Computer Methods in Applied Mechanics and Engineering 418 (2024) 116535

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

GPLaSDI: Gaussian Process-based interpretable Latent Space Dynamics Identification through deep autoencoder

Christophe Bonneville [a,*], Youngsoo Choi [b], Debojyoti Ghosh [b], Jonathan L. Belof [b]

[a] Cornell University, Ithaca, NY 14850, United States
[b] Lawrence Livermore National Laboratory, Livermore, CA 94550, United States

ARTICLE INFO

Keywords:
Autoencoders
Gaussian processes
Partial differential equation
Reduced-order-model
Latent-space identification

ABSTRACT

Numerically solving partial differential equations (PDEs) can be challenging and computationally expensive. This has led to the development of reduced-order models (ROMs) that are accurate but faster than full order models (FOMs). Recently, machine learning advances have enabled the creation of non-linear projection methods, such as Latent Space Dynamics Identification (LaSDI). LaSDI maps full-order PDE solutions to a latent space using autoencoders and learns the system of ODEs governing the latent space dynamics. By interpolating and solving the ODE system in the reduced latent space, fast and accurate ROM predictions can be made by feeding the predicted latent space dynamics into the decoder. In this paper, we introduce GPLaSDI, a novel LaSDI-based framework that relies on Gaussian process (GP) for latent space ODE interpolations. Using GPs offers two significant advantages. First, it enables the quantification of uncertainty over the ROM predictions. Second, leveraging this prediction uncertainty allows for efficient adaptive training through a greedy selection of additional training data points. This approach does not require prior knowledge of the underlying PDEs. Consequently, GPLaSDI is inherently non-intrusive and can be applied to problems without a known PDE or its residual. We demonstrate the effectiveness of our approach on the Burgers equation, Vlasov equation for plasma physics, and a rising thermal bubble problem. Our proposed method achieves between 200 and 100,000 times speed-up, with up to 7% relative error.

1. Introduction

Over the past few decades, the advancement of numerical simulation techniques for understanding physical phenomena has been remarkable, leading to increased sophistication and accuracy. Simultaneously, computational hardware has undergone significant improvements, becoming more powerful and affordable. As a result, numerical simulations are now extensively utilized across various domains, including engineering design, digital twins, decision making [1–5], and diverse fields such as aerospace, automotive, electronics, physics, biology [6–13].

In engineering and physics, computational simulations often involve solving partial differential equations (PDEs) through different numerical techniques like finite difference/volume/element methods, particle methods, and more. While these methods have demonstrated their accuracy and capability to provide high-fidelity simulations when applied correctly, they can be computationally demanding. This is particularly evident in time-dependent multiscale problems that encompass complex physical phenomena (e.g., turbulent fluid flows, fusion device plasma dynamics, astrodynamic flows) on highly refined grids or meshes. Consequently,

* Corresponding author.
E-mail address: cpb97@cornell.edu (C. Bonneville).

https://doi.org/10.1016/j.cma.2023.116535
Received 10 August 2023; Received in revised form 7 October 2023; Accepted 9 October 2023
Available online 14 October 2023

performing a large number of forward simulations using high-fidelity solvers can pose significant computational challenges, especially in situations like uncertainty quantification [14–16], inverse problems [14,16–18], design optimization [19,20], and optimal control [21].

The computational bottleneck associated with high-fidelity simulations has prompted the development of reduced-order models (ROMs). The primary objective of ROMs is to simplify the computations involved in the full-order model (i.e., the high-fidelity simulation) by reducing the order of the problem. Although ROM predictions are generally less accurate, they offer significantly faster results, making them highly appealing when a slight decrease in accuracy is tolerable. Several ROM techniques rely on the projection of snapshot data from the full-order model.

Linear projection methods such as the proper orthogonal decomposition (POD) [22], the reduced basis method [23], and the balanced truncation method [24] have gained popularity in ROM applications. These linear-subspace methods have been successfully applied to various equations such as the Burgers and Navier–Stokes equations [25–28], Lagrangian hydrodynamics [29,30], advection–diffusion problems [31,32], and design optimization [33,34]. More recently, non-linear projection methods [35–38] utilizing autoencoders [39,40] have emerged as an alternative and have shown superior performance in advection-dominated problems [35,41,42].

Data-driven projection-based ROM methods can generally be categorized into two types: intrusive and non-intrusive. Intrusive ROMs are physics-informed models that require knowledge of the governing equation [23,26,28,29,32,36–38,43]. This characteristic enhances the robustness of predictions and often necessitates less data from the full-order model (FOM). However, intrusive ROMs also demand access to the FOM solver and specific implementation details, such as the discretized residual of the PDE.

In contrast, non-intrusive ROMs are independent of the governing equation and rely solely on data-driven techniques. These methods typically utilize interpolation techniques to map parameters to their corresponding ROM predictions [44–47]. However, being purely black-box approaches, these methods lack interpretability and robustness. Sometimes, they struggle to generalize accurately and may exhibit limitations in their performance.

To overcome these challenges, recent approaches have focused on combining projection methods with latent space dynamics learning. These approaches view the latent space as a dynamical system governed by a set of ordinary differential equations (ODEs). By accurately identifying these ODEs, the dynamics of the latent space can be predicted and projected back into the space of full-order solutions.

Various methods have been proposed for identifying governing equations from data [48,49], including the widely used *Sparse Identification of Non-Linear Dynamics* (SINDy) [50]. SINDy constructs a library of terms that could potentially be part of the governing ODEs and estimates the associated coefficients using linear regression. This method has gained popularity and has been extended to a broad range of SINDy-based identification algorithms [51–59].

Champion et al. [60] introduced an approach that combines an autoencoder with SINDy to identify sets of ODEs governing the dynamics of the latent space. While promising, the identified ODEs are not parameterized based on simulation parameters, limiting the method's generalizability. Bai and Peng [61] proposed a similar approach but with a linear projection using proper orthogonal decomposition (POD). They also introduced parameterization of the latent space ODEs, enabling ROM predictions for any point in the parameter space. However, this method exhibits limitations when applied to advection-dominated problems due to the constraints of POD.

Fries et al. [41] proposed a framework called the Latent Space Dynamic Identification (LaSDI), which combines autoencoders with a parametric SINDy identification of the latent space. In LaSDI, a set of ODEs corresponding to each training data point from the full-order model (FOM) is estimated. The coefficients of each ODE are then interpolated based on the FOM parameters using radial basis functions (RBF). However, the sequential training of the autoencoder and SINDy identifications in LaSDI can sometimes lead to a complex latent space with poorly conditioned and/or inaccurate sets of governing ODEs. Additionally, the training FOM dataset in LaSDI is generated on a predefined grid in the parameter space, which may not be optimal. The uniform grid may result in insufficient data in certain regions of the parameter space and excessive data in others, affecting the accuracy of the model.

To address these issues, He et al. [42] introduced the greedy-LaSDI (gLaSDI) framework. In gLaSDI, the autoencoder and SINDy latent space identifications are trained simultaneously, and the FOM data points are sampled sequentially during training. The training starts with only a few points, and at a fixed sampling rate, the model's prediction accuracy is evaluated by plugging the decoder output into the PDE residual. The parameter yielding the largest residual error is selected as the new sampling point, and a FOM simulation is performed to obtain the solution, which is then added to the training dataset. This iterative process ensures that the model focuses on regions of the parameter space where accuracy is needed. Although gLaSDI is robust and accurate, it inherently requires knowledge of the PDE residual, making it an intrusive ROM method. Therefore, gLaSDI may not be suitable for problems where the residual is unknown, difficult to implement, or computationally expensive to evaluate.

In this paper, we present the (greedy) Gaussian-Process Latent Space Identification (GPLaSDI), a non-intrusive extension of the greedy LaSDI framework. One of the main sources of error in LaSDI and gLaSDI arises from potential inaccuracies in the interpolation of the ODE coefficients. To address this issue, we propose replacing deterministic interpolation methods (such as RBF interpolation in LaSDI [41] and k-NN convex interpolation in gLaSDI [42]) with a Bayesian interpolation technique called Gaussian Process (GP) [62].

A GP can provide confidence intervals for its predictions, allowing us to quantify the uncertainty in the sets of ODEs governing the latent space. This uncertainty can then be propagated to the decoder, enabling the generation of ROM prediction confidence intervals for any test point in the parameter space. We adopt a sequential procedure for sampling additional FOM training data, similar to gLaSDI. However, instead of relying on the PDE residual to select the next parameter for sampling, we choose the parameter that yields the highest predictive uncertainty.

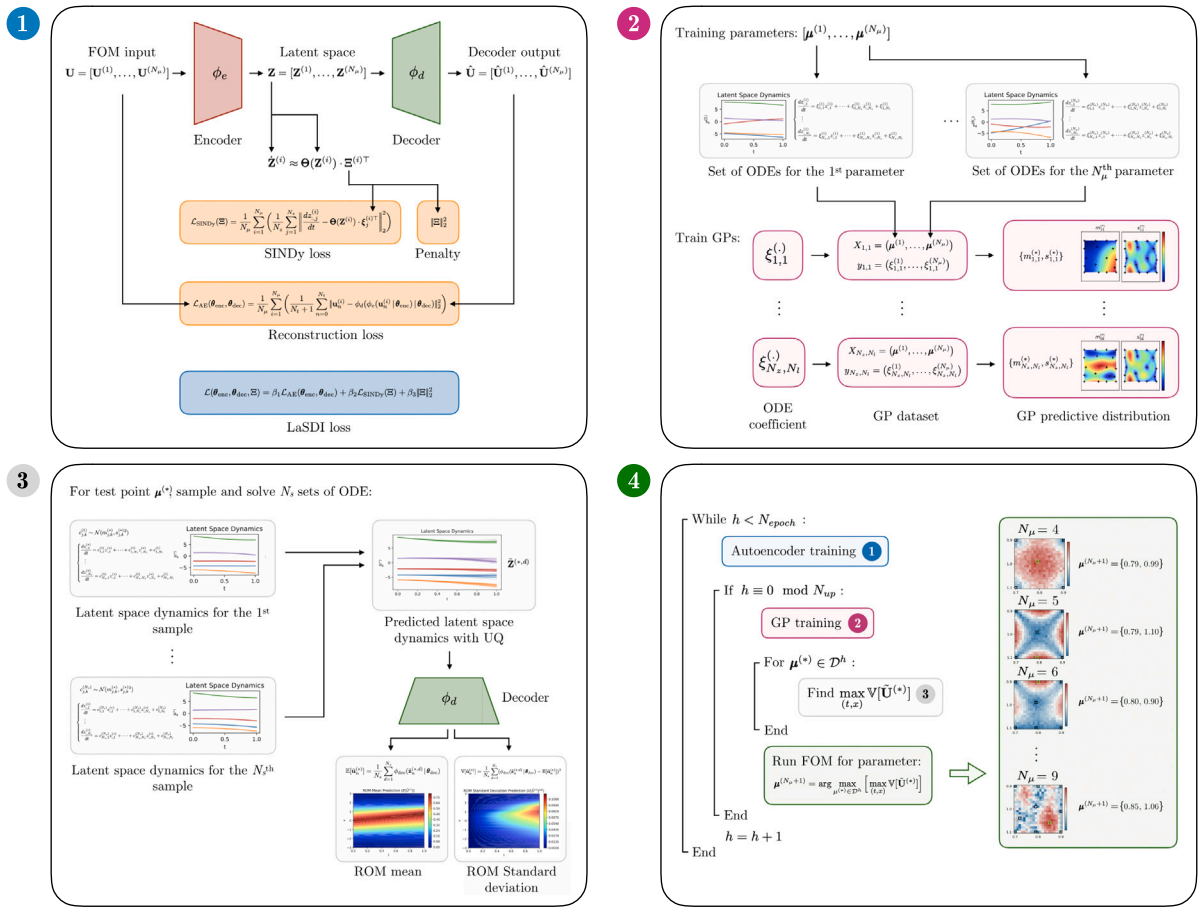


Fig. 1. GPLaSDI general framework. (1) Combined training of the autoencoder and the SINDy latent space identification. (2) Interpolation of the latent space governing sets of ODEs using Gaussian Processes. (3) ROM Prediction methodology using GPLaSDI. (4) FOM training data greedy sampling algorithm, using (1), (2) and (3).

This approach offers two significant advantages. First, it makes our framework fully independent of the specific PDE and its residual, making it applicable to a wide range of problems. Second, our method can generate meaningful confidence intervals for ROM predictions, providing valuable information for assessing the reliability of the simulations.

The details of the GPLaSDI framework are presented in Section 2, where we introduce the utilization of autoencoders in Section 2.2, the application of SINDy in Section 2.3, the adoption of GP interpolation in Section 2.4, and the incorporation of variance-based greedy sampling in Section 2.6. The overall structure of the GPLaSDI framework is summarized in Fig. 1, and the algorithmic procedure is outlined in Algorithm 1. To demonstrate the effectiveness of GPLaSDI, we provide case studies in Section 3. Specifically, we investigate the application of GPLaSDI to the 1D Burgers equation in Section 3.1, the 2D Burgers equation in Section 3.2, the 1D1V Vlasov equation for plasma physics in Section 3.3, and the 2D rising thermal bubble problem in Section 3.4. Through these examples, we highlight the performance and capabilities of the GPLaSDI framework. In conclusion, we summarize the key findings and contributions of the paper in Section 4, providing a comprehensive overview of the work conducted in this study.

2. GPLaSDI framework

2.1. Governing equation of physical systems

In the following sections, we consider physical phenomenons described by governing PDEs with the following form:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(\mathbf{u}, t, x \mid \boldsymbol{\mu}) & (t,x) \in [0, t_{\max}] \times \Omega \\ \mathbf{u}(t=0, x \mid \boldsymbol{\mu}) = \mathbf{u}_0(x \mid \boldsymbol{\mu}) & \boldsymbol{\mu} \in \mathcal{D} \end{cases} \tag{1}$$

Table 1
Table of notations.

Notation	Description
$\mathbf{u}_n^{(i)}$	FOM snapshot at time step n and parameter $\boldsymbol{\mu}^{(i)}$
$\hat{\mathbf{u}}_n^{(i)}$	Reconstructed autoencoder FOM snapshot at time step n and parameter $\boldsymbol{\mu}^{(i)}$
$\tilde{\mathbf{u}}_n^{(*,d)}$	Predicted ROM solution at time step n and test parameter $\boldsymbol{\mu}^{(*)}$
$\mathbf{z}_n^{(i)}$	Latent space representation of snapshot $\mathbf{u}_n^{(i)}$
$\tilde{\mathbf{z}}_n^{(*,d)}$	Predicted latent space dynamics at time step n, for test parameter $\boldsymbol{\mu}^{(*)}$ and sample d
$\boldsymbol{\mu}^{(i)}/\boldsymbol{\mu}^{(*)}$	Training parameter/Test parameter
ϕ_e/ϕ_d	Encoder/Decoder
$\boldsymbol{\theta}_{\text{enc}}/\boldsymbol{\theta}_{\text{dec}}$	Encoder weights/Decoder weights
$\boldsymbol{\Xi}^{(i)} = \{\xi_{j,k}^{(i)}\}$	SINDy coefficients for the set of ODEs associated to parameter $\boldsymbol{\mu}^{(i)}$
$\boldsymbol{\theta}_{\text{gp}} = \{\boldsymbol{\theta}_{\text{gp}}^{j,k}\}$	Sets of GP hyperparameters
$\boldsymbol{\Theta}(\cdot)$	SINDy dictionary
$\mathcal{D}/\mathcal{D}^h$	Parameter space/Discretized parameter space
$[\![1, N]\!]$	A set of integers, i.e., $\{1, \dots, N\}$
n_μ	Dimension of $\mathcal{D}$
N_u	Number of degrees of freedom in the FOM solution
N_t	Number of time steps in the FOM solution
N_μ	Number of parameters in the training dataset
N_z	Number of variables in the latent space
N_l	Number of SINDy candidates
N_s	Number of samples taken from each GP predictive distribution
N_{epoch}	Number of training epochs
N_{up}	Greedy sampling rate
$\beta_1/\beta_2/\beta_3$	Loss hyperparameters
$m_{j,k}^{(*)}/s_{j,k}^{(*)}$	Predictive mean/standard deviation of each GP
$c_{j,k}^{(d)}$	Sample d from $\mathcal{N}(m_{j,k}^{(*)}/s_{j,k}^{(*)2})$
n	Dummy variable for time step indexing
i	Dummy variable for parameter indexing
j	Dummy variable for latent variable indexing
k	Dummy variable for SINDy candidate indexing
d	Dummy variable for GP sampling indexing
h	Dummy variable for epoch count

In Eq. (1), the solution $\mathbf{u}$ may either represent a scalar or vector field, defined over the time–space domain $[0, t_{\text{max}}] \times \Omega$. The spatial domain Ω may be of any dimensions, and the differential operator $\mathbf{f}$ may contain linear and/or non–linear combinations of spatial derivatives and source terms. The governing equations and their initial/boundary conditions are parameterized by a parameter vector $\boldsymbol{\mu}$. The parameter space is defined as $\mathcal{D} \subseteq \mathbb{R}^{n_\mu}$ and may take any dimension (in the following sections, we consider a 2D-parameter space, i.e. $n_\mu = 2$). For a given parameter vector $\boldsymbol{\mu}^{(i)} \in \mathcal{D}$, we assume in the following sections that we have access to the corresponding discretized solution of Eq. (1), denoted as $\mathbf{U}^{(i)}$. $\mathbf{U}^{(i)}$ is a matrix of concatenated snapshots $\mathbf{u}_n^{(i)}$ at each time step n, such that $\mathbf{U}^{(i)} = [\mathbf{u}_0^{(i)}, \dots, \mathbf{u}_{N_t}^{(i)}]^\top \in \mathbb{R}^{(N_t+1)\times N_u}$. This solution is obtained using either a full order model solver or an experiment. In the following sections, every FOM solutions are organized into a 3rd order tensor dataset $\mathbf{U} \in \mathbb{R}^{N_\mu \times (N_t+1) \times N_u}$ of the form:

$$\mathbf{U} = [\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(N_\mu)}]_{N_\mu \times (N_t+1) \times N_u} \tag{2}$$

N_t and N_u are the number of time steps and degrees of freedom, respectively, and N_μ is the number of available FOM solutions. The notations employed in this paper are summarized in Table 1.

2.2. Autoencoders

An autoencoder [39,63] is a neural network designed specifically for compressing large datasets by reducing the dimensions of the data through nonlinear transformation. It consists of two stacked neural networks: the encoder, denoted as ϕ_e and parameterized by $\boldsymbol{\theta}_{\text{enc}}$, and the decoder, denoted as ϕ_d and parameterized by $\boldsymbol{\theta}_{\text{dec}}$. The encoder takes an input data snapshot $\mathbf{u}_n^{(i)} \in \mathbb{R}^{N_u}$ and produces a compressed representation $\mathbf{z}_n^{(i)} \in \mathbb{R}^{N_z}$ in a latent space. Here, N_z represents the number of latent space variables, which is an arbitrary design choice, but typically chosen such that $N_z \ll N_u$. Similar to the matrix $\mathbf{U}^{(i)}$, we concatenate the latent representations at each time step into a matrix $\mathbf{Z}^{(i)} = [\mathbf{z}_0^{(i)}, \dots, \mathbf{z}_{N_t}^{(i)}]^\top \in \mathbb{R}^{(N_t+1)\times N_z}$. The latent variables for each time step and each parameter $\boldsymbol{\mu}^{(i)}$ are stored in a third-order tensor $\mathbf{Z} \in \mathbb{R}^{N_\mu \times (N_t+1) \times N_z}$, analogous to the tensor $\mathbf{U}$, and have the following form:

$$\mathbf{Z} = [\mathbf{Z}^{(1)}, \dots, \mathbf{Z}^{(N_\mu)}]_{N_\mu \times (N_t+1) \times N_z} \tag{3}$$

The decoder takes each $\mathbf{z}_n^{(i)}$ as input and generates a reconstructed version of $\mathbf{u}_n^{(i)}$, denoted as $\hat{\mathbf{u}}_n^{(i)}$.

$$\begin{aligned}\mathbf{z}_n^{(i)} &= \phi_e(\mathbf{u}_n^{(i)} \mid \boldsymbol{\theta}_{\text{enc}})\\ \hat{\mathbf{u}}_n^{(i)} &= \phi_d(\mathbf{z}_n^{(i)} \mid \boldsymbol{\theta}_{\text{dec}})\end{aligned} \tag{4}$$

The parameters of the autoencoder, denoted as $\boldsymbol{\theta}_{\text{enc}}$ and $\boldsymbol{\theta}_{\text{dec}}$, are learned using a numerical optimization algorithm that minimizes the L_2 norm of the difference between the set of input solutions $\mathbf{U}$ and the set of reconstructed solutions $\hat{\mathbf{U}}$. This is achieved by defining the reconstruction loss as follows:

$$\mathcal{L}_{\text{AE}}(\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}) = \|\mathbf{U} - \hat{\mathbf{U}}\|_2^2 = \frac{1}{N_\mu}\sum_{i=1}^{N_\mu}\left(\frac{1}{N_t+1}\sum_{n=0}^{N_t}\|\mathbf{u}_n^{(i)} - \phi_d(\phi_e(\mathbf{u}_n^{(i)} \mid \boldsymbol{\theta}_{\text{enc}}) \mid \boldsymbol{\theta}_{\text{dec}})\|_2^2\right) \tag{5}$$

2.3. Identification of latent space dynamics

The encoder performs compression of the high-dimensional physical data, e.g., a solution of PDEs, defined over space and time, into a reduced set of discrete latent variables that are defined solely over time. Consequently, the latent space can be regarded as a dynamical system governed by sets of ordinary differential equations (ODEs). This characteristic forms a fundamental aspect of LaSDI (Latent Space Dynamics Identification) algorithms, which enable the compression of dynamical systems governed by PDEs into systems governed by ODEs [41]. At each time step, the dynamics of the latent variables in the latent space can be described by an equation of the following form:

$$\frac{d\mathbf{z}_n^{(i)}}{dt} = \psi_{DI}(\mathbf{z}_n^{(i)} \mid \boldsymbol{\mu}^{(i)}) \qquad (i,n) \in [\![1, N_\mu]\!] \times [\![0, N_t]\!] \tag{6}$$

This equation can be unified across all time steps as:

$$\dot{\mathbf{Z}}^{(i)} = \psi_{DI}(\mathbf{Z}^{(i)} \mid \boldsymbol{\mu}^{(i)}) \tag{7}$$

The system of ODEs governing the dynamics of the latent space can be determined using a technique called SINDy (Sparse Identification of Nonlinear Dynamics) [50]. In SINDy, a dictionary $\boldsymbol{\Theta}(\mathbf{Z}^{(i)}) \in \mathbb{R}^{(N_t+1)\times N_l}$ is constructed, consisting of N_l linear and nonlinear candidate terms that may be involved in the set of ODEs. The approach assumes that the time derivatives $\dot{\mathbf{Z}}$ can be expressed as a linear combination of these candidate terms. Hence, the right-hand side of Eq. (7) can be approximated as follows:

$$\dot{\mathbf{Z}}^{(i)} \approx \boldsymbol{\Theta}(\mathbf{Z}^{(i)}) \cdot \boldsymbol{\Xi}^{(i)\top} \tag{8}$$

where $\boldsymbol{\Xi}^{(i)} \in \mathbb{R}^{N_z \times N_l}$ denotes a coefficient matrix. The selection of terms in $\boldsymbol{\Theta}(\cdot)$ is arbitrary. Including a broader range of SINDy terms may potentially capture the latent dynamics more accurately, but it can also result in sets of ODEs that are more challenging to solve numerically. In the subsequent sections, we will typically limit the dictionary to constant and linear terms. Interestingly, we have found that this restricted choice of terms is generally adequate to achieve satisfactory performance:

$$\boldsymbol{\Theta}(\mathbf{Z}^{(i)}) = \begin{bmatrix} 1 & \mathbf{z}_0^{(i)} \\ \vdots & \vdots \\ 1 & \mathbf{z}_{N_t}^{(i)} \end{bmatrix} = \begin{bmatrix} 1 & z_{0,1}^{(i)} & \cdots & z_{0,j}^{(i)} & \cdots & z_{0,N_z}^{(i)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & z_{N_t,1}^{(i)} & \cdots & z_{N_t,j}^{(i)} & \cdots & z_{N_t,N_z}^{(i)} \end{bmatrix}_{(N_t+1)\times N_l} \tag{9}$$

In the system, $z_{n,j}^{(i)}$ represents the jth latent variable for parameter $\boldsymbol{\mu}^{(i)}$ at time step n, such that $\mathbf{z}_n^{(i)} = [z_{n,1}^{(i)}, \ldots, z_{n,j}^{(i)}, \ldots, z_{n,N_z}^{(i)}]$. Sparse linear regressions are conducted between $\boldsymbol{\Theta}(\mathbf{Z}^{(i)})$ and $dz_{:,j}^{(i)}/dt$ for each $j \in [\![1, N_z]\!]$. The resulting coefficients associated with the SINDy terms are stored in a vector $\boldsymbol{\xi}_j^{(i)} = [\xi_{j,1}^{(i)}, \ldots, \xi_{j,N_l}^{(i)}] \in \mathbb{R}^{N_l}$. More formally, the system of SINDy regressions is expressed as:

$$\frac{dz_{:,j}^{(i)}}{dt} = \begin{bmatrix} \dot{z}_{0,j}^{(i)} \\ \vdots \\ \dot{z}_{N_t,j}^{(i)} \end{bmatrix} = \boldsymbol{\Theta}(\mathbf{Z}^{(i)}) \cdot \boldsymbol{\xi}_j^{(i)\top} \tag{10}$$

Each $\boldsymbol{\xi}_j^{(i)}$ is concatenated into a coefficient matrix $\boldsymbol{\Xi}^{(i)} = [\boldsymbol{\xi}_1^{(i)}, \ldots, \boldsymbol{\xi}_{N_z}^{(i)}]^\top \in \mathbb{R}^{N_z \times N_l}$. In this study, the time derivatives $\dot{z}_{n,j}^{(i)}$ are estimated using a first-order finite difference with a time step Δt that matches the time step used in the full order model (FOM) data. As there is a distinct set of ODEs governing the dynamics of the latent space for each parameter $\boldsymbol{\mu}^{(i)} \in \mathcal{D}$, multiple SINDy regressions are performed concurrently to obtain the corresponding sets of ODE coefficients $\boldsymbol{\Xi}^{(i)}$, where $i \in [\![1, N_\mu]\!]$. The collection of ODE coefficient matrices $\boldsymbol{\Xi} = [\boldsymbol{\Xi}^{(1)}, \ldots, \boldsymbol{\Xi}^{(N_\mu)}] \in \mathbb{R}^{N_\mu \times N_z \times N_l}$, associated with each system of ODEs, is determined by minimizing the following mean-squared-error SINDy loss:

$$\mathcal{L}_{\text{SINDy}}(\boldsymbol{\Xi}) = \|\dot{\mathbf{Z}} - \hat{\dot{\mathbf{Z}}}\|_2^2 = \frac{1}{N_\mu}\sum_{i=1}^{N_\mu}\left(\frac{1}{N_z}\sum_{j=1}^{N_z}\left\|\frac{dz_{:,j}^{(i)}}{dt} - \boldsymbol{\Theta}(\mathbf{Z}^{(i)}) \cdot \boldsymbol{\xi}_j^{(i)\top}\right\|_2^2\right) \tag{11}$$

In the LaSDI framework, the autoencoder and the SINDy sparse regressions are jointly trained using a single loss function. To prevent extreme values for the SINDy coefficients, which can result in ill-conditioned sets of ODEs, a penalty term is incorporated into the loss. Thus, the LaSDI loss function is defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}, \boldsymbol{\Xi}) = \beta_1 \mathcal{L}_{\text{AE}}(\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}) + \beta_2 \mathcal{L}_{\text{SINDy}}(\boldsymbol{\Xi}) + \beta_3 \|\boldsymbol{\Xi}\|_2^2, \tag{12}$$

where β_1, β_2, and β_3 are weighting hyperparameters. Note that another reconstruction term to the loss function can be added, namely the L_2 norm between the velocity $\dot{\boldsymbol{U}}$ and the reconstructed velocity $\dot{\hat{\boldsymbol{U}}}$ for additional stability and accuracy. However, doing so requires to have access to $\dot{\boldsymbol{U}}$, and computing $\dot{\hat{\boldsymbol{U}}}$ requires to backpropagate the autoencoder a second time [42]. This can be expensive, especially with deeper autoencoders. Consequently in this paper, we do not use such loss term and only rely on the penalty term to maintain stability of the SINDy coefficients. As shown in the example sections, this is sufficient to obtain satisfactory accuracy. The autoencoder and SINDy coefficients can be simultaneously trained by minimizing Eq. (12) using a gradient-descent-based optimizer. In the following sections, we utilize the Adam optimizer [64] with a learning rate of α.

2.4. Parameterization through coefficient interpolation

The autoencoder learns a mapping to and from the latent space and identifies the set of ODEs governing the latent dynamics, but it does so only for each parameter $\boldsymbol{\mu}_i \in \mathcal{D}$ associated with each training data point $\mathbf{U}^{(i)}$ (with $i \in [\![1, N_\mu]\!]$). To make prediction for any new parameter $\boldsymbol{\mu}^{(*)}$, the system of ODEs associated to this later parameter value needs to be estimated. This can be done by finding a mapping $f : \boldsymbol{\mu}^{(*)} \mapsto \boldsymbol{\Xi}^{(*)}$, where (j,k)th element of $\boldsymbol{\Xi}^{(*)}$ is denoted as $\{\xi^{(*)}_{j,k}\}_{(j,k)\in[\![1,N_z]\!]\times[\![1,N_l]\!]}$. In LaSDI and gLaSDI, f is estimated through RBF interpolation [41] or $k-$NN convex interpolation [42] of each pair of data $(\boldsymbol{\mu}^{(i)}, \{\xi^{(i)}_{j,k}\})_{i\in[\![1,N_\mu]\!]}$. In this paper, we introduce a replacement for the deterministic interpolation methods, utilizing GP regression [62]. The utilization of GPs provides three significant advantages, which will be elaborated in the subsequent subsections:

- GPs have the inherent capability to automatically quantify interpolation uncertainties by generating confidence intervals for their predictions.
- GPs are known for their robustness to noise, making them less prone to overfitting and mitigating the risk of incorporating incorrect SINDy coefficients.
- Being part of the family of Bayesian methods, GPs allow us to incorporate prior knowledge about the latent space behavior, particularly in terms of smoothness. By specifying a prior distribution, we can provide useful hints that enhance interpolation accuracy within the GP framework.

The subsequent two subsections cover essential background information on Gaussian processes (GPs) and outline their specific utilization within the context of this paper. In the first subsection, we provide an overview of GPs, while the second subsection focuses on detailing the application of GPs in our approach.

2.4.1. Gaussian processes

Let us consider a dataset consisting of N_μ input–output pairs in the form of $S = (X, y)$. Here, X represents a set of input vectors, and y represents the corresponding continuous scalar output. We assume that the output y may be affected by Gaussian noise with a variance of σ^2. Our objective is to find a mapping function f such that:

$$y = f(X) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I_{N_\mu}) \tag{13}$$

In the Gaussian process (GP) paradigm, the function f is assumed to be drawn from a Gaussian prior distribution (Eq. (14)) [62]. In the literature, it is common to set the mean of the prior to 0, while the covariance is determined by a kernel function k with parameters $\boldsymbol{\theta}_{\text{gp}}$. The choice of kernel is arbitrary, and in this context, we adopt the radial basis function (RBF) kernel (Eq. (15)). This choice is particularly suitable because the RBF kernel can effectively approximate any C^∞ function. Furthermore, there is no a priori reason to assume that the space of ordinary differential equation (ODE) coefficients lacks smoothness.

$$p(f \mid X) = \mathcal{N}(f \mid 0, k(X, X \mid \boldsymbol{\theta}_{\text{gp}})) \tag{14}$$

$$k(X, X \mid \boldsymbol{\theta}_{\text{gp}}) = \gamma \exp\left(-\frac{\|X - X^\top\|_2^2}{2\lambda^2}\right) \qquad \boldsymbol{\theta}_{\text{gp}} = \{\gamma, \lambda\} \tag{15}$$

The likelihood (Eq. (16)) is obtained as a direct consequence of Eq. (13) and by applying Bayes' rule in combination with the prior distribution. This allows us to derive the posterior distribution over f (Eq. (17)).

$$p(y \mid f, X) = \mathcal{N}(y \mid f(X), \sigma^2 I_{N_\mu}) \tag{16}$$

$$p(f \mid X, y) = \frac{p(y \mid f, X) p(f \mid X)}{p(y \mid X)} \tag{17}$$

The denominator in Eq. (17), often referred to as the marginal likelihood, plays a crucial role in Bayesian inference. Typically, the hyperparameters $\boldsymbol{\theta}_{\text{gp}}$ are selected to maximize this marginal likelihood. Alternatively, it is common to minimize the negative log–marginal–likelihood in Eq. (19), which is equivalent but computationally more efficient in practice.

$$p(y \mid X) = \int p(y \mid f, X) p(f \mid X) df \tag{18}$$

$$\mathcal{L}_{\mathrm{GP}}(\boldsymbol{\theta}_{\mathrm{gp}}) = -\log(p(y \mid X)) \tag{19}$$

Unlike parametric Bayesian machine learning models such as Bayesian neural networks, where the posterior is defined over the model parameter space, Gaussian process models define the posterior over the function space. Consequently, predicting outputs for test input points is not as straightforward. To derive the distribution over the predictive output $y^{(*)}$ for a test input $x^{(*)}$, we need to apply both the sum rule and the product rule in combination:

$$p(y^{(*)} \mid X, y, x^{(*)}) = \int p(y^{(*)} \mid f, x^{(*)}) p(f \mid X, y) df \tag{20}$$

In the present scenario, Bayesian inference proves to be highly tractable due to the Gaussian nature of the posterior distribution [62]. This implies that both the posterior distribution and Eq. (20) can be computed analytically, resulting in Gaussian distributions for both:

$$p(y^{(*)} \mid X, y, x^{(*)}) = \mathcal{N}(y^{(*)} \mid m^{(*)}, s^{(*)2}) \tag{21}$$

$$\begin{cases} m^{(*)} = k(x^{(*)}, X \mid \boldsymbol{\theta}_{\mathrm{gp}})(k(X, X \mid \boldsymbol{\theta}_{\mathrm{gp}}) + \sigma^2 I_{N_\mu})^{-1} y \\ s^{(*)2} = k(x^{(*)}, x^{(*)} \mid \boldsymbol{\theta}_{\mathrm{gp}}) - k(x^{(*)}, X \mid \boldsymbol{\theta}_{\mathrm{gp}})(k(X, X \mid \boldsymbol{\theta}_{\mathrm{gp}}) + \sigma^2 I_{N_\mu})^{-1} k(X, x^{(*)} \mid \boldsymbol{\theta}_{\mathrm{gp}}) \end{cases} \tag{22}$$

2.4.2. GP interpolation of the SINDy coefficients

After training the autoencoder and obtaining the SINDy coefficients, we proceed to construct $N_{gp} = N_z \times N_l$ regression datasets, where each dataset corresponds to an ODE coefficient and consists of N_μ data points: $(X^{(i)}_{j,k}, y^{(i)}_{j,k}) = (\boldsymbol{\mu}^{(i)}, \{\xi^{(i)}_{j,k}\})_{i \in [\![0, N_\mu]\!]}$. Subsequently, a Gaussian Process (GP) is trained for each dataset. For a given test parameter $\boldsymbol{\mu}^{(*)}$, the predictive mean and standard deviation are denoted as $m^{(*)}_{j,k}$ and $s^{(*)}_{j,k}$, respectively. To illustrate, let us consider a LaSDI system that solely incorporates constant and linear candidate terms (i.e., $N_l = N_z + 1$). The system of ODEs, considering a 1-standard deviation uncertainty for the test parameter $\boldsymbol{\mu}^{(*)}$, is as follows:

$$\begin{cases} \dfrac{dz^{(*)}_{:,1}}{dt} = \left(m^{(*)}_{1,1} \pm s^{(*)}_{1,1}\right) z^{(*)}_{:,1} + \cdots + \left(m^{(*)}_{1,N_z} \pm s^{(*)}_{1,N_z}\right) z^{(*)}_{:,N_z} + \left(m^{(*)}_{1,N_l} \pm s^{(*)}_{1,N_l}\right) \\ \vdots \\ \dfrac{dz^{(*)}_{:,N_z}}{dt} = \left(m^{(*)}_{N_z,1} \pm s^{(*)}_{N_z,1}\right) z^{(*)}_{:,1} + \cdots + \left(m^{(*)}_{N_z,N_z} \pm s^{(*)}_{N_z,N_z}\right) z^{(*)}_{:,N_z} + \left(m^{(*)}_{N_z,N_l} \pm s^{(*)}_{N_z,N_l}\right) \end{cases} \tag{23}$$

with:

$$\begin{cases} m^{(*)}_{j,k} = k(\boldsymbol{\mu}^{(*)}, X_{j,k} \mid \boldsymbol{\theta}^{j,k}_{\mathrm{gp}})(k(X_{j,k}, X_{j,k} \mid \boldsymbol{\theta}^{j,k}_{\mathrm{gp}}) + \sigma^2 I_{N_\mu})^{-1} y_{j,k} \\ s^{(*)2}_{j,k} = k(\boldsymbol{\mu}^{(*)}, \boldsymbol{\mu}^{(*)} \mid \boldsymbol{\theta}^{j,k}_{\mathrm{gp}}) - k(\boldsymbol{\mu}^{(*)}, X_{j,k} \mid \boldsymbol{\theta}^{j,k}_{\mathrm{gp}})(k(X_{j,k}, X_{j,k} \mid \boldsymbol{\theta}^{j,k}_{\mathrm{gp}}) + \sigma^2 I_{N_\mu})^{-1} k(X_{j,k}, \boldsymbol{\mu}^{(*)} \mid \boldsymbol{\theta}^{j,k}_{\mathrm{gp}}) \end{cases} \tag{24}$$

GP training typically faces scalability challenges when dealing with large datasets due to the need to invert the $N_\mu \times N_\mu$ kernel matrix, which has a computational complexity of $\mathcal{O}(N_\mu^3)$. While the computational cost of training the GPs should only become an issue which larger values of N_μ (e.g. $N_\mu \gg 500$), GP scalability can be addressed in different ways. One approach is to partition the parameter space into multiple subdomains and construct a separate GP for each subdomain [65]. This strategy effectively mitigates the computational burden associated with GPs when dealing with large datasets. Another approach is to use approximate scalable GP methods such as kernel interpolation [66,67] or hyperparameter cross-validation [68].

2.5. Predicting solutions

By employing the GP interpolation approach discussed in the preceding sub-section, we are able to make predictions for the governing equations of the latent space dynamics, along with associated uncertainty, for any test point $\boldsymbol{\mu}^{(*)} \in \mathcal{D}$ within the parameter space. Consequently, we can now proceed with ease to generate Reduced Order Model (ROM) predictions using the following steps:

- Compute the set of $\{m^{(*)}_{j,k}, s^{(*)}_{j,k}\}$ for each $(j,k) \in [\![1, N_z]\!] \times [\![1, N_l]\!]$ and determine the governing Ordinary Differential Equations (ODEs), utilizing Eq. (23) and Eq. (24), respectively.
- Convert the initial conditions $\mathbf{u}^{(*)}_0$ into latent space initial conditions $\mathbf{z}^{(*)}_0$ by performing a forward pass through the encoder network:

$$\mathbf{z}^{(*)}_0 = \phi_{\mathrm{enc}}(\mathbf{u}^{(*)}_0 \mid \boldsymbol{\theta}_{\mathrm{enc}}) \tag{25}$$

- Perform the simulation of the latent space dynamics by numerically solving the system of ODEs using commonly used integration schemes such as backward Euler, Runge–Kutta, or other suitable methods. The resulting simulated latent space dynamics is denoted as $\tilde{\mathbf{Z}}^{(*)} = [\mathbf{z}^{(*)}_0, \tilde{\mathbf{z}}^{(*)}_1, \ldots, \tilde{\mathbf{z}}^{(*)}_{N_t}] \in \mathbb{R}^{N_z \times (N_t+1)}$, where N_t represents the number of time steps.

- Utilize the forward pass of $\tilde{\mathbf{Z}}^{(*)}$ through the decoder network to generate predictions of the Full Order Model (FOM) physics. The output of the decoder is denoted as $\tilde{\mathbf{U}}^{(*)} = [\tilde{\mathbf{u}}_0^{(*)}, \dots, \tilde{\mathbf{u}}_{N_t}^{(*)}] \in \mathbb{R}^{N_u \times (N_t+1)}$.

The uncertainty associated with each ODE coefficient is effectively quantified through GP interpolation. While it is possible to utilize the predictive GP mean $m_{j,k}^{(*)}$ for each coefficient to obtain a mean prediction for FOM, an alternative approach is to use random samples. This approach yields multiple sets of ODEs for a single test point $\boldsymbol{\mu}^{(*)}$, resulting in multiple corresponding latent space dynamics, each with varying likelihood. For instance, considering linear and constant candidate terms, we can generate N_s sets of ODEs:

$$\begin{cases} \dfrac{dz_{:,1}^{(*)}}{dt} = c_{1,1}^{(d)} z_{:,1}^{(*)} + \cdots + c_{1,N_z}^{(d)} z_{:,N_z}^{(*)} + c_{1,N_l}^{(d)} \\ \vdots \\ \dfrac{dz_{:,N_z}^{(*)}}{dt} = c_{N_z,1}^{(d)} z_{:,1}^{(*)} + \cdots + c_{N_z,N_z}^{(d)} z_{:,N_z}^{(*)} + c_{N_z,N_l}^{(d)} \end{cases} \tag{26}$$

with:

$$c_{j,k}^{(d)} \sim \mathcal{N}(m_{j,k}^{(*)}, s_{j,k}^{(*)2}) \tag{27}$$

Next, we proceed to solve the corresponding latent space dynamics $\tilde{\mathbf{Z}}^{(*,d)}$ for each sample $d \in [\![1, N_s]\!]$ (in the following sections, we use $N_s = 20$). By making forward passes through the decoder network, we can estimate the uncertainty over the Full Order Model (FOM) prediction:

$$\mathbb{E}[\tilde{\mathbf{u}}_n^{(*)}] = \frac{1}{N_s} \sum_{d=1}^{N_s} \phi_{\text{dec}}(\tilde{\mathbf{z}}_n^{(*,d)} \mid \boldsymbol{\theta}_{\text{dec}}) \qquad n \in [\![0, N_t]\!] \tag{28}$$

$$\mathbb{V}[\tilde{\mathbf{u}}_n^{(*)}] = \frac{1}{N_s} \sum_{d=1}^{N_s} (\phi_{\text{dec}}(\tilde{\mathbf{z}}_n^{(*,d)} \mid \boldsymbol{\theta}_{\text{dec}}) - \mathbb{E}[\tilde{\mathbf{u}}_n^{(*)}])^2 \qquad n \in [\![0, N_t]\!] \tag{29}$$

The expected solution and variance of the ROM are denoted as $\mathbb{E}[\tilde{\mathbf{U}}^{(*)}] = [\mathbb{E}[\tilde{\mathbf{u}}_0^{(*)}], \dots, \mathbb{E}[\tilde{\mathbf{u}}_{N_t}^{(*)}]]$ and $\mathbb{V}[\tilde{\mathbf{U}}^{(*)}] = [\mathbb{V}[\tilde{\mathbf{u}}_0^{(*)}], \dots, \mathbb{V}[\tilde{\mathbf{u}}_{N_t}^{(*)}]]$, respectively.

2.6. Variance-based greedy sampling

In most applications, the accuracy of machine learning models, including the one described in this paper, is expected to improve with the availability of additional training data [69]. Hence, selecting an appropriate parameter value, denoted as $\boldsymbol{\mu}^{(N_\mu+1)}$, for running a FOM simulation and acquiring more training data is crucial. One intuitive approach is to choose a value of $\boldsymbol{\mu}^{(N_\mu+1)}$ that hinders the most the model's ability from providing satisfactory accuracy. To achieve this, it is necessary to accurately assess the model's performance for any given $\boldsymbol{\mu}^{(*)}$. In the gLaSDI framework [42], this evaluation is accomplished by examining the decoder predictions for values of $\boldsymbol{\mu}$ sampled from a discretized grid of the parameter space, denoted as $D^h \subseteq D$. These predictions are then incorporated into the PDE residual. However, this approach requires explicit knowledge of the PDE and can be computationally expensive and cumbersome to implement. In GPLaSDI, we propose an alternative method based on the uncertainty associated with the decoder predictions. Specifically, if the variance, denoted as $\mathbb{V}[\tilde{\mathbf{U}}^{(*)}]$, is large compared to other points in the parameter space, then the expected value, denoted as $\mathbb{E}[\tilde{\mathbf{U}}^{(*)}]$, is more likely to be inaccurate. Consequently, we select $\boldsymbol{\mu}^{(N_\mu+1)}$ such that:

$$\boldsymbol{\mu}^{(N_\mu+1)} = \arg \max_{\boldsymbol{\mu}^{(*)} \in D^h} \left[\max_{(t,x)} \mathbb{V}[\tilde{\mathbf{U}}^{(*)}] \right] \tag{30}$$

In this paper, we evaluate $\mathbb{V}[\tilde{\mathbf{U}}^{(*)}]$ at each and every point $\boldsymbol{\mu}^{(*)}$ of D^h. An alternative approach, faster, but less accurate, is to use only a handful of random points within D^h [42]. As for the acquisition of new data, we adopt a consistent sampling rate (i.e., every fixed N_{up} training iterations). The steps involved in GPLaSDI, as discussed in Section 2, are summarized in Algorithm 1.

3. Application

In the following sections, we demonstrate the effectiveness of our method on multiple examples. To quantify the accuracy of GPLaSDI, we use the maximum relative error, defined as:

$$e(\tilde{\mathbf{U}}^{(*)}, \mathbf{U}^{(*)}) = \max_n \left(\frac{\|\tilde{\mathbf{u}}_n^{(*)} - \mathbf{u}_n^{(*)}\|_2}{\|\mathbf{u}_n^{(*)}\|_2} \right) \tag{31}$$

where $\tilde{\mathbf{u}}_n^{(*)}$ and $\mathbf{u}_n^{(*)}$ are the decoder ROM prediction and the ground truth at each time step, respectively (with GPLaSDI, we take $\tilde{\mathbf{u}}_n^{(*)} \equiv \mathbb{E}[\tilde{\mathbf{u}}_n^{(*)}]$). Each example are trained and tested on a compute node of the Livermore Computing Lassen supercomputer at the Lawrence Livermore National Laboratory, using a NVIDIA V100 (Volta) 64Gb GDDR5 GPU. Our GPLaSDI model is implemented using the open–source libraries `PyTorch` [70] (for the autoencoder component) and `sklearn` (for the GP component) [71]. All the numerical examples shown in this paper can be regenerated by our open source code, GPLaSDI, which is available at GitHub page, i.e., https://github.com/LLNL/GPLaSDI.

Algorithm 1 Autoencoder Training with Variance–based Greedy Sampling

Require: $\mathbf{U} = [\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(N_\mu)}]$, N_μ, N_{epoch}, N_z, N_l, N_{up}, α, β_1, β_2, β_3, $\boldsymbol{\Theta}(\cdot)$, $\mathcal{D}^h$, ϕ_{enc}, ϕ_{dec}

Initialize $\boldsymbol{\theta}_{\text{enc}}$, $\boldsymbol{\theta}_{\text{dec}}$, $\boldsymbol{\Xi}$, $\boldsymbol{\theta}_{\text{gp}}^{j,k}$ randomly, and $h = 0$
while $h < N_{epoch}$ **do**
  Compute $\mathbf{Z} = \phi_{\text{enc}}(\mathbf{U} \,|\, \boldsymbol{\theta}_{\text{enc}})$ and $\hat{\mathbf{U}} = \phi_{\text{dec}}(\mathbf{Z} \,|\, \boldsymbol{\theta}_{\text{dec}})$ (See Eq. (4))
  Compute $\mathcal{L}(\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}, \boldsymbol{\Xi})$ (See Eq. (12))
  Update $\boldsymbol{\theta}_{\text{enc}}$, $\boldsymbol{\theta}_{\text{dec}}$, and $\boldsymbol{\Xi}$ using Adam algorithm, α and $\nabla\mathcal{L}(\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}, \boldsymbol{\Xi})$
  if $h \mod N_{up} \equiv 0$ **then**
    for $(j,k) \in [\![1, N_z]\!] \times [\![1, N_l]\!]$ **do**
      Build dataset $(X_{j,k}, y_{j,k}) = (\boldsymbol{\mu}^{(i)}, \{\xi_{j,k}^{(i)}\})_{i \in [\![1, N_\mu]\!]}$
      Find $\boldsymbol{\theta}_{\text{gp}}^{j,k} = \arg\min \mathcal{L}_{\text{GP}}(\boldsymbol{\theta}_{\text{gp}})$ (See Eq. (19))
    end for
    for $\boldsymbol{\mu}^{(*)} \in \mathcal{D}^h$ **do**
      for $(j,k) \in [\![1, N_z]\!] \times [\![1, N_l]\!]$ **do**
        Compute $\{m_{j,k}^{(*)}, s_{j,k}^{(*)}\}$ (See Eq. (24))
        for $d \in [\![1, N_s]\!]$ **do**
          Sample $c_{j,k}^{(d)} \sim \mathcal{N}(m_{j,k}^{(*)}, s_{j,k}^{(*)2})$ (See Eq. (27))
        end for
      end for
      for $d \in [\![1, N_s]\!]$ **do**
        Build the system of ODEs using $\{c_{j,k}^{(d)}\}_{(j,k) \in [\![1,N_z]\!] \times [\![1,N_l]\!]}$
        Solve for $\tilde{\mathbf{Z}}^{(*,d)}$ and evaluate $\tilde{\mathbf{U}}^{(*,d)} = \phi_{\text{dec}}(\tilde{\mathbf{Z}}^{(*,d)} \,|\, \boldsymbol{\theta}_{\text{dec}})$
      end for
      Compute $\max_{(t,x)} \mathbb{V}[\tilde{\mathbf{U}}^{(*)}]$ (See Eq. (29))
    end for
    Find $\boldsymbol{\mu}^{(N_\mu+1)} = \arg\max_{\mu^{(*)} \in \mathcal{D}^h} \left[\max_{(t,x)} \mathbb{V}[\tilde{\mathbf{U}}^{(*)}] \right]$ (See Eq. (30))
    Collect $\mathbf{U}^{(N_\mu+1)}$ by running the FOM
    Update $\mathbf{U} = [\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(N_\mu)}, \mathbf{U}^{(N_\mu+1)}]$ and $N_\mu = N_\mu + 1$
  end if
  Update $h = h + 1$
end while

3.1. 1D Burgers equation

We first consider the inviscid 1D Burgers Equation, which was initially introduced in [41] and further discussed in [42]:

$$\begin{cases} \dfrac{\partial u}{\partial t} + u \dfrac{\partial u}{\partial x} = 0 & (t,x) \in [0,1] \times [-3,3] \\ u(t, x=3) = u(t, x=-3) \end{cases} \tag{32}$$

The initial condition is parameterized by $\boldsymbol{\mu} = \{a, w\} \in \mathcal{D}$, and the parameter space is defined as $\mathcal{D} = [0.7, 0.9] \times [0.9, 1.1]$:

$$u(t=0,x) = a \exp\left(-\frac{x^2}{2w^2}\right) \qquad \boldsymbol{\mu} = \{a, w\} \tag{33}$$

The FOM solver utilizes an implicit backward Euler time integration scheme and a backward finite difference discretization in space. The spatial stepping is set to $\Delta x = 6 \cdot 10^{-3}$ and the time stepping to $\Delta t = 10^{-3}$. In this section and in Section 3.2, the discretization is based on the one used in [41,42], and is chosen to ensure stability at any point of the parameter space. For the purpose of greedy sampling, the parameter space is discretized into a square grid $\mathcal{D}^h$ with a stepping of $\Delta a = \Delta w = 0.01$, resulting in a total of 441 grid points (21 values in each dimension). The initial training dataset consists of $N_\mu = 4$ FOM simulations, corresponding to the parameters located at each corner of $\mathcal{D}^h$. Specifically, the parameter values are $\boldsymbol{\mu}^{(1)} = \{0.7, 0.9\}$, $\boldsymbol{\mu}^{(2)} = \{0.9, 0.9\}$, $\boldsymbol{\mu}^{(3)} = \{0.7, 1.1\}$ and $\boldsymbol{\mu}^{(4)} = \{0.9, 1.1\}$.

The encoder architecture follows a 1001–100–5 structure, comprising one hidden layer with 100 hidden units and $N_z = 5$ latent variables. The decoder has a symmetric configuration to the encoder. It employs a sigmoid activation function. Note that in this paper, we select the autoencoder architecture through a random search using only the initial four corner training datapoints, on trainings that are run for at most N_{up} epochs (i.e. no sampling of additional FOM data). This is done to ensure that the training loss

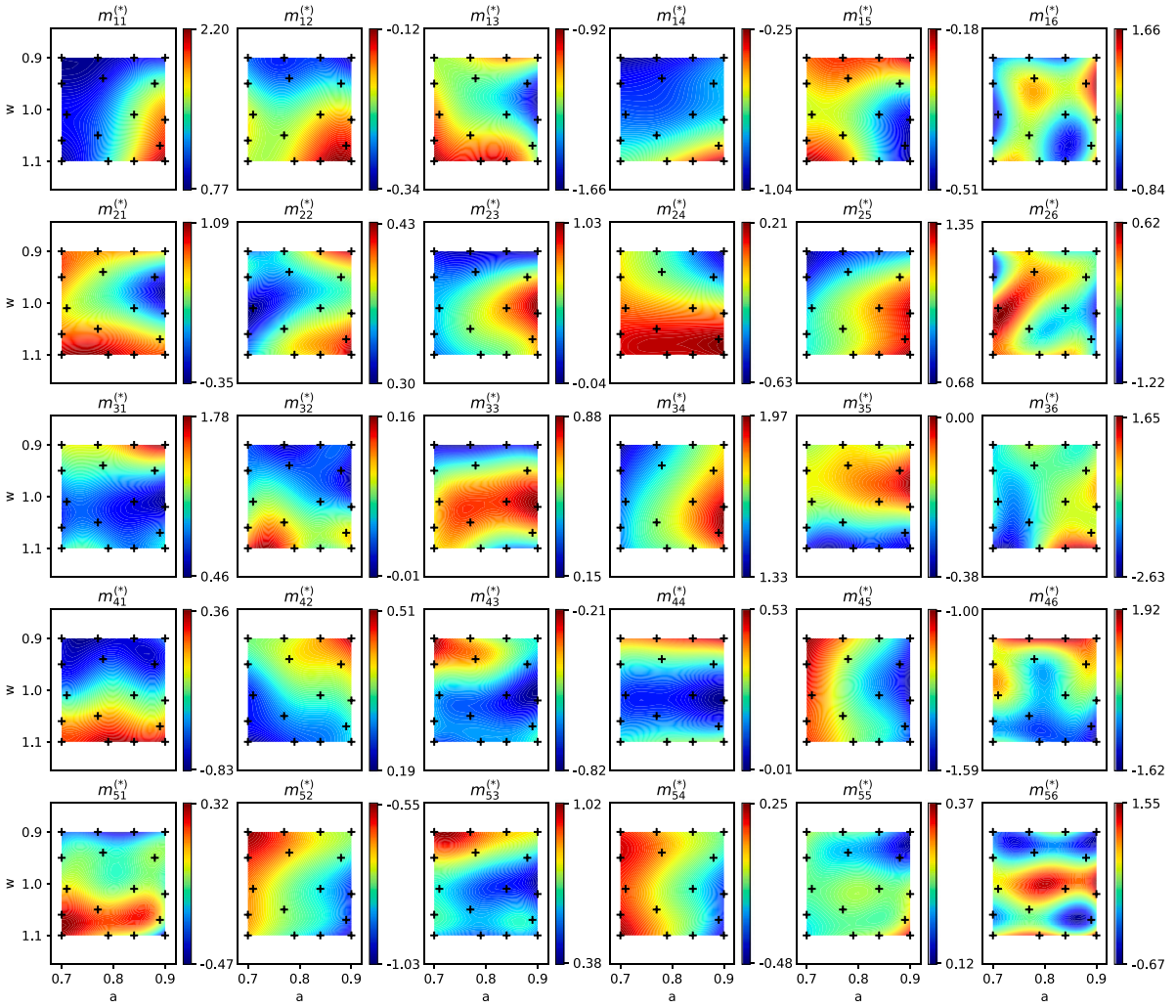


Fig. 2. 1D Burgers Equation — Predictive mean of each ODE coefficients ($m^{(*)}_{j,k}$) given $\boldsymbol{\mu}^{(*)}$ at the end of the training. The black marks represent each sampled data point.

does not get stuck and that the learning rate is appropriate. The cost of running these search is cheap since there is few training points and no FOM sampling. Throughout each examples, as a rule of thumb, we have found that decreasing the width of each hidden layer by an order of magnitude compared to the previous layer generally yields satisfactory performance. It is possible that other architectures may lead to equivalent, better, or worse performance (e.g. convolutional layers, etc.).

To identify the latent space dynamics, the dictionary of possible terms $\boldsymbol{\Theta}(\cdot)$ includes only linear terms and a constant, resulting in $N_l = 6$ terms. The autoencoder is trained for $N_{epoch} = 2.8 \cdot 10^4$ iterations, with a learning rate $\alpha = 10^{-3}$. A new FOM data point is sampled every $N_{up} = 2000$ iterations (resulting in adding 13 data points during training, for a total of 17 training points). For estimating the prediction variance, 20 samples are used (i.e., $N_s = 20$). The loss hyperparameters are set as $\beta_1 = 1$, $\beta_2 = 0.1$, and $\beta_3 = 10^{-6}$. The hyperparameters employed in this example are based on [42]. Additional details on the effects of hyperparameter selection can be found in [41,42].

For baseline comparison, a gLaSDI model [42] is also trained using identical hyperparameter settings, autoencoder architecture, and GP interpolation of the latent space dynamics. The key difference lies in the data sampling strategy: gLaSDI employs PDE residual error-based sampling, while GPLaSDI utilizes uncertainty-based sampling.

The system of governing ODEs for the latent space dynamics consists of 30 coefficients, each of which is interpolated by a GP. Fig. 2 illustrates the predictive mean of each GP, while Fig. 3 displays the corresponding standard deviation. Fig. 4 showcases the maximum relative error (in percentage) for each test point in $\mathcal{D}^h$, obtained using GPLaSDI and gLaSDI [42]. Remarkably, our GPLaSDI framework achieves outstanding performance, with the worst maximum relative error being below 5%, and the majority of predictions exhibiting less than 3.5% error. Moreover, for this particular example, GPLaSDI slightly outperforms gLaSDI, where the worst maximum relative error reaches 5%.

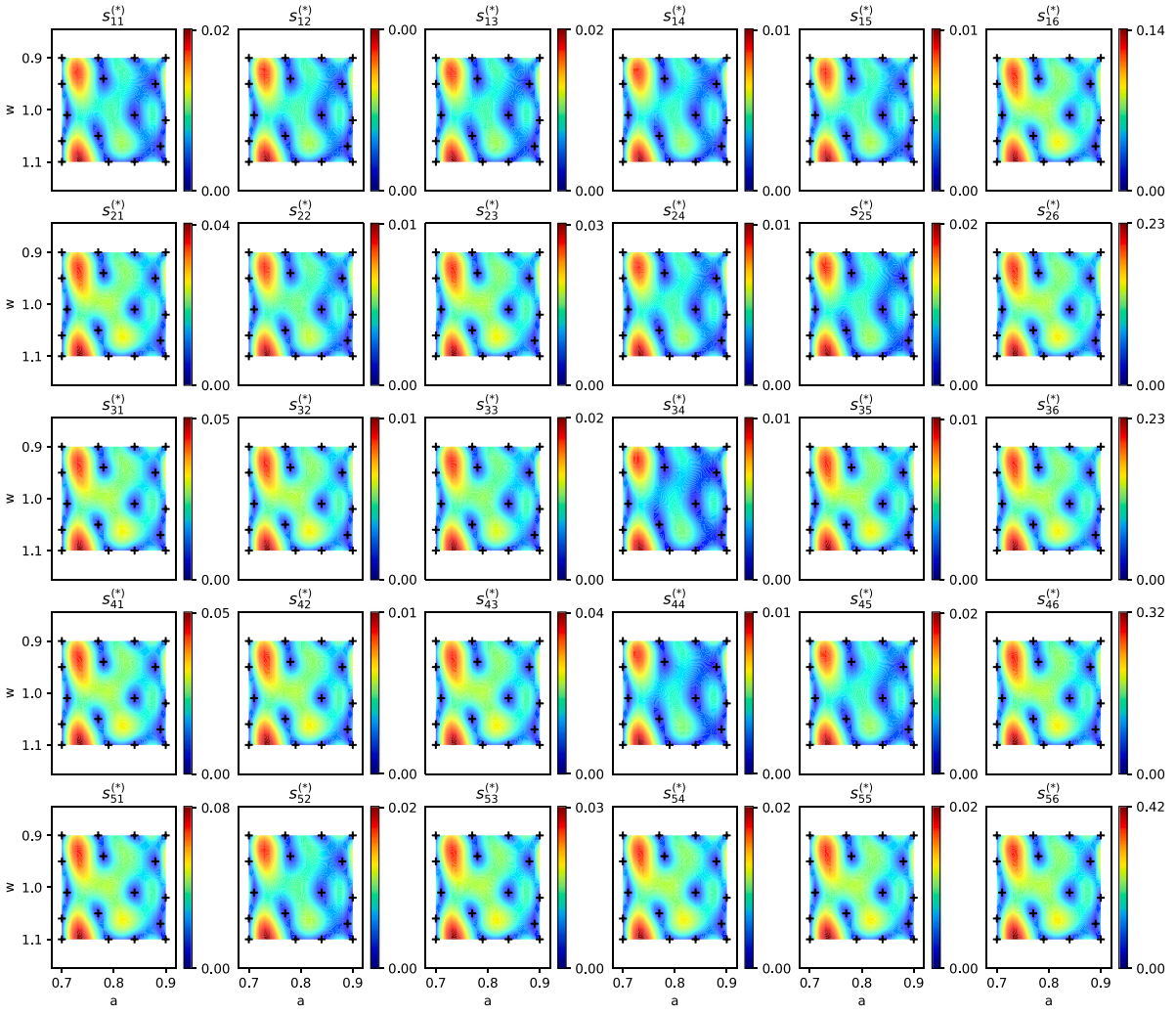


Fig. 3. 1D Burgers Equation — Predictive standard deviation of each ODE coefficients ($s_{j,k}^{(*)}$) given $\boldsymbol{\mu}^{(*)}$. The heatmaps are similar for each coefficients, which is expected because here the input point locations within $\mathcal{D}^h$ are all the same. Notice that the uncertainty is higher in regions with no training data points, as one might intuitively expect.

In Fig. 5, the maximum standard deviation $\max_{(t,x)} \mathbb{V}[\tilde{\mathbf{U}}^{(*)}]^{1/2}$ is depicted, representing the uncertainty in the decoder ROM predictions. As expected, the standard deviation patterns closely match the predictive standard deviation of each GP (Fig. 3). There is also a significant correlation between high standard deviation and high maximum relative error (Fig. 4). This correlation is valuable as it indicates that the uncertainty in the ROM model can be reliably quantified. This observation may also explain why GPLaSDI performs similarly to gLaSDI. A ROM that incorporates knowledge of the underlying physics is generally expected to outperform purely data-driven models due to additional insights. However in this case, GPLaSDI, despite being agnostic to the PDE (unlike gLaSDI), effectively captures the correlation between prediction uncertainty and ROM error. It therefore serves as a robust surrogate for the PDE residual error.

Fig. 6 illustrates the model prediction for parameter $\boldsymbol{\mu}^{(*)} = \{0.73, 0.92\}$, corresponding to the case with the largest relative error. Despite the larger error, the model predictions remain reasonable, and there is a clear correlation between the predictive standard deviation and the absolute error compared to the ground truth. The absolute error consistently falls within one standard deviation.

During 50 test runs, the FOM wall clock run-time averages at 1.31 s using a single core. On the other hand, the ROM model requires an average of $6.36 \cdot 10^{-3}$ seconds, resulting in an average speed-up of 206×. It is important to note that in this case, the system of ODEs is solved using the GP predictive means ($m_{j,k}^{(*)}$), requiring only one integration of the system of ODEs and one forward pass through the decoder. This approach does not allow for predictive standard deviation estimations since multiple sets of ODEs would need to be solved, and an equivalent number of forward passes would be required. Therefore, if we aim to make a ROM prediction with uncertainty estimation using, for example, 10 samples, the speed-up would reduce to roughly 20×. Note that running the ROM predictions for multiple samples could also be done in an embarrassingly parallel way, which would limit the deterioration of speed-up performances.

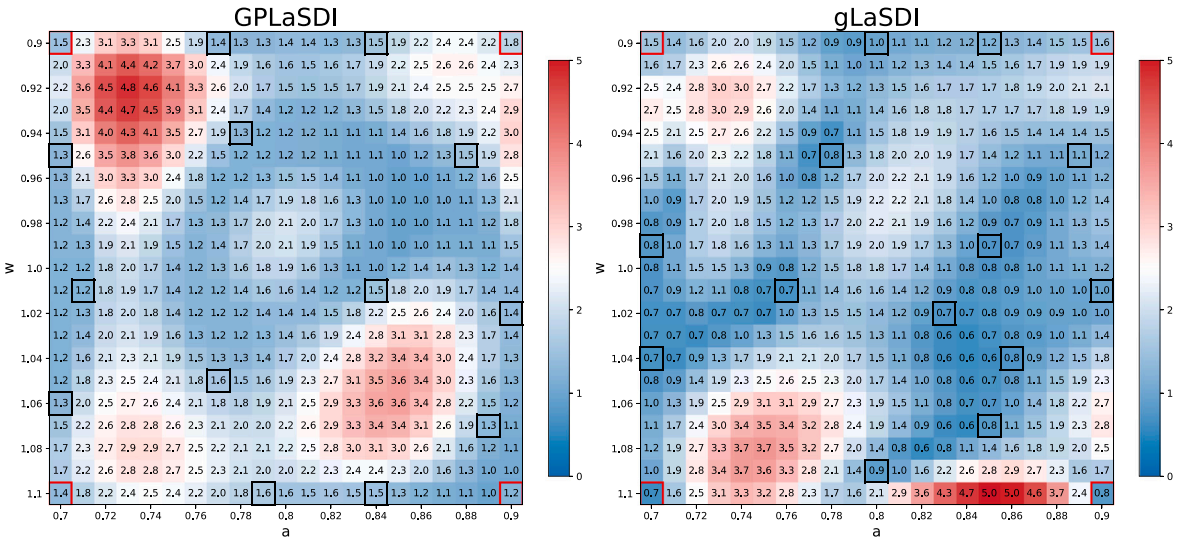


Fig. 4. 1D Burgers Equation — Maximum relative error (%) using GPLaSDI (left) and gLaSDI (right). The values in a red square correspond to the original FOM data at the beginning of the training (located at the four corners). The values in a black square correspond to parameters and FOM runs that were sampled during training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

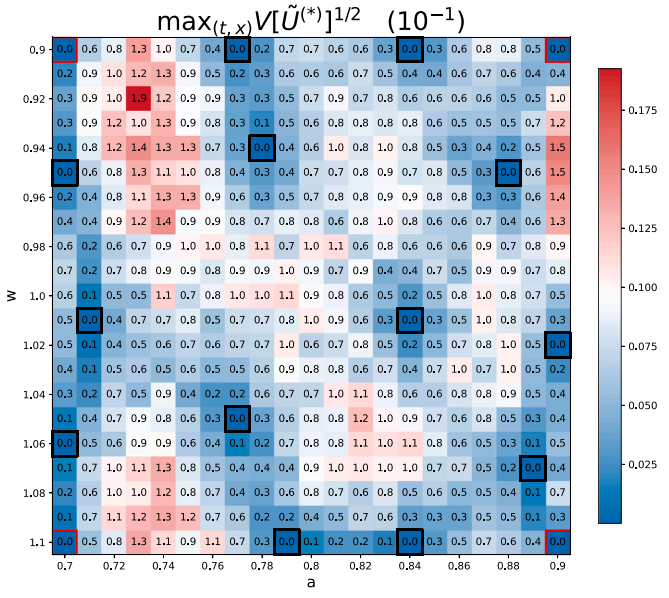


Fig. 5. 1D Burgers Equation — Maximum predictive standard deviation for GPLaSDI. The numbers inside each box are in scientific notations, scaled by the factor of 10 specified in the title. For example, the maximum value across the figure is $1.9 \cdot 10^{-1}$.

As in any neural-network-based algorithm, precisely understanding the source of errors can be challenging. There exist however several ways of pinpointing and mitigating possible errors in GPLaSDI:

- Compute the maximum relative error for points belonging to the training set (i.e. reproductive case), where ground truth data is available. Note that the training data comes from deterministic numerical solvers with unchanged simulation settings, and can thus be considered as noiseless data (as opposed to experimental data for instance). Therefore, overfitting of the autoencoder is not a primary concern and is not expected to be a major source of error. On the other hand, underfitting and/or using an autoencoder that does not have an architecture capable of capturing all the complexity in the physics is always a risk. This can be easily assessed by looking at the autoencoder prediction error on the training data (e.g. mean-squared-error between $\mathbf{U}$ and $\hat{\mathbf{U}}$).

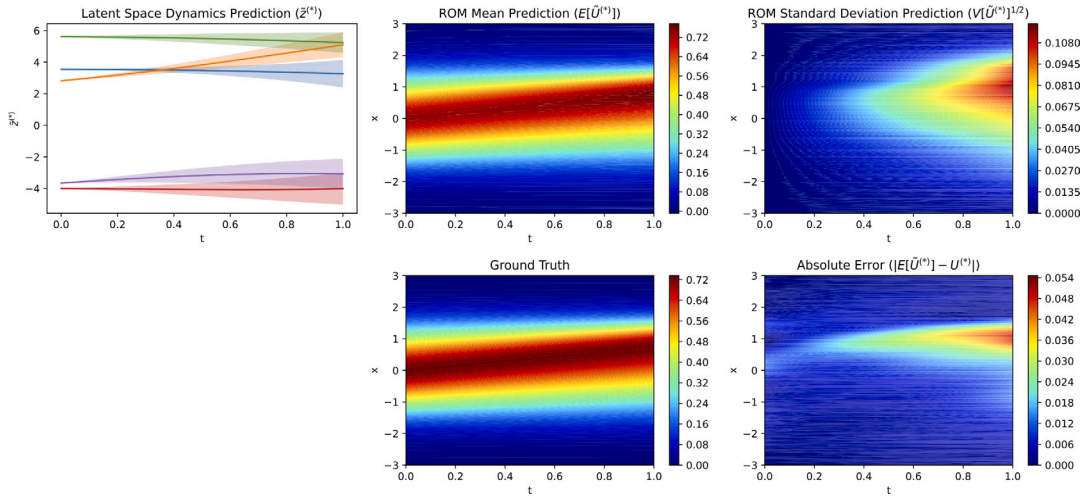


Fig. 6. 1D Burgers Equation — Predictions for $\boldsymbol{\mu}^{(*)} = \{0.73, 0.92\}$. This plot shows the predicted latent space dynamics $\mathbb{E}[\tilde{\mathbf{Z}}^{(*)}]$ with a 95% confidence interval, the ROM mean prediction $\mathbb{E}[\tilde{\mathbf{U}}^{(*)}]$ (decoder output) and standard deviation, the ground truth, and the absolute error.

- Comparing $\hat{\mathbf{Z}}$ and $\tilde{\mathbf{Z}}$ through preliminary visual inspection and error metrics such as mean-squared-error. If the error between $\mathbf{U}$ and $\hat{\mathbf{U}}$ is low, but the error between $\hat{\mathbf{Z}}$ and $\tilde{\mathbf{Z}}$ is high, this would indicate that the autoencoder is well trained, but that the model fails to reproduce the dynamics of $\hat{\mathbf{Z}}$, and more emphasis needs to be put on the SINDy loss (i.e. increase β_2).
- In this paper, to demonstrate the performance of GPLaSDI, we have generated FOM testing data for every single point in $\mathcal{D}^h$ (to compute the maximum relative error). In practice, doing so may be very expensive and would most likely defeat the purpose of a ROM. It may be possible however to generate only a handful of test FOM datapoints at random locations of the parameter space. Another way of estimating the error could be to feed the ROM predictions into the PDE residual and compute the residual error, if available. This would however defeat the purpose of a non-intrusive ROM, and in such case, using an intrusive approach such as *gLaSDI* [42] may be desirable. Note that a hybrid algorithm, using both the PDE residual (for physics information) and GPs to interpolate the sets of ODEs (for uncertainty quantification) is perfectly conceivable, but left to future work.

3.2. 2D Burgers equation

We now consider the 2D Burgers equation with a viscous term, as introduced in [41,42]:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \dfrac{1}{Re} \Delta \mathbf{u} & (t, x, y) \in [0, 1] \times [-3, 3] \times [-3, 3] \\ \mathbf{u}(t, x = \pm 3, y = \pm 3) = \mathbf{0} \end{cases} \tag{34}$$

The initial condition is analogous to the 1D case:

$$u(t = 0, x, y) = v(t = 0, x, y) = a \exp\left(-\frac{x^2 + y^2}{2w^2}\right) \qquad \begin{cases} \boldsymbol{\mu} = \{a, w\} \\ \mathbf{u} = \{u, v\} \end{cases} \tag{35}$$

We consider a Reynolds number of $Re = 10^5$. The FOM solver employs an implicit backward Euler time integration scheme, a backward finite difference for the nonlinear term, and a central difference discretization for the diffusion term. The spatial resolution is set to $\Delta x = \Delta y = 0.1$, and the time stepping is set to $\Delta t = 5 \cdot 10^{-3}$. These settings remain consistent with the 1D case. However, in this scenario, we employ a neural network architecture of 7200–100–20–20–20-5 for the encoder, and a symmetric architecture for the decoder. The activation function used is softplus. The autoencoder is trained for $N_{epoch} = 7.5 \cdot 10^5$ iterations, with a sampling rate of $N_{up} = 5 \cdot 10^4$ (resulting in adding 14 data points during training, for a total of 18 training points).

Fig. 7 depicts the maximum relative error for each point in the parameter space, comparing the performance of GPLaSDI and gLaSDI. GPLaSDI achieves consistently low maximum relative errors across most of the parameter space, typically ranging from 0.5% to 1%, with a maximum error not exceeding 3.8%. On the other hand, gLaSDI exhibits a larger maximum relative error, reaching up to 4.6%.

In Fig. 8, we observe the maximum standard deviation, which, similar to the 1D case, exhibits a clear correlation with the maximum relative error.

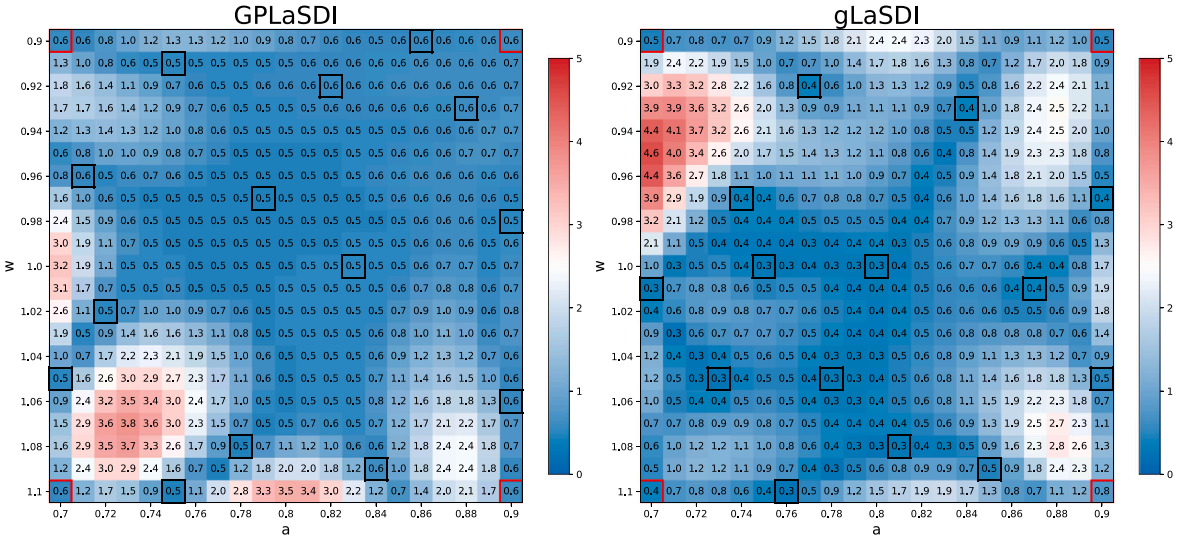


Fig. 7. 2D Burgers Equation – Maximum relative error (%) using GPLaSDI (left) and gLaSDI (right).

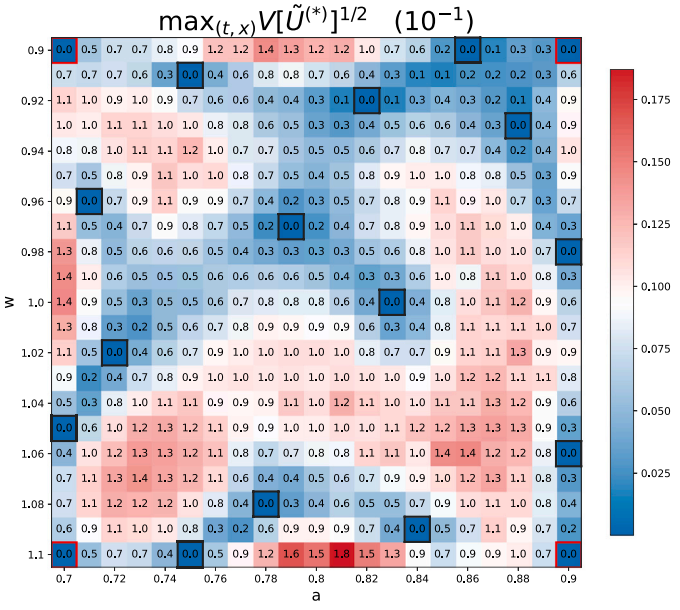


Fig. 8. 2D Burgers Equation — Maximum predictive standard deviation for GPLaSDI.

Fig. 9 presents the dynamics of the latent space, the predicted and ground truth **u** fields, the absolute error, and the fields' predictive standard deviations for the least favorable case ($\boldsymbol{\mu}^{(*)} = \{0.73, 1.07\}$) at time points $t = 0.25$ and $t = 0.75$. The error predominantly concentrates along the shock front, where the discontinuity forms. However, the error remains well within the predictive standard deviation, affirming the capability of GPLaSDI to provide meaningful confidence intervals.

During 50 test runs, the FOM wall clock run-time averaged 63.5 s using a single core. In contrast, the ROM model achieved an average runtime of blue$9.13 \cdot 10^{-3}$ seconds, resulting in an average speed-up of 6949×. Similarly to the 1D case, this speed-up is attained solely using the mean prediction. To incorporate uncertainty prediction, additional ODE integrations and decoder forward passes would be required, thereby slightly diminishing the speed-up gains.

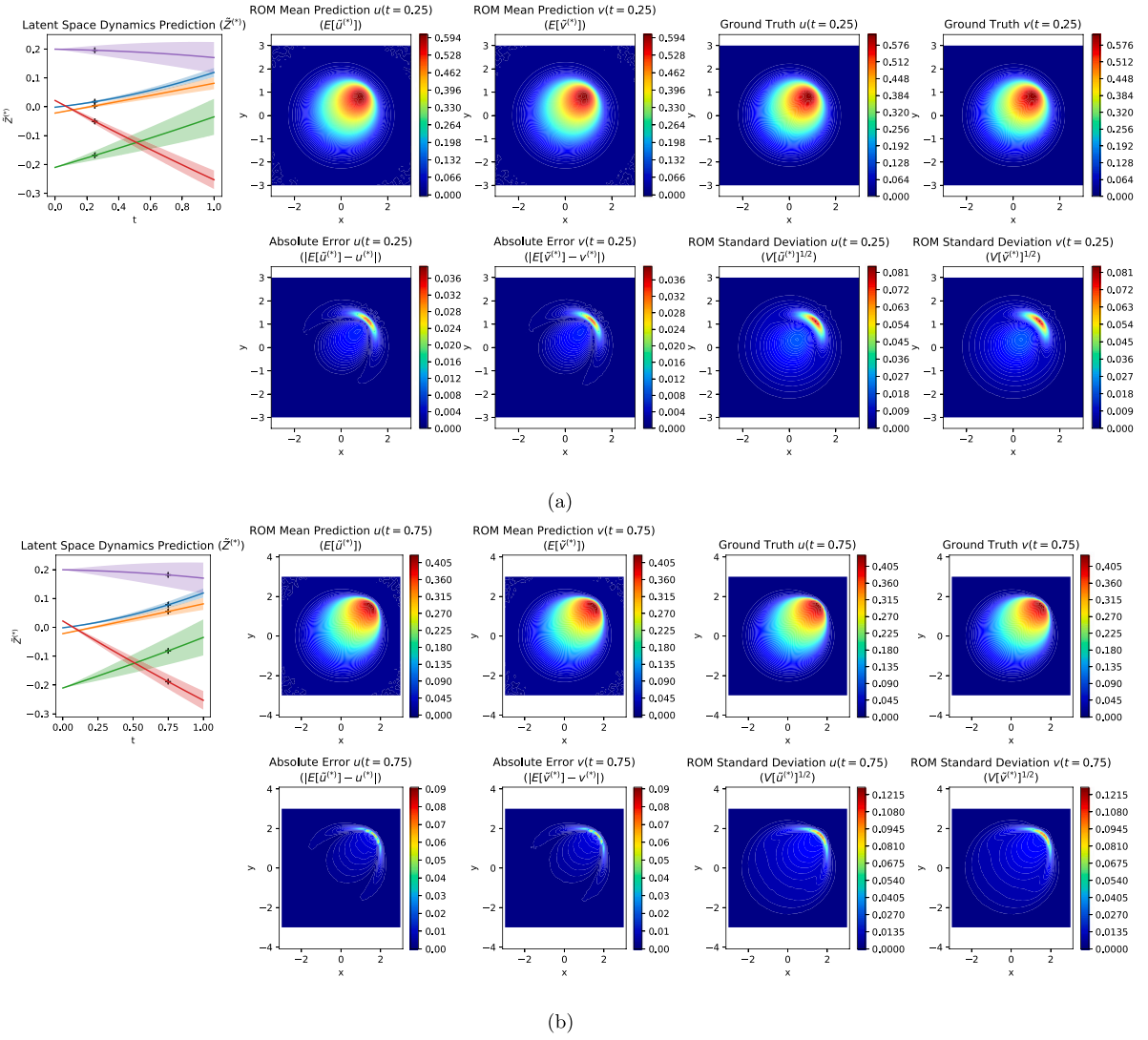


Fig. 9. 2D Burgers Equation — Predictions for $\boldsymbol{\mu}^{(*)} = \{0.73, 1.07\}$ at $t = 0.25$ (a) and $t = 0.75$ (b). These plot show the predicted latent space dynamics $\mathbb{E}[\tilde{\mathbf{Z}}^{(*)}]$ with a 95% confidence interval, the ROM mean prediction for $\mathbf{u} = \{u, v\} \equiv \{\mathbb{E}[\tilde{u}^{(*)}], \mathbb{E}[\tilde{v}^{(*)}]\}$ (decoder output) and standard deviation, the ground truth, and the absolute error.

3.3. Two–stream plasma instability

In this section, we consider the simplified 1D–1V Vlasov–Poisson equation:

$$\begin{cases} \dfrac{\partial f}{\partial t} + \dfrac{\partial}{\partial x}(vf) + \dfrac{\partial}{\partial v}\left(\dfrac{d\phi}{dx} f\right) = 0, & (t, x, v) \in [0,5] \times [0, 2\pi] \times [-7, 7], \\ \dfrac{d^2\phi}{dx^2} = \displaystyle\int_v f\, dv, \end{cases} \tag{36}$$

where we consider a plasma distribution function denoted as $f \equiv f(x, v)$, which depends on the variables x (physical coordinate) and v (velocity coordinate). The equation also involves the electrostatic potential ϕ. This simplified model governs 1D collisionless electrostatic plasma dynamics and is representative of complex models for plasma behavior in various applications, including proposed fusion power plant designs. It is important to note that kinetic modeling of plasmas leads to high-dimensional PDEs; although Eq. (36) describes one-dimensional dynamics, it is a two-dimensional PDE.

In this example, we focus on solving the two-stream instability problem that is a canonical problem used to simulate the excitation of a plasma wave from counterstreaming plasmas. The initial solution is:

$$f(t=0, x, v) = \frac{4}{\pi T}\left[1 + \frac{1}{10}\cos(k\pi x)\right]\left[\exp\left(-\frac{(v-2)^2}{2T}\right) + \exp\left(-\frac{(v+2)^2}{2T}\right)\right], \tag{37}$$

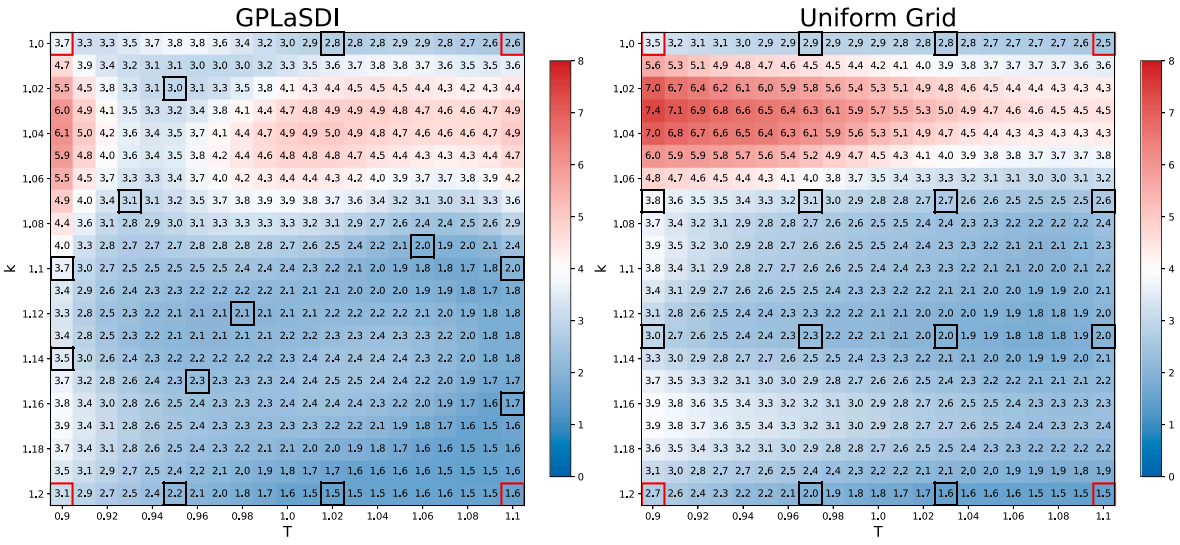


Fig. 10. 1D1V Vlasov Equation — Maximum relative error (%) using GPLaSDI and a uniform training grid (non-greedy).

where T represents the plasma temperature. The parameters involved are denoted as $\boldsymbol{\mu} = \{T, k\}$, where T ranges from 0.9 to 1.1, and k ranges from 1.0 to 1.2. We discretize the parameter space over a 21×21 grid $\mathcal{D}^h$, with a step size of $\Delta T = \Delta k = 0.01$. The training process is initialized with $N_\mu = 4$ training data points located at the four corners of the parameter space. The FOM data is sampled using `HyPar` [72], a conservative finite difference PDE code. It utilizes the fifth order WENO discretization [73] in space ($\Delta x = 2\pi/128$, $\Delta v = 7/128$) and the classical four-stage, fourth-order Runge–Kutta time integration scheme ($\Delta t = 5 \cdot 10^{-3}$). In this section and in Section 3.4, the discretization is based on the showcasing examples of `HyPar` [72], and is chosen to ensure stability at any point of the parameter space.

For the neural network architecture, we use a 16384–1000–200–50–50–50–5 configuration for the encoder (and a symmetric architecture for the decoder). The activation function employed is softplus. The latent space consists of $N_z = 5$ variables, and only linear terms are considered for the SINDy library. The loss hyperparameters are set as $\beta_1 = 1$, $\beta_2 = 0.1$, and $\beta_3 = 10^{-5}$. To estimate the prediction variance, we utilize $N_s = 20$ samples. The training process involves $N_{epoch} = 6.5 \cdot 10^5$ epochs with a learning rate of $\alpha = 10^{-5}$ and $N_{up} = 5 \cdot 10^4$ updates (resulting in adding 12 data points during training, for a total of 16 training points).

To provide a baseline for comparison, we evaluate the performance of GPLaSDI against an autoencoder trained with the same settings and hyperparameters on a uniform parameter grid. The uniform grid consists of 16 data points arranged in a 4×4 grid. In the baseline model, the interpolations of the latent space ODE coefficients are performed using GPs. Similar to GPLaSDI, the training process for the baseline model also incorporates active learning. It begins with the initial four corner points, and at every $N_{up} = 5 \cdot 10^4$ iterations, a new point is randomly selected from the uniform grid.

Fig. 10 presents the maximum relative error for each point in the parameter space obtained using GPLaSDI and the baseline model. With GPLaSDI, the worst maximum relative error is 6.1%, and in most regions of the parameter space, the error remains within the range of $1.5 - 3.5\%$. The highest errors are concentrated towards smaller values of k (typically $k < 1.07$). Compared to uniform sampling, GPLaSDI outperforms the baseline model, which achieves a maximum relative error of 7.4%.

Fig. 11 illustrates the maximum standard deviation. Although it correlates with the relative error, the correlation is only partial in this example. The standard deviation is low for parameters that correspond to a training point, indicating reproductive cases. However, the relative error can still be somewhat high in these cases. For example, for $\boldsymbol{\mu}^{(*)} = \{0.96, 1.15\}$, the maximum standard deviation $\max_{(t,x,v)} \mathbb{V}[\tilde{f}^{(*)}]^{1/2}$ is 0.0, while the relative error $e(\tilde{f}^{(*)}, f^{(*)})$ is 2.3%. The GPs interpolate the sets of ODEs governing the latent space dynamics, so the uncertainty quantification reflects the uncertainty in the latent space rather than the uncertainty in the training of the encoder and/or decoder. Therefore, it is possible that in some case, the uncertainty quantification in GPLaSDI may only provide a partial depiction of the model uncertainty. Using a Bayesian neural network (BNN) in place of the encoder and decoder could provide a fuller picture of the model uncertainty, but training BNNs is notoriously difficult and expensive [74].

Fig. 12 displays the latent space dynamics, including the predicted and ground truth values of f, the absolute error, and the predictive standard deviation. The results correspond to the least favorable case ($\boldsymbol{\mu}^{(*)} = \{0.9, 1.04\}$) at two different time instances: $t = 1$ and $t = 4$. The standard deviation of the reduced-order model (ROM) exhibits qualitative similarity to the absolute error, and the error generally falls within the range of 1 to 1.5 standard deviations.

In 20 separate test runs, the FOM requires an average wall clock run–time of 22.5 s when utilizing four cores, and 57.9 s when using a single core. In contrast, the ROM model achieves an average run–time of $1.18 \cdot 10^{-2}$ seconds, resulting in a remarkable average speed-up of 4906× (1906× when compared to the parallel FOM). It is important to note that, similar to the Burgers equation cases, this speed-up is obtained solely using the mean prediction and does not take advantage of the full predictive distribution.

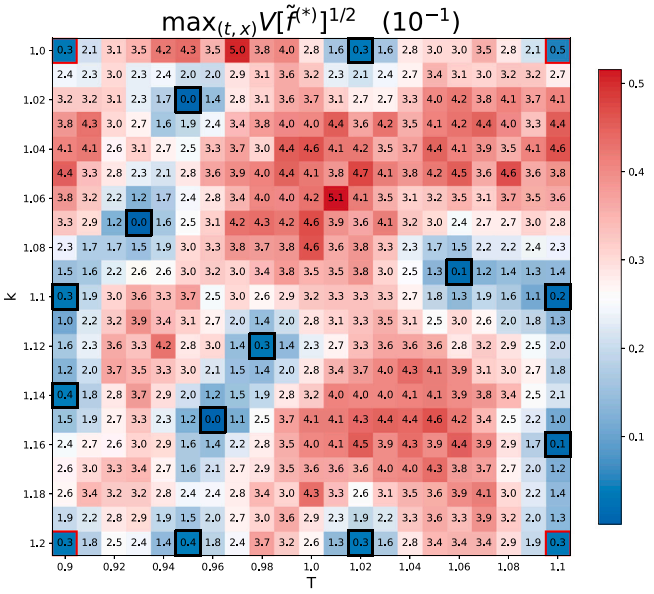


Fig. 11. 1D1V Vlasov Equation — Maximum predictive standard deviation for GPLaSDI.

3.4. Rising thermal bubble

We explore a rising thermal bubble scenario, where an initially warm bubble is introduced into a cold ambient atmosphere [75]. As time progresses, the bubble rises and dissipates, forming a mushroom pattern. The governing equations for this problem are the two-dimensional compressible Euler equations with gravitational source terms:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 & (t, x, y) \in [0, 300] \times [0, 1000] \times [0, 1000] \\ \dfrac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p\mathcal{I}) = -\rho \mathbf{g} & \mathbf{g} = \{0, 9.8\} \\ \dfrac{\partial e}{\partial t} + \nabla \cdot (e + p)\,\mathbf{u} = -\rho \mathbf{g} \cdot \mathbf{u}, \end{cases} \tag{38}$$

where ρ represents the fluid density, $\mathbf{u} = \{u, v\}$ represents the velocity, p denotes the pressure, and $\mathbf{g}$ represents the gravitational acceleration. The internal energy e is given by:

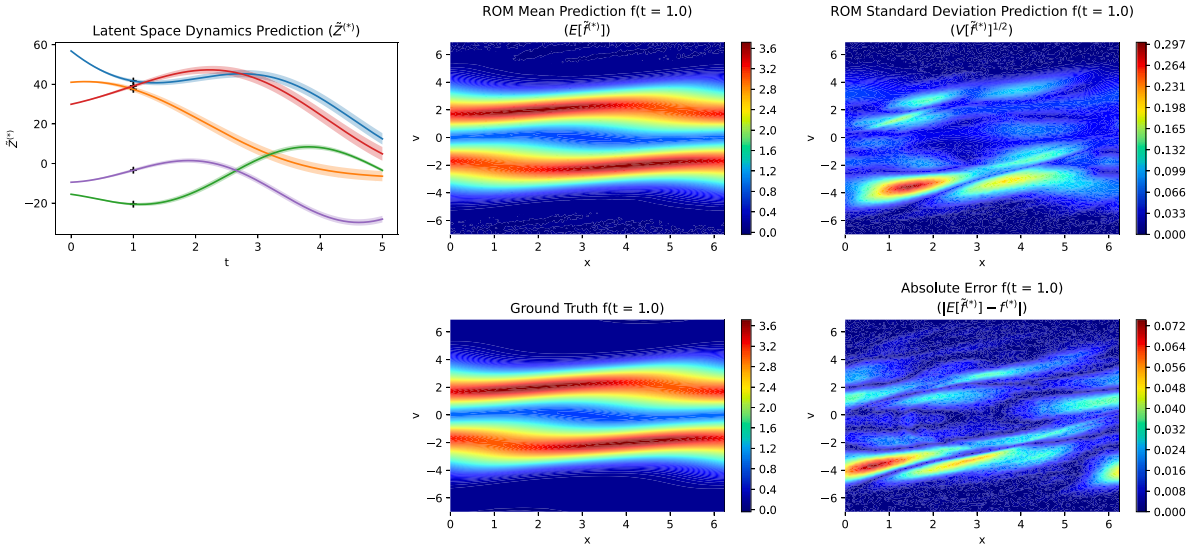
$$e = \frac{p}{\gamma - 1} + \frac{1}{2}\rho \mathbf{u} \cdot \mathbf{u}, \tag{39}$$

where $\gamma = 1.4$ is the specific heat ratio. It is important to note that Eq. (38) is solved in its dimensional form. Slip-wall boundary conditions are enforced on the velocity field $\mathbf{u}$ at all boundaries. The ambient atmosphere is a hydrostatically-balanced stratified air with a constant potential temperature $\theta = 300$ and a reference pressure $p_0 = 10^5$. The potential temperature is defined as $\theta = T\left(\frac{p}{p_0}\right)^{\frac{\gamma}{\gamma-1}}$. A warm bubble is introduced as a potential temperature perturbation:

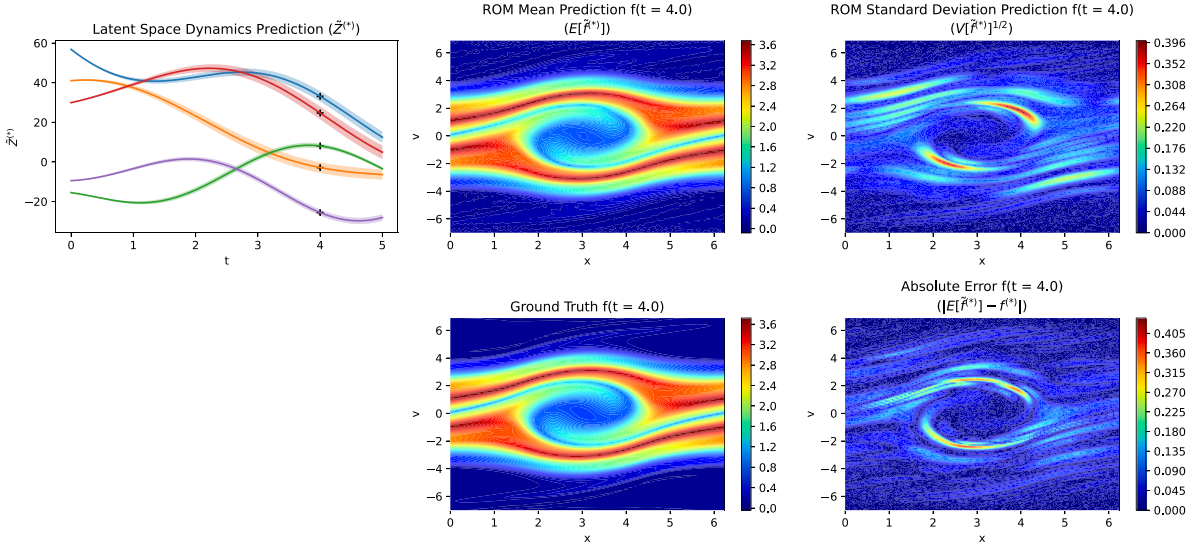
$$\theta\left(t = 0, r = \sqrt{x^2 + y^2}\right) = \begin{cases} \frac{1}{2}\theta_c(1 + \cos(\pi \frac{r}{R_c})) & r < R_c \\ 0 & r > R_c \end{cases} \tag{40}$$

The parameters of interest are denoted as $\boldsymbol{\mu} = \{\theta_c, R_c\}$, representing the perturbation strength and bubble radius, respectively. The parameter θ_c ranges from 0.5 to 0.6, while R_c ranges from 150 to 160. The parameter space is discretized using a 21×21 grid $\mathcal{D}^h$ with step sizes $\Delta\theta_c = 0.005$ and $\Delta R_c = 0.5$. The training process begins with $N_\mu = 4$ training data points located at the four corners of the parameter space. Similarly to the Vlasov equation example, the full-order model (FOM) data is sampled using `HyPar` [72,75]. The FOM is solved using a fifth order WENO discretization [73] in space with grid spacings of $\Delta x = \Delta y = 10$, and a third order strong-stability-preserving Runge–Kutta time integration scheme with a time step size of $\Delta t = 0.01$.

For the neural network architecture, we utilize a 10100–1000–200–50–20–5 configuration for the encoder (and a symmetric architecture for the decoder). The activation function employed is softplus. The latent space consists of $N_z = 5$ variables, and only linear terms are considered for the SINDy library. The loss hyperparameters are set to $\beta_1 = 1$, $\beta_2 = 0.25$, and $\beta_3 = 10^{-6}$. To estimate the prediction variance, we use $N_s = 20$ samples. The training process involves $N_{epoch} = 6.8 \cdot 10^5$ epochs with a learning rate of $\alpha = 10^{-4}$, and $N_{up} = 4 \cdot 10^4$ updates (resulting in adding 16 data points during training, for a total of 20 training points).



(a) $t = 1.0$



(b) $t = 4.0$

Fig. 12. 1D1V Vlasov Equation — Prediction for $\boldsymbol{\mu}^{(*)} = \{0.9, 1.04\}$ at (a) $t = 1.0$ and (b) $t = 4.0$. These plots illustrate the predicted latent space dynamics $\mathbb{E}[\tilde{\mathbf{Z}}^{(*)}]$ along with a 95% confidence interval. Additionally, the plots display the ROM mean prediction (decoder output) and standard deviation, the ground truth values, and the absolute error.

Similar to the Vlasov equation example, we employ a baseline for comparison by training an autoencoder with the same settings and hyperparameters on a uniform parameter grid. The uniform grid consists of 20 data points arranged in a 5×4 grid. Fig. 13 illustrates the maximum relative error for each point in the parameter space obtained using GPLaSDI and the baseline model. With GPLaSDI, the worst maximum relative error is 6.2%, and the largest errors occur for parameter values located towards the bottom right corner of the parameter space. GPLaSDI slightly outperforms the baseline, which exhibits higher errors for smaller values of R_c ($R_c < 153$), with the worst maximum relative error reaching 8.3%.

Fig. 14 displays the maximum standard deviation. It generally correlates reasonably well with the relative error. However, in the lower left corner of the parameter space, where large maximum relative errors are observed ($\theta_c < 0.52$ and $R_c > 158$), the standard deviation is unexpectedly low, erroneously indicating a high confidence in the model's predictions.

Fig. 15 depicts the latent space dynamics, specifically the predicted and ground truth values of θ, along with the absolute error and the predictive standard deviation. The results are presented for the least favorable case ($\boldsymbol{\mu}^{(*)} = \{0.59, 159\}$) at two time instances: $t = 100$ and $t = 300$. The absolute error typically falls within one standard deviation, and its pattern closely matches that of the

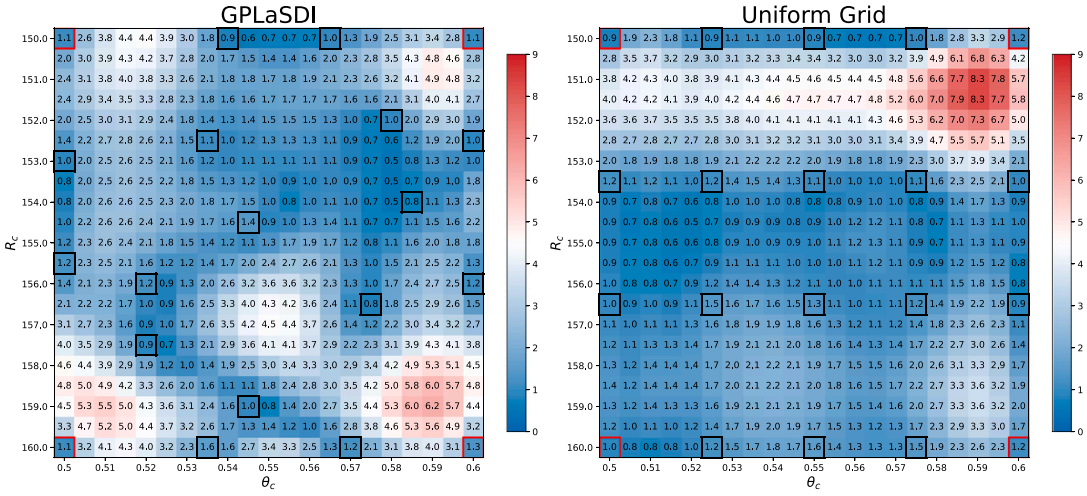


Fig. 13. 2D Rising Thermal Bubble — Maximum relative error (%) using GPLaSDI and a uniform training grid (non-greedy).

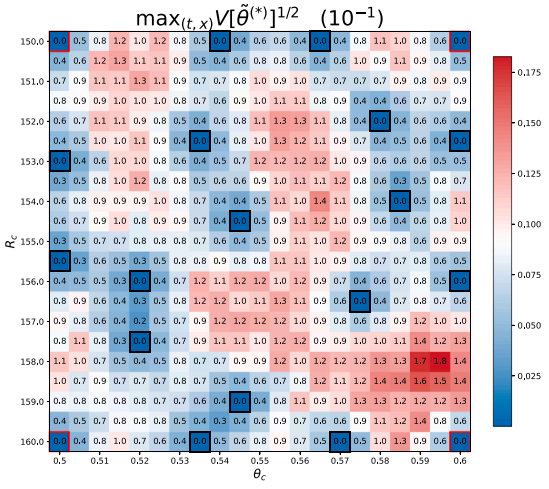


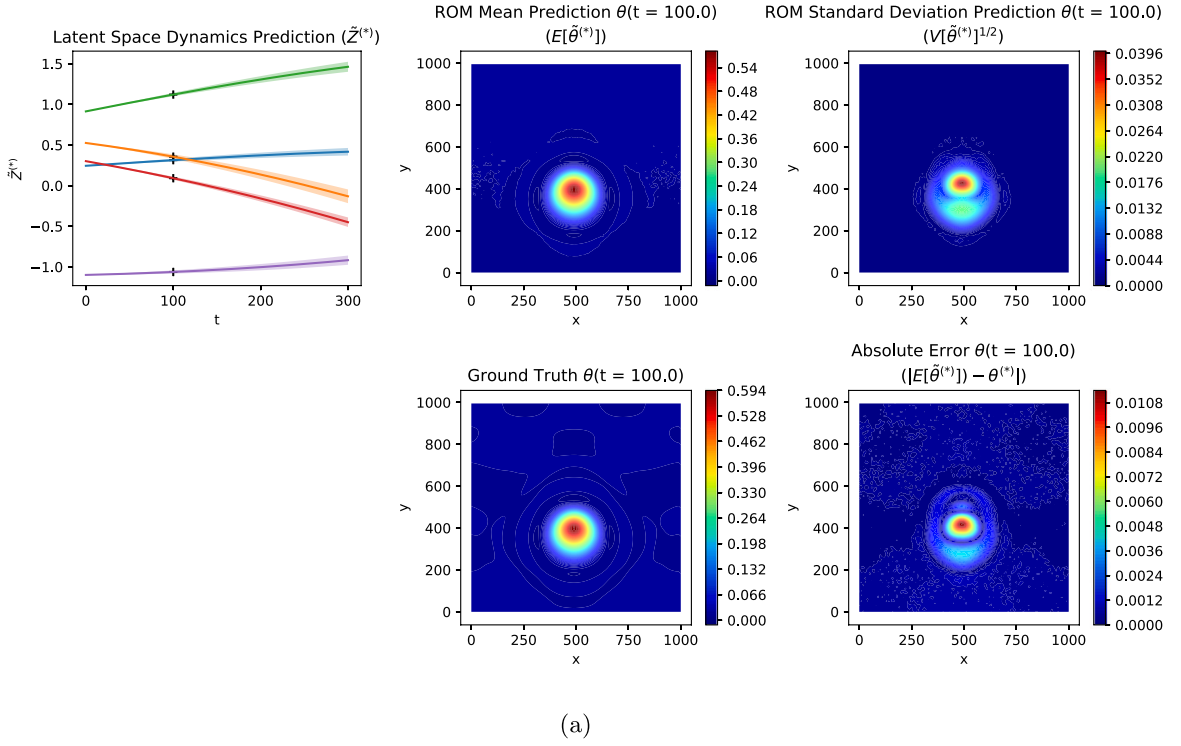
Fig. 14. 2D Rising Thermal Bubble — Maximum predictive standard deviation for GPLaSDI.

standard deviation. This consistent observation aligns with the findings from the Burgers equation and Vlasov equation examples, indicating that the confidence intervals provided by GPLaSDI are meaningful and closely correlated with the prediction error.

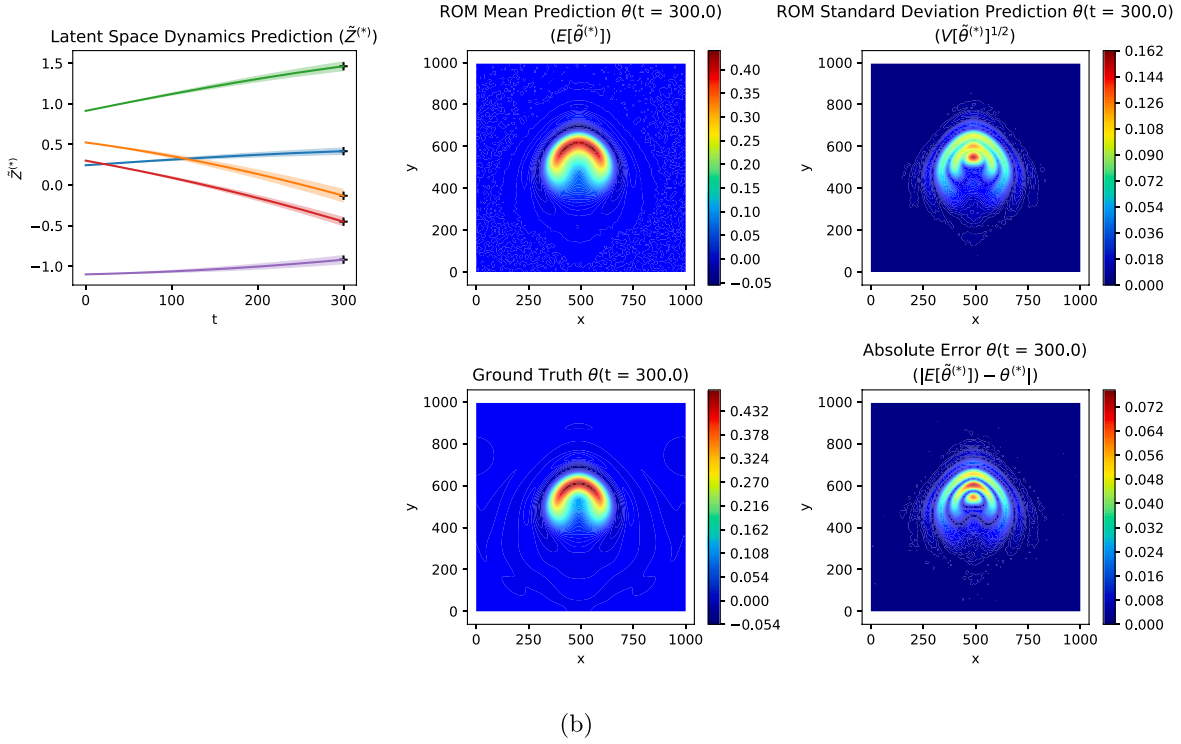
During 20 test runs, the FOM requires an average wall clock run-time of 89.1 s when utilizing 16 cores, and 1246.8 s when using a single core. On the other hand, the ROM model achieves an average run-time of $1.25 \cdot 10^{-2}$ seconds, resulting in an impressive average speed-up of 99744× (7128× when compared to the parallel FOM). It is worth noting that, similar to the Burgers and Vlasov equation cases, this speed-up is obtained solely using the mean prediction. It is also interesting to note that throughout each example, the run time of GPLaSDI remains relatively consistent (in the order of 10^{-2} seconds), even though the discretization of the high-fidelity problem varied widely (and with it, the number of parameters in the autoencoder). This would indicate that GPLaSDI prediction time can scale well to problems requiring finer discretizations.

4. Conclusion

We have presented GPLaSDI, a non-intrusive greedy LaSDI framework that incorporates Gaussian process latent space interpolation. Our proposed framework offers several key advantages. First, GPLaSDI efficiently captures the latent space dynamics and successfully interpolates the governing sets of ODEs while providing uncertainty quantification. This allows for meaningful confidence intervals to be obtained over the reduced-order model (ROM) predictions. These confidence intervals play a crucial role in identifying regions of high uncertainty in the parameter space. Furthermore, GPLaSDI intelligently selects additional training data points in these uncertain regions, thereby maximizing prediction accuracy while providing confidence intervals. Notably, GPLaSDI accomplishes this without requiring any prior knowledge of the underlying partial differential equation (PDE) or its residual.



(a)



(b)

Fig. 15. 2D Rising Thermal Bubble — Predictions for $\boldsymbol{\mu}^{(*)} = \{0.59, 159\}$ at (a) $t = 100$ and (b) $t = 300$. These plots illustrate the predicted latent space dynamics $\mathbb{E}[\tilde{\mathbf{Z}}^{(*)}]$ along with a 95% confidence interval. Additionally, the plots display the ROM mean prediction (decoder output) and standard deviation, the ground truth values, and the absolute error.

We have demonstrated the effectiveness of GPLaSDI through four numerical examples, showcasing its superiority over uniform sampling baselines and its competitive performance compared to intrusive methods such as gLaSDI. GPLaSDI consistently achieved maximum relative errors of less than 6–7%, while achieving significant speed-ups ranging from several hundred to several tens of thousands of times.

Overall, GPLaSDI offers a powerful and efficient approach for capturing latent space dynamics, accurately interpolating ODEs, and providing uncertainty quantification in the context of reduced-order modeling. Its ability to autonomously select training data points and generate confidence intervals makes it a valuable tool for various scientific and engineering applications.

Currently the number of training iteration and sampling rate (and thus, the number of points that will be sampled) are all predetermined. In future work, an early termination strategy when satisfactory accuracy is obtained could be designed. Additionally, the multiple GP training could become intractable in case of a large number of latent space variables (large N_z) and SINDy candidates (large N_l), since the number of ODE coefficients would grow in $\mathcal{O}(N_z N_l)$. In such case, a parallel implementation of GP training would likely be necessary.

Declaration of competing interest

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

Data availability

The code github link is provided in the manuscript.

Acknowledgments

This research was conducted at Lawrence Livermore National Laboratory and received support from the LDRD program under project number 21-SI-006. Y. Choi also acknowledges support from the CHaRMNET Mathematical Multifaceted Integrated Capability Center (MMICC). Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344 and LLNL-JRNL-852707.

References

[1] S. Raczynski, Modeling and simulation : The computer science of illusion / Stanislaw Raczynski, in: Modeling and Simulation : The Computer Science of Illusion, in: RSP Series in Computer Simulation and Modeling, John Wiley & Sons, Ltd, Hertfordshire, England, 2006.
[2] D. Jones, C. Snider, A. Nassehi, J. Yon, B. Hicks, Characterising the digital twin: A systematic literature review, CIRP J. Manuf. Sci. Technol. 29 (2020) 36–52, http://dx.doi.org/10.1016/j.cirpj.2020.02.002, URL https://www.sciencedirect.com/science/article/pii/S1755581720300110.
[3] Review of digital twin about concepts, technologies, and industrial applications, J. Manuf. Syst. 58 (2020) 346–361, http://dx.doi.org/10.1016/J.JMSY.2020.06.017.
[4] M. Calder, C. Craig, D. Culley, R. Cani, C. Donnelly, R. Douglas, B. Edmonds, J. Gascoigne, N. Gilbert, C. Hargrove, D. Hinds, D. Lane, D. Mitchell, G. Pavey, D. Robertson, B. Rosewell, S. Sherwin, M. Walport, A. Wilson, Computational modelling for decision-making: Where, why, what, who and how, R. Soc. Open Sci. 5 (2018) 172096, http://dx.doi.org/10.1098/rsos.172096.
[5] E. Winsberg, Computer Simulations in Science, in: E.N. Zalta, U. Nodelman (Eds.), The Stanford Encyclopedia of Philosophy, Winter 2022 ed., Metaphysics Research Lab, Stanford University, 2022.
[6] R.M. Cummings, W.H. Mason, S.A. Morton, D.R. McDaniel, Applied Computational Aerodynamics: A Modern Engineering Approach, in: Cambridge Aerospace Series, Cambridge University Press, 2015, http://dx.doi.org/10.1017/CBO9781107284166.
[7] D. Diston, Computational Modelling and Simulation of Aircraft and the Environment: Platform Kinematics and Synthetic Environment, first ed., in: Aerospace Series, vol. 1, John Wiley & Sons Ltd, United Kingdom, 2009, http://dx.doi.org/10.1002/9780470744130.
[8] K. Kurec, M. Remer, J. Broniszewski, P. Bibik, S. Tudruj, J. Piechna, Advanced modeling and simulation of vehicle active aerodynamic safety, J. Adv. Transp. 2019 (2019) 1–17, http://dx.doi.org/10.1155/2019/7308590.
[9] A. Muhammad, I.H. Shanono, Simulation of a car crash using ANSYS, in: 2019 15th International Conference on Electronics, Computer and Computation, ICECCO, 2019, pp. 1–5, http://dx.doi.org/10.1109/ICECCO48375.2019.9043275.
[10] A. Peterson, S. Ray, R. Mittra, Computational Methods for Electromagnetics, Wiley, John & Sons, 1997.
[11] T. Rylander, P. Ingelström, A. Bondeson, Computational Electromagnetics, Springer, 2013.
[12] J. Thijssen, Computational Physics, second ed., Cambridge University Press, 2007, http://dx.doi.org/10.1017/CBO9781139171397.
[13] R. Schwartz, Biological Modeling and Simulation, MIT Press, 2008.
[14] L. Biegler, G. Biros, O. Ghattas, M. Heinkenschloss, D. Keyes, B. Mallick, Y. Marzouk, L. Tenorio, B. van Bloemen Waanders, K. Willcox, Large-Scale Inverse Problems and Quantification of Uncertainty, Wiley, 2010, http://dx.doi.org/10.1002/9780470685853, URL http://hdl.handle.net/10754/656260.
[15] R.C. Smith, Uncertainty quantification - theory, implementation, and applications, in: Computational Science and Engineering, 2013.
[16] R. Sternfels, C.J. Earls, Reduced-order model tracking and interpolation to solve PDE-based Bayesian inverse problems, Inverse Problems 29 (7) (2013) 075014, http://dx.doi.org/10.1088/0266-5611/29/7/075014.
[17] D. Galbally, K. Fidkowski, K. Willcox, O. Ghattas, Non-linear model reduction for uncertainty quantification in large-scale inverse problems, Internat. J. Numer. Methods Engrg. 81 (12) (2010) 1581–1608, http://dx.doi.org/10.1002/nme.2746, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2746.
[18] V. Fountoulakis, C. Earls, Duct heights inferred from radar sea clutter using proper orthogonal bases, Radio Sci. 51 (10) (2016) 1614–1626, http://dx.doi.org/10.1002/2016RS005998, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2016RS005998.
[19] S. Wang, E.d. Sturler, G.H. Paulino, Large-scale topology optimization using preconditioned Krylov subspace methods with recycling, Internat. J. Numer. Methods Engrg. 69 (12) (2007) 2441–2468, http://dx.doi.org/10.1002/nme.1798, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1798.
[20] D. White, Y. Choi, J. Kudo, A dual mesh method with adaptivity for stress-constrained topology optimization, Struct. Multidiscip. Optim. 61 (2020) http://dx.doi.org/10.1007/s00158-019-02393-6.

[21] Y. Choi, C. Farhat, W. Murray, M. Saunders, A practical factorization of a Schur complement for PDE-constrained distributed optimal control, 2013, http://dx.doi.org/10.48550/ARXIV.1312.5653, arXiv. URL https://arxiv.org/abs/1312.5653.
[22] G. Berkooz, P. Holmes, J.L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annu. Rev. Fluid Mech. 25 (1) (1993) 539–575, http://dx.doi.org/10.1146/annurev.fl.25.010193.002543.
[23] G. Rozza, D. Huynh, A. Patera, Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations, Arch. Comput. Methods Eng. 15 (2007) 1–47, http://dx.doi.org/10.1007/BF03024948.
[24] M.G. Safonov, R.Y. Chiang, A Schur method for balanced model reduction, in: 1988 American Control Conference, 1988, pp. 1036–1040.
[25] J.T. Lauzon, S.W. Cheung, Y. Shin, Y. Choi, D.M. Copeland, K. Huynh, S-OPT: A points selection algorithm for hyper-reduction in reduced order models, 2022, http://dx.doi.org/10.48550/ARXIV.2203.16494, arXiv. URL https://arxiv.org/abs/2203.16494.
[26] Y. Choi, D. Coombs, R. Anderson, SNS: A solution-based nonlinear subspace method for time-dependent model order reduction, SIAM J. Sci. Comput. 42 (2) (2020) A1116–A1146, http://dx.doi.org/10.1137/19M1242963.
[27] G. Stabile, G. Rozza, Finite volume POD-Galerkin stabilised reduced order methods for the parametrised incompressible Navier–Stokes equations, Comput. & Fluids 173 (2018) 273–284, http://dx.doi.org/10.1016/j.compfluid.2018.01.035.
[28] T. Iliescu, Z. Wang, Variational multiscale proper orthogonal decomposition: Navier-stokes equations, Numer. Methods Partial Differential Equations 30 (2) (2014) 641–663, http://dx.doi.org/10.1002/num.21835, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/num.21835.
[29] D.M. Copeland, S.W. Cheung, K. Huynh, Y. Choi, Reduced order models for Lagrangian hydrodynamics, Comput. Methods Appl. Mech. Engrg. 388 (2022) 114259, http://dx.doi.org/10.1016/j.cma.2021.114259.
[30] S.W. Cheung, Y. Choi, D.M. Copeland, K. Huynh, Local Lagrangian reduced-order modeling for Rayleigh-Taylor instability by solution manifold decomposition, 2022, http://dx.doi.org/10.48550/ARXIV.2201.07335, arXiv. URL https://arxiv.org/abs/2201.07335.
[31] B. McLaughlin, J. Peterson, M. Ye, Stabilized reduced order models for the advection–diffusion–reaction equation using operator splitting, Comput. Math. Appl. 71 (11) (2016) 2407–2420, http://dx.doi.org/10.1016/j.camwa.2016.01.032, Proceedings of the conference on Advances in Scientific Computing and Applied Mathematics. A special issue in honor of Max Gunzburger's 70th birthday. URL https://www.sciencedirect.com/science/article/pii/S0898122116300281.
[32] Y. Kim, K. Wang, Y. Choi, Efficient space–time reduced order model for linear dynamical systems in python using less than 120 lines of code, Mathematics 9 (14) (2021) http://dx.doi.org/10.3390/math9141690, URL https://www.mdpi.com/2227-7390/9/14/1690.
[33] Y. Choi, G. Oxberry, D. White, T. Kirchdoerfer, Accelerating design optimization using reduced order models, 2019, http://dx.doi.org/10.48550/ARXIV.1909.11320, arXiv. URL https://arxiv.org/abs/1909.11320.
[34] S. McBane, Y. Choi, Component-wise reduced order model lattice-type structure design, Comput. Methods Appl. Mech. Engrg. 381 (2021) 113813, http://dx.doi.org/10.1016/j.cma.2021.113813.
[35] Y. Kim, Y. Choi, D. Widemann, T. Zohdi, A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder, 2020, http://dx.doi.org/10.48550/ARXIV.2009.11990, arXiv. URL https://arxiv.org/abs/2009.11990.
[36] Y. Kim, Y. Choi, D. Widemann, T. Zohdi, Efficient nonlinear manifold reduced order model, 2020, http://dx.doi.org/10.48550/ARXIV.2011.07727, arXiv. URL https://arxiv.org/abs/2011.07727.
[37] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973, http://dx.doi.org/10.1016/j.jcp.2019.108973, URL https://www.sciencedirect.com/science/article/pii/S0021999119306783.
[38] A.N. Diaz, Y. Choi, M. Heinkenschloss, A fast and accurate domain-decomposition nonlinear manifold reduced order model, 2023, arXiv:2305.15163.
[39] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507, http://dx.doi.org/10.1126/science.1127647, arXiv:https://www.science.org/doi/pdf/10.1126/science.1127647.
[40] D. DeMers, G. Cottrell, Non-linear dimensionality reduction, in: S. Hanson, J. Cowan, C. Giles (Eds.), Advances in Neural Information Processing Systems. Vol. 5, Morgan-Kaufmann, 1992, URL https://proceedings.neurips.cc/paper/1992/file/cdc0d6e63aa8e41c89689f54970bb35f-Paper.pdf.
[41] W.D. Fries, X. He, Y. Choi, LaSDI: Parametric latent space dynamics identification, Comput. Methods Appl. Mech. Engrg. 399 (2022) 115436, http://dx.doi.org/10.1016/j.cma.2022.115436.
[42] X. He, Y. Choi, W.D. Fries, J. Belof, J.-S. Chen, gLaSDI: Parametric physics-informed greedy latent space dynamics identification, J. Comput. Phys. 489 (2023) 112267, http://dx.doi.org/10.1016/j.jcp.2023.112267, URL https://www.sciencedirect.com/science/article/abs/pii/S0021999123003625.
[43] S. McBane, Y. Choi, K. Willcox, Stress-constrained topology optimization of lattice-like structures using component-wise reduced order models, Comput. Methods Appl. Mech. Engrg. 400 (2022) 115525.
[44] G. Tapia, S.A. Khairallah, M.J. Matthews, W.E. King, A. Elwany, Gaussian process-based surrogate modeling framework for process planning in laser powder-bed fusion additive manufacturing of 316L stainless steel, Int. J. Adv. Manuf. Technol. 94 (9–12) (2017) http://dx.doi.org/10.1007/s00170-017-1045-z.
[45] D. Marjavaara, CFD driven optimization of hydraulic turbine draft tubes using surrogate models, 2006.
[46] K. Cheng, R. Zimmermann, Sliced gradient-enhanced Kriging for high-dimensional function approximation and aerodynamic modeling, 2022.
[47] J.N. Kutz, Deep learning in fluid dynamics, J. Fluid Mech. 814 (2017) 1–4, http://dx.doi.org/10.1017/jfm.2016.803.
[48] J.R. Koza, Genetic programming as a means for programming computers by natural selection, Stat. Comput. 4 (2) (1994) 87–112, http://dx.doi.org/10.1007/BF00175355.
[49] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (5923) (2009) 81–85, http://dx.doi.org/10.1126/science.1165893, arXiv:https://www.science.org/doi/pdf/10.1126/science.1165893.
[50] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. 113 (15) (2016) 3932–3937, http://dx.doi.org/10.1073/pnas.1517384113, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1517384113.
[51] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (4) (2017) e1602614, http://dx.doi.org/10.1126/sciadv.1602614, arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.1602614.
[52] L.M. Gao, J.N. Kutz, Bayesian autoencoders for data-driven discovery of coordinates, governing equations and fundamental constants, 2022, http://dx.doi.org/10.48550/ARXIV.2211.10575, arXiv. URL https://arxiv.org/abs/2211.10575.
[53] K. Owens, J.N. Kutz, Data-driven discovery of governing equations for coarse-grained heterogeneous network dynamics, 2022, http://dx.doi.org/10.48550/ARXIV.2205.10965, arXiv. URL https://arxiv.org/abs/2205.10965.
[54] S.M. Hirsh, D.A. Barajas-Solano, J.N. Kutz, Sparsifying priors for Bayesian uncertainty quantification in model discovery, R. Soc. Open Sci. 9 (2) (2022) 211823, http://dx.doi.org/10.1098/rsos.211823, arXiv:https://royalsocietypublishing.org/doi/pdf/10.1098/rsos.211823.
[55] D.A. Messenger, D.M. Bortz, Weak SINDy for partial differential equations, J. Comput. Phys. 443 (2021) 110525, http://dx.doi.org/10.1016/j.jcp.2021.110525.
[56] Z. Chen, Y. Liu, H. Sun, Physics-informed learning of governing equations from scarce data, Nature Commun. 12 (1) (2021) http://dx.doi.org/10.1038/s41467-021-26434-1.
[57] C. Bonneville, C. Earls, Bayesian deep learning for partial differential equation parameter discovery with sparse and noisy data, J. Comput. Phys.: X 16 (2022) 100115, http://dx.doi.org/10.1016/j.jcpx.2022.100115, URL https://www.sciencedirect.com/science/article/pii/S2590055222000117.
[58] R. Stephany, C. Earls, PDE-READ: Human-readable partial differential equation discovery using deep learning, Neural Netw. 154 (2022) 360–382, http://dx.doi.org/10.1016/j.neunet.2022.07.008, URL https://www.sciencedirect.com/science/article/pii/S0893608022002660.

[59] R. Stephany, C. Earls, PDE-LEARN: Using deep learning to discover partial differential equations from noisy, limited data, 2022, http://dx.doi.org/10.48550/ARXIV.2212.04971, arXiv. URL https://arxiv.org/abs/2212.04971.
[60] K. Champion, B. Lusch, J.N. Kutz, S.L. Brunton, Data-driven discovery of coordinates and governing equations, Proc. Natl. Acad. Sci. 116 (45) (2019) 22445–22451, http://dx.doi.org/10.1073/pnas.1906995116, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1906995116.
[61] Z. Bai, L. Peng, Non-intrusive nonlinear model reduction via machine learning approximations to low-dimensional operators, 2021, http://dx.doi.org/10.48550/ARXIV.2106.09658, arXiv. URL https://arxiv.org/abs/2106.09658.
[62] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning., in: Adaptive computation and machine learning, MIT Press, 2006, pp. I–XVIII, 1–248.
[63] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[64] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, http://dx.doi.org/10.48550/ARXIV.1412.6980, arXiv. URL https://arxiv.org/abs/1412.6980.
[65] J.N. Fuhg, M. Marino, N. Bouklas, Local approximate Gaussian process regression for data-driven constitutive models: Development and comparison with neural networks, Comput. Methods Appl. Mech. Engrg. 388 (2022) 114217, http://dx.doi.org/10.1016/j.cma.2021.114217, URL https://www.sciencedirect.com/science/article/pii/S004578252100548X.
[66] A.G. Wilson, H. Nickisch, Kernel interpolation for scalable structured Gaussian processes (KISS-GP), 2015, arXiv:1503.01057.
[67] A.G. Wilson, C. Dann, H. Nickisch, Thoughts on massively scalable Gaussian processes, 2015, arXiv:1511.01870.
[68] A. Muyskens, B. Priest, I. Goumiri, M. Schneider, MuyGPs: Scalable Gaussian process hyperparameter estimation using local cross-validation, 2021, arXiv:2104.14581.
[69] C.M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), first ed., Springer, 2007, URL http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738.
[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (Oct) (2011) 2825–2830.
[72] HyPar Repository, https://bitbucket.org/deboghosh/hypar.
[73] G.-S. Jiang, C.-W. Shu, Efficient implementation of weighted ENO schemes, J. Comput. Phys. 126 (1) (1996) 202–228, http://dx.doi.org/10.1006/jcph.1996.0130.
[74] R.M. Neal, Bayesian learning for neural networks, 1995, URL https://api.semanticscholar.org/CorpusID:60809283.
[75] D. Ghosh, E.M. Constantinescu, Well-balanced, conservative finite-difference algorithm for atmospheric flows, AIAA J. 54 (4) (2016) 1370–1385, http://dx.doi.org/10.2514/1.J054580.