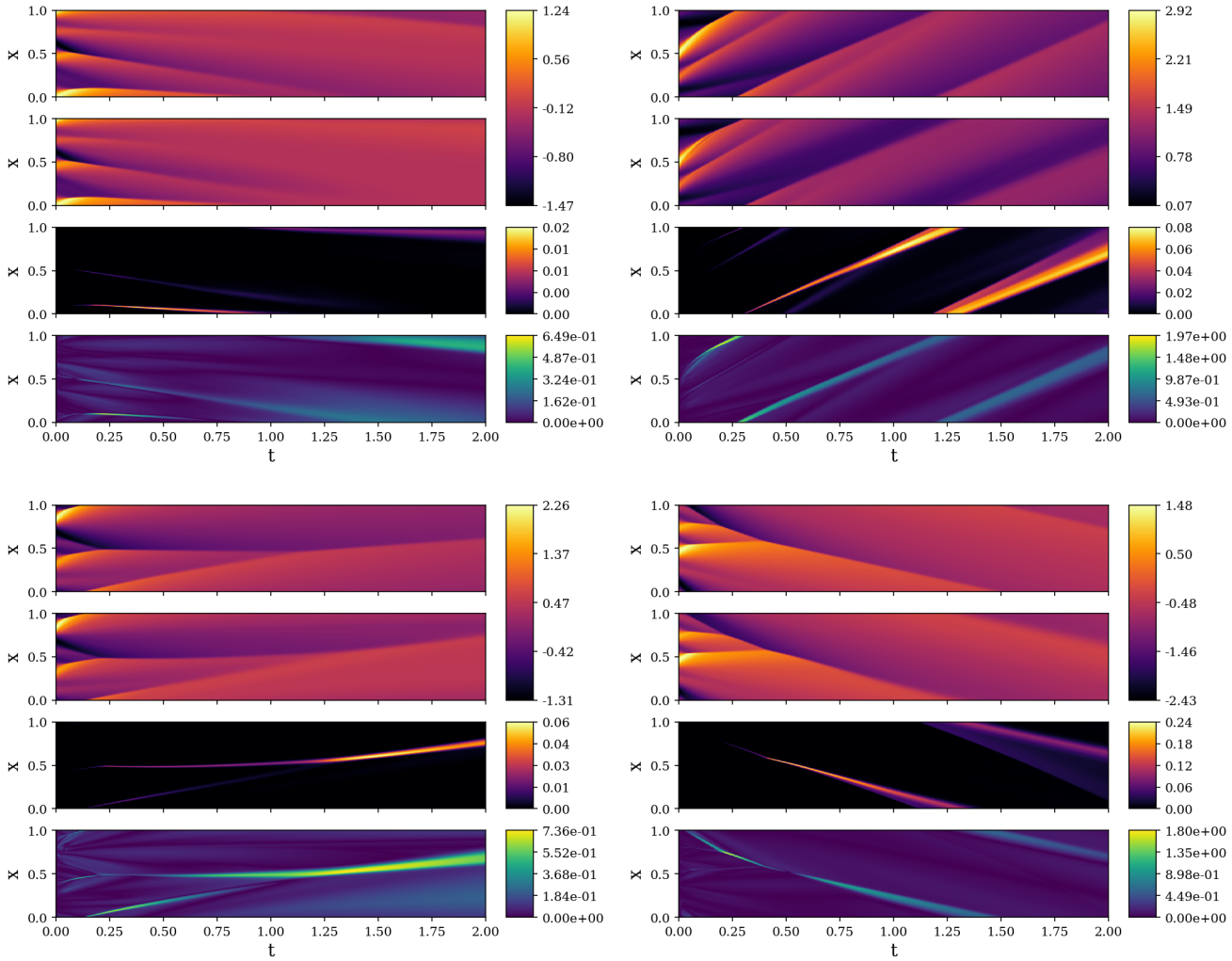


Figure: BAR-DenseED predictions for four test initial conditions of the 1D Burgers' system. (Top to bottom) FEM target solution, BAR-DenseED expected

BAR-DenseED Probabilistic Predictions

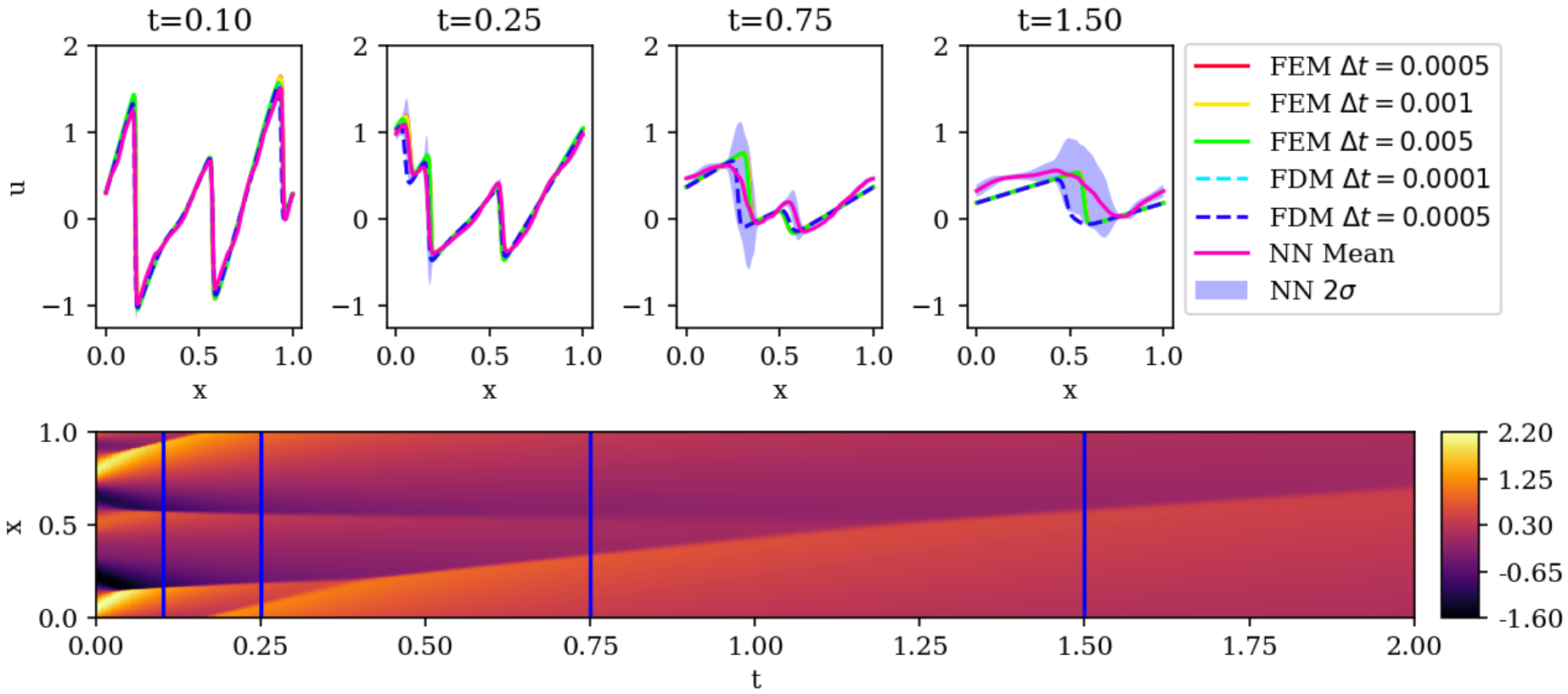


Figure: Instantaneous profiles of both the finite element method (FEM) and finite difference method (FDM) numerical solvers along with BAR-DenseED neural network (NN) predictive expectation and standard deviation at four various times of a test case. The bottom contour is the ideal target calculated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

BAR-DenseED Probabilistic Predictions

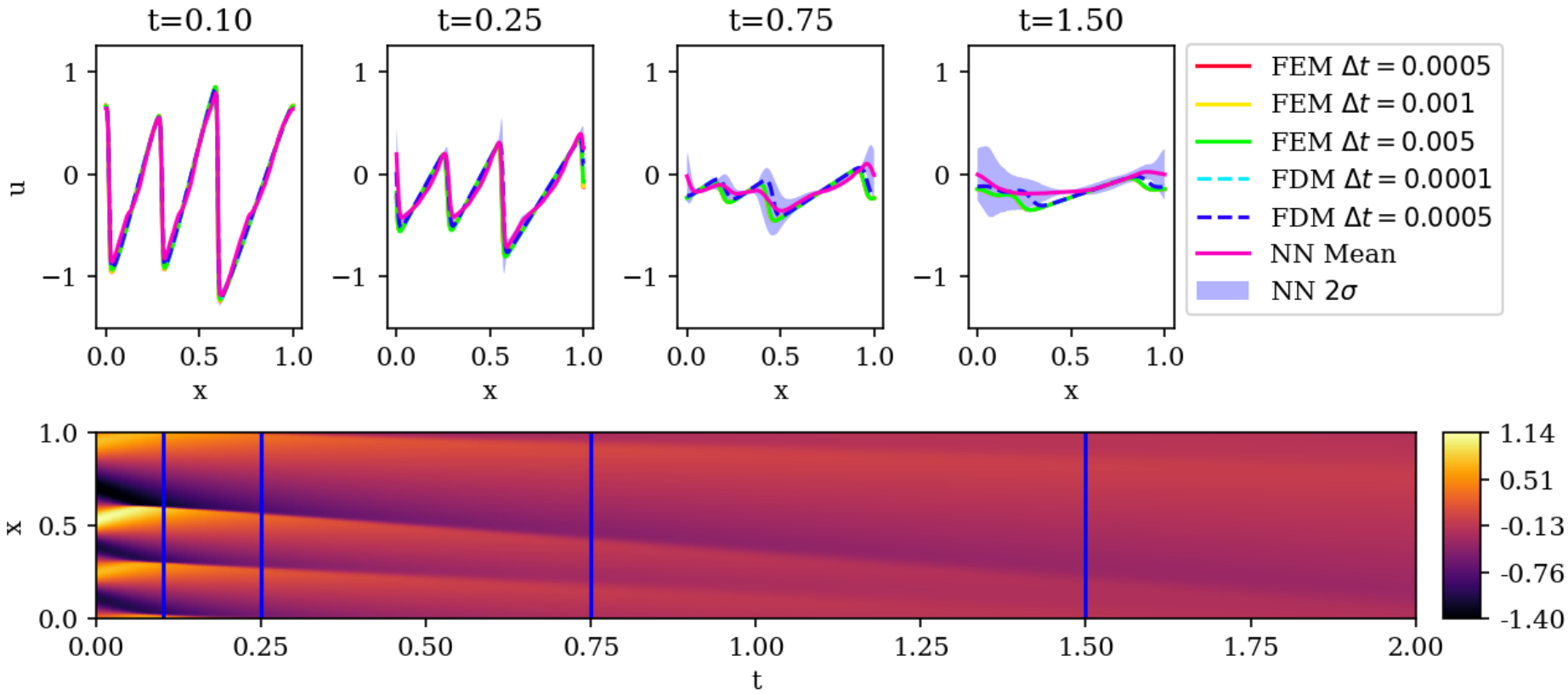


Figure: Instantaneous profiles of both finite element method (FEM) and finite difference method (FDM) numerical solvers along with BAR-DenseED neural network (NN) predictive expectation and standard deviation at four various times of a test case. The bottom contour is the ideal target calculated using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

2D Coupled Burgers' System

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

$$\mathrm{u}_t + \mathrm{u} \cdot \nabla \mathrm{u} - \nu \Delta \mathrm{u} = 0, \tag{33}$$

$$\mathrm{u}(0, y, t) = \mathrm{u}(L, y, t), \quad \mathrm{u}(x, 0, t) = \mathrm{u}(x, L, t), \tag{34}$$

$$\{x, y\} \in [0, L], \quad t \in [0, T], \tag{35}$$

which when expanded into its components takes the following form:

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0, \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0, \end{aligned} \tag{36}$$

2D Coupled Burgers' System

where, ν 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.

2D Coupled Burgers' System

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:

$$\begin{aligned} \mathrm{w}(x,y) &= \sum_{i=-L}^{L}\sum_{j=-L}^{L} \mathrm{a}_{ij}\sin(2\pi\,(ix+jy)) + \mathrm{b}_{ij}\cos(2\pi\,(ix+jy)), \\ \mathrm{u}(x,y,0) &= \frac{2\mathrm{w}(x,y)}{\max_{\{x,y\}}|\mathrm{w}(x,y)|} + \mathrm{c}, \end{aligned} \tag{37}$$

where $\mathrm{a}_{ij}, \mathrm{b}_{ij} \sim \mathcal{N}(0, \mathrm{I}_2)$, $L = 4$ and $\mathrm{c} \sim \mathcal{U}(-1,1) \in \mathbb{R}^2$. Several of these randomly generated initial conditions are illustrated in Fig. 15.

2D Coupled Burgers' System

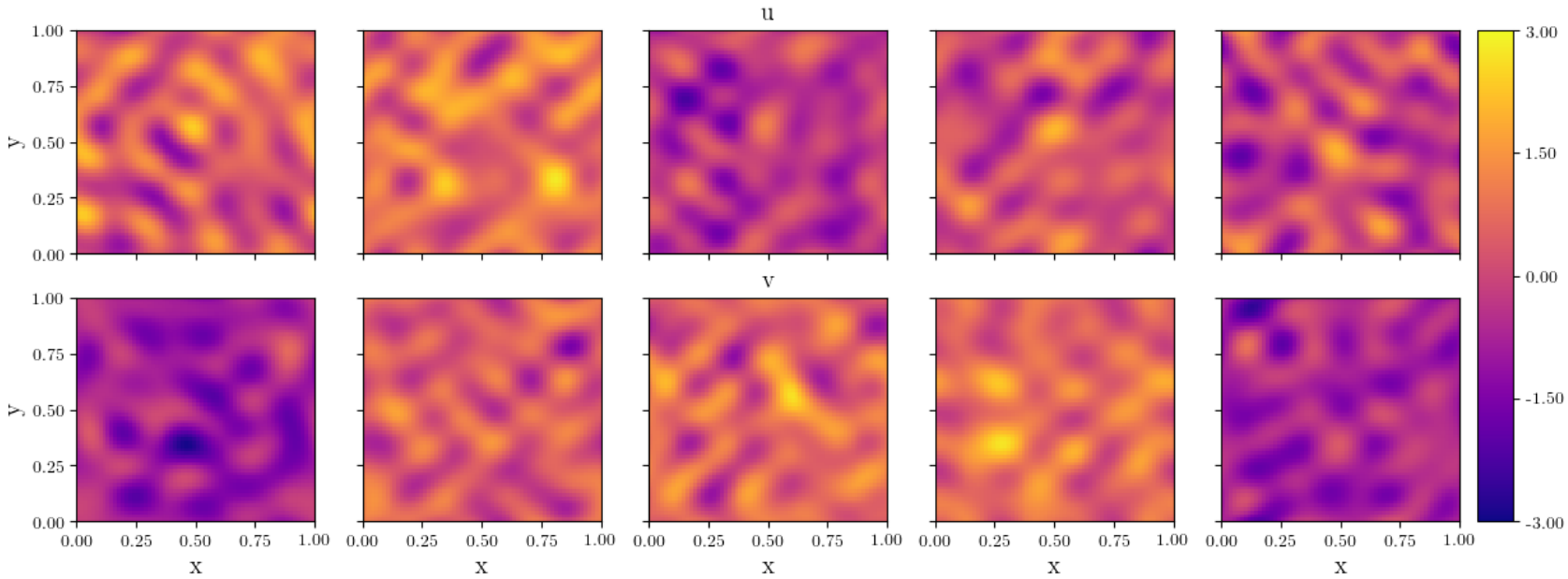
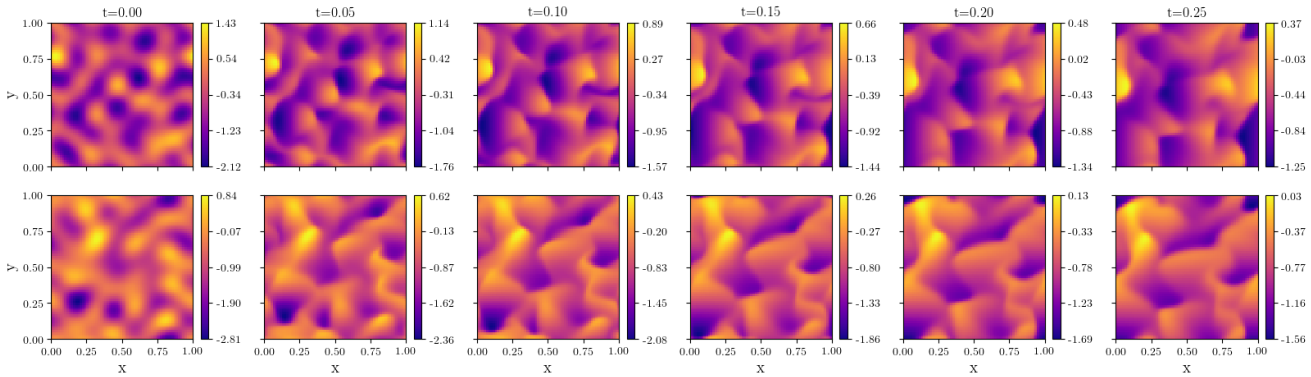
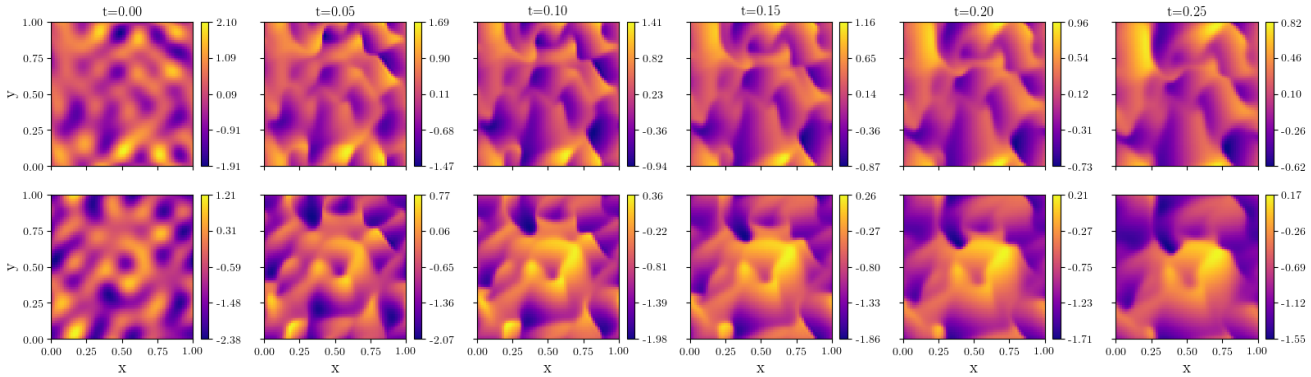


Figure: Randomly generated initial conditions for the 2D coupled Burgers' system. (Top to bottom) The x-velocity and y-velocity components. (Left to right) Different samples of the random initial condition.

2D Coupled Burgers' System



(a) Simulation 1



(b) Simulation 2

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

2D Coupled Burgers' System

$$T_{\Delta t}(\mathcal{U}^n, F_{\Delta x}) = \mathrm{u}^n + \Delta t \left[-0.5 \left(F_{\Delta x}(\mathrm{x}, \mathrm{u}^{n+1}) + F_{\Delta x}(\mathrm{x}, \mathrm{u}^n)\right)\right], \qquad (38)$$
$$F_{\Delta x}(\mathrm{x}, \mathrm{u}^n) = \mathrm{u}^n \cdot \nabla \mathrm{u}^n - \nu \Delta \mathrm{u}^n,$$

$$\mathrm{u}_x = \frac{1}{8\Delta x} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathrm{u}, \quad \mathrm{u}_y = \frac{1}{8\Delta x} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathrm{u}, \qquad (39)$$

$$\Delta \mathrm{u} = \frac{1}{2\Delta x^2} \begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \mathrm{u}, \qquad (40)$$

2D Coupled Burgers' System

where, the spatial gradients are approximated using Sobel filter 2D convolutions which are analogous to smoothed second-order accurate finite difference approximations [**?**].
Using the Sobel filter was found to increase the stability of training and significantly reduce spurious oscillations in the model's predictions.

AR-DenseED Deterministic Predictions

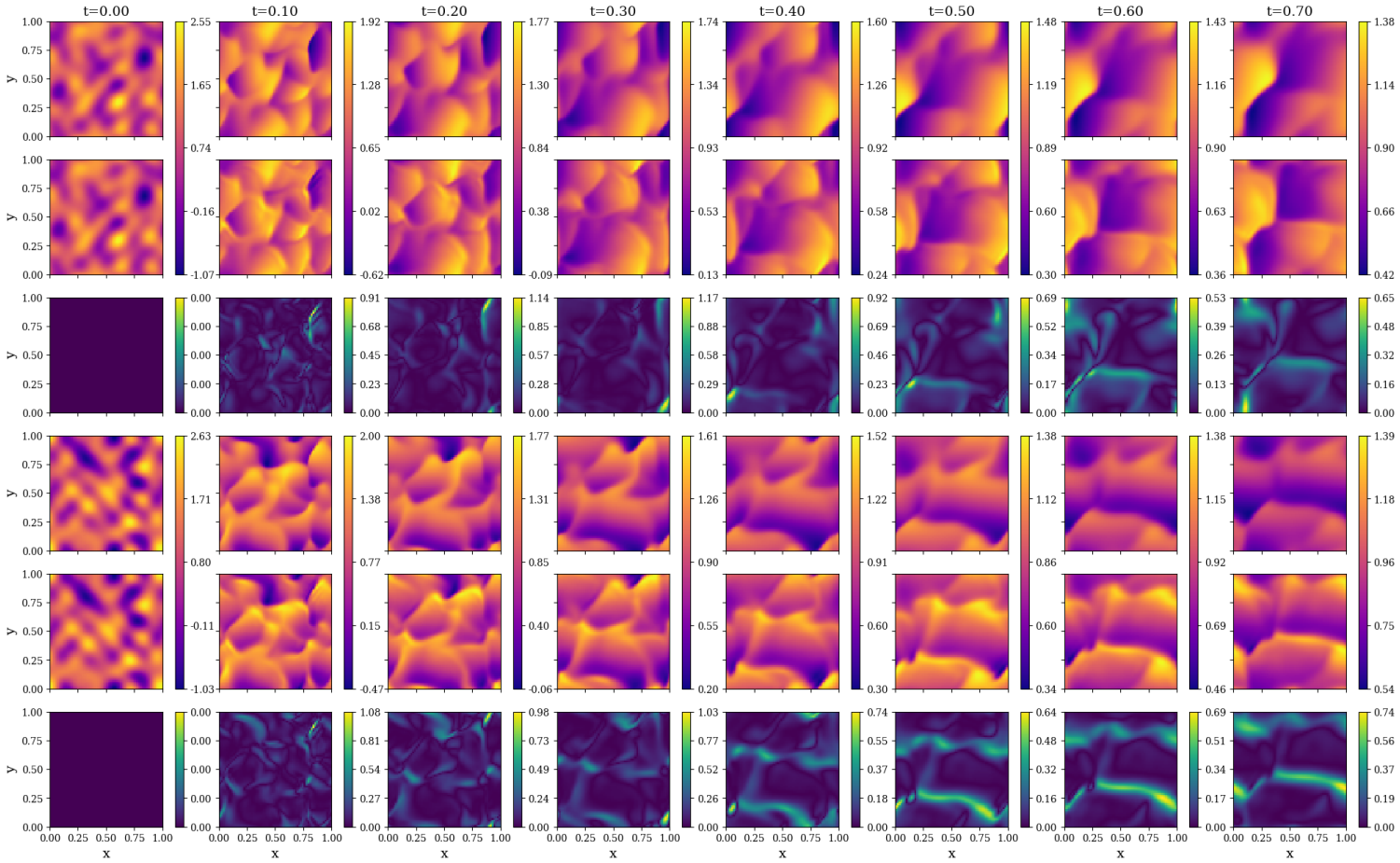


Figure: AR-DenseED predictions of a 2D coupled Burgers' test case. (Top to bottom) x-velocity FEM target solution, x-velocity AR-DenseED prediction, x-velocity L_1 error, y-velocity FEM target solution, y-velocity AR-DenseED prediction and y-velocity L_1 error.

AR-DenseED Deterministic Predictions

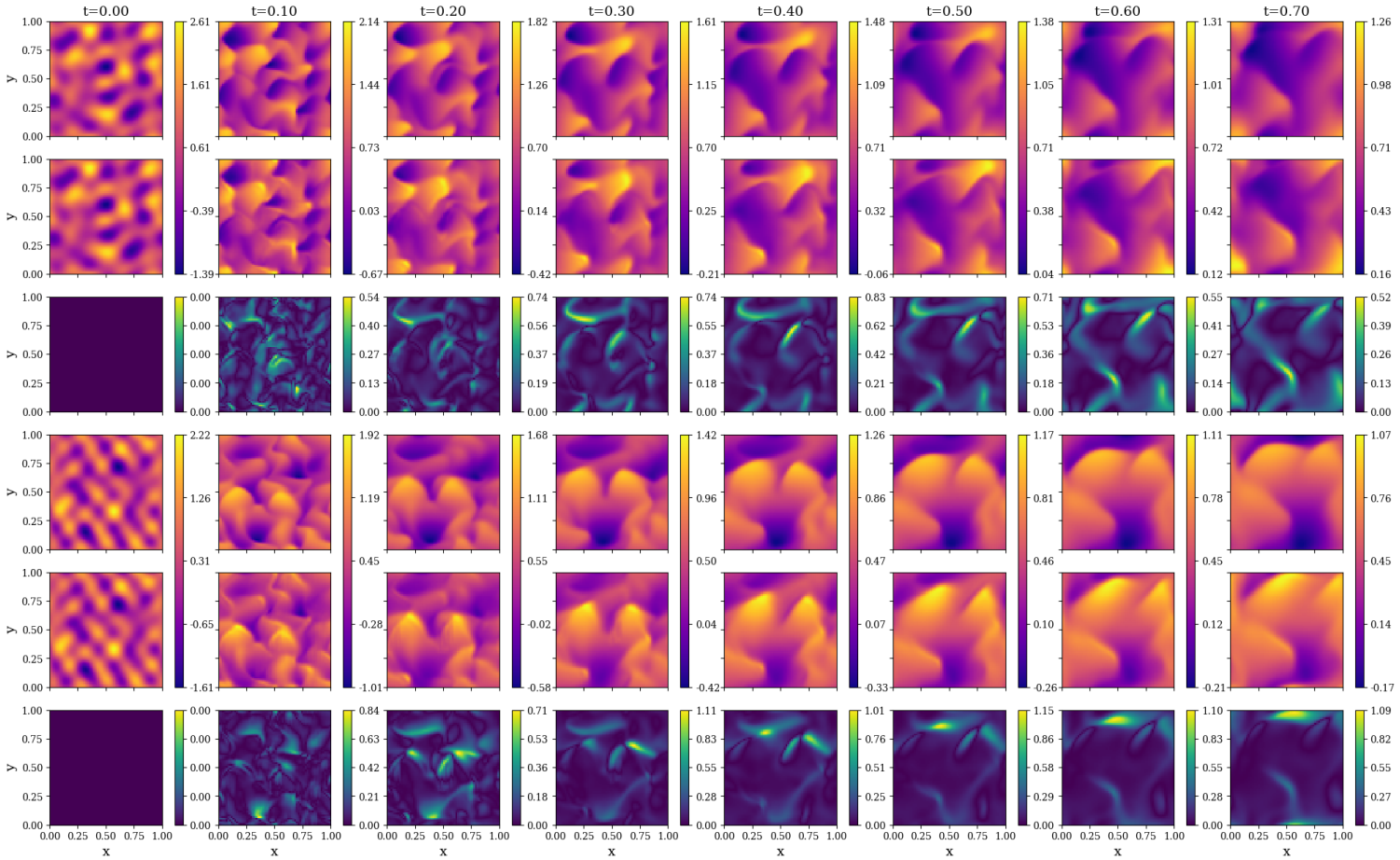


Figure: AR-DenseED predictions of a 2D coupled Burgers' test case. (Top to bottom) x-velocity FEM target solution, x-velocity AR-DenseED prediction, x-velocity L_1 error, y-velocity FEM target solution, y-velocity AR-DenseED prediction and y-velocity L_1 error.

AR-DenseED Deterministic Predictions

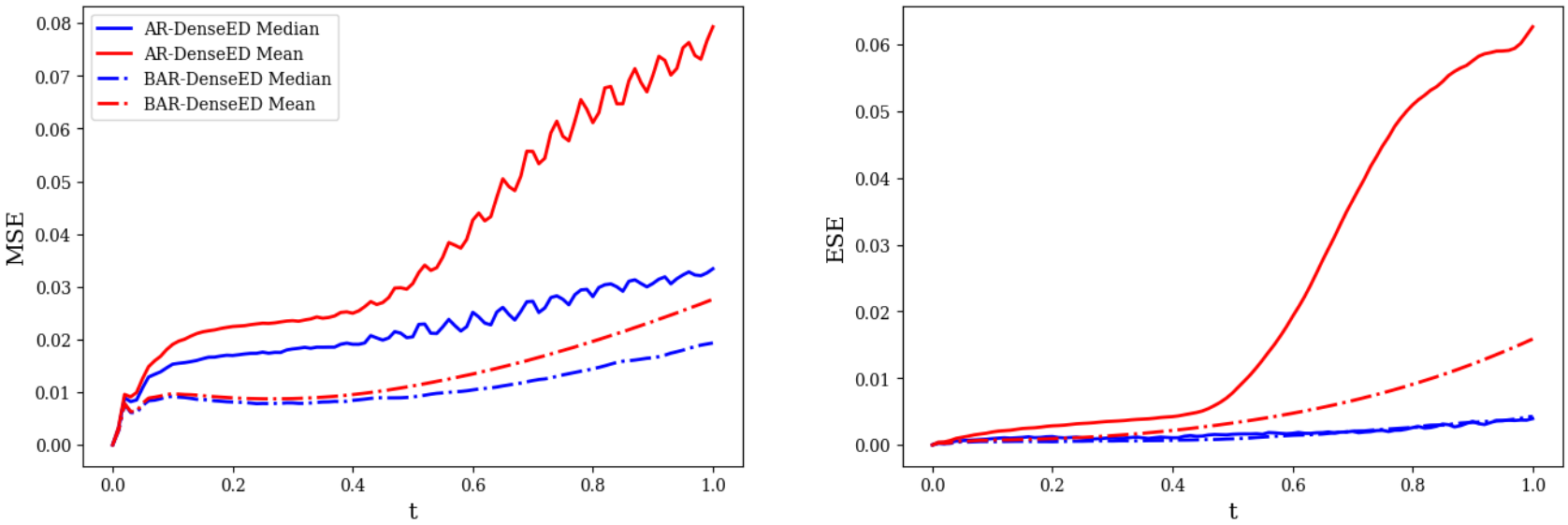


Figure: (Left to right) The mean square error (MSE) and energy squared error (ESE) as a function of time for a test set of 200 cases for the 2D coupled Burgers' system. The error of BAR-DenseED is calculated using the expected value of the predictive distribution approximating using 30 samples of the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

2D Coupled Burgers' System

Table: 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	Δt	Δx	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

BAR-DenseED Probabilistic Predictions

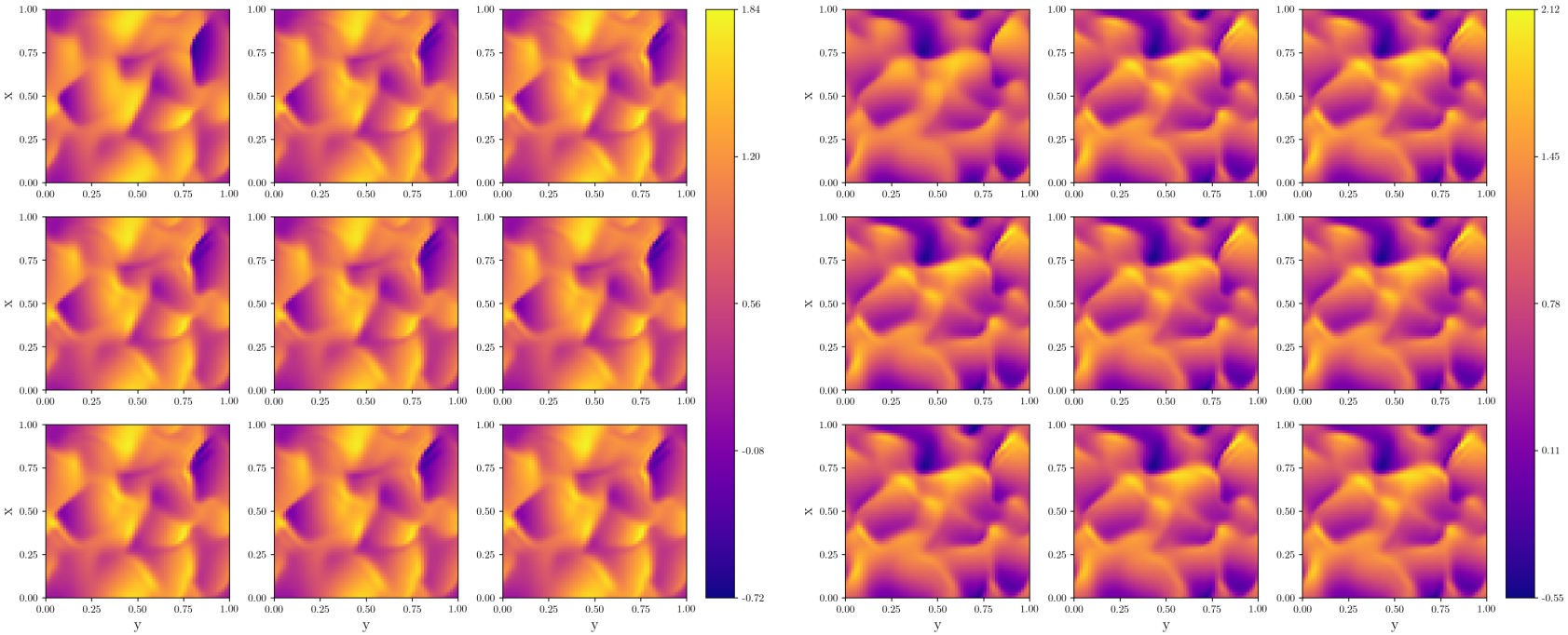


Figure: (Left to right) Samples of the x-velocity and y-velocity component from the posterior of BAR-DenseED approximated using SWAG at $t = 0.1$ for the 2D coupled Burgers' system. The top left in each grid is the simulated result using FEM.

BAR-DenseED Probabilistic Predictions

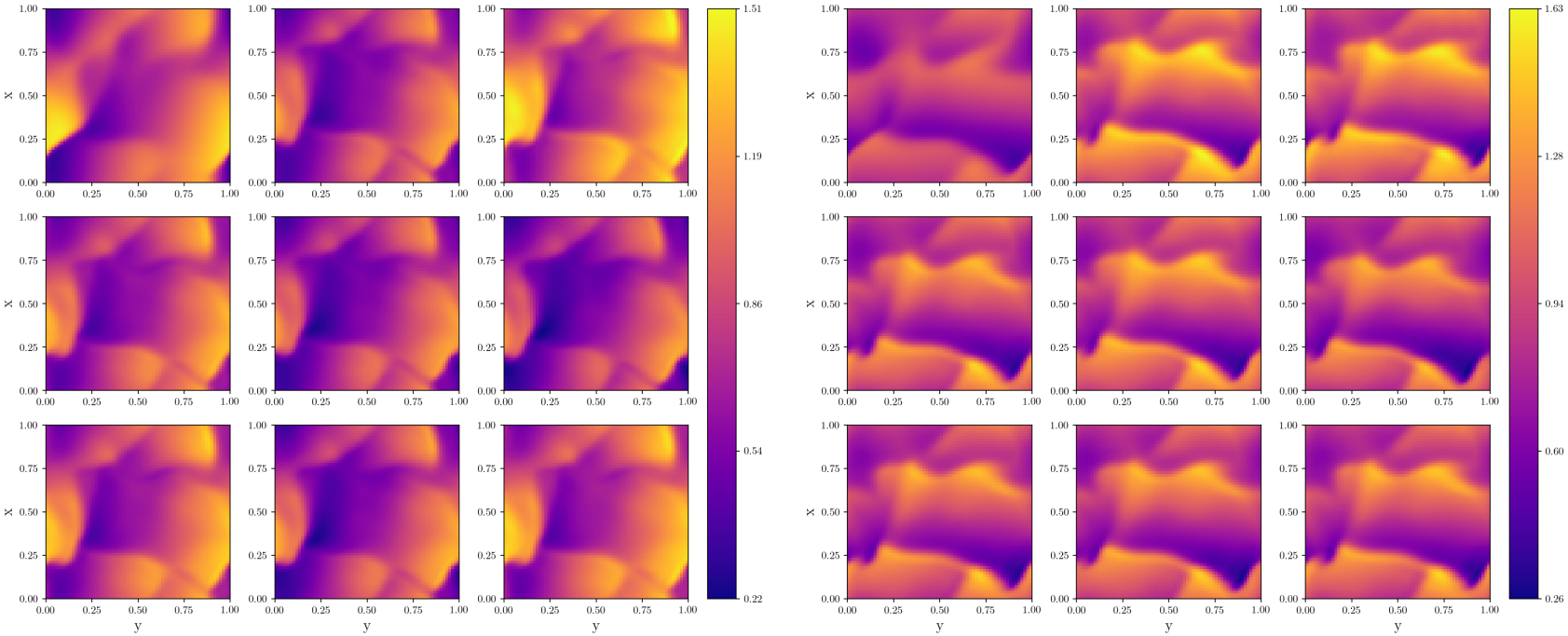


Figure: (Left to right) Samples of the x-velocity and y-velocity component from the posterior of BAR-DenseED approximated using SWAG at $t = 0.5$ for the 2D coupled Burgers' system. The top left in each grid is the simulated result using FEM.

BAR-DenseED Probabilistic Predictions

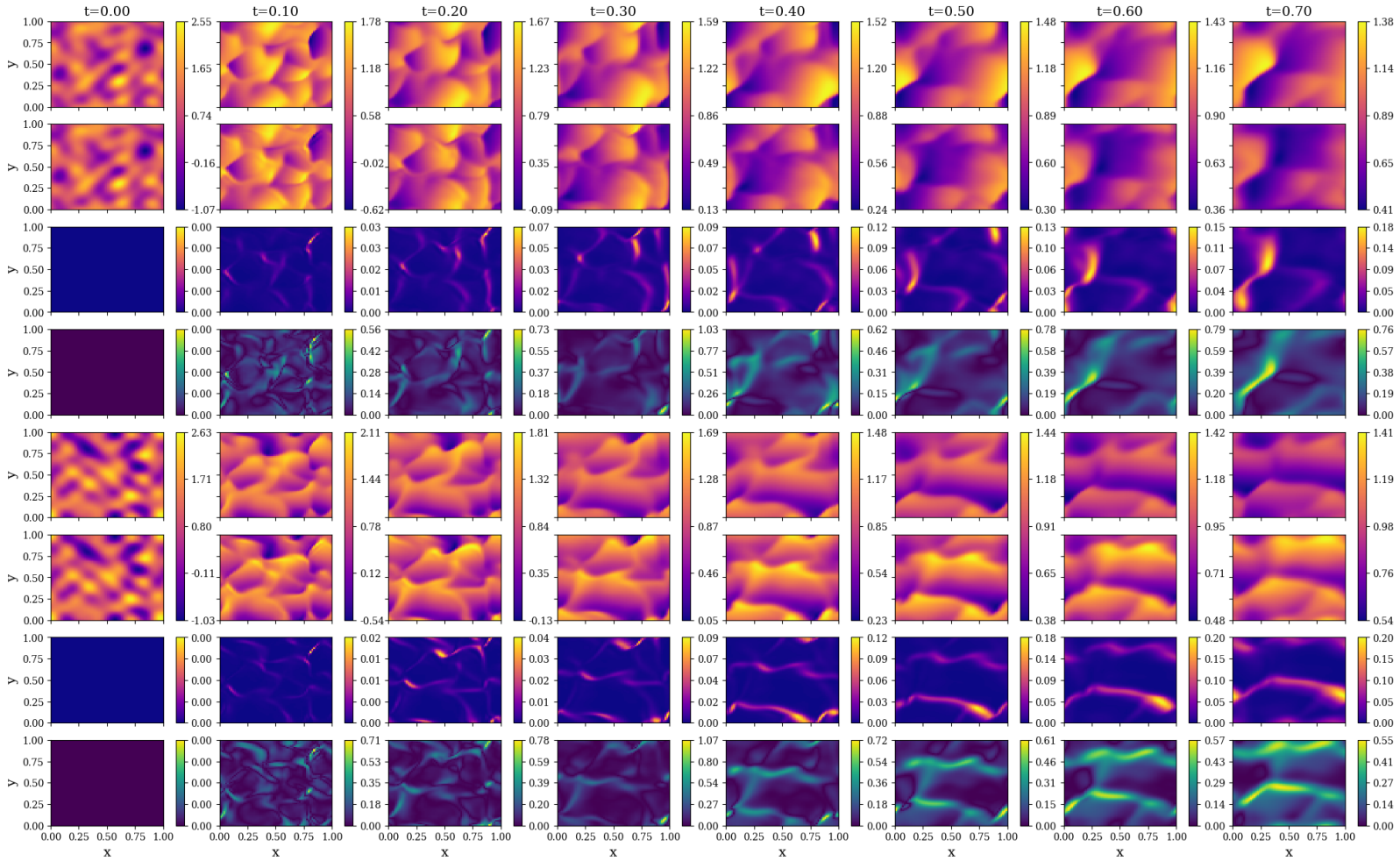


Figure: BAR-DenseED predictions for a 2D coupled Burgers' test case. (Top to bottom) x-velocity FEM target solution, BAR-DenseED expected response, BAR-DenseED variance, L_1 error between the target and expected values and

BAR-DenseED Probabilistic Predictions

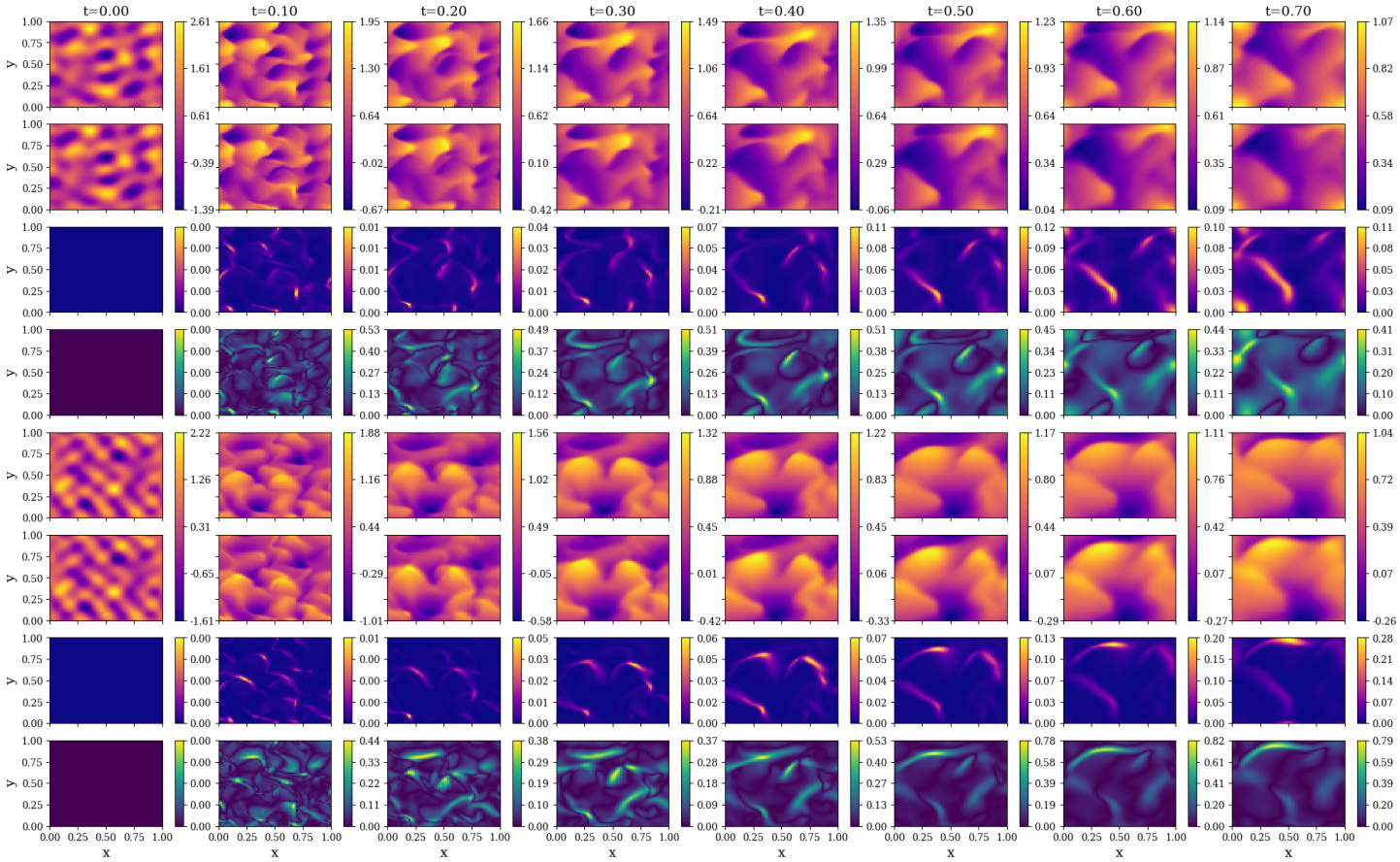


Figure: BAR-DenseED predictions for a 2D coupled Burgers' test case. (Top to bottom) x-velocity FEM target solution, BAR-DenseED expected response, BAR-DenseED variance, L_1 error between the target and expected values and similarly followed by the y-velocity component.

BAR-DenseED Probabilistic Predictions

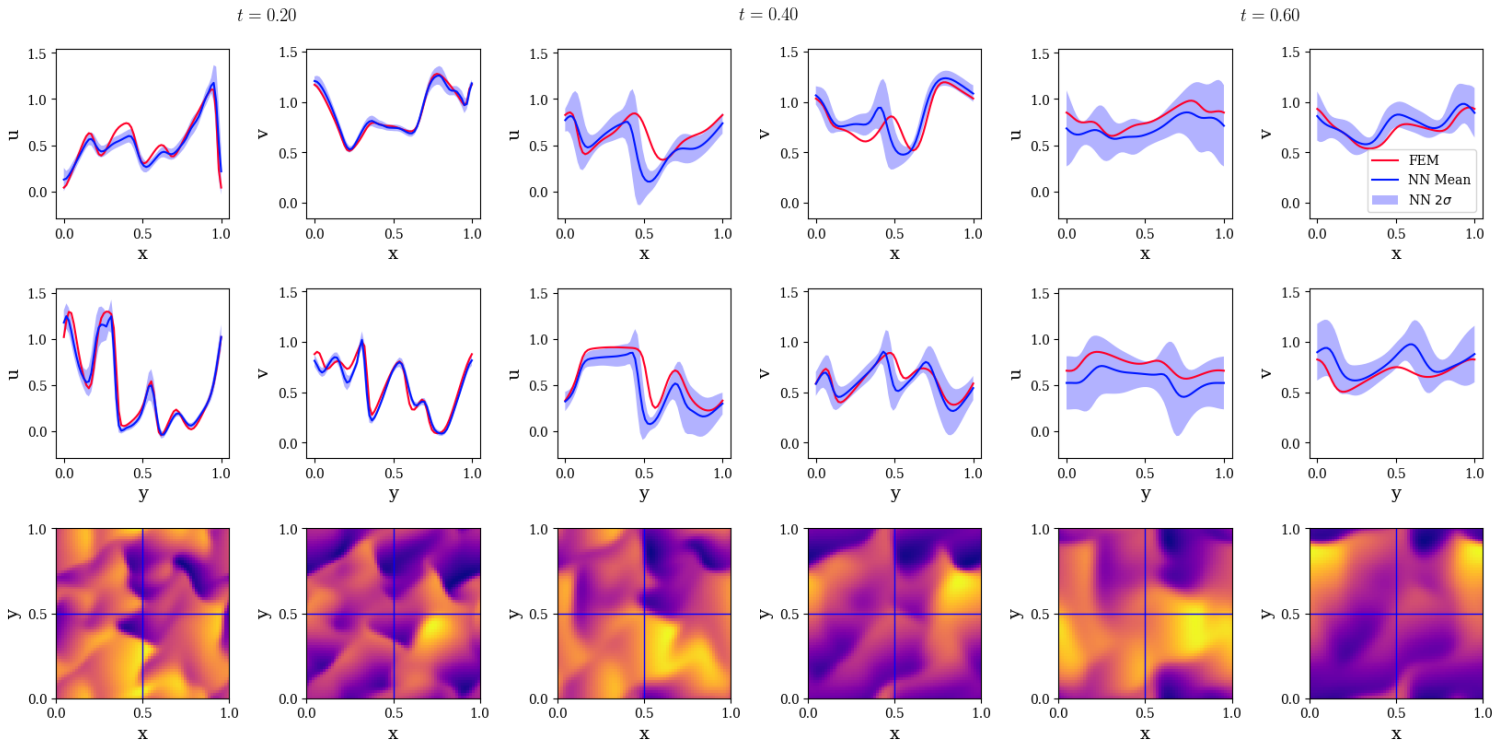


Figure: Instantaneous profiles of the finite element method (FEM) solver and BAR-DenseED (NN) predictive expectation and standard deviation at three various times of a test case. (Top to bottom) Horizontal profile at $y = 0.5$, vertical profile at $x = 0.5$ and target FEM contour with blue lines to show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

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).

Conclusion

- 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.

Limitations

- 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 draw-back 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.

Future Directions

- 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.

More Future Directions

- 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.