

Active-Learning-Driven Surrogate Modeling for Efficient Simulation of Parametric Nonlinear Systems

Harshit Kapadia* Lihong Feng[†] Peter Benner[‡]

*Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.
Corresponding author, Email: kapadia@mpi-magdeburg.mpg.de, ORCID: [0000-0003-3214-0713](https://orcid.org/0000-0003-3214-0713)

[†]Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.
Email: feng@mpi-magdeburg.mpg.de, ORCID: [0000-0002-1885-3269](https://orcid.org/0000-0002-1885-3269)

[‡]Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.
Email: benner@mpi-magdeburg.mpg.de, ORCID: [0000-0003-3362-4103](https://orcid.org/0000-0003-3362-4103)

Abstract: When repeated evaluations for varying parameter configurations of a high-fidelity physical model are required, surrogate modeling techniques based on model order reduction are desired. In absence of the governing equations describing the dynamics, we need to construct the parametric reduced-order surrogate model in a non-intrusive fashion. In this setting, the usual residual-based error estimate for optimal parameter sampling associated with the reduced basis method is not directly available. Our work provides a non-intrusive optimality criterion to efficiently populate the parameter snapshots, thereby, enabling us to effectively construct a parametric surrogate model. We consider separate parameter-specific proper orthogonal decomposition (POD) subspaces and propose an active-learning-driven surrogate model using kernel-based shallow neural networks, abbreviated as ActLearn-POD-KSNN surrogate model. To demonstrate the validity of our proposed ideas, we present numerical experiments using two physical models, namely Burgers' equation and shallow water equations. Both the models have mixed—convective and diffusive—effects within their respective parameter domains, with each of them dominating in certain regions. The proposed ActLearn-POD-KSNN surrogate model efficiently predicts the solution at new parameter locations, even for a setting with multiple interacting shock profiles.

Keywords: Active Learning, Data-driven Surrogate Modeling, Non-intrusive Model Order Reduction, Shallow Neural Networks, Parametric Dynamical Systems

Novelty statement:

- ActLearn-POD-KSNN: A novel surrogate modeling framework for parametric systems that is driven by actively learning the high-fidelity solution snapshots.
- A new non-intrusive error estimator based on the surrogate solution approximated in parameter-specific POD subspace which enables active learning.
- The active learning framework identifies areas in the parameter space with high variation in solution features and generates new snapshots in those regions, enhancing surrogate solution accuracy and refining the learning process iteratively.
- The parameter-specific adaptive POD subspaces makes our approach efficient for problems with mixed—convective and diffusive—phenomena, even in settings with multiple interacting shock profiles under convection domination.
- The offline training and the online querying is fast due to the shallow neural network architecture used in the construction of ActLearn-POD-KSNN.

a vast amount of training data is computationally expensive since it is obtained by repeated evaluations of a high-fidelity model. Alternately, if the solution data is collected from experimental measurements, conducting repeated experiments for a vast pool of parametric configurations could become practically infeasible. To alleviate this situation, we propose a surrogate modeling framework for parametric nonlinear dynamical systems that actively generates solution snapshots at new parameter locations by querying the high-fidelity model only when necessary. This enables us to iteratively arrive at a set of optimal training data corresponding to important parameter values. By doing this, we improve the surrogate model in an efficient fashion—by relaxing the vast data requirement to some extent and also providing an accuracy estimation of the constructed surrogate model.

Figure 1.1 provides an overview of our proposed active-learning-driven surrogate modeling paradigm. An initial coarse sampling of the parameter space is first considered, and the corresponding high-fidelity solution snapshots are generated and stored for a particular discrete time trajectory. With this initial set of snapshots, a data-driven surrogate model is trained. By designing an appropriate optimality criterion that helps us to pick new important parameter locations, we can actively improve the accuracy of the surrogate model. Such an optimality criterion can be designed by using the error caused in the surrogate approximation. However, in this work, we employ a new strategy to design the optimality criterion—constructing an error estimator from the parameter-specific POD-based solution approximations. We utilize a shallow neural network architecture equipped with an RBF kernel as the nonlinear activation to construct the error-estimate-based optimality criterion, as well as to construct the actively learned reduced-order surrogate model. The shallow architecture renders a fast offline training phase, as well as a fast online evaluation phase.

The proposed ActLearn-POD-KSNN surrogate iteratively detects locations in the parameter domain where the variation between solution features is high, and queries the FOM solver in those regions to generate new training snapshots. During this iterative procedure, a POD subspace for the new parameter sample is created and appropriately enriched in an adaptive fashion by using the error estimator. Such a POD subspace enrichment results in a varying number of POD bases between each of the parameter-specific subspaces, corresponding to different levels of energy (information) retention in each subspace. This enables us to choose an appropriate energy criterion for creating POD subspaces at newly queried parameter samples in the online phase such that the subspaces are expressive enough to provide a solution approximation up to a desired accuracy. For a setting that requires a multi-query parametric generation of solution, the proposed active learning framework becomes useful to build a surrogate model in an efficient fashion—by limiting the generation of the expensive FOM snapshots to an optimal set of parameter samples which still provide a sufficient exploration of the parameter space.

1.2 Relation to Previous Work

Estimating the error of the reduced approximations is crucial to assess their quality. For the reduced basis method [16, 27], an a posteriori error estimator is constructed by using the governing equations which then drives the greedy algorithm for constructing the reduced-order model. To reduce the offline time of the reduced basis method when a large training set of parameter samples are required, authors in [4] propose a RBF-interpolation-based surrogate for the error estimator. This reduces the numerous ROM evaluations that are required for the error estimator construction, while enabling sufficient exploration of the parameter space for the reduced basis method. However, efficient error estimation for non-intrusive MOR is still rarely discussed in the literature. In [32], a machine learning technique is applied to learn the error of the RBF-ROM in [33]. The error is the error of the approximate solution computed from the ROM and is a long vector of the FOM dimension. The error estimator is obtained via two ROMs: the ROM of the FOM and the ROM of the error, so that machine learning via Gaussian processes (GP) is done on the ROM of the error. However, the learning process needs to be implemented for each element of the error vector, i.e., one GP error model is learned for each element of the error vector.

In this work, we propose a non-intrusive error estimator, built using a KSNN which is equipped with RBF kernels, to assess the quality of a data-driven surrogate model that emulates the physics of a nonlinear parametric dynamical system. The error estimator is constructed by learning the

norm of the POD-approximate state-vector error using interpolation in the parameter-time space. No extra ROM for the error vector needs to be constructed as in [32]. Furthermore, a single error model is learned rather than quite a few GP models for all the elements of the error vector in [32]. The proposed error estimator is computationally much cheaper. The training data for the KSNN-based interpolation are the snapshots at certain samples of the parameter and time instances that can then be updated adaptively. We employ the RBF-MOR from [33] to show the robustness of the proposed error estimator. Beyond the RBF-MOR method in [33], we propose a greedy procedure in order to actively learn the POD-KSNN surrogate by adaptively and iteratively updating the snapshot data. This process iteratively improves the accuracy of the POD-KSNN surrogate model and updates the proposed error estimator at the same time. We further propose to use an energy criterion to identify different POD bases corresponding to different parameters. An adaptive technique for enriching the identified POD basis is proposed. This further significantly improves the accuracy of the RBF-MOR method from [33], especially for convection-dominated problems.

Active learning is also proposed in [5, 7, 21, 34]. The method in [5] proposes a greedy non-intrusive method that selects the parameters iteratively according to a proposed indicator. However, the indicator has nothing to do with the error of the approximate solution. The method in [7] proposes RBF interpolation for predicting the reduced state vector in the future time instances. Greedy algorithms are proposed to adaptively select the snapshots of the reduced state vector that are called the projected snapshots. Moreover, the projected snapshots are greedily selected according to a residual and a power function, rather than by error estimation of the approximate solution. The projected snapshots are selected from precomputed solution snapshots at a given set of time instances that need a lot of offline computations. The method applies only to non-parametric time-dependent cases.

In [21], Gaussian process regression (GPR) is proposed to learn the reduced state vector as a function of parameters. Active learning using deviation of GP as an indicator to iteratively enrich the training data (snapshots) that are then used for retraining GPR. Again, the deviation of GP cannot tell the error of the approximate solution computed from the proposed method there. Steady-state problems are only addressed in [21], and extension of the method to time-dependent problems is not straightforward. Similarly, in the most recent work [34], an error estimator based GPR is proposed to perform active learning by using single-time step snapshots of the parametric system states. Their method works for time-dependent problems, but its performance for models with mixed—convective and diffusive—effects is unclear.

The non-intrusive error estimator in our method is built from the error arising in a parameter-specific POD-approximation of the solution states. To the best of our knowledge, this is in contrast to all the previously proposed frameworks for active learning. As a result, our error estimator based non-intrusive optimality criteria allows us to actively learn important solution snapshots at new parameter locations, completely in the offline phase, without the need to repeatedly evaluate and retrain the entire surrogate model or non-intrusive ROM. This further reduces the computational burden. During the online phase, we do not need to evaluate the high-fidelity model in real-time, but can simply query the actively learned reduced-order surrogate model and obtain efficient approximation of the physics.

1.3 Organization

The remaining article is organized as follows. In [Section 2](#), the general setting for the parametric nonlinear dynamical system is introduced. This is followed by introducing the kernel-based shallow neural network (KSNN) which uses radial basis functions (RBFs) as the kernel functions. Finally, we formulate the POD-based data-driven surrogate model using KSNNs. Next, in [Section 3](#), we propose a non-intrusive optimality criterion based on an error estimator which can be used for actively learning any POD-based surrogate model. [Section 4](#) summarizes the novel ActLearn-POD-KSNN surrogate model by detailing its complete algorithm. Then, we provide detailed numerical experiments for models with mixed—convective and diffusive—physical phenomena in [Section 5](#). At the end, we draw some conclusions in [Section 6](#).

2 Data-Driven Surrogate Model for Parametric Systems

We can represent a full-order nonlinear dynamical system arising from the spatial discretization of a parametric partial differential equation as

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}, t; \boldsymbol{\mu}), \quad \mathbf{u}(0) = \mathbf{u}_0(\boldsymbol{\mu}), \quad t \in [0, T], \quad (2.1)$$

where $T \in \mathbb{R}^+$ denotes the final time; $\mathbf{u} \equiv \mathbf{u}(t, \boldsymbol{\mu})$ with $\mathbf{u} : [0, T] \times \mathcal{D} \rightarrow \mathbb{R}^N$ denotes the solution; $\mathbf{u}_0 : \mathcal{D} \rightarrow \mathbb{R}^N$ denotes the parameterized initial condition; $\boldsymbol{\mu} \in \mathcal{D} \subseteq \mathbb{R}^{N_\mu}$ denotes the parameters; and $\mathbf{f} : \mathbb{R}^N \times [0, T] \times \mathcal{D} \rightarrow \mathbb{R}^N$ denotes a nonlinear function. In this section, we provide a formulation of a reduced-order surrogate model which can be constructed directly from the high-fidelity solution snapshots of (2.1). The surrogate model is built by employing a series of neural networks with a shallow neural network architecture. The shallowness enables a fast offline training procedure as well as a rapid online querying of the surrogate model at new out-of-training parameter locations.

2.1 Kernel-Based Shallow Neural Network

We formulate the interpolation technique that will be used in this work as a radial kernel-based shallow neural network (KSNN). The network is as shown in Figure 2.1 with an input, a hidden, and an output layer. The input layer includes the data points \mathbf{x}_j where $j = 1, \dots, \ell$. The activation functions in the hidden layer are $\phi_i := \phi(\|\bar{\mathbf{x}} - \bar{\mathbf{x}}_i\|)$. The output f of the network is used to learn (approximate) a scalar-valued function $\hat{f}(\bar{\mathbf{x}})$. Mathematically, this can be expressed as follows,

$$\hat{f}(\bar{\mathbf{x}}) \approx f(\bar{\mathbf{x}}) = \sum_{i=1}^r w_i \phi_i(\|\bar{\mathbf{x}} - \bar{\mathbf{x}}_i\|). \quad (2.2)$$

where $\{w_i\}_{i=1}^r$ are the network weights; $\{\bar{\mathbf{x}}_i\}_{i=1}^r$ are the centers; and ϕ is a kernel function depending on the radial distance of input $\bar{\mathbf{x}}$ from a specified center $\bar{\mathbf{x}}_i$. As an example, Table 2.1 lists a few different radial basis kernels with the shape factor ϵ , and the radial distance d . Note that we indicate the vectors of input and centers with a bar at the top to highlight that they can attain any generic input and center values.

Name	Function
Gaussian	$e^{-(d/\epsilon)^2}$
Multi-quadric	$\sqrt{(d/\epsilon)^2 + 1}$
Inverse multi-quadric	$1/(\sqrt{(d/\epsilon)^2 + 1})$
Linear spline	d
Cubic spline	d^3
Quintic spline	d^5
Thin-plate spline	$d^2 \log(d)$

Table 2.1: List of radial basis kernels.

The radial kernel or basis functions operate on multivariate input data, which in turn reduces to a scalar function of the Euclidean norm of $(\bar{\mathbf{x}} - \bar{\mathbf{x}}_i)$. For exact interpolation, we take $\bar{\mathbf{x}}_i = \mathbf{x}_i$, $r = \ell$, and enforce $\hat{f}(\mathbf{x}_j) = f(\mathbf{x}_j)$ with $j = 1, \dots, \ell$ in (2.2). This reduces the training step to a linear system solve for the weights $\{w_i\}_{i=1}^r$. The coefficient matrix of the linear system is a distance matrix $D \in \mathbb{R}^{\ell \times \ell}$, where the entries of D are the kernel values evaluated at all the data points ($D_{j,i} = \phi_i(\mathbf{x}_j)$ with $i = j = 1, \dots, \ell$). After training, we can evaluate the scalar value \hat{f} at any new input $\bar{\mathbf{x}}$.

To learn a vector-valued function $\hat{\mathbf{y}}(\bar{\mathbf{x}}) \in \mathbb{R}^q$, the output layer needs to have width q . More

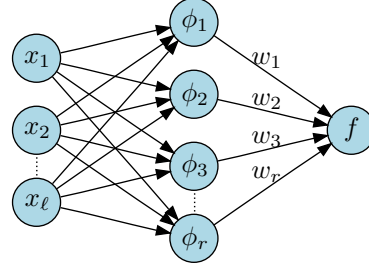


Figure 2.1: Illustration of a radial kernel-based shallow neural network with a scalar output.

precisely, the output $\mathbf{y}(\bar{\mathbf{x}})$ of the KSNN can be written as follows,

$$\hat{\mathbf{y}}(\bar{\mathbf{x}}) \approx \mathbf{y}(\bar{\mathbf{x}}) = [y_1(\bar{\mathbf{x}}), y_2(\bar{\mathbf{x}}), \dots, y_q(\bar{\mathbf{x}})]^T \quad (2.3)$$

$$= \left[\sum_{i=1}^r w_i^{(1)} \phi_i(\|\bar{\mathbf{x}} - \bar{\mathbf{x}}_i\|), \sum_{i=1}^r w_i^{(2)} \phi_i(\|\bar{\mathbf{x}} - \bar{\mathbf{x}}_i\|), \dots, \sum_{i=1}^r w_i^{(q)} \phi_i(\|\bar{\mathbf{x}} - \bar{\mathbf{x}}_i\|) \right]^T \quad (2.4)$$

where, for exact interpolation, we again take $\bar{\mathbf{x}}_i = \mathbf{x}_i$, $r = l$, and enforce $\hat{y}_k(\mathbf{x}_j) = y_k(\mathbf{x}_j)$ with $j = 1, \dots, l$ and $k = 1, \dots, q$. An illustration of such a network is provided in Figure 2.2.

All the weights $\{w_i^{(k)}\}_{i=1}^l$ corresponding to components $k = 1, \dots, q$ of $\mathbf{y}(\bar{\mathbf{x}})$ can be collected in a weight matrix W such that its entries are defined as below,

$$W_{i,k} := w_i^{(k)}. \quad (2.5)$$

To obtain the weights in each column of W , we need to solve a linear system with the coefficient matrix $D \in \mathbb{R}^{\ell \times \ell}$ during the training process. We avoid directly solving q linear systems and first perform a pivoted LU decomposition of the distance matrix D ,

$$D = PLU \quad (2.6)$$

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal elements, and U is an upper triangular matrix. When such a decomposition is available, each column of W can be obtained by simply performing forward and back substitutions. As a result, instead of performing q computations in $\mathcal{O}(l^3)$, we just perform one LU factorization in $\mathcal{O}(l^3)$ followed by q operations in $\mathcal{O}(l^2)$. The benefit of such an adjacent training approach for vectorial interpolation becomes more prominent as the number of data points $\{\mathbf{x}_j\}_{j=1}^l$ increases, as well as when the size

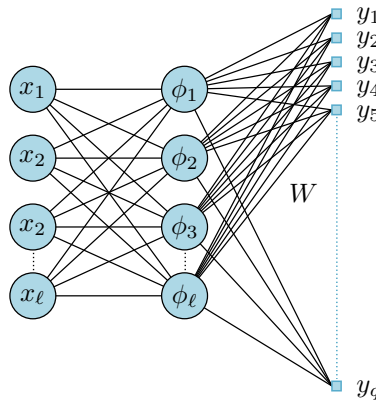


Figure 2.2: Illustration of a radial kernel-based shallow neural network with a vectorial output. For interpolation, the width of the hidden layer is the same as the number of inputs ℓ .

of the vectors to be interpolated becomes significantly large, i.e., $q \gg l$. We employ the multi-quadratic kernel function (which is not positive definite) to conduct all our numerical experiments, so LU decomposition is used. But when a symmetric positive definite kernel is considered, we can use the Cholesky decomposition instead of LU decomposition and obtain a further reduction in computational complexity from $\frac{2}{3}\ell^3$ to $\frac{1}{3}\ell^3$ during the first step of matrix factorization while training any KSNN with a vectorial output.

Remark 2.1 (Preserving positivity when interpolating error values). *In our work, we repeatedly build or retrain KSNNs to construct interpolants for the norm of the relative error caused in the POD-approximate solution, which will be discussed at length in Section 3. While interpolating these small error values, it could happen that the result is a negative value close to zero, which would be nonphysical. This phenomenon is dependent on the distribution of the training data, as well as on the shape factor's (ϵ) value. To ensure the positivity of the error values, we modify the training data by taking the logarithm of all the error values used for training. After querying the network, we need to take the exponential of the result, to obtain the correct (positive) interpolated error value.*

2.2 POD-KSNN Surrogate Model

Consider that we have the solution snapshots along discrete time trajectories $\{t_0, t_1, \dots, t_{N_t}\}$ with $t_0 = 0$ and $t_{N_t} = T$. The snapshots can be collected in matrices $U(\boldsymbol{\mu}_i)$ sized $N \times (N_t + 1)$ corresponding to each parameter sample $\boldsymbol{\mu}_i$ for $i \in \{1, \dots, m\}$,

$$U(\boldsymbol{\mu}_i) = [\mathbf{u}(t_0, \boldsymbol{\mu}_i) \mid \mathbf{u}(t_1, \boldsymbol{\mu}_i) \mid \dots \mid \mathbf{u}(t_{N_t}, \boldsymbol{\mu}_i)]. \quad (2.7)$$

We follow a two-step interpolation approach [33] to construct the non-intrusive reduced-order surrogate model. In the first step, by building $(N_t + 1)$ KSNNs (refer to (2.4)), we interpolate the snapshot data $U(\boldsymbol{\mu}_i)$ in the parameter space corresponding to all time-instances t_j with $j \in \{0, \dots, N_t\}$. The KSNNs' construction and training is done in the offline phase, whereas in the online phase, they are queried at a new parameter instance $\boldsymbol{\mu}^*$. The result is an estimation of the snapshot matrix corresponding to any new $\boldsymbol{\mu}^*$,

$$U^I(\boldsymbol{\mu}^*) = [\mathcal{I}_{t_0}^\mu(\boldsymbol{\mu}^*) \mid \mathcal{I}_{t_1}^\mu(\boldsymbol{\mu}^*) \mid \dots \mid \mathcal{I}_{t_{N_t}}^\mu(\boldsymbol{\mu}^*)]. \quad (2.8)$$

In this case, \mathbf{y} in (2.3) corresponds to each $\mathcal{I}_{t_j}^\mu$ in (2.8) (an N -dimensional vector), $\bar{\mathbf{x}}_i$ in (2.4) are $\boldsymbol{\mu}_i$, and $\bar{\mathbf{x}}$ is $\boldsymbol{\mu}^*$ when the KSNN is queried. Each column of U^I is obtained by performing a vectorial interpolation. So, $\{\mathcal{I}_{t_j}^\mu\}_{j=0}^{N_t}$ denote $(N_t + 1)$ KSNNs interpolating the snapshots in the parameter space. Due to this reason, we interchangeably refer to the KSNNs as interpolants. It is crucial to note that once the KSNNs $\{\mathcal{I}_{t_j}^\mu\}_{j=0}^{N_t}$ are constructed, we do not need to store any snapshot matrices $U(\boldsymbol{\mu}_i)$ that were used to train the KSNNs.

The optimal linear subspace spanned by the approximate snapshot data for $\boldsymbol{\mu}^*$ can be computed via proper orthogonal decomposition (POD). In practice, we can compute the POD bases $\tilde{\Phi}(\boldsymbol{\mu}^*) := (\tilde{\phi}_1(\boldsymbol{\mu}^*), \tilde{\phi}_2(\boldsymbol{\mu}^*), \dots, \tilde{\phi}_s(\boldsymbol{\mu}^*))$ either using the singular value decomposition or (for large-scale problems) using the method of snapshots, when N is very large and $N_t \ll N$ [29]. The interpolated snapshots at $\boldsymbol{\mu}^*$ can then be written as a linear combination of the bases $\tilde{\Phi}(\boldsymbol{\mu}^*)$. By

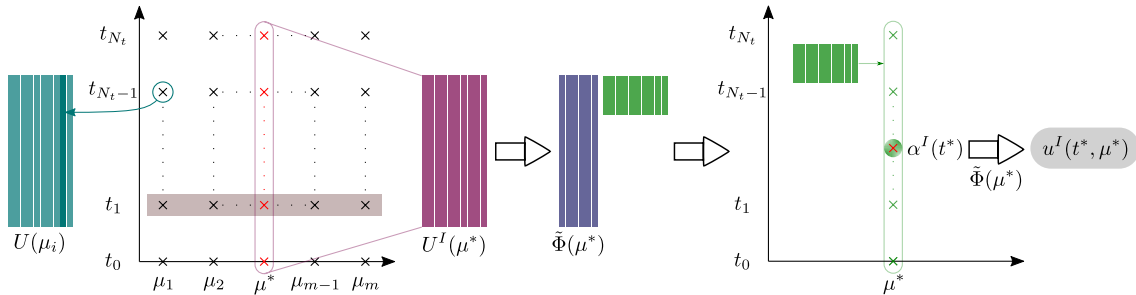


Figure 2.3: Framework for POD-KSNN data-driven surrogate modeling.

collecting all the coefficients for such a linear combination in a matrix $A(\boldsymbol{\mu}^*)$, the interpolated snapshots $U^I(\boldsymbol{\mu}^*)$ can be represented as,

$$U^I(\boldsymbol{\mu}^*) = \tilde{\Phi}(\boldsymbol{\mu}^*)A(\boldsymbol{\mu}^*), \quad (2.9)$$

$$A(\boldsymbol{\mu}^*) = [\alpha^0(\boldsymbol{\mu}^*) \mid \alpha^1(\boldsymbol{\mu}^*) \mid \dots \mid \alpha^{N_t}(\boldsymbol{\mu}^*)], \quad (2.10)$$

where $\alpha^j \in \mathbb{R}^s$ is the vector of coefficients at time t_j . For the complete POD space, s equals to the number of nonzero singular values of $U^I(\boldsymbol{\mu}^*)$. However, in practice, usually the POD space is truncated. In that scenario, s corresponds to the number of retained singular values.

To obtain the approximation of the solution corresponding to $\boldsymbol{\mu}^*$ at a new time instance $t^* \in [0, T]$, a second interpolation step is carried out in the time domain by building and training an additional KSNN (refer to (2.4)). We construct an interpolant $\mathcal{I}_{\mu^*}^t$ for the reduced coordinates $\alpha^j(\boldsymbol{\mu}^*)$ in the time domain,

$$\alpha^I(t^*) := \mathcal{I}_{\mu^*}^t(t^*). \quad (2.11)$$

In this case, \mathbf{y} in (2.3) corresponds to α^I in (2.11) (an s -dimensional vector), $\bar{\mathbf{x}}_i$ in (2.4) is t_j , and $\bar{\mathbf{x}}$ is t^* when the interpolant is queried. The surrogate approximation of the solution is obtained as follows,

$$\mathbf{u}_s(t^*, \boldsymbol{\mu}^*) = \tilde{\Phi}(\boldsymbol{\mu}^*) \alpha^I(t^*). \quad (2.12)$$

The complete POD-KSNN surrogate model is summarized in Figure 2.3. We adhere to a two-step interpolation approach to divide the function complexity between space and time domains, thereby allowing us to work with multiple reduced-sized KSNNs. The network size (layer width) is directly related to the number of centers or data points under consideration. More data points lead to a wider KSNN, whose training will require a larger linear system solve for the weights. As the number of centers increases, the memory required for the linear system solve goes up considerably. This is due to the quadratic dependence of the required memory on the number of centers. The training could become infeasible in such a scenario. However, the two-step interpolation strategy enables us to isolate the centers between space and time, thereby relaxing the total permissible center count.

Remark 2.2 (Extension to system of parametric PDEs). *One approach is to prepare different snapshot matrices for all the solution components that are present in the system of equations. Then prepare separate KSNNs for all components to interpolate between the snapshots in the parameter domain. Later, in the second step, we create POD subspaces, individually, for each of the components. An additional KSNN for each component interpolates the reduced coordinates in the time domain. We employ this methodology in our numerical experiments with shallow water equations.*

An alternate approach is to merge the solutions for all the components and form one big snapshot matrix corresponding to each parameter instance. This allows us to proceed in the same fashion as detailed above in this section. Undertaking this approach of first merging the component snapshots and then applying POD leads to a single ROM, but with larger reduced size.

3 Active Learning for POD-Based Data-Driven Surrogates

We are concerned with dynamical systems which are parametric in nature. For this setting, one typically requires a substantial amount of training data at several parameter samples to create a good reduced-order surrogate model. Our aim is to be efficient and choose a set of optimal training samples corresponding to different parameters, from a vast pool of parameters. However, there is no trivial notion of optimality. We address this by proposing a non-intrusive error estimator as an optimality criterion. This is further used to actively create the training or snapshot data and leverage the most out of the POD-KSNN surrogate.

3.1 Non-Intrusive Optimality Criterion

To assess the quality of the POD-based data-driven surrogate solution in Section 2, we require a way to estimate the error in its approximate solution, in comparison with the full-order (or

high-fidelity) solution. There are two types of errors induced while constructing and deploying the POD-based surrogate: the error caused due to restricting the solution corresponding to each parameter sample in an (active) linear subspace obtained via POD, and the amalgamation of errors arising from the chosen interpolation or regression technique. The total error $\mathcal{E} \in \mathbb{R}^N$ in the spatial domain for some parameter $\boldsymbol{\mu}$ corresponding to a time instance t can be represented as

$$\mathcal{E}(t, \boldsymbol{\mu}) = \mathcal{E}_{POD}(t, \boldsymbol{\mu}) + \mathcal{E}_I(t, \boldsymbol{\mu}), \quad (3.1)$$

where $\mathcal{E}_{POD} \in \mathbb{R}^N$ represents the POD projection error, and $\mathcal{E}_I \in \mathbb{R}^N$ represents the total interpolation error when the surrogate model is evaluated.

To understand the additive decomposition of the total error $\mathcal{E}(t, \boldsymbol{\mu})$, let us look more concretely at the error in the surrogate solution $\mathbf{u}_s(t, \boldsymbol{\mu})$,

$$\mathcal{E}(t, \boldsymbol{\mu}) := \mathbf{u}(t, \boldsymbol{\mu}) - \mathbf{u}_s(t, \boldsymbol{\mu}). \quad (3.2)$$

This can be written in the following fashion for the POD-KSNN surrogate model:

$$\mathcal{E} = (\mathbf{u} - \Phi\Phi^\top \mathbf{u}) + (\Phi\Phi^\top \mathbf{u} - \Phi\Phi^\top \mathbf{u}^I) + (\Phi\Phi^\top \mathbf{u}^I - \tilde{\Phi}\tilde{\Phi}^\top \mathbf{u}^I) + (\tilde{\Phi}\tilde{\Phi}^\top \mathbf{u}^I - \mathbf{u}_s). \quad (3.3)$$

Here, $\Phi(\boldsymbol{\mu}) = (\phi_1(\boldsymbol{\mu}), \phi_2(\boldsymbol{\mu}), \dots, \phi_s(\boldsymbol{\mu}))$ are the bases obtained by performing POD of the solution \mathbf{u} ; $\mathbf{u}^I(t, \boldsymbol{\mu}) = \mathcal{I}_t^\mu(\boldsymbol{\mu})$ is an approximation of \mathbf{u} obtained via interpolating the solutions in the parameter domain using KSNNs, as shown in (2.8); $\tilde{\Phi}(\boldsymbol{\mu}) = (\tilde{\phi}_1(\boldsymbol{\mu}), \tilde{\phi}_2(\boldsymbol{\mu}), \dots, \tilde{\phi}_s(\boldsymbol{\mu}))$ are the bases obtained by performing POD of the approximate solution \mathbf{u}^I .

By following (2.12), we can write the surrogate solution as $\mathbf{u}_s(t, \boldsymbol{\mu}) = \tilde{\Phi}(\boldsymbol{\mu})\alpha^I(t)$. Here, an approximation of the reduced solution coordinates, $\alpha^I(t)$, is obtained by interpolating $\{\alpha^j(\boldsymbol{\mu})\}_{j=0}^{N_t}$ from (2.10) in the time domain using a KSNN, i.e., $\alpha^I(t) = \mathcal{I}_\mu^t(t)$. This allows us to write (3.3) in the following way:

$$\mathcal{E} = (\mathbf{u} - \Phi\Phi^\top \mathbf{u}) + (\Phi\Phi^\top \mathbf{u} - \Phi\Phi^\top \mathbf{u}^I) + (\Phi\Phi^\top \mathbf{u}^I - \tilde{\Phi}\tilde{\Phi}^\top \mathbf{u}^I) + (\tilde{\Phi}\tilde{\Phi}^\top \mathbf{u}^I - \tilde{\Phi}\alpha^I). \quad (3.4)$$

The first term in (3.4) arises due to the retention of only the leading s POD modes. This results in a parameter-specific linear subspace that captures most of the solution, but not in its entirety. We refer to this omitted contribution as the POD approximation error and define it as,

$$\mathcal{E}_{POD} := (\mathbf{u} - \Phi\Phi^\top \mathbf{u}). \quad (3.5)$$

The second and third terms in (3.4) arise from the solution approximation \mathbf{u}^I , due to interpolation in the parameter domain. More precisely, the second term accounts for the solution error resulting from the reduced representation of the approximate solution obtained by projection onto the true solution bases Φ . Whereas, the third term accounts for the solution error caused due to projection of the approximate solution \mathbf{u}^I onto the bases $\tilde{\Phi}$ instead of Φ , which is obtained from a POD of \mathbf{u}^I . Finally, the fourth term in (3.4) provides the solution error caused because of an approximation of the reduced coordinates α^I via interpolation in the time domain. As a result, we can define the total interpolation error in the following way,

$$\mathcal{E}_I := (\Phi\Phi^\top \mathbf{u} - \Phi\Phi^\top \mathbf{u}^I) + (\Phi\Phi^\top \mathbf{u}^I - \tilde{\Phi}\tilde{\Phi}^\top \mathbf{u}^I) + (\tilde{\Phi}\tilde{\Phi}^\top \mathbf{u}^I - \mathbf{u}_s). \quad (3.6)$$

From (3.4)–(3.6) it is clear that we can decompose the total error \mathcal{E} in an additive fashion as represented in (3.1). Now, consider the norm of the total error in the spatial domain,

$$\|\mathcal{E}(t, \boldsymbol{\mu})\| = \|\mathcal{E}_{POD}(t, \boldsymbol{\mu}) + \mathcal{E}_I(t, \boldsymbol{\mu})\|. \quad (3.7)$$

Upon application of the triangle inequality, we obtain

$$\|\mathcal{E}(t, \boldsymbol{\mu})\| \leq \|\mathcal{E}_{POD}(t, \boldsymbol{\mu})\| + \|\mathcal{E}_I(t, \boldsymbol{\mu})\|, \quad (3.8)$$

$$\epsilon(t, \boldsymbol{\mu}) \leq \epsilon_{POD}(t, \boldsymbol{\mu}) + \epsilon_I(t, \boldsymbol{\mu}), \quad (3.9)$$

where ϵ , ϵ_{POD} , and ϵ_I represents the spatial norms of the total, POD projection, and interpolation errors respectively. More precisely, they are defined as $\epsilon := \|\mathcal{E}\|$, $\epsilon_{POD} := \|\mathcal{E}_{POD}\|$, and $\epsilon_I := \|\mathcal{E}_I\|$.

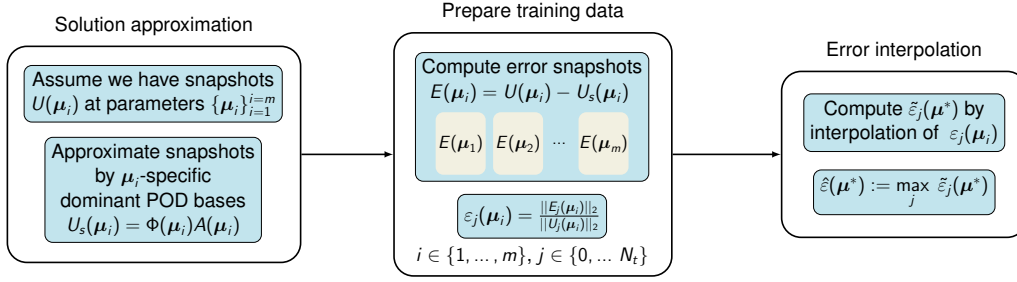


Figure 3.1: Procedure to non-intrusively estimate the error at new parameter locations.

During the construction of the POD-KSNN surrogate model, the KSNN interpolation procedure reproduces the training data exactly which results in $\epsilon_I = 0$ for all the training parameter samples. This makes ϵ_{POD} the bound for the total error ϵ . As a result, we can construct an estimator for the error ϵ caused in the surrogate solution by interpolating between the errors ϵ_{POD} corresponding to all the training parameter samples. Afterward, this error estimator can be queried at new parameter samples, providing an approximation for ϵ at any out-of-training parameters. Through our numerical experiments in [Section 5](#), we see that employing such an error estimate is a reasonable strategy to drive the active learning procedure. The key benefit is that we can carry out active learning entirely in the offline phase without the need to repeatedly evaluate the POD-KSNN surrogate model at the training parameter samples.

Let us now dive into the details of the error estimator construction, which we will use as the optimality criterion. Consider a successful parameter sampling, followed by snapshot data collection of all the chosen parameter points corresponding to the same time horizon by simulating the high-fidelity model. For instance, following the setup for [\(2.7\)](#), we have the snapshot matrices $U(\boldsymbol{\mu}_i)$. Let us consider a POD approximation of the solution snapshots,

$$U_{POD}(\boldsymbol{\mu}_i) = \Phi(\boldsymbol{\mu}_i)A(\boldsymbol{\mu}_i) \quad (3.10)$$

with $\Phi(\boldsymbol{\mu}_i) \in \mathbb{R}^{N_x \times r_i}$ and $A(\boldsymbol{\mu}_i) \in \mathbb{R}^{r_i \times (N_t + 1)}$. Here, $r_i \leq \min\{N_x, (N_t + 1)\}$ denotes the level of POD space truncation for each parameter $\boldsymbol{\mu}_i$.

The error of the POD approximate solution at the parameters $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_m\}$ is given by

$$E(\boldsymbol{\mu}_i) = U(\boldsymbol{\mu}_i) - U_{POD}(\boldsymbol{\mu}_i). \quad (3.11)$$

Note that $E(\boldsymbol{\mu}_i)$ is also the POD-KSNN surrogate error because $U_{surrogate} = U_{POD} = U_s$ for all $\boldsymbol{\mu}_i$ (since they are the training parameter samples). We avoid interpolating $E(\boldsymbol{\mu}_i)$ entry-wise at a new parameter location as the computational cost for doing that is equivalent to interpolating the high-dimensional snapshots. Our motivation is to alleviate this computational burden, but at the same time, have an informative indication for the error. This is accomplished by taking the relative norm of the solution error $E_j(\boldsymbol{\mu}_i) \in \mathbb{R}^{N_x}$ at each time instance t_j . In our experiments, we have tested using l^2 , l^1 , and l^∞ norms. We noticed similar qualitative results for all the norms. Depending on the problem setting, one can be preferred over the other, for instance, physical systems prone to advective effects might benefit from l^1 norm usage. As a generic choice, we use the l^2 norm for our discussion because the considered test problems in [Section 5](#) have mixed diffusive and convective effects. We denote the relative norm by,

$$\epsilon_j(\boldsymbol{\mu}_i) = \frac{\|E_j(\boldsymbol{\mu}_i)\|_2}{\|U_j(\boldsymbol{\mu}_i)\|_2}, \quad (3.12)$$

where $U_j(\boldsymbol{\mu}_i)$ denotes the j^{th} column (corresponding to t_j) of the snapshot matrix for $\boldsymbol{\mu}_i$.

The relative error values $\epsilon_j(\boldsymbol{\mu}_i)$ are used to train $(N_t + 1)$ KSNNs and obtain their weight values $w_i^{(j)}$. The interpolated relative l^2 error at time t_j for any new parameter value $\boldsymbol{\mu}^*$ becomes

$$\tilde{\epsilon}_j(\boldsymbol{\mu}^*) = \sum_{i=1}^m w_i^{(j)} \phi_i^{(j)} (\|\boldsymbol{\mu}^* - \boldsymbol{\mu}_i\|). \quad (3.13)$$

This can be written compactly as a vector with entries corresponding to each time instance,

$$\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{\mu}^*) = [\tilde{\varepsilon}_0(\boldsymbol{\mu}^*) \mid \tilde{\varepsilon}_1(\boldsymbol{\mu}^*) \mid \dots \mid \tilde{\varepsilon}_{N_t}(\boldsymbol{\mu}^*)]. \quad (3.14)$$

The final error estimate is taken to be the maximum interpolated relative error in time, given by

$$\hat{\varepsilon}(\boldsymbol{\mu}^*) := \|\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{\mu}^*)\|_\infty = \max(|\tilde{\varepsilon}_0(\boldsymbol{\mu}^*)|, |\tilde{\varepsilon}_1(\boldsymbol{\mu}^*)|, \dots, |\tilde{\varepsilon}_{N_t}(\boldsymbol{\mu}^*)|). \quad (3.15)$$

The entire procedure to compute the non-intrusive error estimator is summarized in [Figure 3.1](#).

3.2 Active Learning Framework

The intention of the active learning procedure is to enrich the snapshot data in a fashion that is most beneficial for the reduced-order surrogate model. In other words, each enrichment of the training snapshots lead to an optimal or near-optimal improvement of the approximate dynamics. The motivation is similar to the greedy procedure used for the reduced basis method [16]. However, in our setting we do not have access to the first principle models, so we cannot leverage the equations to decide the choice of new parameter samples for efficient training of the non-intrusive ROM. Instead, we utilize the non-intrusive error estimator as the optimality criterion to enable active learning.

We initialize the parameter set P with a coarse sampling of the parameter space \mathcal{D} . Additionally, a second set P^* is prepared which holds all the candidate parameter values that could be included in set P as the active learning progresses. Consider the coarsest initial sampling set P , i.e., $P = \{\boldsymbol{\mu}_i \mid i \in I\}$ with index set $I = \{1, 2\}$. And the candidate set P^* is composed of a fine sampling in $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)$, given by

$$P^* = \{\bar{\boldsymbol{\mu}}_1, \bar{\boldsymbol{\mu}}_2, \dots, \bar{\boldsymbol{\mu}}_q\} \text{ with } \bar{\boldsymbol{\mu}}_j \neq \boldsymbol{\mu}_i; \quad i = 1, 2; \quad j = 1, \dots, q. \quad (3.16)$$

The high-fidelity solution is computed for all the parameters in set P . This is followed by performing a POD approximation (3.10) for the snapshot matrix at each $\boldsymbol{\mu}_i$ in P . The level of POD space truncation r_i for each $\boldsymbol{\mu}_i \in P$ is obtained by maintaining a constant energy criterion $\eta(\boldsymbol{\mu}_i)$ defined at $\boldsymbol{\mu}_i$ as

$$\eta(\boldsymbol{\mu}_i) = 1 - \frac{\sum_{k=1}^{k=r_i} (\sigma_k^{(i)})^2}{\sum_{k=1}^{k=s_i} (\sigma_k^{(i)})^2}. \quad (3.17)$$

Here, $\sigma_k^{(i)}$ with $k = 1, \dots, s_i$, are the nonzero singular values obtained from SVD of the snapshot matrix $U(\boldsymbol{\mu}_i)$.

Using the high-fidelity and POD approximate solutions, error snapshots for the parameters in set P are computed,

$$E^{(1)}(P) := \{E(\boldsymbol{\mu}_1), E(\boldsymbol{\mu}_2)\} \quad (3.18)$$

where $E(\boldsymbol{\mu}_i)$ is given by (3.11) and the superscript in $E^{(1)}$ denotes the first iteration of the active learning loop, i.e. $iter = 1$. An error estimator is constructed for the parameters in the candidate set P^* ,

$$\hat{\varepsilon}^{(1)}(P^*) := \{\hat{\varepsilon}(\bar{\boldsymbol{\mu}}_1), \hat{\varepsilon}(\bar{\boldsymbol{\mu}}_2), \dots, \hat{\varepsilon}(\bar{\boldsymbol{\mu}}_q)\}, \quad (3.19)$$

where $\hat{\varepsilon}(\cdot)$ is computed by (3.15).

The parameter sample corresponding to the maximal value of the error estimate is chosen ($iter = 1$),

$$\bar{\boldsymbol{\mu}}^{(iter)} = \operatorname{argmax}_{\bar{\boldsymbol{\mu}}' \in P^*} \hat{\varepsilon}(\bar{\boldsymbol{\mu}}'). \quad (3.20)$$

If $\hat{\varepsilon}(\bar{\boldsymbol{\mu}}^{(iter)}) > tol$, the set P is extended by including the chosen parameter $\bar{\boldsymbol{\mu}}^{(iter)}$. The candidate set is also updated, $P^* = P^* \setminus \bar{\boldsymbol{\mu}}^{(iter)}$. Here, tol is a predefined tolerance level that we intend to achieve. In other words, we terminate the active learning process as soon as the error estimator reaches a value which is less than the target tolerance.

Next, we compute the high-fidelity snapshots $U(\bar{\boldsymbol{\mu}}^{(iter)})$, and the POD approximation error at $\bar{\boldsymbol{\mu}}^{(iter)}$,

$$E(\bar{\boldsymbol{\mu}}^{(iter)}) = U(\bar{\boldsymbol{\mu}}^{(iter)}) - U_{POD}(\bar{\boldsymbol{\mu}}^{(iter)}), \quad (3.21)$$

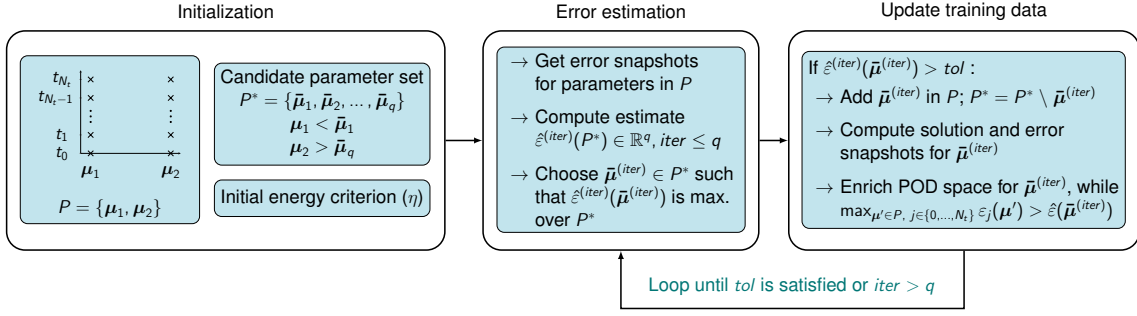


Figure 3.2: Procedure to actively learn the POD-KSNN surrogate model by greedy refinement of the high-fidelity solution snapshots along the parameter domain.

where U_{POD} is constructed by adhering to the energy criterion η in (3.17). The subspace of the POD bases for $\bar{\mu}^{(iter)}$ is further enriched if the maximal relative-error-norm at any time instance in the extended set P corresponds to $\bar{\mu}^{(iter)}$. The POD basis is incremented until the following holds true:

$$\hat{\varepsilon}(\bar{\mu}^{(iter)}) < \max_{\mu' \in P, j \in \{0, \dots, N_t\}} \varepsilon_j(\mu'). \quad (3.22)$$

This step essentially makes sure that the POD subspace of the newly selected parameter $\bar{\mu}^{(iter)}$ is at least as expressive as all the other parameter-specific POD subspaces corresponding to samples in P . Moreover, due to this procedure, we can obtain an update of the energy criterion $\eta(\bar{\mu}^{(iter)})$, which is later used during the online phase to construct an accurate POD subspace for any newly queried parameter μ^* . A more detailed discussion about obtaining and using such an updated energy criterion is provided in Section 4.

With the necessary POD space enrichment for $\bar{\mu}^{(iter)}$ and the updated parameter set P , the error snapshots $E^{(iter)}(P)$ are collected to start the next iteration. This is used as the training data for the interpolation procedure to construct the new error estimator $\hat{\varepsilon}^{(iter)}(P^*)$. The next parameter $\bar{\mu}^{(iter)}$ is then greedily picked using (3.20), followed by an adaptive refinement of the POD space using (3.22). In this fashion, we iteratively expand the solution snapshots by greedy selection of new parameters along with their POD space adaptation, until the tolerance tol is satisfied. The complete active learning procedure is outlined in Figure 3.2.

Remark 3.1 (Choice of candidate parameter set P^*). *The preparation of P^* is not trivial, in fact, a decent sampling procedure needs to be maintained while choosing the candidate samples for P^* . As the parameter values range over several orders of magnitude for the physical models presented in our numerical experiments, we consider a uniform sampling of the logarithm of the entire parameter space. This ensures a reasonable selection of parameters through all of the parameter space.*

Remark 3.2 (Extension to system of parametric PDEs). *We prepare different snapshot matrices corresponding to all the solution components in the system of equations, i.e., $\{U^{(k)}\}_{k=1}^q$ for q components. This is followed by forming the error matrices $\{E^{(k)}\}_{k=1}^q$, and then proceeding to apply the aforementioned active learning procedure, but now obtaining component-wise error estimators $\{(\hat{\varepsilon}^{(iter)})^{(k)}\}_{k=1}^q$ by following (3.19) for each component. The new parameter sample is picked based on the maximal value of the average between all the component-wise error estimates. So, (3.20) takes the following form:*

$$\bar{\mu}^{(iter)} = \operatorname{argmax}_{\bar{\mu}' \in P^*} \left(\frac{1}{q} \sum_{k=1}^q (\hat{\varepsilon}^{(iter)})^{(k)}(\bar{\mu}') \right). \quad (3.23)$$

Similar to the scalar equation setting, now, if $\frac{1}{q} \sum_{k=1}^q (\hat{\varepsilon}^{(iter)})^{(k)}(\bar{\mu}') > tol$, the set P is extended by adding $\bar{\mu}^{(iter)}$ to it. The candidate set is also updated, $P^* = P^* \setminus \bar{\mu}^{(iter)}$. The POD bases are incremented until the component-averaged error estimate value at the newly selected parameter $\bar{\mu}^{(iter)}$ is less than the maximal component-averaged relative-error-norm values among all the

parameter samples already present in P . So, (3.22) takes the following form:

$$\frac{1}{q} \sum_{k=1}^q (\hat{\varepsilon}^{(iter)})^{(k)}(\bar{\boldsymbol{\mu}}^{(iter)}) < \max_{\boldsymbol{\mu}' \in P, j \in \{0, \dots, N_t\}} \frac{1}{q} \sum_{k=1}^q (\varepsilon_j)^{(k)}(\boldsymbol{\mu}'). \quad (3.24)$$

4 ActLearn-POD-KSNN Surrogate Model

We summarize the complete methodology to construct and deploy the ActLearn-POD-KSNN reduced-order surrogate model in this section. Algorithm 1 details the complete offline phase. To iteratively construct the non-intrusive error estimator, a new KSNN (refer to (2.2)) is automatically built, trained, and queried at steps 5 and 18 of the algorithm. Solution snapshots corresponding to new parameter values are actively selected, and the parameter set P is updated during the offline phase. Since the POD subspace is refined adaptively during the active learning process, the relative energy corresponding to the retained POD modes could be different for the selected parameters in the final pool of P . We denote it by $\hat{\eta}(\boldsymbol{\mu}_i)$ for each $\boldsymbol{\mu}_i \in P$. The minimum energy criteria would be

$$\hat{\eta} = \min_{\boldsymbol{\mu}' \in P} \hat{\eta}(\boldsymbol{\mu}'). \quad (4.1)$$

We adhere to this updated energy criterion, $\hat{\eta}$, to construct the POD subspace for any new parameter during the online phase.

Algorithm 1 Offline phase: Constructing the ActLearn-POD-KSNN surrogate

Input: Initial parameter set P , snapshots $U(\boldsymbol{\mu}_i)$ for $\boldsymbol{\mu}_i \in P$, candidate parameter set P^* , initial energy criterion η , tolerance value (*tol*) to terminate the active learning loop, $iter = 1$.

Output: Updated parameter set P , energy criterion $\hat{\eta}$, KSNN surrogates $\{\mathcal{I}_{t_j}^\mu\}_{j=0}^{N_t}$.

- 1: Based on η , calculate the POD truncation level r_i for all $\boldsymbol{\mu}_i \in P$ such that (3.17) holds.
 - 2: Compute nonlinear reduced bases, i.e., the truncated parameter-specific POD subspaces $\Phi(\boldsymbol{\mu}_i)$.
 - 3: Construct the POD approximate solutions $U_{POD}(\boldsymbol{\mu}_i)$.
 - 4: Obtain the error snapshots $E(\boldsymbol{\mu}_i)$ following (3.11) and (3.18).
 - 5: Compute the error estimate $\hat{\varepsilon}(\bar{\boldsymbol{\mu}}_j)$ using (3.15) and (3.19), where $\bar{\boldsymbol{\mu}}_j \in P^*$.
 - 6: Pick a parameter $\bar{\boldsymbol{\mu}}^{(iter)}$ from P^* following (3.20). Store $\mathcal{E}^{(iter)} = \hat{\varepsilon}(\bar{\boldsymbol{\mu}}^{(iter)})$.
 - 7: **while** $\mathcal{E}^{(iter)} > tol$ **do**
 - 8: Extend P by including $\bar{\boldsymbol{\mu}}^{(iter)}$. Update the candidate set, $P^* = P^* \setminus \bar{\boldsymbol{\mu}}^{(iter)}$.
 - 9: Get the solution snapshots $U(\bar{\boldsymbol{\mu}}^{(iter)})$ from a high-fidelity model or experiments.
 - 10: Based on η , calculate the POD truncation level for $\bar{\boldsymbol{\mu}}^{(iter)}$.
 - 11: Compute the truncated POD subspace $\Phi(\bar{\boldsymbol{\mu}}^{(iter)})$. Construct $U_{POD}(\bar{\boldsymbol{\mu}}^{(iter)})$.
 - 12: Compute the error snapshots $E(\bar{\boldsymbol{\mu}}^{(iter)})$.
 - 13: **while** $\max_{\boldsymbol{\mu}' \in P, j \in \{0, \dots, N_t\}} \varepsilon_j(\boldsymbol{\mu}') > \mathcal{E}^{(iter)}$ **do**
 - 14: Enhance $\Phi(\bar{\boldsymbol{\mu}}^{(iter)})$ by incrementing the truncation level for $\bar{\boldsymbol{\mu}}^{(iter)}$, i.e., adding a new basis.
 - 15: Recompute $U(\bar{\boldsymbol{\mu}}^{(iter)})$ and $E(\bar{\boldsymbol{\mu}}^{(iter)})$.
 - 16: **end while**
 - 17: $iter = iter + 1$
 - 18: Build the error estimate $\hat{\varepsilon}(\boldsymbol{\mu}_j)$ using (3.15) for each $\boldsymbol{\mu}_j \in P^*$.
 - 19: Following (3.20), pick a parameter $\bar{\boldsymbol{\mu}}^{(iter)}$ from P^* . Store $\mathcal{E}^{(iter)} = \hat{\varepsilon}(\bar{\boldsymbol{\mu}}^{(iter)})$.
 - 20: **end while**
 - 21: Compute $\hat{\eta}$ following (4.1).
 - 22: Using $U(\boldsymbol{\mu}_i)$ ($\boldsymbol{\mu}_i \in P$) as training data, construct $\{\mathcal{I}_{t_j}^\mu\}_{j=0}^{N_t}$ by building and training $(N_t + 1)$ KSNNs (2.4). As per (2.8), they will be queried in the online phase at new parameter location $\boldsymbol{\mu}^*$.
-

Building upon the successful active learning procedure carried out during the offline phase, in the online phase, we acquire the surrogate solution at a new time t^* and parameter $\boldsymbol{\mu}^*$ value as described in Algorithm 2. We do not query the full-order model in the online phase. The bases $\tilde{\Phi}(\boldsymbol{\mu}^*)$ is obtained from the snapshot approximation $U^I(\boldsymbol{\mu}^*)$ by performing its SVD to extract the

leading r^* POD basis. Consequently, we have to do computations with the cost in $\mathcal{O}(NN_t^2)$, where $N_t < N$. The truncation level r^* is obtained by maintaining the energy criterion $\hat{\eta}$. If r^* can be learned without relying on the energy criterion, the cost of evaluating the bases $\Phi(\boldsymbol{\mu}^*)$ can be reduced to $\mathcal{O}(NN_t \log(r^*))$ with negligible loss of accuracy by leveraging randomized SVD [14]. This will be included in our subsequent work. Although the online computational cost depends on the dimension of the full-order model, N , the online phase is fast, as evidenced by the numerical results presented in Section 5.

Algorithm 2 Online phase: Querying the ActLearn-POD-KSNN surrogate

Input: New parameter $\boldsymbol{\mu}^*$, new time t^* , energy criterion $\hat{\eta}$, KSNN surrogates $\{\mathcal{I}_{t_j}^\mu\}_{j=0}^{N_t}$.

Output: Surrogate solution at $(t^*, \boldsymbol{\mu}^*)$.

- 1: Evaluate $U^I(\boldsymbol{\mu}^*)$, the KSNN approximate snapshots for $\boldsymbol{\mu}^*$, on the training time grid via (2.4) and (2.8).
 - 2: Compute POD bases $\tilde{\Phi}(\boldsymbol{\mu}^*)$ by deciding the truncation level from the energy criterion $\hat{\eta}$.
 - 3: Compute reduced coordinates $A(\boldsymbol{\mu}^*)$ with (2.9), after projecting $U^I(\boldsymbol{\mu}^*)$ onto $\tilde{\Phi}(\boldsymbol{\mu}^*)$ via (2.10).
 - 4: Build and train a KSNN (2.4), $\mathcal{I}_{\boldsymbol{\mu}^*}^t$, by using $A(\boldsymbol{\mu}^*)$ as the training data.
 - 5: Obtain a vector of approximate reduced coordinates, $\alpha^I(t^*)$, at t^* by evaluating $\mathcal{I}_{\boldsymbol{\mu}^*}^t$ via (2.4) and (2.11).
 - 6: Compute the surrogate solution $u_s(t^*, \boldsymbol{\mu}^*)$ using (2.12).
-

5 Numerical Results

We validate the proposed active learning framework with POD-KSNN reduced-order surrogate models by performing numerical experiments on two test cases. The first test case is the Burgers' equation, which is parametrized by the viscosity. It is known to develop an advecting shock in finite time, even when starting with smooth solutions, given a low enough viscosity value. Additionally, we also parametrize the initial condition with viscosity. The second test case is the shallow water equation, which is parametrized by the viscosity and the mean-free path. It is used to model the flow under a pressure surface in a fluid. Its solution comprises two waves moving with opposing characteristic speeds. Due to a periodic boundary condition, the traveling waves repeatedly interact with each other over time. In the remainder of this section, we provide details about both the problem setups and our results for them.

5.1 Burgers' Equation

We consider the following viscous Burgers' equation in a 1D spatial domain with Dirichlet boundary conditions:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad (5.1)$$

where the solution is denoted by u , changing with time $t \geq 0$, space $x \in [0, 1]$, and viscosity ν . The viscosity can be viewed as a parameter that controls the competing effects of diffusion and advection, resulting in a varying solution behavior for a wide range of ν . We choose the following initial condition, such that the solution space at $t = 0$ is also parameter-dependent:

$$u(x, 0) = \frac{x}{1 + \sqrt{\frac{1}{\kappa'}} \exp\left(\text{Re} \frac{x^2}{4}\right)}; \quad \kappa' = \exp(\text{Re}/8), \quad \text{Re} = 1/\nu. \quad (5.2)$$

Instead of simulating the equation using a numerical discretization technique, we opt to utilize the exact solution by converting (5.1) into a parabolic nonlinear PDE through the application of the Cole-Hopf transformation [8]. The exact solution takes the following form:

$$u(x, t) = \frac{\frac{x}{t+1}}{1 + \sqrt{\frac{t+1}{\kappa'}} \exp\left(\text{Re} \frac{x^2}{4t+4}\right)}; \quad \kappa' = \exp(\text{Re}/8), \quad \text{Re} = 1/\nu, \quad (5.3)$$

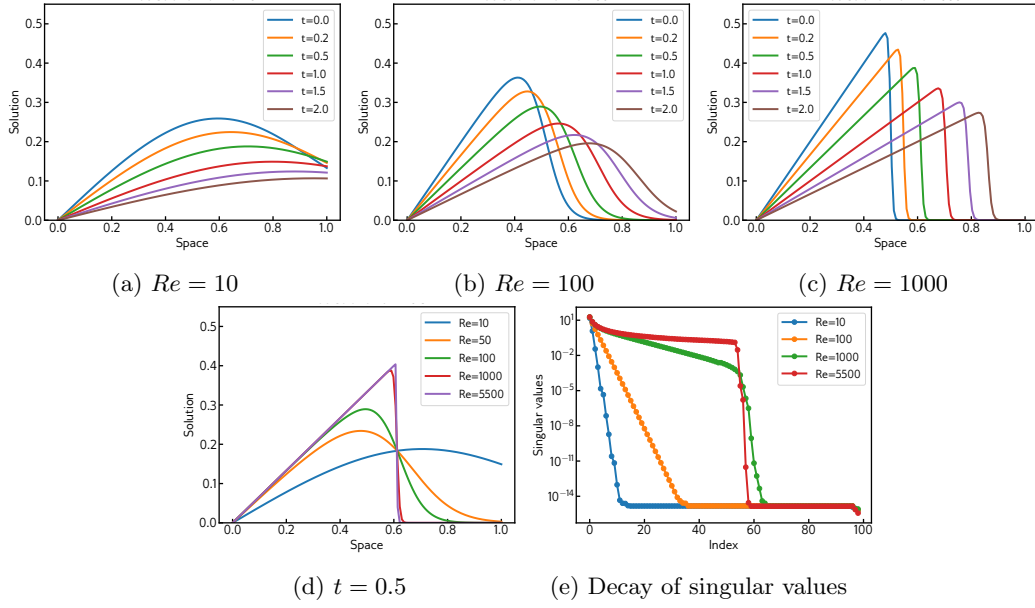


Figure 5.1: Burgers' equation: Evolution of exact solution over time $t \in [0, 2]$ in (a), (b), (c) for various Reynolds numbers; Comparison of solution at $t = 0.5$ in (d) for different Reynolds number; Comparison of the singular value decay in (e) for different Reynolds numbers.

where we refer to Re as the Reynolds number, denoting the inverse of viscosity.

Figures 5.1a to 5.1d show the behavior of the solution for selected time instances $t \in [0, 2]$ corresponding to various Reynolds numbers. To highlight the competing diffusive and convective effects of the equation, we select representative Re values from different parametric regimes. Upon singular value decomposition of the snapshot matrices, we observe a progressively slow singular value decay in Figure 5.1e as we go towards higher Re , which is due to the convective effects dominating the flow.

For our experiments, the spatial domain has 150 grid nodes, and the time-domain is $t \in [0, 2]$ with 100 time-steps. The parametric range for the Reynolds number is taken from 10 to 5500. The total number of discrete parameters we consider when accounting for both the candidate set P^* and the parameter set P are 100. These 100 values of Re are picked by uniformly dividing the logarithmic ν values ($\log_{10}(\frac{1}{5500})$ and $\log_{10}(\frac{1}{10})$) into 99 intervals, ensuring a decent pool of parameters.

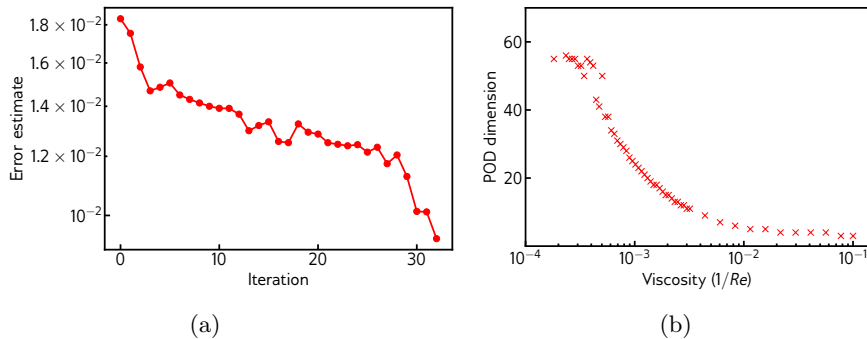


Figure 5.2: Active learning for Burgers' equation: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer Algorithm 1) varying with greedy iterations of the active learning process; (b) shows the final POD subspace dimension for each of the chosen parameter samples.

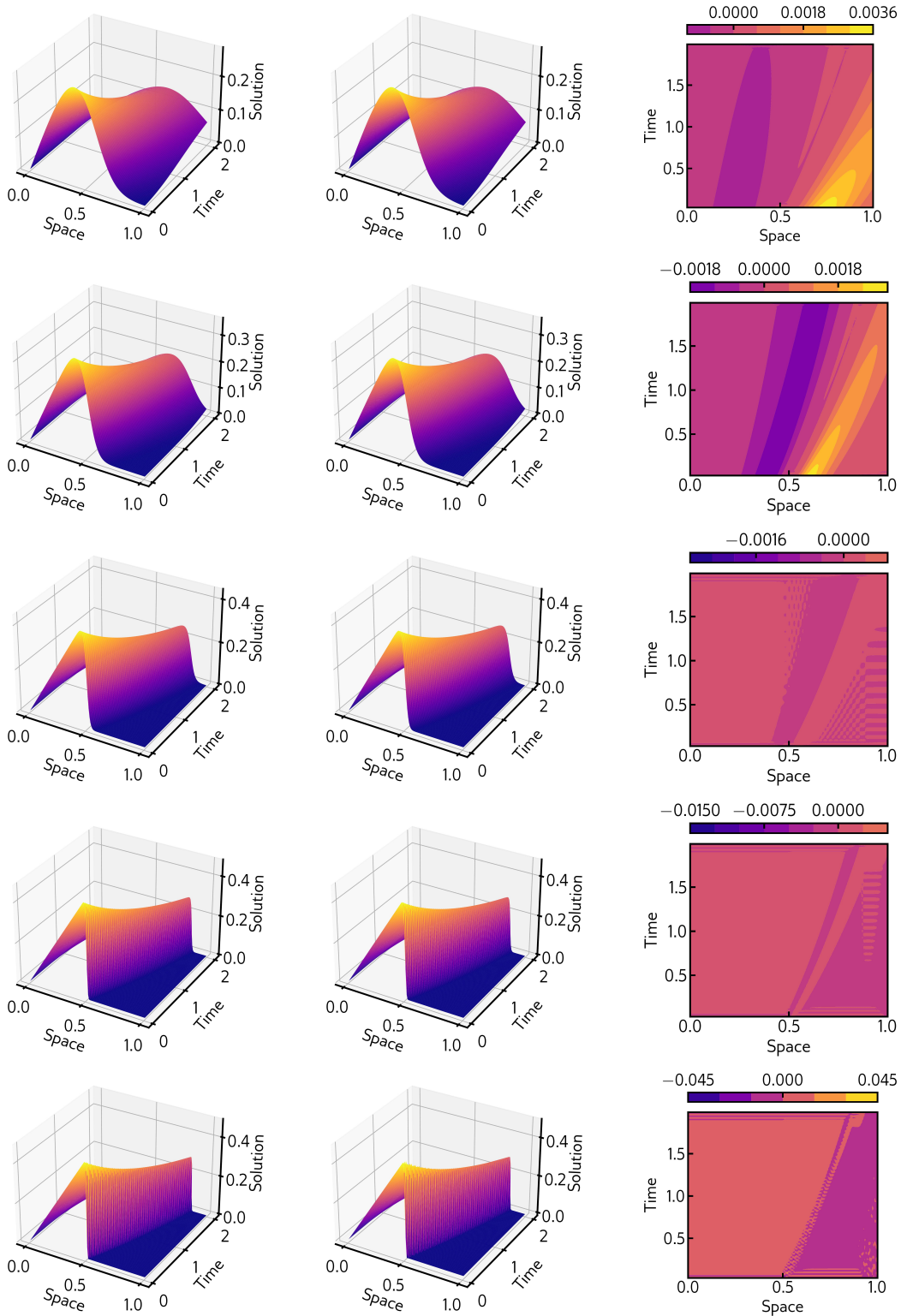


Figure 5.3: Burgers' equation: The true solution (shown in first column), ActLearn-POD-KSNN solution (shown in second column), and the solution error (shown in third column). The error values correspond to the point-wise difference in the space-time domain between the ActLearn-POD-KSNN solution and the true solution. The Re values going from top to bottom in the rows are in the following order: $\{40, 100, 350, 1250, 3000\}$. All these Re values and discrete time instances required to generate the plots are outside of the training set.

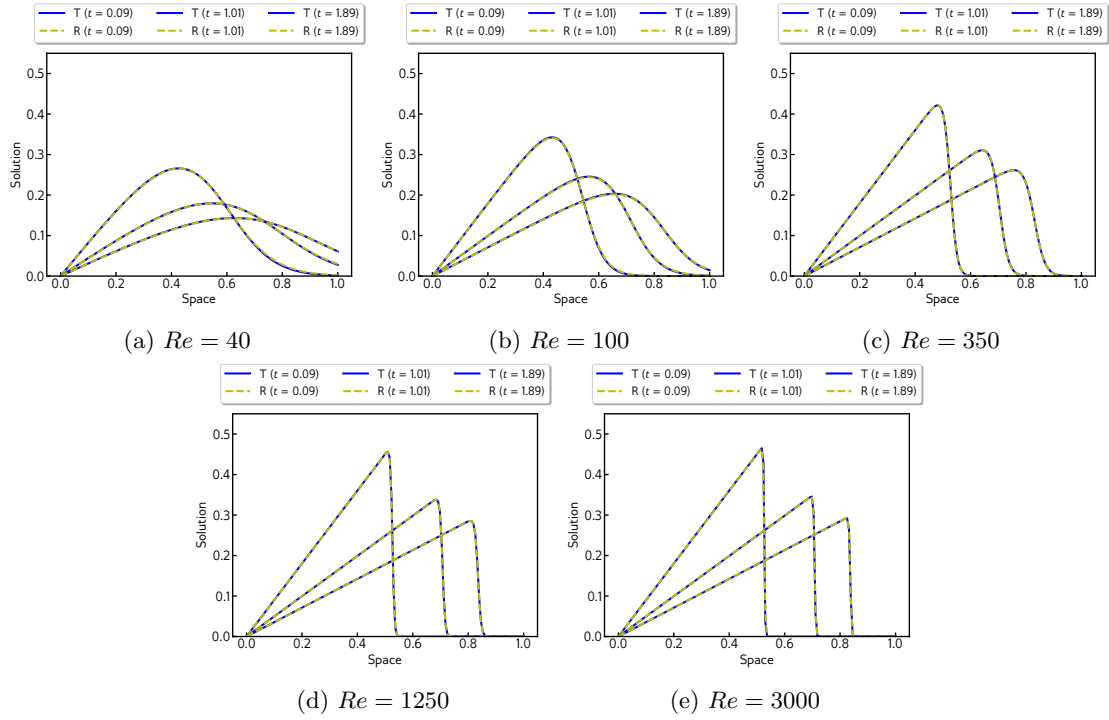


Figure 5.4: Comparison of the ActLearn-POD-KSNN solution (denoted by R) and true solution (denoted by T) for Burgers' equation. All the Reynolds numbers and time instances (t) are outside of the training set.

To begin the active learning procedure, the parameter set P is initiated by 21 viscosity values corresponding to the following indices,

$$\{0, 99, 10, 20, 30, 40, 50, 60, 70, 80, 90, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95\}.$$

Here, the ν values are ordered from lowest to highest and the index starts from 0 when counting the 100 values. We report the indices instead of exact values for ease of readability. High-fidelity snapshots are generated for all the viscosities in P at 100 instances of $t \in [0, 2]$. Using these snapshots, POD-approximate solutions are computed such that the POD subspace retains 99.99% of the energy, i.e., $\eta(\nu) = 10^{-4}$, $\forall \nu \in P$.

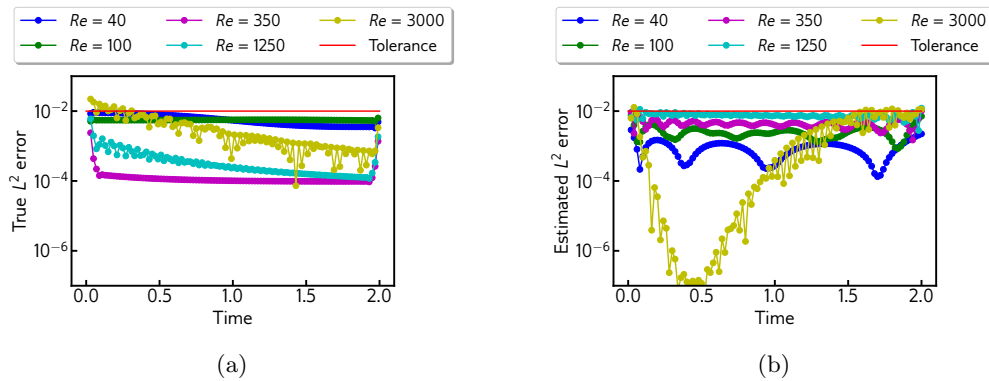


Figure 5.5: Plot (a) shows the true error of the ActLearn-POD-KSNN solution for the Burgers' equation on a new time grid corresponding to several out-of-training Reynolds numbers. The tolerance used for termination of the active learning procedure is also shown for comparison. Plot (b) shows the estimated error on the original time grid.

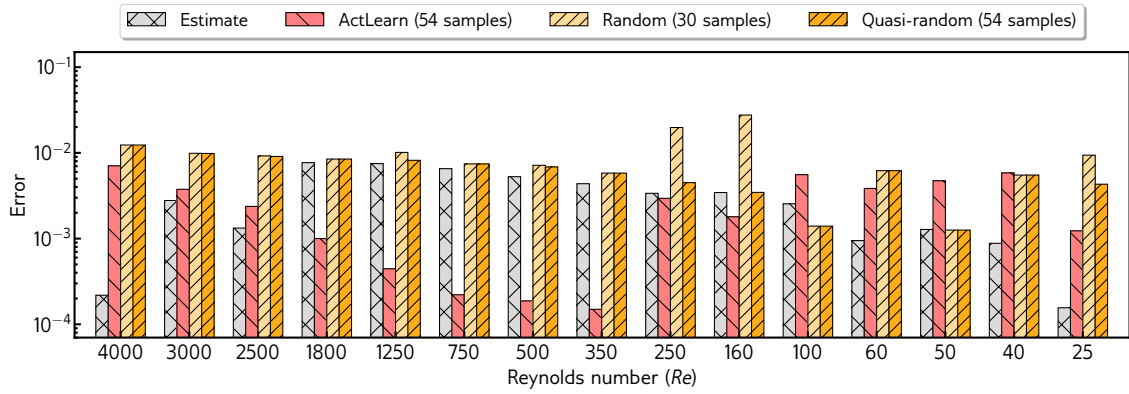


Figure 5.6: Burgers’ equation: Error comparison between the ActLearn-POD-KSNN solution and the POD-KSNN solutions upon a random (30 samples, seed 10) and quasi-random (54 samples, seed 10) selection of parametric training data. The values labeled ‘ActLearn’, ‘Random’, and ‘Quasi-random’ are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. The values labeled ‘Estimate’ are the time-average (over the training time grid) of the error estimate values given by (3.13). All the reported Reynolds numbers are outside of the training set.

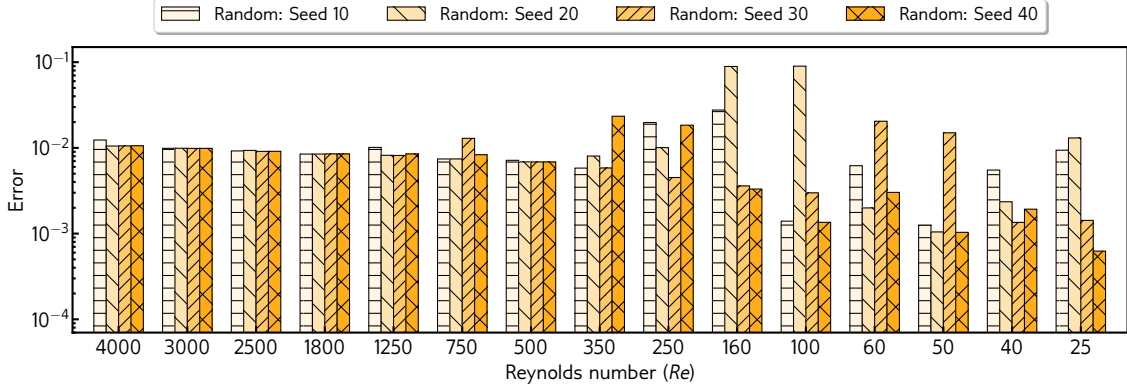
During the active learning loop, we sample from the candidate set P^* at each iteration. The initial state of P^* for the reported results comprises 79 candidate values – upon exclusion of 21 values present in P (initially) from the total 100 values. Figure 5.2a shows the estimated error $\mathcal{E}^{(iter)}$ (see Algorithm 1) varying with greedy iterations of the active learning process. The error decreases over iterations, and we stop the loop after a tolerance of 10^{-2} is met. So, 33 new parameter points are selected in a greedy fashion and their corresponding estimated error values are reported in Figure 5.2a.

The ultimate choice of viscosity values and their corresponding POD subspace dimensions are shown in Figure 5.2b. The reported dimensions also account for the POD space adaptation (refinement) when required through the iterations, as discussed in Section 3.2. For low viscosities, the subspace dimension is comparatively higher. And new selections are mostly concentrated in regions where the nature of the POD subspace changes significantly. Among all the individual parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in (4.1) is 2.676×10^{-11} . This is used as a condition for deciding the POD subspace dimension for any newly queried parameter.

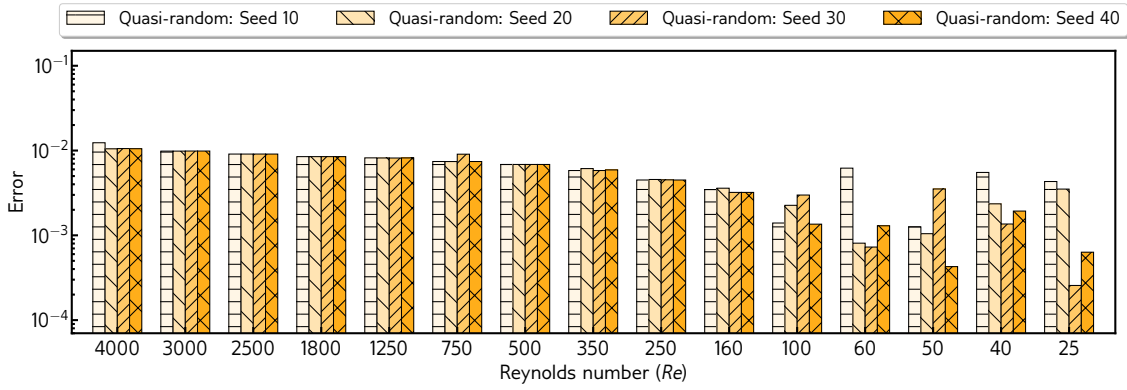
In Figure 5.3, the solution to the ActLearn-POD-KSNN surrogate model is compared with the true solution over the entire space-time domain. This allows us to see the evolution of the solutions in time. The Reynolds numbers are taken outside the training set – $\{40, 100, 350, 1250, 3000\}$. The solution is computed on a new test time grid with 99 instances starting from 0.01 with a step size of 0.02. We can see that the ActLearn-POD-KSNN solution agrees well with the ground truth. To further visualize and compare the solution with the truth, solutions are plotted in Figure 5.4 for three representative time instances from the start, middle, and end of the time domain. The error contours in the third column of Figure 5.3 show the point-wise difference in the space-time domain between the surrogate solution and the ground truth. The solution error stays reasonable for the entire range of Re values.

To obtain an estimate of the relative solution error in the surrogate’s approximation at a newly queried parameter μ^* , we train the KSNNs using the relative error data given by (3.12) for all $\mu_i \in P$. Doing this, we obtain an error estimate $\tilde{\varepsilon}(Re)$ for all the time instances, which is given by $\tilde{\varepsilon}(\mu^*)$ in (3.14). Such an estimation of the error is shown in Figure 5.5b for the training (original) time grid corresponding to several out-of-training Reynolds number. Whereas, Figure 5.5a shows the true relative error values of the ActLearn-POD-KSNN solution. Here, the time instances are different from the training time grid. For the most part, the true relative errors are bounded by the tolerance criterion 10^{-2} which is used for the active learning loop.

In Figure 5.6, we provide a comparative study between ActLearn-POD-KSNN solution error and POD-KSNN solution errors that are obtained by randomly picking 30 and 54 parameter samples for



(a) Error in the solution upon random selection of parametric training data.



(b) Error in the solution upon quasi-random selection of parametric training data.

Figure 5.7: Burgers' equation: Error comparison between the POD-KSNN solutions upon a random (30 samples) and quasi-random (54 samples) selection of parametric training data with four different starting seeds: $\{10, 20, 30, 40\}$. The error values are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. All the reported Reynolds numbers are outside of the training set.

preparing the training snapshots of the surrogate. We label the surrogate solution error obtained by training with 54 random samples as quasi-random because this choice is actually informed by the total number of parameter samples ($21 + 33 = 54$) upon termination of the active learning loop. The random selection from a pool of 100 values of Re (which are the same as described before during the preparation of sets P and P^*) is done with a fixed representative seed value of 10 using the NumPy library written for Python programming language. The surrogate solutions $(\mathbf{u}_s)_k(Re) := \mathbf{u}_s(t_k, Re)$ are first computed over the previously described test time grid (t_k with $k = 1, \dots, 99$, $t_1 = 0.01$, a uniform step size of 0.02, and the total discrete time instances are $\tilde{N}_t = 99$) after which the relative l^2 error values in space are obtained,

$$\varepsilon_k(Re) := \frac{\|\mathbf{u}_k(Re) - (\mathbf{u}_s)_k(Re)\|_2}{\|\mathbf{u}_k(Re)\|_2} \quad (5.4)$$

We then take the average of these l^2 errors over all the discrete time instances, i.e., Error := $(1/\tilde{N}_t) \sum_k \varepsilon_k(Re)$, and report the final result for active, random, and quasi-random sampling in Figure 5.6. In a similar fashion, we report the time-average of the error estimate values $\tilde{\varepsilon}_j(Re)$ defined in (3.13) on the training time grid (t_j with $j = 1, \dots, 100$, $t_1 = 0.0$, a uniform step size of 0.02, and the total discrete time instances are $N_t = 100$), i.e., Estimate := $(1/N_t) \sum_j \tilde{\varepsilon}_j(Re)$. We observe that the error in the ActLearn-POD-KSNN solution is bounded by the tolerance of 10^{-2} specified for termination of the active learning procedure.

In Figure 5.7, we compare the POD-KSNN solution error for scenarios when 30 and 54 pa-

parameters are randomly picked with four different starting seed values: $\{10, 20, 30, 40\}$. Similar to Figure 5.6, the reported error values are the time-average (over the test time grid) of the relative l^2 error in space. Figure 5.7a shows that with 30 random samples the error goes up to 10^{-1} , and there is also significant difference among the solution error values corresponding to certain testing Reynolds numbers for different starting seeds. We observe from Figure 5.7b that the error values reduce by adding more samples, but there is still a noticeable variation between the values for different starting seeds. This situation is addressed by the active learning framework. Moreover, the procedure provides an idea about the number of parameter samples required for preparing the snapshot training data such that the constructed surrogate model approximates the solution (at any new parameter and time instance) up to some predefined accuracy level, as specified by the tolerance value.

5.2 Shallow Water Equations

The free-surface flows in water bodies like channels or rivers can be modeled using the shallow water equations [3]. They are obtained from the incompressible Navier-Stokes equations under the condition that the fluid flow's vertical extent is significantly smaller than its horizontal extent. The conservation of mass and momentum takes the following form:

$$\partial_t h + \partial_x(hu) = 0, \quad (5.5)$$

$$\partial_t(hu) + \partial_x\left(hu^2 + \frac{1}{2}gh^2\right) = -\frac{\nu}{\lambda}u, \quad (5.6)$$

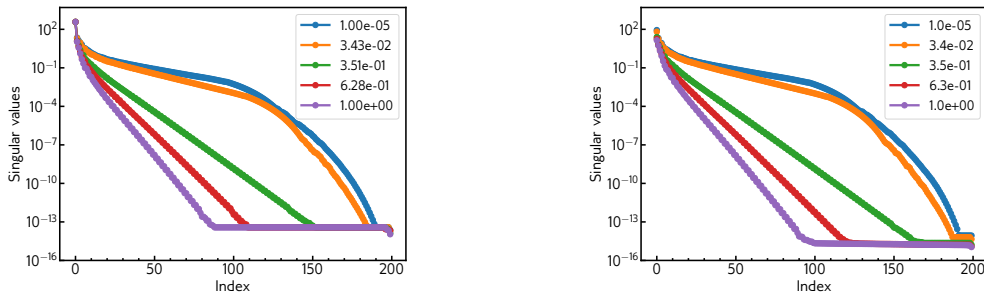
where $h(t, x)$ is the depth of the channel, $u(t, x)$ is the depth-averaged velocity along the length of the channel, ν is the dynamic viscosity, λ is the mean-free path, g is the gravitational acceleration. This form of the shallow water equations is also known as the Saint-Venant system, which naturally allows us to capture a constant vertical velocity profile in a channel. To carry out the experiments, the initial condition of the height is taken as a smooth bump described by the following nonlinear function:

$$h(0, x) = 1 + \exp(3 \cos(\pi(x + 0.5)) - 4). \quad (5.7)$$

The initial velocity is taken to be constant along x ,

$$u(0, x) = 0.25. \quad (5.8)$$

The periodic spatial domain is $\Omega \in [-1, 1]$ with 601 grid nodes, and the time-domain is $t \in [0, 2]$ in which the solution is stored at 200 uniform time steps. The high-fidelity solution of the system of equations is computed using a discontinuous Galerkin solver with local polynomial reconstruction of degree 1 [20]. The Riemann solver utilized to compute the numerical flux is local Lax-Friedrichs. For integration in time, we use a second-order strong stability preserving Runge-Kutta scheme. The equations are solved in non-dimensional form, and all the reported parameter values are non-dimensional. For details about the conversion to dimensionless form, we suggest the reader to refer [23]. We perform a singular value decomposition of the parameter-specific snapshot matrices



(a) Singular value decay for the fluid height.

(b) Singular value decay for the fluid velocity.

Figure 5.8: Shallow water equations: The decay in singular values for various viscosity samples ν .

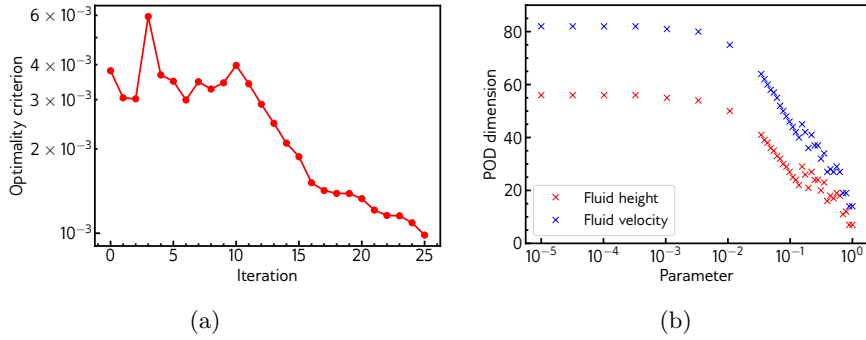


Figure 5.9: Active learning for shallow water equations: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer [Algorithm 1](#)) varying with greedy iterations of the active learning process; (b) shows the final POD subspace dimension for each of the chosen parameter samples.

for the fluid height and velocity and plot their respective singular value decays in [Figure 5.8](#). The decay is already not very fast for higher viscosity values, which gets exacerbated further as the viscosity values are decreased. The hyperbolic nature of these equations renders an even greater challenge for constructing an efficient reduced-order surrogate.

For our experiments, we fix $\lambda = 0.1$, and vary the viscosity ν from 1 to 10^{-5} . The total number of discrete parameters we consider when accounting for both the candidate set P^* and the parameter set P are 100. These 100 samples of ν are picked by uniformly dividing the logarithmic ν values ($\log_{10}(10^{-5})$ and $\log_{10}(1)$) into 99 intervals, ensuring a decent pool of parameters.

To begin the active learning procedure, the parameter set P is initiated by 11 viscosity values corresponding to the following indices,

$$\{0, 99, 10, 20, 30, 40, 50, 60, 70, 80, 90\}.$$

Here, the ν values are ordered from lowest to highest and the index starts from 0 when counting the 100 values. Like done for Burgers' equation, we report the indices instead of exact values for ease of readability. High-fidelity snapshots are generated for all the viscosities in P for 200 time instances. Using these snapshots, POD-approximate solutions are computed such that the POD subspace retains 99.99997% of the energy.

During the active learning loop, we sample from the candidate set P^* at each iteration. The initial state of P^* for the reported results comprises 89 candidate values—upon exclusion of 11 values present in P (initially) from the total 100 values. [Figure 5.9a](#) shows variation of the estimated error $\mathcal{E}^{(iter)}$ (see [Algorithm 1](#)) with greedy iterations of the active learning process. The error decreases over iterations, and we stop the loop after a tolerance of 10^{-3} is met. So, 26 new viscosity samples are picked in a greedy fashion for which the estimated error values are reported in [Figure 5.9a](#).

The ultimate choice of viscosity values and their corresponding POD subspace dimensions are shown in [Figure 5.9b](#). Like for the Burgers' equation, here as well the reported dimensions account for the POD subspace refinement through the iterations. Also like Burgers' equation, for low values of viscosities, the subspace dimension is comparatively higher. New selections are mostly concentrated in regions where the nature of the POD subspace changes significantly. Here, this is towards the moderate to high viscosity regions. Among all the individual parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in (4.1) for the fluid height is 2.989×10^{-10} , and for the fluid velocity is 6.362×10^{-10} . These are used as conditions for deciding the POD subspace dimension for the fluid height and velocity at any newly queried value of ν in the online phase (refer [Algorithm 2](#)).

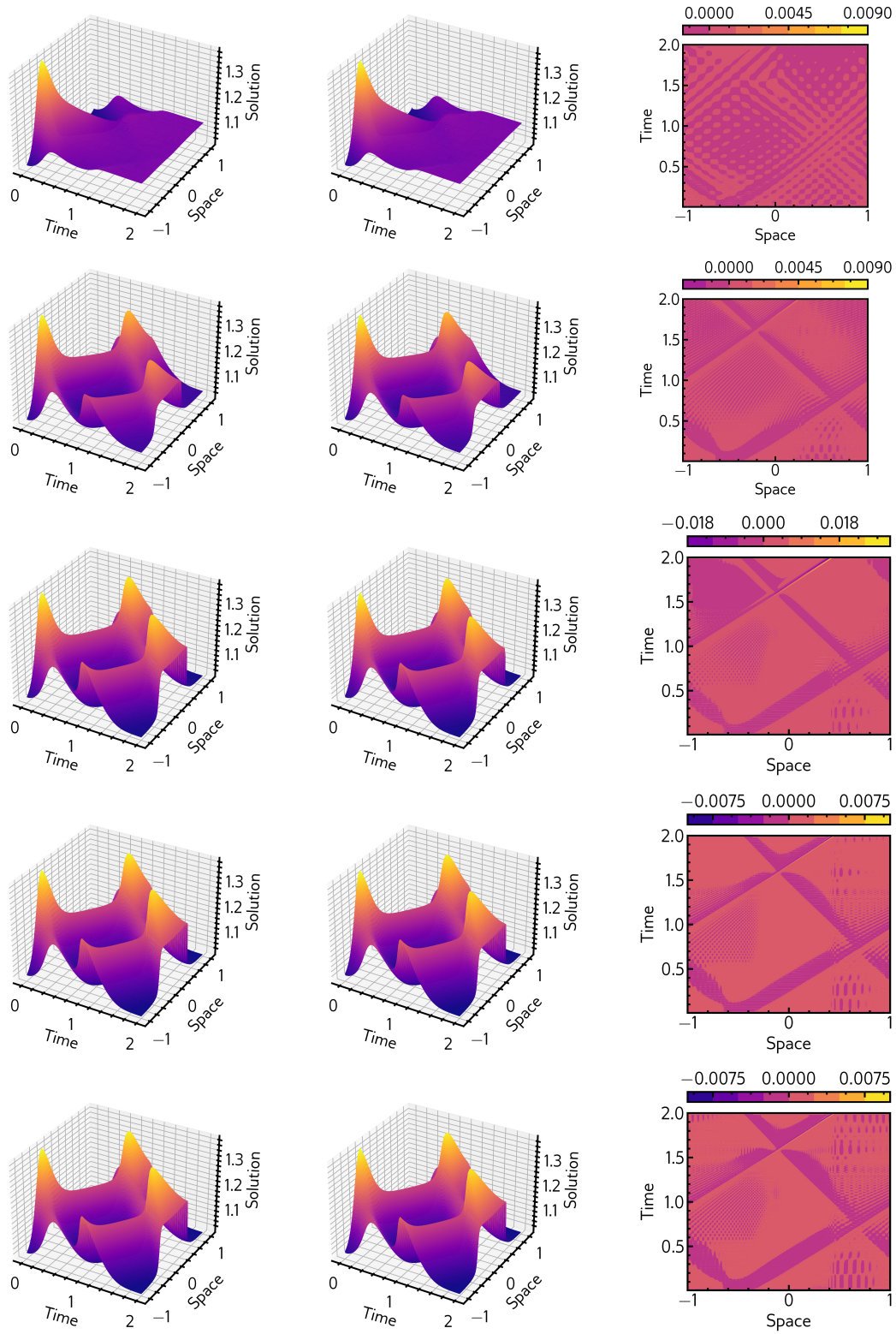


Figure 5.10: Shallow water equations' fluid height: The true solution (shown in first column), ActLearn-POD-KSNN solution (shown in second column), and the solution error (shown in third column). The error values correspond to the point-wise difference in the space-time domain between the ActLearn-POD-KSNN solution and the true solution. The viscosity ν going from top to bottom in the rows are in the following order: $\{5 \times 10^{-1}, 5 \times 10^{-2}, 5 \times 10^{-3}, 5 \times 10^{-4}, 5 \times 10^{-5}\}$. All these ν values and time instances are outside of the training set.

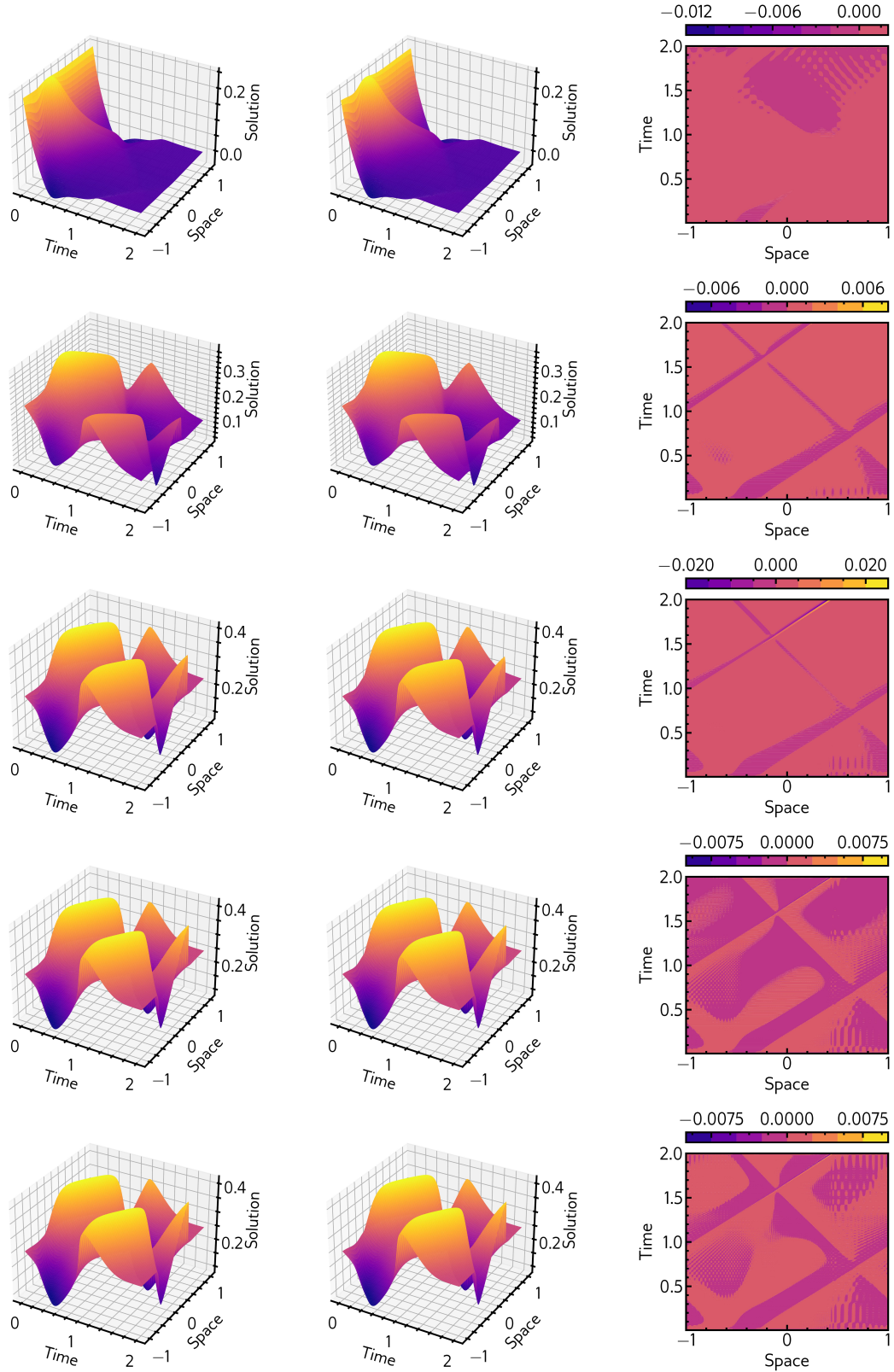


Figure 5.11: Shallow water equations’ fluid velocity: The true solution (shown in first column), ActLearn-POD-KSNN solution (shown in second column), and the solution error (shown in third column). The error values correspond to the point-wise difference in the space-time domain between the ActLearn-POD-KSNN solution and the true solution. The viscosity ν going from top to bottom in the rows are in the following order: $\{5 \times 10^{-1}, 5 \times 10^{-2}, 5 \times 10^{-3}, 5 \times 10^{-4}, 5 \times 10^{-5}\}$. All these ν values and time instances are outside of the training set.

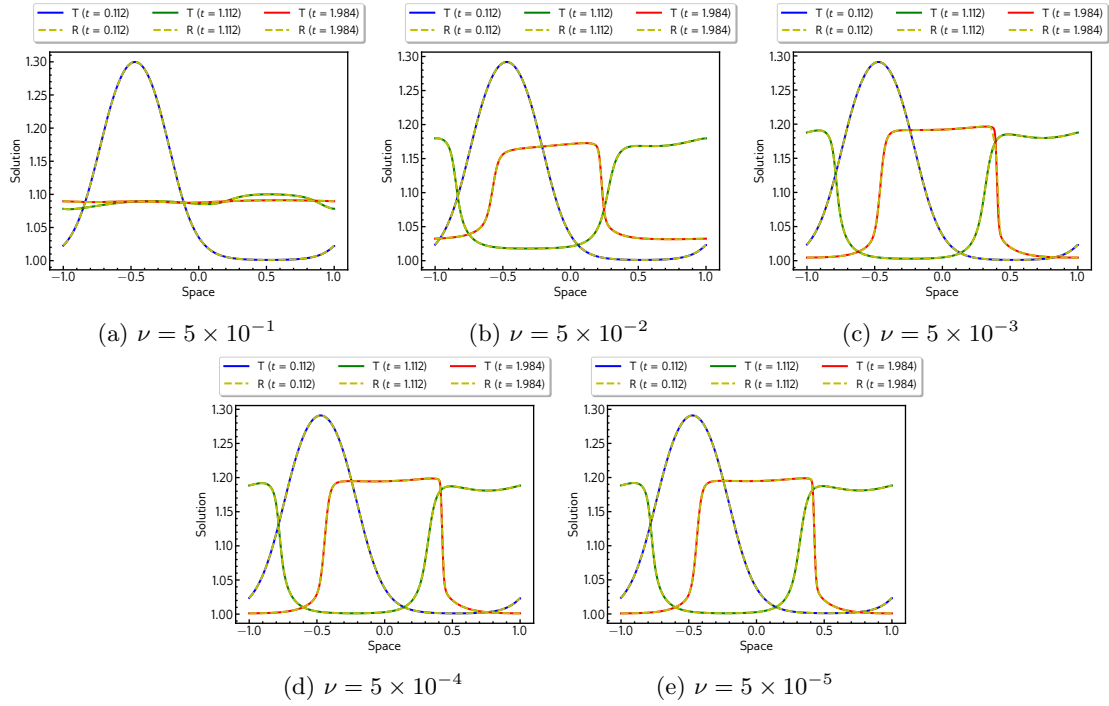


Figure 5.12: Comparison of the ActLearn-POD-KSNN solution (denoted by R) and true solution (denoted by T) for the fluid height in the shallow water equations. All the ν values and time instances t are outside of the training set.

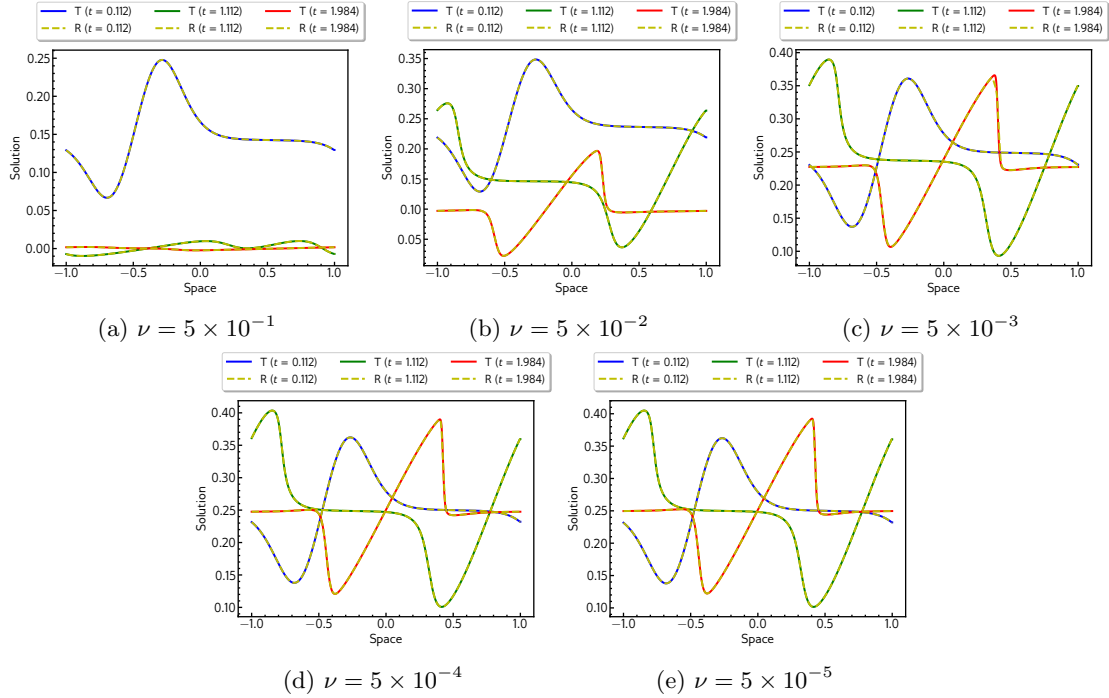


Figure 5.13: Comparison of the ActLearn-POD-KSNN solution (denoted by R) and true solution (denoted by T) for the fluid velocity in the shallow water equations. All the ν values and time instances t are outside of the training set.

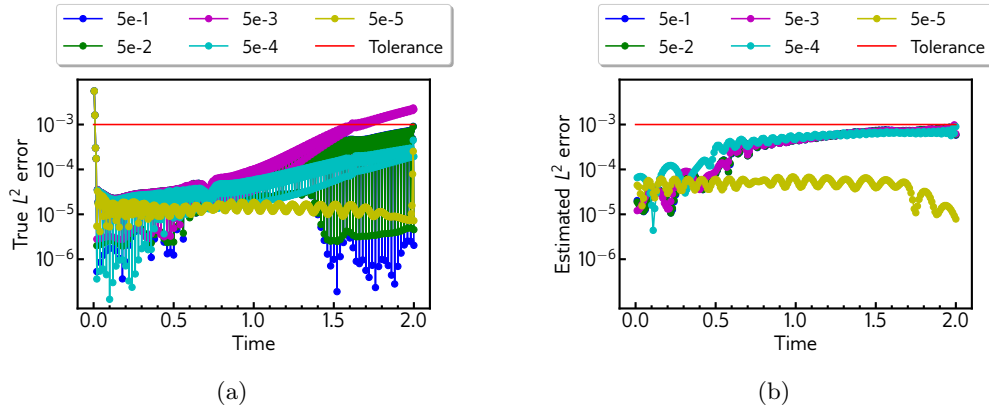


Figure 5.14: Plot (a) shows the true error of the ActLearn-POD-KSNN solution for the fluid height of the shallow water equations on a new time grid corresponding to several out-of-training samples of ν . The tolerance used for termination of the active learning procedure is also shown for comparison. Plot (b) shows the estimated error for the fluid height on the original time grid.

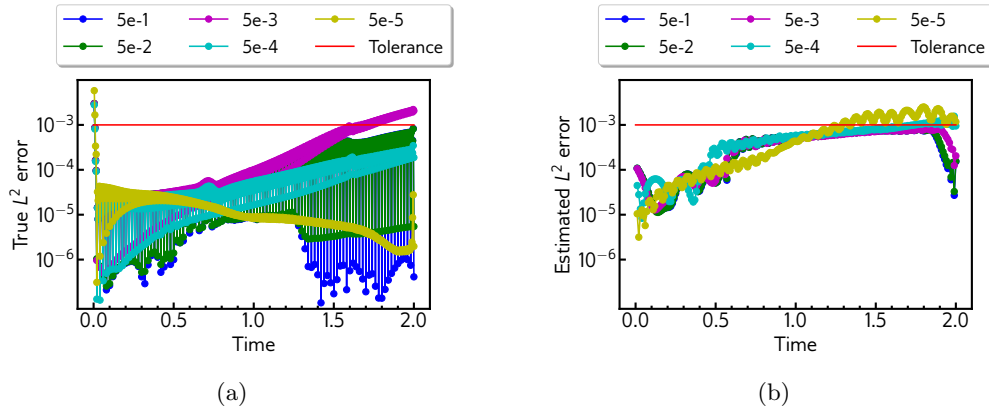


Figure 5.15: Plot (a) shows the true error of the ActLearn-POD-KSNN solution for the fluid velocity of the shallow water equations on a new time grid corresponding to several out-of-training samples of ν . The tolerance used for termination of the active learning procedure is also shown for comparison. Plot (b) shows the estimated error for the fluid velocity on the original time grid.

In Figures 5.10 and 5.11, the fluid height and velocity obtained from the ActLearn-POD-KSNN surrogate model are compared with the true height and velocity over the entire space-time domain. We can see that the ActLearn-POD-KSNN solutions are able to capture the multiple shock interactions over time, in both the fluid height and the fluid velocity. The viscosity values are taken outside the training set: $\{5 \times 10^{-1}, 5 \times 10^{-2}, 5 \times 10^{-3}, 5 \times 10^{-4}, 5 \times 10^{-5}\}$. The solution is computed on a new test time grid with 499 instances starting from 0.004 with a step size of 0.004. We can see that the ActLearn-POD-KSNN solution agrees well with the ground truth. To further visualize and compare the surrogate solutions with the true solutions, we plot them in Figures 5.12 and 5.13 at three representative time instances from the start, middle, and end of the time domain. The error contours in the third column of Figures 5.10 and 5.11 show the point-wise difference in the space-time domain between the surrogate solution and the ground truth.

An estimation of the error in the fluid height and velocity at several out-of-training parameters is shown in Figures 5.14b and 5.15b, respectively, for the training time grid. Similar to the Burgers' equation, the reported estimates are computed by training KSNNs and obtaining values for $\tilde{\epsilon}(\mu^*)$ in (3.14). Figures 5.14a and 5.15a show the true relative error values of the fluid height and velocity obtained from the ActLearn-POD-KSNN surrogate. Here, the time instances are different from

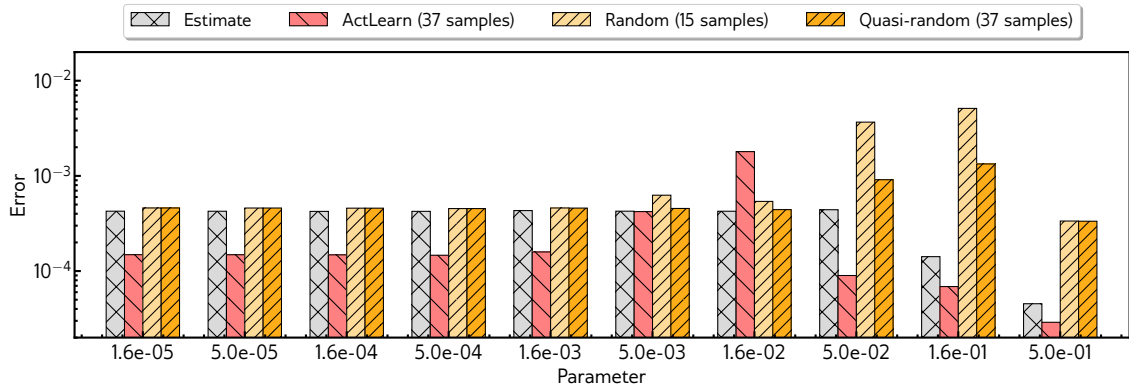
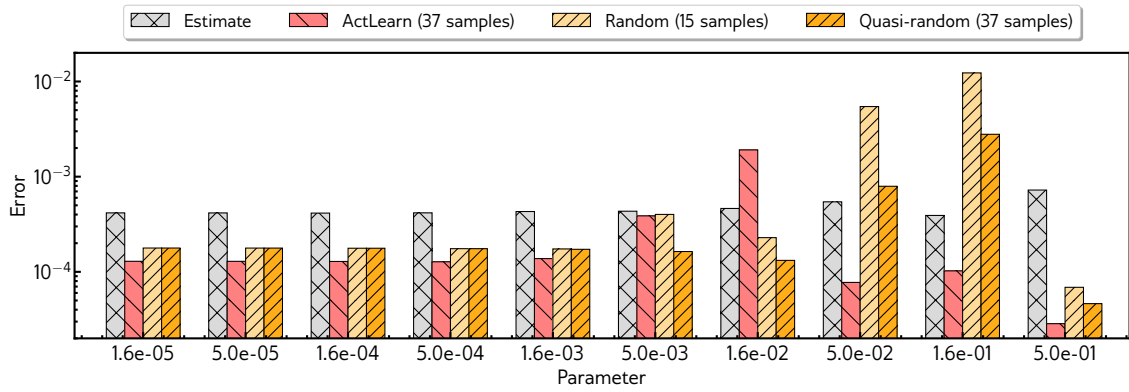
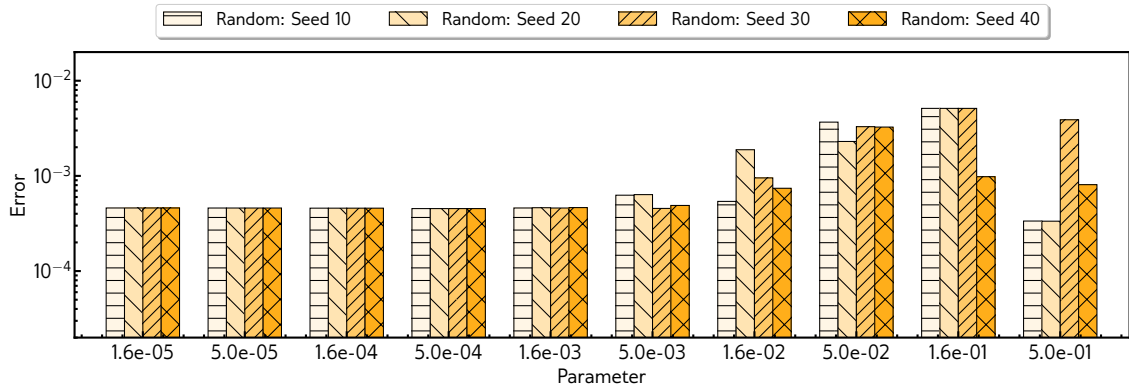
(a) Error in the fluid height for several out-of-training parameter samples ν .(b) Error in the fluid velocity for several out-of-training parameter samples ν .

Figure 5.16: Shallow water equations: Error comparison between the ActLearn-POD-KSNN solution and the POD-KSNN solutions upon a random (15 samples, seed 10) and quasi-random (37 samples, seed 10) selection of parametric training data. The values labeled 'ActLearn', 'Random', and 'Quasi-random' are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. The values labeled 'Estimate' are the time-average (over the training time grid) of the error estimate values given by (3.13).

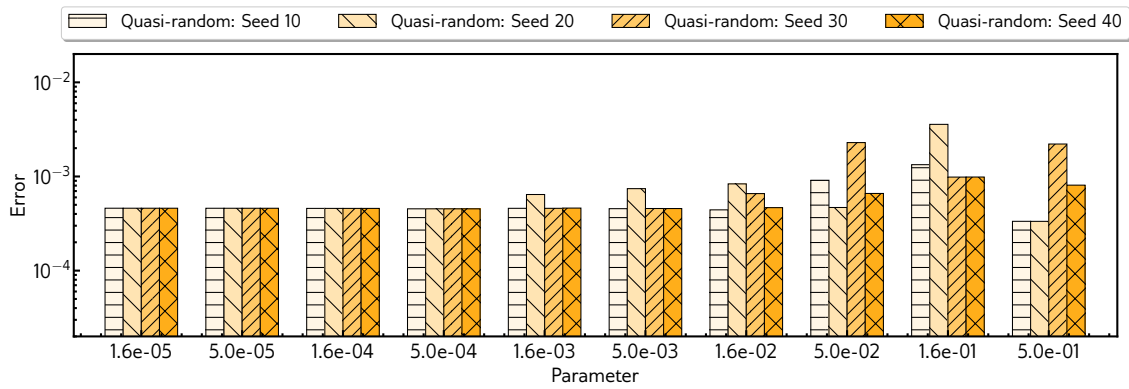
the training time grid. For the most part, the true relative errors are bounded by the tolerance criterion 10^{-3} which is used for the active learning loop.

Figure 5.16 provides a comparative study between the ActLearn-POD-KSNN solution error and the POD-KSNN solution errors that are obtained by randomly picking 15 and 37 parameter samples. The errors in fluid height and velocity are respectively reported in Figures 5.16a and 5.16b. Similar to the Burgers' equation example, we label the surrogate solution error obtained by training with 37 random samples as quasi-random, because this choice is inspired from the active learning procedure. For the random selection from 100 values of ν (which are the same as described before during the preparation of sets P and P^*), we again fix the random seed in NumPy to 10. All the surrogate error values and the estimates in Figure 5.16 are computed in the same way as those in Figure 5.6, i.e., the relative l^2 errors in the spatial domain are time-averaged over the test time grid (t_k with $k = 1, \dots, 499$, $t_1 = 0.004$, a uniform step size of 0.004, and the total discrete time instances are $\tilde{N}_t = 499$), whereas, the reported error estimate values are time-averaged over the training time grid (t_j with $j = 1, \dots, 200$, $t_1 = 0.0$, a uniform step size of 0.01, and the total discrete time instances are $N_t = 200$).

In Figures 5.17 and 5.18, we compare the error in the fluid height and velocity approximated by the POD-KSNN surrogate for scenarios when 15 and 37 parameters are randomly picked with four different starting seed values: $\{10, 20, 30, 40\}$. Similar to Figure 5.16, the reported error values are the time-average (over the test time grid) of the relative l^2 error in space. Figures 5.17a and 5.18a



(a) Error in the fluid height upon random selection of parametric training data.



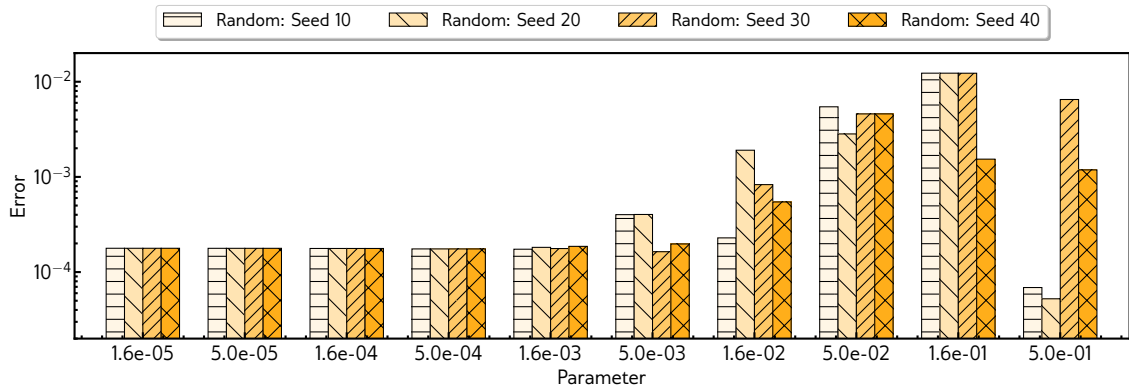
(b) Error in the fluid height upon quasi-random selection of parametric training data.

Figure 5.17: Shallow water equations: Error comparison between the POD-KSNN fluid height solution upon a random (15 samples) and quasi-random (37 samples) selection of parametric training data with four different starting seeds: $\{10, 20, 30, 40\}$. The error values are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. All the reported parameter samples ν are outside of the training set.

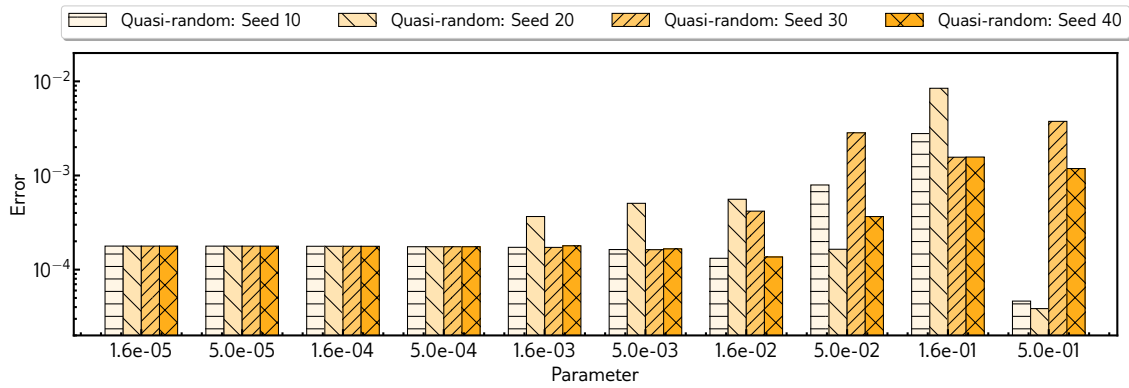
show that with 15 random samples, the error goes up to 10^{-2} , and similar to our observation for Burgers' equation, there is also noticeable difference among the solution error values corresponding to certain testing viscosities for different starting seeds. The error values reduce by adding more samples as seen in Figures 5.17b and 5.18b, but there is still a noticeable variation between the values for different starting seeds. So, the accuracy of the surrogate solution is dependent on how the random sampling is done, i.e., the choice of the starting seed. The active learning procedure resolves this situation by picking the parameter samples based on an optimality criterion.

From Figure 5.16 we notice that the error of the ActLearn-POD-KSNN solution is generally the lowest and bounded by the tolerance of 10^{-3} specified for termination of the active learning procedure. For $\nu = 1.6 \times 10^{-2}$, the error in the fluid height is 1.79×10^{-3} and the fluid velocity is 1.91×10^{-3} , which is slightly higher than the tolerance. However, these error values are still lower than the maximum error values we observe with random and quasi-random sampling in Figures 5.17 and 5.18, i.e., for $\nu = 1.6 \times 10^{-1}$ with a starting seed of 20. The active learning procedure gives an idea about the most informative parameter samples useful for preparing the snapshot training data. This way the ActLearn-POD-KSNN surrogate solution provides a reasonable accuracy without oversampling the parameter space for preparation of the training snapshot data, thereby staying computationally efficient.

We report runtime of the full-order shallow water equation solver and the ActLearn-POD-KSNN surrogate model in Table 5.1. The numerical tests are carried out on a laptop with Intel[®] Core[™] i5-1035G1 CPU @ 1.00GHz and 16 GB of RAM. All the reported timings are the average of



(a) Error in the fluid velocity upon random selection of parametric training data.



(b) Error in the fluid velocity upon quasi-random selection of parametric training data.

Figure 5.18: Shallow water equations: Error comparison between the POD-KSNN fluid velocity solution upon a random (15 samples) and quasi-random (37 samples) selection of parametric training data with four different starting seeds: $\{10, 20, 30, 40\}$. The error values are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. All the reported parameter samples ν are outside of the training set.

three independent executions. The timings reported under offline phase and online phase of the surrogate model are the total execution times for Algorithms 1 and 2 respectively. The time for active sampling and other offline computations required for building the surrogate, excluding the full-order model query time, is only 6.23 seconds for a grid size of 601, and 7.63 seconds for a grid size of 1201. The total full-order model query time during the offline phase depends on the number of parameter samples picked by the active learning procedure until its termination, and on the

Number of grid nodes	FOM solver	ActLearn-POD-KSNN surrogate model	
		Offline phase	Online phase
601	249.56	$6.23 + (26 \times 249.56) = 6494.78$	0.04
1201	894.62	$7.63 + (17 \times 894.62) = 15216.17$	0.08

Table 5.1: Comparison between runtime (in seconds) of the full-order model (FOM) and the ActLearn-POD-KSNN surrogate model. The simulations are performed for the shallow water equations at two spatial grid sizes. All the reported timings are the average of three independent executions.

time it takes for generating the full-order solution for one parameter sample. Once the surrogate is built, a fast approximation of the solution is possible, in just 0.04 seconds for a grid size of 601, and 0.08 seconds for a grid size of 1201. Compared to evaluating the full-order solver at a new parameter sample, we can query the surrogate and obtain the approximate solution with a speedup of greater than $\mathcal{O}(10^3)$, as evident from Table 5.1. Note that the surrogate model becomes efficient as soon as it is called more often for unseen parameter values than used in the offline time.

6 Conclusions

We have proposed an active learning framework for parametric non-linear dynamical systems that generates solution snapshots at new parameter locations by evaluating the high-fidelity model when necessary. This, in turn, improves the accuracy of the data-driven surrogate model. The central driving force of the active learning process is a non-intrusive error-estimation-based optimality criterion. It is designed from the parameter-specific relative POD approximation errors. Through active learning, we iteratively arrive at a good selection of solution snapshots which are then used to train the data-driven surrogate. In doing so, we relax the vast data requirement for training data-driven surrogate models to some extent, and also provide an estimation of the surrogate accuracy.

The numerical results show that the developed active learning framework iteratively detects locations in the parameter domain where the variation in solution features is high, and prefers new snapshot generation in those regions. For the Burgers' equation, the ActLearn-POD-KSNN surrogate model is able to successfully gauge the variation in its initial conditions and capture the transport of shock profile accurately in time, over the entire range of viscosity values. Moreover, for the shallow water equations, the surrogate model is able to efficiently predict, at new parameter locations, the interacting shock waves that morph into each other over time. The parameter-specific adaptive POD subspaces make our approach efficient, even for problems with mixed—convective and diffusive—phenomena, where each of them dominate in certain regions. Additionally, we observe that the true surrogate errors stay under or are very close to the tolerance level used to terminate the active learning procedure. This indicates reliability of the proposed error estimate that provides us a good measure to gauge the accuracy of the constructed ActLearn-POD-KSNN surrogate model.

The interpolation steps in the active learning loop as well as within the surrogate model's construction are carried out by automatically building, training, and evaluating several kernel-based shallow neural networks. Such a shallow architecture results in a fast offline training stage, as well as a fast online evaluation stage, further reducing the overall computational burden. The training strategy for our ActLearn-POD-KSNN surrogate model is problem independent, and automatically selects the parameter locations whose additional solution snapshots would most improve the non-linear reduced basis space. This minimizes the user interaction for data-driven surrogates built using machine-learning, and the fast online deployment phase brings us a step closer to real-time simulations for high-fidelity parametric physical systems.

Acknowledgments

Harshit Kapadia is supported by the International Max Planck Research School for Advanced Methods in Process and Systems Engineering (IMPRS-ProEng).

References

- [1] A. Bērziņš, J. Helmig, F. Key, and S. Elgeti. Standardized non-intrusive reduced order modeling using different regression models with application to complex flow problems. e-prints 2006.13706, arXiv, 2020. physics.comp-ph. URL: <https://arxiv.org/abs/2006.13706>.
- [2] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI Journal of Computational Mathematics*, 7:121–157, 2021. doi:10.5802/smai-jcm.74.

-
- [3] H. Chanson. *Environmental Hydraulics of Open Channel Flows*. Butterworth-Heinemann, Oxford, 2004. doi:10.1016/B978-0-7506-6165-2.X5028-0.
- [4] S. Chellappa, L. Feng, and P. Benner. An adaptive sampling approach for the reduced basis method. In *Realization and Model Reduction of Dynamical Systems - A Festschrift in Honor of the 70th Birthday of Thanos Antoulas*, pages 137–155. Springer, Cham, 2022. doi:10.1007/978-3-030-95157-3_8.
- [5] W. Chen, J. S. Hesthaven, B. Junqiang, Y. Qiu, Z. Yang, and Y. Tihao. Greedy nonintrusive reduced order model for fluid dynamics. *AIAA Journal*, 56(12):4927–4943, 2018. doi:10.2514/1.J056161.
- [6] W. Chen, Q. Wang, J. S. Hesthaven, and C. Zhang. Physics-informed machine learning for reduced-order modeling of nonlinear problems. *Journal of Computational Physics*, 446:110666, 2021. doi:10.1016/j.jcp.2021.110666.
- [7] S. Dutta, M. W. Farthing, E. Perracchione, G. Savant, and M. Putti. A greedy non-intrusive reduced order model for shallow water equations. *Journal of Computational Physics*, 439:110378, 2021. doi:10.1016/j.jcp.2021.110378.
- [8] L. C. Evans. *Partial differential equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, second edition, 2010. ISBN: 9780821849743.
- [9] B. A. Freno and K. T. Carlberg. Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations. *Computer Methods in Applied Mechanics and Engineering*, 348:250–296, 2019. doi:10.1016/j.cma.2019.01.024.
- [10] S. Fresca, L. Dedé, and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87(61):1–36, 2021. doi:10.1007/s10915-021-01462-7.
- [11] F. J. Gonzalez and M. Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. e-prints 1808.01346v2, arXiv, 2018. math.DS. URL: <https://arxiv.org/abs/1808.01346>.
- [12] M. Guo and J. S. Hesthaven. Reduced order modeling for nonlinear structural analysis using Gaussian process regression. *Computer Methods in Applied Mechanics and Engineering*, 341:807–826, 2018. doi:10.1016/j.cma.2018.07.017.
- [13] M. Guo and J. S. Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering*, 345:75–99, 2019. doi:10.1016/j.cma.2018.10.029.
- [14] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011. doi:10.1137/090771806.
- [15] D. Hartmann and L. K. Mestha. A deep learning framework for model reduction of dynamical systems. In *Proceedings of 2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1917–1922, 2017. doi:10.1109/CCTA.2017.8062736.
- [16] J. S. Hesthaven, G. Rozza, and B. Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. SpringerBriefs in Mathematics. Springer International Publishing, 2016. doi:10.1007/978-3-319-22470-1.
- [17] J. S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018. doi:10.1016/j.jcp.2018.02.037.
- [18] J. N. Kani and A. H. Elsheikh. Reduced-order modeling of subsurface multi-phase flow models using deep residual recurrent neural networks. *Transport in Porous Media*, 126:713–741, 2018. doi:10.1007/s11242-018-1170-7.

- [19] J. N. Kani and H. Elsheikh. DR-RNN: A deep recurrent neural network for model reduction. e-prints 1709.00939, arXiv, 2017. cs.CE. URL: <https://arxiv.org/abs/1709.00939>.
- [20] H. Kapadia. Discontinuous Galerkin schemes for extended shallow water models. Master's thesis, RWTH Aachen University, Aachen, Germany, 2019.
- [21] M. Kast, M. Guo, and J. S. Hesthaven. A non-intrusive multifidelity method for the reduced order modeling of nonlinear problems. *Computer Methods in Applied Mechanics and Engineering*, 364:112947, 2020. doi:10.1016/j.cma.2020.112947.
- [22] W. J. Kostorz, A. H. Muggeridge, and M. D. Jackson. An efficient and robust method for parameterized non-intrusive reduced-order modeling. *International Journal for Numerical Methods in Engineering*, 121:4674–4688, 2020. doi:10.1002/nme.6461.
- [23] J. Kowalski and M. Torrilhon. Moment approximations and model cascades for shallow flow. *Communications in Computational Physics*, 25(3):669–702, 2018. doi:10.4208/cicp.0A-2017-0263.
- [24] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020. doi:10.1016/j.jcp.2019.108973.
- [25] A. Mohan and D. V. Gaitonde. A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks. e-prints 1804.0926, arXiv, 2018. physics.comp-ph. URL: <https://arxiv.org/abs/1804.09269>.
- [26] S. E. Otto and C. W. Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019. doi:10.1137/18M1177846.
- [27] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced Basis Methods for Partial Differential Equations*, volume 92 of *La Matematica per il 3+2*. Springer International Publishing, 2016. doi:10.1007/978-3-319-15431-2.
- [28] S. A. Renganathan, R. Maulik, and V. Rao. Machine learning for nonintrusive model order reduction of the parametric inviscid transonic flow past an airfoil. *Physics of Fluids*, 32(4):047110, 2020. doi:10.1063/1.5144661.
- [29] L. Sirovich. Turbulence and the dynamics of coherent structures part I: Coherent structures. *Quarterly of Applied Mathematics*, 45(3):561–571, 1987. doi:10.1090/qam/910462.
- [30] Z. Wan, P. Vlachas, P. Koumoutsakos, and T. Sapsis. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. *PLOS ONE*, 13(5):1–22, 2018. doi:10.1371/journal.pone.0197704.
- [31] Q. Wang, J. S. Hesthaven, and D. Ray. Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem. *Journal of Computational Physics*, 384:289–307, 2019. doi:10.1016/j.jcp.2019.01.031.
- [32] D. Xiao. Error estimation of the parametric non-intrusive reduced order model using machine learning. *Computer Methods in Applied Mechanics and Engineering*, 355:513–534, 2019. doi:10.1016/j.cma.2019.06.018.
- [33] D. Xiao, F. Fang, C.C. Pain, and I.M. Navon. A parameterized non-intrusive reduced order model and error analysis for general time-dependent nonlinear partial differential equations and its applications. *Computer Methods in Applied Mechanics and Engineering*, 317:868–889, 2017. doi:10.1016/j.cma.2016.12.033.
- [34] Q. Zhuang, D. Hartmann, H. J. Bungartz, and J. M. Lorenzi. Active-learning-based nonintrusive model order reduction. *Data-Centric Engineering*, 4:e2, 2023. doi:10.1017/dce.2022.39.