InVAErt networks: a data-driven framework for model synthesis and identifiability analysis

Guoxiang Grayson Tong[1], **Carlos A. Sing Long**[2], **Daniele E. Schiavazzi**[1*]
[1]Department of Applied and Computational Mathematics and Statistics, University of Notre Dame, Notre Dame, 46656, IN, United States
[2]Institute for Mathematical and Computational Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

Keywords: Model synthesis; Data-driven identifiability analysis; Variational autoencoders; Direct and inverse problems; Deep neural network

Abstract

Use of generative models and deep learning for physics-based systems is currently dominated by the task of emulation. However, the remarkable flexibility offered by data-driven architectures would suggest to extend this representation to other aspects of system analysis including model inversion and identifiability. We introduce inVAErt (pronounced *invert*) networks, a comprehensive framework for data-driven analysis and synthesis of parametric physical systems which uses a deterministic encoder and decoder to represent the forward and inverse solution maps, a normalizing flow to capture the probabilistic distribution of system outputs, and a variational encoder designed to learn a compact latent representation for the lack of bijectivity between inputs and outputs. We formally analyze how changes in the penalty coefficients affect the stationarity condition of the loss function, the phenomenon of posterior collapse, and propose strategies for latent space sampling, since we find that all these aspects significantly affect both training and testing performance. We verify our framework through extensive numerical examples, including simple linear, nonlinear, and periodic maps, dynamical systems, and spatio-temporal PDEs.

1 Introduction

In the simulation of physical systems, an increase in model complexity directly corresponds to an increase in the simulation time, posing substantial limitations to the use of such models for applications that depend on time-sensitive decisions. Therefore, fast emulators learned by data-driven architectures and integrated in algorithms for the solution of forward and inverse problems are becoming increasingly successful.

Several contributions in the literature have proposed architectures for physics-based emulators designed to limit the number of model evaluations during training. These include, for example, physics-informed neural networks (PINN) [27], deep operator networks (DeepONet) [35], and transformers-based architectures [18]. Other approaches include Gaussian Processes [42], Bayesian networks [62], generative adversarial networks (GAN) [63], diffusion models [58], optimal transport [37], normalizing flows [33, 40] and Variational Auto-Encoders (VAE) [65].

When using data-driven emulators in the context of inverse problems, other difficulties arise. Inverse problems are often ill-posed as a result of non-uniqueness of solutions, or of ill-conditioning due to high-dimensionality, data-sparsity, noise-corruption, and nonlinear response of the physical systems [26, 5, 53, 4, 19]. Thus, robust solutions heavily rely on regularization of the Tikhonov-

Phillips type [31, 26, 5], on prior specification in Bayesian inversion [53, 11, 32] or, more recently, on learning data-driven regularizers (see, e.g., the Network Tikhonov approach [34]).

First, we would like to emphasize that, even if forward and inverse problems are generally treated separately in the literature, they are strongly related. A uniform emulator accuracy in the entire input space, for example, is not required for the accurate solution of inverse problems, and accuracy only around regions of high posterior density might suffice. In this context, incremental improvements of data-driven emulators during inversion is explored in [59] in the context of variational inference, [8] for residual-based error correction of neural operators, and [10] for adaptive annealing. In addition, an increasing number of studies in the literature are looking at data-driven architectures that can jointly learn the forward and inverse solution map [20, 57, 54].

Second we remark that, in many of the previous contributions, regularization and strong assumptions on the distribution of model outputs or parameters are used to promote certain solutions in ill-posed inverse problems. However, this may not be fully informative on the nature of such problems. In other words, there may be several possible inputs belonging to a *manifold* (or, more generally, a *fiber* or a *level set*) embedded in the ambient input space, that constitute the *preimage* of a given observation. In this case, instead of *selecting* a solution with a particular structure, we would like to characterize the *entire* manifold of possible solutions to the problem. Discovering this manifold is analogous to the study of the identifiability of a physical system with a non-injective input-to-output map. Thus, understanding and characterizing such manifold becomes essential for gaining insights into the system and generating accurate emulators efficiently.

In this paper, we introduce inVAErt networks, a new approach for jointly learning the forward and inverse model maps, the distribution of the model outputs, and also discovering non-identifiable input manifolds. We also claim the following novel contributions

- An inVAErt network goes beyond emulation, learning all essential aspects of a physics-based model and combining these aspects in a unique surrogate, or, in other words, performing model *synthesis*.
- Our approach does not require strong assumptions about the distribution of inputs and outputs, and provides a comprehensive sampling-based (nonparametric) characterization of the set of possible solutions to an inverse problem.
- We formally analyze the interaction between penalty selection and accuracy in the emulator and decoder through first-order loss stationarity. In addition we investigate the phenomenon of posterior collapse, and explore several latent space sampling strategies to improve inferential performance.

After an inVAErt network is trained, it greatly enhances the possibility of interacting in real-time with a given physical model, by providing input parameters, spatial locations and time corresponding to a single or multiple observations. For the solution of Bayesian inverse problems, an inVAErt network can be seen as jointly learning both an emulator and a proposal distribution, hence it can be easily integrated, for example, in MCMC simulation. For the interested reader, the source code and examples can be found at `https://github.com/desResLab/InVAErt-networks`.

While finalizing this paper, we became aware of a similar network architecture studied in [1] and later applied to the problem of quantum chromodynamics [2]. However, there are significant differences between the proposed inVAErt networks and the architecture in [1]. First we separate the emulator from the variational encoder, opening new possibilities to use architectures specifically designed to learn space/time dependent solutions on structured or unstructured variable arrangements. Second we train a density estimator for the model outputs that enables fast selection of representative outputs in the training dataset, and is particularly useful to rank output combinations for inference with missing data. Third, we investigate how the choice of the latent space dimensionality affects inversion accuracy, and propose three possible approaches to generate latent space samples that are better adapted to specific model outputs. Fourth, we investigate an extensive number of numerical tests, from simple maps to ODEs and PDEs where time and space are explicitly considered as parameters. Fifth, we formally analyze the interaction between penalty selection, emulator and decoder accuracy under first-order loss stationarity, and discuss the phenomenon of posterior collapse. Finally, we have performed endless tests to minimize the number of network parameters and training samples without compromising accuracy.

In addition, we would like to point out the similarities between our approach and the emerging paradigm of simulation-based (or likelihood-free) inference (SBI, see, e.g., [12]). SBI combines amortized inference with a neural approximation of the posterior distribution, likelihood function or likelihood ratio [41, 13, 38]. Once trained, the conditional distribution corresponding to a new set of observations is estimated without requiring additional model solutions. The same can be achieved by training our inVAErt networks from deterministic or noisy inputs and outputs. In the first case, the analysis of the latent space is indicative of the *structural* identifiability of the system, in the second structural and *practical* identifiability coexist.

The paper is organized as follows. In Section 2, we provide an in-depth description of each component in the inVAErt network. Section 3 presents a rigorous analysis of its properties, with a focus on assessing the stationarity of the loss function gradients and latent space sampling. A number of numerical examples, covering input-to-output maps and the solution of nonlinear ODEs and PDEs in both spatial and temporal domains, are discussed in Section 4.

2 The inVAErt network

The network has three main components, an architecture-agnostic neural emulator, a density estimator for the model outputs, and a decoder network equipped with a VAE for latent space discovery and model inversion. Each of these components is explained in detail in the following sections.

2.1 Neural emulator

Consider a physics-based system defined through an input-to-output map or the solution of an ordinary or partial differential equation

$$\boldsymbol{y} = \mathcal{F}\left(\boldsymbol{x}, t, \boldsymbol{\Phi}\right). \tag{1}$$

The operator $\mathcal{F}$ can be described as a generic *emulator* (sometimes referred to as *observation operator* [53]), mapping the spatial coordinates $\boldsymbol{x} \in \mathbb{R}^D$, time t and a set of additional problem-dependent parameters $\boldsymbol{\Phi} = \{\phi_1, \phi_2, \cdots\}$ to the system outputs $\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}$. To simplify the notation, we lump the model *inputs* as $\boldsymbol{v} = [\boldsymbol{x}, t, \boldsymbol{\Phi}] \in \boldsymbol{\mathcal{V}}$ such that this forward process can be rewritten as

$$\boldsymbol{y} = \mathcal{F}\left(\boldsymbol{v}\right) = NN_e\left(\boldsymbol{v}; \boldsymbol{\theta}_e\right), \tag{2}$$

where a generic *encoder*, or *neural emulator* NN_e with trainable parameters $\boldsymbol{\theta}_e$ is used in place of the abstract map $\mathcal{F}\left(\cdot\right)$.

In practice, training NN_e from $(\boldsymbol{v}, \boldsymbol{y})$ pairs can be challenging for complex physics-based systems and one usually provides additional information as input, or designs specific network architectures to improve accuracy. For example, recurrent or residual networks [43, 17, 56] are used to emulate the evolution operator (flow map) of dynamical systems. These networks utilize one or multiple solutions in the past to predict the future response. When a physical system involves space, convolutional neural networks (CNN) or message-passing graph neural networks (GNN) [52, 47, 22] can be used to gather neighbor information in structured and unstructured discretized domains, respectively. Thus, to approximate the system output $\boldsymbol{y}$ at time instance t_n and location $\boldsymbol{x}$, i.e. $\boldsymbol{y}(t_n, \boldsymbol{x}, \boldsymbol{\Phi})$, we introduce the auxiliary set $\mathcal{D}_{\boldsymbol{v}}$ of the form:

$$\mathcal{D}_{\boldsymbol{v}} = \left\{ \cdots, \boldsymbol{y}\big(t_{n-2}, \mathcal{B}(\boldsymbol{x}), \boldsymbol{\Phi}\big), \boldsymbol{y}\big(t_{n-1}, \mathcal{B}(\boldsymbol{x}), \boldsymbol{\Phi}\big) \right\},$$

where $\mathcal{B}(\boldsymbol{x})$ represents the *neighborhood* of node $\boldsymbol{x}$ when dealing with a mesh (graph)-based discretization of a physical system that involves space, e.g. k-hop in GNNs [22]. For instance, in 1D, we can have $\mathcal{B}(x) = \{x - \Delta x, x + \Delta x\}$ as a collection of the left/right neighbors at each node. Consequently, our neural emulators can be redefined in general as: $\boldsymbol{y} = NN_e(\boldsymbol{v}, \mathcal{D}_{\boldsymbol{v}}; \boldsymbol{\theta}_e)$, and $\mathcal{D}_{\boldsymbol{v}}$ is disregarded when dealing with easy-to-learn forward maps.

In the numerical examples of Section 4 involving time discretizations, we adopt the ResNet [43] model, which learns to update the value of the state $\boldsymbol{y}$ from two successive time steps as

$$\boldsymbol{y}(t_n, \boldsymbol{x}, \boldsymbol{\Phi}) = \boldsymbol{y}(t_{n-1}, \boldsymbol{x}, \boldsymbol{\Phi}) + NN_e\left(\boldsymbol{v}, \mathcal{D}_{\boldsymbol{v}}; \boldsymbol{\theta}_e\right), \tag{3}$$

and $\mathcal{D}_{\boldsymbol{v}}$ can be formulated to contain solutions from a larger number of steps in the past. This simple change in architecture with respect to fully-connected networks leads to significantly more accurate emulators.

2.2 Density estimator for model outputs

Besides the forward emulator, we would also like the ability to generate representative output samples $\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}$. Consider the situation where a collection of input samples $\boldsymbol{v} \sim p_{\boldsymbol{v}}(\boldsymbol{v})$ (this is usually taken as an uniform distribution for the test cases in Section 4) is available, and the corresponding outputs $\boldsymbol{y}$ have distribution $p_{\boldsymbol{y}}(\boldsymbol{y})$, i.e. $\mathcal{F}(\boldsymbol{v}) = \boldsymbol{y} \sim p_{\boldsymbol{y}}(\boldsymbol{y})$. In this context, we train a K-layer Real-NVP normalizing flow density estimator [14]

$$\boldsymbol{y} \approx \boldsymbol{z}_K = NN_f\left(\boldsymbol{z}_0; \boldsymbol{\theta}_f\right), \text{ where } \boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) . \tag{4}$$

Normalizing flows [46] (in their discrete form) consist of a collection of K bijective transformations $\mathscr{T} = \boldsymbol{f}_K \circ \boldsymbol{f}_{K-1} \circ \cdots \circ \boldsymbol{f}_1$ (parameterized using $\boldsymbol{\theta}_f = \boldsymbol{\theta}_{f,1} \cup \cdots \cup \boldsymbol{\theta}_{f,K}$) trained to learn the mapping between an easy-to-sample distribution to the density of the available samples. Under this transformation, the underlying density is modified following the change of variable formula

$$p_{\boldsymbol{y}}(\boldsymbol{y}) = p_{\boldsymbol{z}}(\boldsymbol{z}) |\det \mathbf{J}|^{-1}, \text{ or, equivalently } \log p_{\boldsymbol{y}}(\boldsymbol{y}) = \log q_0(\boldsymbol{z}_0) + \sum_{k=1}^{K} \log \left| \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{z}_{k-1}} \right|^{-1}, \tag{5}$$

where $q_0(\cdot)$ is usually the multivariate standard normal $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\boldsymbol{z}_k = \boldsymbol{f}_k(\boldsymbol{z}_{k-1}; \boldsymbol{\theta}_{f,k})$, $k = 1, \ldots, K$. The parameters $\boldsymbol{\theta}_f$ are determined by maximizing the log-likelihood (MLE) of the available samples $\{\boldsymbol{y}_i\}_{i=1}^N$, i.e.

$$\sum_{i=1}^{N} \log p_{\boldsymbol{y}}(\boldsymbol{y}_i) = \sum_{i=1}^{N} \log q_0(\boldsymbol{z}_{i,0}) - \sum_{i=1}^{N} \sum_{k=1}^{K} \log \left| \frac{\partial \boldsymbol{z}_{i,k}}{\partial \boldsymbol{z}_{i,k-1}} \right| . \tag{6}$$

Of the many possible choices for the transformation $\boldsymbol{f}_k$, $k = 1, \ldots, K$, an ideal candidate should be easy to invert and the computational complexity of computing the Jacobian determinant should increase linearly with the input dimensionality. Dinh et al. propose a block triangular autoregressive transformation, introducing the widely used real-valued non-volume preserving transformations (Real-NVP [14], but several other types of flows are discussed in the literature, see, e.g. [33, 40]). Given $\dim(\boldsymbol{y}) = m$, the method of Real-NVP considers a $m^* < m$ and defines the bijection $\boldsymbol{z}_k = \boldsymbol{f}_k(\boldsymbol{z}_{k-1}; \boldsymbol{\theta}_{f,k})$ through the easily invertible *affine coupling*

$$\begin{aligned} [\boldsymbol{z}_k]_{1:m^*} &= [\boldsymbol{z}_{k-1}]_{1:m^*} , \\ [\boldsymbol{z}_k]_{m^*+1:m} &= [\boldsymbol{z}_{k-1}]_{m^*+1:m} \odot \exp[\boldsymbol{s}_k([\boldsymbol{z}_{k-1}]_{1:m^*})] + \boldsymbol{t}_k([\boldsymbol{z}_{k-1}]_{1:m^*}) , \end{aligned} \tag{7}$$

where $[\cdot]_{a:b}$ is used to denote the range of components from a to b included, $\boldsymbol{s}_k \in \mathbb{R}^{m^*} \to \mathbb{R}^{m-m^*}$ and $\boldsymbol{t}_k \in \mathbb{R}^{m^*} \to \mathbb{R}^{m-m^*}$ are the scaling and translation functions implemented through dense neural networks, and $\odot$ denotes the Hadamard product. We alternate the variables that remain unchanged in each layer (see Section 3.5 in [14]) and use batch normalization as proposed in [14]. Finally, observe how the above coupling does not apply to one-dimensional transformations, i.e. $m = 1, y \in \mathbb{R}$. For such cases, we concatenate to y auxiliary independent standard Gaussian samples.

By training a density estimator on $\boldsymbol{y} \sim p_{\boldsymbol{y}}(\boldsymbol{y})$, we can quickly evaluate the likelihood of a given output $\boldsymbol{y}$, determining if it is representative or rare with respect to the selected training data. This provides an efficient way to quantify the expected performance of the decoder. A normalizing-flow based density estimator is also essential to provide information on the correlation between different outputs in $\boldsymbol{y}$. For situations where we would like to identify model parameters based on experimental measurements with missing data, the trained density estimator provides valuable information on possible ranges for the missing outputs that were seen during training. For example, assume we would like to determine all parameters $\boldsymbol{v}$ that correspond to $\boldsymbol{y}^* = [y_1^*, y_2^*, y_3]$, where the value of y_3 is missing. In such a case, one can sample a number of $\boldsymbol{y}$ from the trained density estimator and then replace y_1, y_2 with the fixed values y_1^*, y_2^*. Next, we rank the samples by the corresponding density values in order to select the most probable y_3 compatible with y_1^* and y_2^*.

2.3 Inverse modeling

Consider the situation where the dimension of the output space $\boldsymbol{\mathcal{Y}}$ is smaller than the dimension of the input space $\boldsymbol{\mathcal{V}}$, i.e., $\dim(\boldsymbol{\mathcal{Y}}) < \dim(\boldsymbol{\mathcal{V}})$.

In this case, to every input $\boldsymbol{v}$ we can associate a manifold[1] $\boldsymbol{\mathcal{M}_y}$ embedded in $\boldsymbol{\mathcal{V}}$ with $\dim(\boldsymbol{\mathcal{M}_y}) \leq \dim(\boldsymbol{\mathcal{V}})$ containing all the inputs producing the same output as $\boldsymbol{v}$, i.e.,

$$\mathcal{F}(\boldsymbol{v}') = \mathcal{F}(\boldsymbol{v}) = \boldsymbol{y}, \quad \forall\, \boldsymbol{v}' \in \boldsymbol{\mathcal{M}_y}.$$

As a result, it will not be possible to distinguish between two inputs $\boldsymbol{v}, \boldsymbol{v}' \in \boldsymbol{\mathcal{M}_y}$ from their outputs, or, in other words, the inputs in $\boldsymbol{\mathcal{M}_y}$ are *non-identifiable* from their common output $\boldsymbol{y}$. A parameter $\boldsymbol{v} \in \boldsymbol{\mathcal{V}}$ is identifiable by the map $\mathcal{F}$ if $\mathcal{F}(\boldsymbol{v}) = \mathcal{F}(\boldsymbol{v}') = \boldsymbol{y}$ implies $\boldsymbol{v} = \boldsymbol{v}'$ [49], that is, when $\boldsymbol{\mathcal{M}_y} = \{\boldsymbol{v}\}$. In general, we have that $\boldsymbol{\mathcal{M}_{y_1}} \cap \boldsymbol{\mathcal{M}_{y_2}} = \varnothing$ if $\boldsymbol{y}_1 \neq \boldsymbol{y}_2$. The collection of distinct manifolds $\boldsymbol{\mathcal{M}_y}$ forms a partition of $\boldsymbol{\mathcal{V}}$ with respect to a certain forward operator $\mathcal{F}$ and the dimensionality of $\boldsymbol{\mathcal{M}_y}$ may not be constant over $\boldsymbol{\mathcal{V}}$, depending on the rank of the Jacobian $\nabla\mathcal{F}$.

A dimensionality mismatch between $\boldsymbol{v}$ and $\boldsymbol{y}$ precludes the existence of an inverse, and may pose difficulties when constructing a pseudo-inverse

$$\boldsymbol{v} = \mathcal{G}\left(\boldsymbol{y}\right), \tag{8}$$

for which $\mathcal{F}\big(\mathcal{G}(\boldsymbol{y})\big) = \boldsymbol{y}$ and $\mathcal{G}$ is the inverse or pseudo-inverse operator. To overcome this problem and restore bijectivity, we introduce a latent space $\boldsymbol{\mathcal{W}}$ such that $\dim(\boldsymbol{\mathcal{V}}) \leq \dim(\widetilde{\boldsymbol{\mathcal{Y}}})$ where $\widetilde{\boldsymbol{\mathcal{Y}}} = \boldsymbol{\mathcal{W}} \times \boldsymbol{\mathcal{Y}}$ and $\tilde{\boldsymbol{y}} \in \widetilde{\boldsymbol{\mathcal{Y}}}$ is obtained via concatenation as $\tilde{\boldsymbol{y}} = [\boldsymbol{w}, \boldsymbol{y}]^T$ with $\boldsymbol{w} \in \boldsymbol{\mathcal{W}}$. This is similar to the concept of invertible neural networks in [3] and can be understood as a $\dim(\boldsymbol{\mathcal{W}})$-dimensional extension of $\boldsymbol{\mathcal{Y}}$. When the dimension of $\boldsymbol{\mathcal{M}_y}$ is constant, each one of the manifolds can be identified with the latent space $\boldsymbol{\mathcal{W}}$. Otherwise, $\boldsymbol{\mathcal{W}}$ must have a sufficiently high dimensionality to include the latent space representation of each $\boldsymbol{\mathcal{M}_y}$. As a concrete example, when all the above spaces are vector spaces and the forward map $\mathcal{F}$ is linear, i.e. with the matrix representation $\mathbf{F} \in \mathbb{R}^{\dim(\boldsymbol{\mathcal{Y}})\times\dim(\boldsymbol{\mathcal{V}})}$, the manifolds have the explicit form $\boldsymbol{\mathcal{M}_y} = \boldsymbol{v}^* + \mathbf{Ker}(\mathbf{F})$, for any particular $\boldsymbol{v}^*$ satisfying $\mathbf{F}\boldsymbol{v}^* = \boldsymbol{y}$. Therefore, each one of them can be identified with the kernel, or the null space $\mathbf{Ker}(\mathbf{F})$ of map $\mathbf{F}$.

We also consider the possibility that $\dim(\widetilde{\boldsymbol{\mathcal{Y}}}) > \dim(\boldsymbol{\mathcal{V}})$. As a consequence of the Whitney embedding theorem [61], any smooth manifold can be embedded into a higher dimensional Euclidean space. Intuitively, it is simpler to approximate a lower dimensional structure in a higher dimensional space. In fact, an embedding into a high-dimensional space may smooth some geometric singularities, i.e., self-intersections, cusps, etc., that appear in a low-dimensional space. Think, for example, about a curve that does not self-intersect if represented in $\mathbb{R}^3$, but it does when projected on $\mathbb{R}^2$.

A graphical representation of this decomposition is illustrated in Figure 1, where $\boldsymbol{v} \in \mathbb{R}^3$, $\boldsymbol{y} \in \mathbb{R}^2$ and $\boldsymbol{w} \in \mathbb{R}$, such that $\boldsymbol{\mathcal{M}_y}$ is an one-dimensional fiber. In this case, the space $\widetilde{\boldsymbol{\mathcal{Y}}}$ is obtained by extending $\boldsymbol{\mathcal{Y}}$ along the $\boldsymbol{w}$ direction. As discussed and shown in Figure 1, all $\boldsymbol{v} \in \boldsymbol{\mathcal{M}_{y^*}}$ are mapped to a single $\boldsymbol{y}^* \in \boldsymbol{\mathcal{Y}}$ through $\mathcal{F}$, while $\mathcal{F}(\boldsymbol{\mathcal{M}_{y^*}})$ varies in $\widetilde{\boldsymbol{\mathcal{Y}}}$ when augmented with $\boldsymbol{w}$. Also, as mentioned above, we would like to remark that under linear settings, i.e. when $\mathcal{F} = \mathbf{F}$, the above discussion is consistent with the rank–nullity theorem, stating that $\dim\left(\boldsymbol{\mathcal{M}_y}\right) + \dim\left(\boldsymbol{\mathcal{Y}}\right) = \dim\left(\boldsymbol{\mathcal{V}}\right)$.

In this paper, our goal is to learn the manifold $\boldsymbol{\mathcal{M}_y}$, conditioned on each model output $\boldsymbol{y}$, i.e. $\mathcal{F}(\boldsymbol{\mathcal{M}_y}) = \boldsymbol{y}$, using a VAE, or more precisely, a conditional VAE [50]. To do so, we consider a $\boldsymbol{\mathcal{W}}$-valued stochastic process $\{\boldsymbol{W_v}\}_{\boldsymbol{v}\in\boldsymbol{\mathcal{V}}}$ indexed by every $\boldsymbol{v} \in \boldsymbol{\mathcal{V}}$, such that the probability density of $\boldsymbol{W_v}$ is concentrated near the image of $\boldsymbol{\mathcal{M}_y}$ in $\boldsymbol{\mathcal{W}}$, which is responsible for the information gap between the spaces $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{Y}}$. Hence, we seek a neural network $NN_v(\cdot\,;\boldsymbol{\theta}_v) : \boldsymbol{\mathcal{V}} \to \boldsymbol{\mathcal{W}}$ such that

$$NN_v(\boldsymbol{v};\boldsymbol{\theta}_v) = [\boldsymbol{\mu_v}, \boldsymbol{\sigma_v}],\ \boldsymbol{W_v} \sim \mathcal{N}\big(\boldsymbol{\mu_v}, \mathbf{diag}(\boldsymbol{\sigma_v^2})\big), \tag{9}$$

and another neural network $NN_d(\cdot,\cdot\,;\boldsymbol{\theta}_d) : \boldsymbol{\mathcal{W}} \times \boldsymbol{\mathcal{Y}} \to \boldsymbol{\mathcal{V}}$ such that the parameters $(\boldsymbol{\theta}_v, \boldsymbol{\theta}_d)$ are optimally trained and

$$\mathbb{E}_{\boldsymbol{W_v}} \|\mathcal{F}\big(NN_d(\boldsymbol{W_v}, \boldsymbol{y}; \boldsymbol{\theta}_d)\big) - \boldsymbol{y}\|^2 \approx 0\,. \tag{10}$$

Therefore, $\boldsymbol{W_v}$ represents the missing information about $\boldsymbol{v}$ with respect to the forward pass $\boldsymbol{y} = \mathcal{F}(\boldsymbol{v})$, with samples generated during training [29] as

$$\boldsymbol{W_v} = \boldsymbol{\mu_v} + \boldsymbol{\sigma_v} \odot \boldsymbol{\epsilon} \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\,.$$

[1] In our discussion we assume the preimage of a single output $\{\boldsymbol{y}\}$ to have a local Euclidean structure, consistent with the nature of the problems we analyze in Section 4. However, the proposed approach is based on sampling and therefore, in principle, we can deal with more general settings and therefore the term *manifold*, with an abuse of terminology, is sometimes used as a synonym for *fiber*, or *level set* of $\mathcal{F}$.

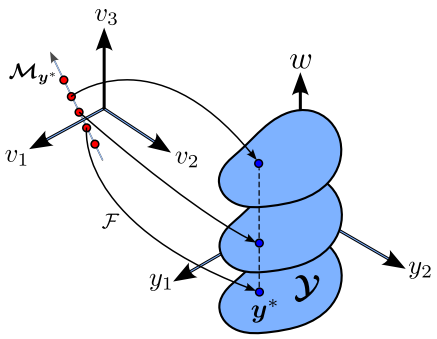


Figure 1: A graphical illustration of a non-identifiable parameter embedding and how input-output bijectivity is restored by extending the output space.

We also impose that the samples of $\boldsymbol{W}_{\boldsymbol{v}}$ provide unbiased information about $\boldsymbol{v}$ in the sense that

$$\mathbb{E}_{\boldsymbol{W}_{\boldsymbol{v}}} \|NN_d(\boldsymbol{W}_{\boldsymbol{v}}, \mathcal{F}(\boldsymbol{v}); \boldsymbol{\theta}_d) - \boldsymbol{v}\|^2 \approx 0, \ \forall \boldsymbol{v} \in \boldsymbol{\mathcal{V}} \, . \tag{11}$$

Remark. *We use the notation $\boldsymbol{w}$ to denote a generic latent variable from the space $\boldsymbol{\mathcal{W}}$, while we utilize $\boldsymbol{W}_{\boldsymbol{v}}$ to show the dependence of latent space w.r.t the training samples of $\boldsymbol{v}$.*

2.3.1 An introduction to variational autoencoders

The VAE uses deep neural network-based parametric models to handle the intractability in traditional inference tasks, as well as to learn a low-dimensional embedding of the input feature $\boldsymbol{v}$ [29, 30, 21, 15]. Given a distribution $\boldsymbol{v} \sim p_{\boldsymbol{v}}(\boldsymbol{v})$ to be learned, we consider an unobserved latent variable $\boldsymbol{w} \sim p_{\boldsymbol{w}}(\boldsymbol{w})$, that correlates to $\boldsymbol{v}$ via the law of total probability

$$p_{\boldsymbol{v}}(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{w} \sim p_{\boldsymbol{w}}(\boldsymbol{w})} \big[p(\boldsymbol{v}|\boldsymbol{w}) \big] \, . \tag{12}$$

The conditional probability $p(\boldsymbol{v}|\boldsymbol{w})$ quantifies the relation between a latent variable $\boldsymbol{w}$ (from an easy-to-sample prior distribution $p_{\boldsymbol{w}}(\boldsymbol{w})$, usually a standard Gaussian) and a specific input feature $\boldsymbol{v} \sim p_{\boldsymbol{v}}(\boldsymbol{v})$. However, computing $p(\boldsymbol{v}|\boldsymbol{w})$ is usually intractable, so one applies Bayes theorem to get

$$p_{\boldsymbol{v}}(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{w} \sim p_{\boldsymbol{w}}(\boldsymbol{w})} \left[\frac{p_{\boldsymbol{v}}(\boldsymbol{v}) p(\boldsymbol{w}|\boldsymbol{v})}{p_{\boldsymbol{w}}(\boldsymbol{w})} \right] = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w}|\boldsymbol{v})} \left[\frac{p(\boldsymbol{v}, \boldsymbol{w})}{p(\boldsymbol{w}|\boldsymbol{v})} \right] \, . \tag{13}$$

Drawing $\boldsymbol{w}$ samples from the exact posterior distribution $p(\boldsymbol{w}|\boldsymbol{v})$ is also intractable but a variational approximation $q(\boldsymbol{w}|\boldsymbol{v}) \approx p(\boldsymbol{w}|\boldsymbol{v})$ can be used instead. The Jensen's inequality and a log transformation are then applied to obtain

$$\log p_{\boldsymbol{v}}(\boldsymbol{v}) \geq -\mathcal{KL}\big[q(\boldsymbol{w}|\boldsymbol{v}) \| p_{\boldsymbol{w}}(\boldsymbol{w})\big] + \mathbb{E}_{\boldsymbol{w} \sim q(\boldsymbol{w}|\boldsymbol{v})} \big[\log p(\boldsymbol{v}|\boldsymbol{w}) \big], \tag{14}$$

where the term $\mathcal{KL}$ denotes the Kullback–Leibler (KL) divergence between two distributions p_1 and p_2, defined as

$$\mathcal{KL}\big[p_1(\boldsymbol{x}) \| p_2(\boldsymbol{x})\big] = \mathbb{E}_{\boldsymbol{x} \sim p_1(\boldsymbol{x})} \big[\log p_1(\boldsymbol{x}) - \log p_2(\boldsymbol{x}) \big], \tag{15}$$

and the right hand side of (14) is also referred to as the *evidence lower bound* (ELBO). Then, maximizing $\log p_{\boldsymbol{v}}(\boldsymbol{v})$ is equivalent to minimize the negative ELBO, leading to an optimization problem of the form

$$q(\boldsymbol{w}|\boldsymbol{v}), p(\boldsymbol{v}|\boldsymbol{w}) = \underset{\tilde{q}(\boldsymbol{w}|\boldsymbol{v};\boldsymbol{\delta}), \tilde{p}(\boldsymbol{v}|\boldsymbol{w};\boldsymbol{\tau})}{\operatorname{argmin}} \big\{ \mathcal{KL}\big[\tilde{q}(\boldsymbol{w}|\boldsymbol{v};\boldsymbol{\delta}) \| p_{\boldsymbol{w}}(\boldsymbol{w})\big] - \mathbb{E}_{\boldsymbol{w} \sim \tilde{q}(\boldsymbol{w}|\boldsymbol{v};\boldsymbol{\delta})} \big[\log \tilde{p}(\boldsymbol{v}|\boldsymbol{w};\boldsymbol{\tau}) \big] \big\} \, . \tag{16}$$

The candidate distributions, $\tilde{q}(\boldsymbol{w}|\boldsymbol{v};\boldsymbol{\delta})$ and $\tilde{p}(\boldsymbol{v}|\boldsymbol{w};\boldsymbol{\tau})$, are parameterized by variational and generative parameters, $\boldsymbol{\delta}$ and $\boldsymbol{\tau}$, respectively [29]. The first objective of equation (16) aims to align $\tilde{q}(\boldsymbol{w}|\boldsymbol{v};\boldsymbol{\delta})$ with the prior distribution of $\boldsymbol{w}$ (usually a standard Gaussian $\boldsymbol{w} = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$), while the second objective focuses on minimizing the reconstruction error. As previously mentioned, VAE models both $\tilde{q}$ and $\tilde{p}$ via deep neural networks and $\tilde{q}$ is usually chosen as an uncorrelated multivariate Gaussian distribution [30], with mean and variance equal to

$$\begin{aligned} \boldsymbol{\delta} &= [\boldsymbol{\mu}_{\boldsymbol{v}}, \boldsymbol{\sigma}_{\boldsymbol{v}}] = NN_v(\boldsymbol{v}; \boldsymbol{\theta}_v) \, , \\ \tilde{q}(\boldsymbol{w}|\boldsymbol{v};\boldsymbol{\delta}) &= \mathcal{N}\big(\boldsymbol{\mu}_{\boldsymbol{v}}, \mathbf{diag}(\boldsymbol{\sigma}_{\boldsymbol{v}}^2)\big). \end{aligned} \tag{17}$$

The network NN_v is referred to as the variational encoder, characterized by the trainable parameter set $\boldsymbol{\theta}_v$. Finally, (16) is solved by stochastic gradient descent using the reparameterization trick [29]. This ensures the gradient and expectation operator can be exchanged, while also requiring minimal modifications to the deterministic back-propagation algorithm.

2.4 InVAErt network training

A complete training workflow of the proposed inVAErt network is illustrated in Figure 2. Forward model evaluations are learned using the deterministic emulator NN_e. The network learns to generate model outputs from their density thanks to the normalizing flow density estimator NN_f. The non-identifiable manifold $\boldsymbol{\mathcal{M}_y}$ and the inverse model map are learned through the variational encoder and decoder (NN_v, NN_d). As mentioned earlier, the auxiliary input data $\mathcal{D}_{\boldsymbol{v}} = \{\mathcal{D}_{\boldsymbol{v}_i}\}_{i=1}^N$ only helps the emulator NN_e to approximate complex forward processes and here we include it for generality. Besides, if not specified otherwise, we adopt notations $\boldsymbol{\mu}, \boldsymbol{\sigma}$ instead of $\boldsymbol{\mu_v}, \boldsymbol{\sigma_v}$ in the following sections for simplicity.

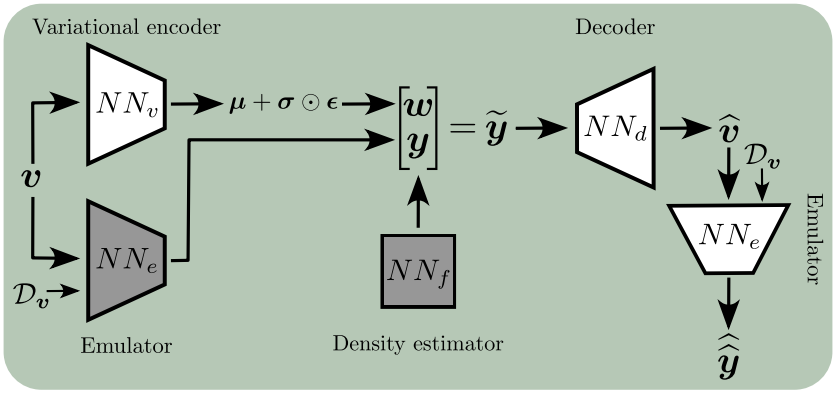


Figure 2: Training diagram for the inVAErt network. Emulator and density estimator are colored in gray, as training can be performed separately for these components using the true outputs $\boldsymbol{y}$.

For a given collection of N independent samples $\boldsymbol{v} \sim p_{\boldsymbol{v}}(\boldsymbol{v})$ in $\boldsymbol{\mathcal{V}}$ (e.g. uniformly distributed as assumed in the numerical examples in Section 4) and their corresponding outputs $\{(\boldsymbol{v}_i, \boldsymbol{y}_i)\}_{i=1}^N$, an optimal set of parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_e \cup \boldsymbol{\theta}_f \cup \boldsymbol{\theta}_v \cup \boldsymbol{\theta}_d$ can be obtained by minimizing the following loss function:

$$\mathcal{L} = \lambda_e \mathcal{L}_e + \lambda_v \mathcal{L}_v + \lambda_f \mathcal{L}_f + \lambda_d \mathcal{L}_d + \lambda_r \mathcal{L}_r \,, \tag{18}$$

where

$$\begin{aligned}
\mathcal{L}_e(\boldsymbol{\theta}_e) &= \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \widehat{\boldsymbol{y}}_i\|_2^2 \,, \\
\mathcal{L}_v(\boldsymbol{\theta}_v) &= \frac{1}{2 \cdot N} \sum_{i=1}^{N} \sum_{k=1}^{\dim(\boldsymbol{w})} \left(\mu_{i,k}^2 + \sigma_{i,k}^2 - \log(\sigma_{i,k}^2) - 1 \right) \,, \\
\mathcal{L}_f(\boldsymbol{\theta}_f) &= -\frac{1}{N} \sum_{i=1}^{N} \left(\log q_0(\boldsymbol{z}_{i,0}) + \sum_{k=1}^{K} \log \left| \frac{\partial \boldsymbol{z}_{i,k}}{\partial \boldsymbol{z}_{i,k-1}} \right|^{-1} \right) \,, \\
\mathcal{L}_d(\boldsymbol{\theta}_v, \boldsymbol{\theta}_d) &= \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{v}_i - \widehat{\boldsymbol{v}}_i\|_2^2 \,, \\
\mathcal{L}_r(\boldsymbol{\theta}_e, \boldsymbol{\theta}_v, \boldsymbol{\theta}_d) &= \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \widehat{\widehat{\boldsymbol{y}}}_i\|_2^2 \,,
\end{aligned} \tag{19}$$

denote the forward MSE loss for the emulator NN_e, KL divergence loss for the VAE encoder NN_v, MLE loss for the Real-NVP density estimator NN_f, reconstruction MSE loss for the decoder model

NN_d and another forward MSE loss to constrain the inverse modeling, respectively. Besides, λ_e, λ_v, λ_f, λ_d, λ_r are penalty coefficients associated with each loss function component.

Note that, in the current implementation, the training of an inVAErt network can be accomplished by independently training (1) the forward emulator (NN_e), (2) the density estimator (NN_f) and (3) the inverse model ($NN_v + NN_d$), using the *labels* $\{\boldsymbol{y}_i\}_{i=1}^N$ (instead of their predicted values) for each of these three tasks. To further emphasize this fact, in Figure 2, we use gray color for the network components that can be trained separately, and denote the output labels as $\boldsymbol{y}$. However, end-to-end training might be required for extensions to stochastic models or noisy inputs and outputs, which are not discussed in this paper. A pseudocode for further explaining the current training workflow is given in Algorithms 1 to 3.

Algorithm 1 Training the emulator NN_e using the dataset $\{\boldsymbol{v}_i, \boldsymbol{y}_i\}_{i=1}^N$

for epoch = $1, 2, \cdots$ **do**
  for $i = 1 : N$ **do**
    $\widehat{\boldsymbol{y}}_i = NN_e(\boldsymbol{v}_i, \mathcal{D}_{\boldsymbol{v}_i}; \boldsymbol{\theta}_e)$
  end for
  Calculate the MSE loss $\mathcal{L}_e$ using the predictions $\widehat{\boldsymbol{y}}_i$ and the labels $\boldsymbol{y}_i$
  Optimize $\boldsymbol{\theta}_e$ through gradient descent w.r.t $\mathcal{L}_e$
end for

Algorithm 2 Training the density estimator NN_f using only the dataset $\{\boldsymbol{y}_i\}_{i=1}^N$

for epoch = $1, 2, \cdots$ **do**
  for $i = 1 : N$ **do**
    $\boldsymbol{z}_{i,0} = NN_f^{-1}(\boldsymbol{y}_i; \boldsymbol{\theta}_f)$
  end for
  Calculate the negative log-likelihood loss $\mathcal{L}_f$ from each map $\boldsymbol{y}_i \to \boldsymbol{z}_i$ through NF
  Optimize $\boldsymbol{\theta}_f$ through gradient descent w.r.t $\mathcal{L}_f$
end for

Algorithm 3 Training the inverse model (NN_v, NN_d) using the dataset $\{\boldsymbol{v}_i, \boldsymbol{y}_i\}_{i=1}^N$

for epoch = $1, 2, \cdots$ **do**
  for $i = 1 : N$ **do**
    $[\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i] = NN_v(\boldsymbol{v}_i; \boldsymbol{\theta}_v)$ ▷ VAE encoding
    $\boldsymbol{w}_i = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}\,, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ $\boldsymbol{w}_i \sim q(\boldsymbol{w}_i|\boldsymbol{v}_i) = \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)$
    $\widehat{\boldsymbol{v}}_i = NN_d(\boldsymbol{w}_i, \boldsymbol{y}_i; \boldsymbol{\theta}_d)$ ▷ Decoding
    $\widehat{\widehat{\boldsymbol{y}}}_i = NN_e(\widehat{\boldsymbol{v}}_i, \mathcal{D}_{\boldsymbol{v}_i}; \boldsymbol{\theta}_e)$ ▷ Emulator re-constraining
  end for
  Calculate the KL divergence loss $\mathcal{L}_v$ using the predicted means $\boldsymbol{\mu}_i$ and variances $\boldsymbol{\sigma}_i$
  Calculate the MSE loss $\mathcal{L}_d$ using the inverse predictions $\widehat{\boldsymbol{v}}_i$ and the exact inputs $\boldsymbol{v}_i$
  Calculate the MSE loss $\mathcal{L}_r$ using the emulator predictions $\widehat{\widehat{\boldsymbol{y}}}_i$ w.r.t $\widehat{\boldsymbol{v}}_i$, and the labels $\boldsymbol{y}_i$
  Optimize $(\boldsymbol{\theta}_v, \boldsymbol{\theta}_d)$ through gradient descent w.r.t the joint loss $\lambda_d\mathcal{L}_d + \lambda_v\mathcal{L}_v + \lambda_r\mathcal{L}_r$
end for

Note that the loss $\mathcal{L}_r$ is consistent with condition (10), where the forward operator $\mathcal{F}$ is approximated by the previously trained neural emulator NN_e (using $\mathcal{L}_r$ to fine-tune the emulator parameter $\boldsymbol{\theta}_e$ is not recommended). However, for systems with less complex inverse processes, imposing $\mathcal{L}_r$ is usually not necessary for achieving good performance. Therefore, we do not enforce $\mathcal{L}_r$ in a few of our numerical experiments (e.g. Section 4.1, non-periodic part of Section 4.2, Section 4.3).

2.5 InVAErt network interrogation

Once trained, an inVAErt network can be used to answer a number of interesting questions about the physical system it synthesizes. Essentially, we have the ability to control two trained samplers on $\boldsymbol{y}$

and $\boldsymbol{w}$ to investigate different types of inverse problems, and use the trained emulator for verification or additional data generation. A diagram of how the network is used during inference is shown in Figure 3. Note that appropriate auxiliary data must be used when evaluating a trained emulator from an inverse prediction $\widehat{\boldsymbol{v}}$, hence the notation $\mathcal{D}_{\widehat{\boldsymbol{v}}}$

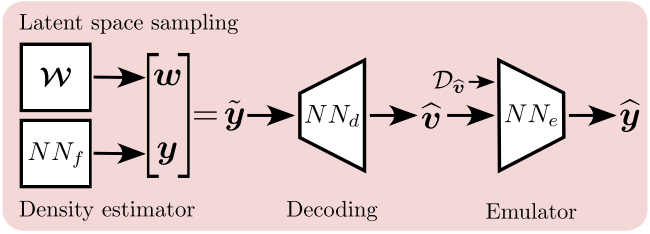


Figure 3: Sampling and inference diagram for the inVAErt network.

We might be interested, as a first use case, in sampling from $\boldsymbol{\mathcal{M}}_{\boldsymbol{y}^*}$, the manifold of input parameters corresponding to the specific model output $\boldsymbol{y}^*$. To do so, we sample a number of latent space realizations $\boldsymbol{w}$ from a multivariate standard normal or, alternatively, we use one of the approaches discussed in Section 3.3. We then concatenate these realizations with the observation $\boldsymbol{y}^*$ and feed the resulting vector to the decoder. Note that this approach produces multiple possible input parameters and therefore characterizes the inverse properties of the system in a way that is superior to most regularization approaches, where a single solution is promoted, instead of the $\boldsymbol{\mathcal{M}}_{\boldsymbol{y}^*}$ manifold. Note also that the inputs $\widehat{\boldsymbol{v}}$ obtained from the decoder can include time and spatial variables (see examples in Section 4.4 and 4.5).

Alternatively, we can sample from the density of the model outputs $\boldsymbol{y}$ using NN_f and set a constant value $\boldsymbol{w}^*$ for the latent variable. Ideally, decoding under this process, as opposed to sampling from $\boldsymbol{\mathcal{M}}_{\boldsymbol{y}}$, allows one to approximate the manifold transverse to $\boldsymbol{\mathcal{M}}_{\boldsymbol{y}}$ (which, in the linear case, coincides with the orthogonal complement ($\boldsymbol{\mathcal{M}}_{\boldsymbol{y}}^{\perp} = \boldsymbol{\mathcal{V}} \setminus \boldsymbol{\mathcal{M}}_{\boldsymbol{y}}$). This transverse manifold is embedded in $\boldsymbol{\mathcal{V}}$, and is associated with the highest sensitivity in the model outputs.

In addition, we can also determine a local collection of non-identifiable neighbors of a given input $\boldsymbol{v}^*$. To do so, we decode the concatenation of $\mathcal{F}(\boldsymbol{v}^*)$ and samples of $\boldsymbol{w}$ from the multivariate normal distribution $\mathcal{N}\big(\boldsymbol{\mu}_{\boldsymbol{v}^*}, \mathbf{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{v}^*})\big)$, as obtained from the variational encoder NN_v for the input $\boldsymbol{v}^*$. This process should help in parameter searches to identify directions along which the cost function, formulated in terms of the system outputs, remains constant.

Additional examples related to the practical interrogation of an inVAErt network are provided in Section 4.

3 Analysis of the method

To successfully train the proposed inVAErt network, we empirically find that is important to select appropriate penalty coefficients in (18) and avoid posterior collapse. Thus we formally investigate these two aspects through first-order stationarity conditions in the loss function. In doing so, we focus on training NN_v, NN_d for inverse predictions, and assume the exact forward model $\mathcal{F}$ is utilized to compute $\mathcal{L}_r$, while in practice, it is replaced by the trained emulator NN_e. Additionally, we assume $\mathcal{F}$ is Fréchet differentiable. As a consequence, its first-order Taylor expansion at the input $\boldsymbol{v}$ has the form:

$$\mathcal{F}(\boldsymbol{v} + \alpha\Delta\boldsymbol{v}) = \mathcal{F}(\boldsymbol{v}) + \alpha\nabla\mathcal{F}(\boldsymbol{v})\Delta\boldsymbol{v} + o(\alpha)\ , \tag{20}$$

where $\nabla\mathcal{F}$ denotes the Jacobian matrix of $\mathcal{F}$ with respect to the input $\boldsymbol{v}$, and the notation $o(\alpha)$ represents any term satisfying $o(\alpha)/\alpha \to 0$ as $\alpha \to 0$.

First, note that the relations (10) and (11) lead us to an optimization problem of the form

$$\begin{aligned} NN_d, NN_v &= \underset{\mathscr{D},\mathscr{V}\in\mathfrak{D},\mathfrak{V}}{\operatorname{argmin}}\ T(\mathscr{D},\mathscr{V})\ , \\ &= \underset{\mathscr{D},\mathscr{V}\in\mathfrak{D},\mathfrak{V}}{\operatorname{argmin}}\ \big[\lambda_r J(\mathscr{D},\mathscr{V}) + \lambda_d H(\mathscr{D},\mathscr{V}) + \lambda_v K(\mathscr{V})\big]\ , \end{aligned} \tag{21}$$

with the smooth candidate functions $\mathscr{D}$, $\mathscr{V}$ belonging to some general vector spaces $\mathfrak{D}$ and $\mathfrak{V}$, respectively. The target functional $T : \mathfrak{D} \times \mathfrak{V} \to \mathbb{R}$ is composed of three objective functionals J, H, K corresponding to our loss function components $\mathcal{L}_r$, $\mathcal{L}_d$ and $\mathcal{L}_v$ in equations (19).

Let $\{(\boldsymbol{v}_i, \boldsymbol{y}_i)\}_{i=1}^N$ (disregarding, for simplicity, the auxiliary data $\mathcal{D}_{\boldsymbol{v}}$) be a set of *i.i.d* training samples, and we associate each sample with a set of standard Gaussian random variables $\{\boldsymbol{\epsilon}_{ij}\}_{i,j=1}^{N,M}$. We then define $\boldsymbol{w}_{ij} = \mathscr{V}(\boldsymbol{v}_i)_{\boldsymbol{\epsilon}_{ij}} = \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_{ij} \odot \boldsymbol{\sigma}_i$ to indicate the latent realization obtained from the i-th training sample by adding noise $\boldsymbol{\epsilon}_{ij}$.

The first functional $J : \mathfrak{D} \times \mathfrak{V} \to \mathbb{R}$, as our main objective, consists of a discrete version of equation (10), and enforces consistency between the forward and inverse problems

$$J(\mathscr{D}, \mathscr{V}) \coloneqq \frac{1}{2 \cdot N \cdot M} \sum_{i,j=1}^{N,M} \|\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\|_2^2 \,, \tag{22}$$

where we use $\sum_{i,j=1}^{N,M}$ in place of the nested sum $\sum_{i=1}^{N} \sum_{j=1}^{M}$. The first constraint $H : \mathfrak{D} \times \mathfrak{V} \to \mathbb{R}$ focuses on minimizing the reconstruction loss of the system input:

$$H(\mathscr{D}, \mathscr{V}) \coloneqq \frac{1}{2 \cdot N \cdot M} \sum_{i,j=1}^{N,M} \|\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\|_2^2 \,. \tag{23}$$

Finally, the second constraint $K : \mathfrak{V} \to \mathbb{R}$ promotes compactness in the support of $\boldsymbol{w}$ by minimizing the divergence between the posterior distribution of the latent variables and a standard normal prior. While any statistical measure of discrepancy between distributions can be used, here we adopt the KL divergence (15), resulting in

$$K\big(\mathscr{V}\big) \coloneqq \frac{1}{2 \cdot N} \sum_{i=1}^{N} \sum_{k=1}^{\dim(\boldsymbol{w})} \big(\mu_{i,k}^2 + \sigma_{i,k}^2 - \log(\sigma_{i,k}^2) - 1\big) \,. \tag{24}$$

In practice, a local minimizer of the full objective may not attain $J(\mathscr{D}, \mathscr{V}) = 0$, $H(\mathscr{D}, \mathscr{V}) = 0$ nor $K(\mathscr{V}) = 0$. However, we can still affirm that the local minimizer must satisfy the stationarity conditions for the full objective. Hence, to better understand and interpret the results obtained using our approach, we determine the first-order conditions for a local minimizer of T. In the following sections, we discuss how these conditions yield insights into the trade-off between forward prediction errors, decoding errors and the phenomenon of posterior collapse.

3.1 Trade-off between forward prediction and decoding errors

The first-order stationarity conditions of the decoder involve only the functionals J and H:

$$0 = \delta T(\mathscr{D}, \mathscr{V}; \Delta\mathscr{D}) = \lambda_r \delta J(\mathscr{D}, \mathscr{V}; \Delta\mathscr{D}) + \lambda_d \delta H(\mathscr{D}, \mathscr{V}; \Delta\mathscr{D}) \,, \tag{25}$$

where the $\delta(\cdot)$ operator computes the Gâteaux derivative. Using the expressions derived in Appendix A, (25) becomes:

$$0 = \sum_{i,j=1}^{N,M} \Big\langle \lambda_r \nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\Big) + \lambda_d \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big), \Delta\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\Big\rangle \,, \tag{26}$$

where in practice, the perturbations $\Delta\mathscr{D}$ depend on the characteristics of the decoder network. If a sufficiently flexible architecture is selected, then for any i, j and an arbitrary $\Delta\boldsymbol{D}$, the decoder can produce a feasible perturbation $\Delta\mathscr{D}$ such that

$$\Delta\mathscr{D}_{ij}(\boldsymbol{w}_{kl}, \boldsymbol{y}_k) = \begin{cases} \Delta\boldsymbol{D} \,, & k = i, l = j \\ \boldsymbol{0} \,, & \text{otherwise} \end{cases} \,. \tag{27}$$

Then (26) implies

$$0 = \lambda_r \nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\Big) + \lambda_d \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big), \qquad \forall\, i, j \,. \tag{28}$$

This relation suggests a trade-off between the forward prediction and the decoding error. On one hand, we deduce

$$\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) = \boldsymbol{v}_i - \frac{\lambda_r}{\lambda_d} \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\Big), \qquad \forall\, i, j\ . \tag{29}$$

and therefore errors in the approximation of $\mathcal{F}$ with NN_e when training (NN_v, NN_d) might negatively affect the decoder. This suggests, for example, increasing the value of λ_d while letting λ_r fixed can reduce bias in $\mathscr{D}$. On the other hand, if the Jacobian of $\mathcal{F}$ is full-rank, we then have

$$\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) = \boldsymbol{y}_i - \frac{\lambda_d}{\lambda_r} \Big(\nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T\Big)^+ \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big), \qquad \forall i, j\ , \tag{30}$$

where $(\cdot)^+$ denotes the Moore-Penrose pseudo-inverse. Therefore, any error in the decoder becomes a forward prediction error. A decoder error is also scaled by the pseudo-inverse of the Jacobian, and thus depends on the local conditioning of the problem at $\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)$. Furthermore, this effect is proportional to λ_d/λ_r , showing that a balance must be reached when modulating the combination of λ_d, λ_r, based on the expected training complexity of the decoder and the accuracy of the trained emulator, respectively. For instance, when dealing with difficult-to-learn forward maps and so the expected emulator accuracy is limited, one would expect a relative large decoder error in equation (29) which can be mitigated by choosing a larger λ_d and a smaller λ_r in (29). This, however, comes at the expense of increasing the forward prediction error in (30). In practice, we find a suitable choice by letting $\lambda_d \gg \lambda_r$, which can effectively improve the decoder training when the forward map $\mathcal{F}$ is replaced by the trained emulator.

As an illustrative example, we evaluate these conditions when $\mathcal{F}$ is a full-rank linear map $\mathbf{F}$. In this case, (28) becomes

$$0 = \mathbf{F}^T\big(\mathbf{F}\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{y}_i\big) + \frac{\lambda_d}{\lambda_r}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big)\ , \qquad \forall i, j\ , \tag{31}$$

whence

$$\begin{aligned} \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) &= (\mathbf{F}^T\mathbf{F} + \frac{\lambda_d}{\lambda_r}\mathbf{I})^{-1}(\frac{\lambda_d}{\lambda_r}\boldsymbol{v}_i + \mathbf{F}^T\boldsymbol{y}_i)\ , \\ &= (\mathbf{F}^T\mathbf{F} + \frac{\lambda_d}{\lambda_r}\mathbf{I})^{-1}(\frac{\lambda_d}{\lambda_r}\mathbf{I} + \mathbf{F}^T\mathbf{F})\boldsymbol{v}_i\ , \\ &= \boldsymbol{v}_i\ , \qquad \forall i, j\ . \end{aligned} \tag{32}$$

Therefore, when the architecture is sufficiently expressive so that (27) holds, any stationary point leads to a perfect decoder in the linear case.

3.2 Analysis of posterior collapse

The phenomenon of posterior collapse affecting VAEs can also impact the proposed inVAErt network. It describes a scenario in which the learned posterior distribution $q(\boldsymbol{w}|\boldsymbol{v})$ becomes remarkably similar to the prior distribution $p_{\boldsymbol{w}}(\boldsymbol{w})$ (usually a standard Gaussian), resulting in a trivial latent space independent on the input $\boldsymbol{v}$. This collapse in the posterior undermines the fundamental nature of a generative model by preventing it from capturing inherent diversity between samples in $\boldsymbol{\mathcal{V}}$. Possible causes of posterior collapse are investigated in a number of articles [64, 9, 24, 16, 36, 45, 60] (see also Appendix B). Here we analyze such a phenomenon using stationarity conditions.

First, we have the first-order condition on T with respect to $\mathscr{V}$ as:

$$0 = \delta T(\mathscr{D}, \mathscr{V}; \Delta\mathscr{V}) = \lambda_r \delta J(\mathscr{D}, \mathscr{V}; \Delta\mathscr{V}) + \lambda_d \delta H(\mathscr{D}, \mathscr{V}; \Delta\mathscr{V}) + \lambda_v \delta K(\mathscr{V}; \Delta\mathscr{V})\ . \tag{33}$$

This condition has an impact on the probability density of the latent variable $\boldsymbol{w}$. In particular, any perturbation $\Delta\mathscr{V}$ induces perturbations in the mean $\Delta\boldsymbol{\mu}_i$ and standard deviation $\Delta\boldsymbol{\sigma}_i$. Therefore, the

condition in (33) can be represented as perturbations on the means and variances as (see Appendix A)

$$
\begin{aligned}
0 = & \frac{\lambda_r}{M} \sum_{j=1}^{M} \left\langle \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i \Big), \Delta \boldsymbol{\mu}_i \right\rangle \\
& + \frac{\lambda_d}{M} \sum_{j=1}^{M} \left\langle \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i \big), \Delta \boldsymbol{\mu}_i \right\rangle \\
& + \lambda_v \left\langle \boldsymbol{\mu}_i, \Delta \boldsymbol{\mu}_i \right\rangle \\
& + \frac{\lambda_r}{M} \sum_{j=1}^{M} \left\langle \boldsymbol{\epsilon}_{ij} \odot \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i \Big), \Delta \boldsymbol{\sigma}_i \right\rangle \\
& + \frac{\lambda_d}{M} \sum_{j=1}^{M} \left\langle \boldsymbol{\epsilon}_{ij} \odot \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i \big), \Delta \boldsymbol{\sigma}_i \right\rangle \\
& + \lambda_v \left\langle \boldsymbol{\sigma}_i - \boldsymbol{\sigma}_i^{-1}, \Delta \boldsymbol{\sigma}_i \right\rangle , \qquad \forall i \, .
\end{aligned}
\tag{34}
$$

Again, similar to (27), given a sufficiently flexible network architecture for the variational encoder, for any choice of $\Delta\boldsymbol{\mu}$, $\Delta\boldsymbol{\sigma}$ and given sample $\boldsymbol{v}_i$ we can find a perturbation $\Delta\mathscr{V}$ such that

$$
\Delta \mathscr{V}_i(\boldsymbol{v}_k) = \begin{cases} (\Delta\boldsymbol{\mu}, \Delta\boldsymbol{\sigma}) \, , & k = i \\ \mathbf{0} \, , & \text{otherwise} \end{cases} \, .
\tag{35}
$$

This yields an implicit relation for the mean

$$
\begin{aligned}
\boldsymbol{\mu}_i = & -\frac{\lambda_r}{\lambda_v} \frac{1}{M} \sum_{j=1}^{M} \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i \Big) \\
& - \frac{\lambda_d}{\lambda_v} \frac{1}{M} \sum_{j=1}^{M} \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i \big) \, , \qquad \forall i \, ,
\end{aligned}
\tag{36}
$$

as well as an implicit relation for the variance (Recall $\boldsymbol{w}_{ij} - \boldsymbol{\mu}_i = \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}_{ij}$)

$$
\begin{aligned}
\boldsymbol{\sigma}_i^2 = & -\frac{\lambda_r}{\lambda_v} \frac{1}{M} \sum_{j=1}^{M} (\boldsymbol{w}_{ij} - \boldsymbol{\mu}_i) \odot \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i \Big) \\
& - \frac{\lambda_d}{\lambda_v} \frac{1}{M} \sum_{j=1}^{M} (\boldsymbol{w}_{ij} - \boldsymbol{\mu}_i) \odot \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i \big) + \mathbf{1} \, , \qquad \forall i \, ,
\end{aligned}
\tag{37}
$$

where $\mathbf{1}$ denotes a vector of ones with size $\dim(\boldsymbol{w})$. Conditions (36) and (37) readily imply that when we have *exact decoding* over the samples, i.e.

$$
\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) = \boldsymbol{v}_i \, , \quad \forall i, j \, ,
$$

then any candidate function $\mathscr{V}$ satisfying the first order conditions will make

$$
\boldsymbol{\mu}_i = \mathbf{0} \quad \text{and} \quad \boldsymbol{\sigma}_i^2 = \mathbf{1} \, .
$$

In other words, we have posterior collapse. Although surprising, this is intuitive. When the decoding is exact, the decoder becomes independent of $\boldsymbol{w}$, which implies that optimal training is achieved at the global optimum of K. Hence, to avoid posterior collapse, the decoder must necessarily incur in some error. This aligns with previous studies which suggest reducing the decoder flexibility (e.g. [9, 6]). This agrees with our experience, i.e., reducing the number of hidden layers and neurons in the decoder can prevent posterior collapse when the penalty coefficients are kept fixed.

3.2.1 The role of discretization in posterior collapse

It is of interest to understand how to mitigate the risk of posterior collapse. In fact, our results show that when conditions (27) and (35) hold then we necessarily have posterior collapse. For any candidate function pair $(\mathscr{D}, \mathscr{V})$ satisfying the stationarity conditions, i.e.

$$
\begin{cases} \delta T(\mathscr{D}, \mathscr{V}; \Delta\mathscr{D}) = 0 \, , \\ \delta T(\mathscr{D}, \mathscr{V}; \Delta\mathscr{V}) = 0 \, . \end{cases}
\tag{38}
$$

If both conditions (27) and (35) are satisfied then the optimality conditions for $\boldsymbol{\mu}_i$ (36) and $\boldsymbol{\sigma}_i^2$ (37), as a consequence of (28), are reduced to

$$\begin{aligned}
\boldsymbol{\mu}_i &= -\frac{1}{\lambda_v \cdot M} \sum_{j=1}^{M} \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \Big(\lambda_r \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\big) \\
&\quad + \lambda_d \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big) \Big) = \mathbf{0} \,, \qquad \forall i \,, \\
\boldsymbol{\sigma}_i^2 &= \frac{1}{\lambda_v \cdot M} \sum_{j=1}^{M} (\boldsymbol{w}_{ij} - \boldsymbol{\mu}_i) \odot \nabla \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \Big(\lambda_r \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\big) \\
&\quad + \lambda_d \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big) \Big) + \mathbf{1} = \mathbf{1} \,, \qquad \forall i \,.
\end{aligned} \tag{39}$$

It would seem that posterior collapse always happens! It can be seen that this is a consequence of condition (27), which is much weaker than perfect decoding, and it occurs even when condition (35) does not hold. For this reason it is illustrative to interpret this condition. In practice, it implicitly represents a trade-off between the complexity of the decoder and the number of samples $\boldsymbol{w}_{ij}$ used in training. If the decoder is *too expressive* relative to the number of samples, then condition (27) states that the *derivatives* of the neural network with respect to its parameters are flexible enough to interpolate *any* function on the set of samples $(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)_{i,j=1}^{N,M}$. Hence, to prevent posterior collapse for a given architecture of the decoder, one can increase M accordingly. In this case, condition (27) may fail to hold, and thus we no longer suffer from posterior collapse at stationary points. Remark that this analysis involves the architecture of the decoder, whereas the variational encoder NN_v may be very flexible without causing posterior collapse. In other words, the decoder is mainly responsible for posterior collapse.

It is illustrative to study the limit $M \to \infty$, in which case we redefine the functionals J, H as:

$$J(\mathscr{D}, \mathscr{V}) \coloneqq \frac{1}{2N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{W}_{\boldsymbol{v}_i}} \Big[\big\| \mathcal{F}\big(\mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_i}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i \big\|_2^2 \Big] \,, \tag{40}$$

$$H(\mathscr{D}, \mathscr{V}) \coloneqq \frac{1}{2N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{W}_{\boldsymbol{v}_i}} \Big[\big\| \mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_i}, \boldsymbol{y}_i) - \boldsymbol{v}_i \big\|_2^2 \Big] \,, \tag{41}$$

where $\boldsymbol{W}_{\boldsymbol{v}_i} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{v}_i}, \boldsymbol{\sigma}_{\boldsymbol{v}_i}^2)$ (see also Section 2.3). Then the first-order condition (26) turns to

$$\begin{aligned}
0 = \sum_{i}^{N} \mathbb{E}_{\boldsymbol{W}_{\boldsymbol{v}_i}} \Big[\big\langle \lambda_r \nabla \mathcal{F}\big(\mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_i}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_i}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i \Big) \\
+ \lambda_d \big(\mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_i}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big), \Delta \mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_i}, \boldsymbol{y}_i) \big\rangle \Big] \,.
\end{aligned} \tag{42}$$

To obtain an expression of the form (28), it would be sufficient for the space generated by the functions

$$(\boldsymbol{W}_{\boldsymbol{v}_1}, \cdots, \boldsymbol{W}_{\boldsymbol{v}_N}) \mapsto \big(\Delta \mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_1}, \boldsymbol{y}_1), \cdots, \Delta \mathscr{D}(\boldsymbol{W}_{\boldsymbol{v}_N}, \boldsymbol{y}_N)\big) \,,$$

to be dense. However, the network architecture always constrains the function spaces it generates, and this fails to happen in practice. Characterizing the space generated by these perturbations, namely the derivatives of the network with respect to the parameters, is related to the concept of *neural tangent kernel* [25]. Therefore, our analysis implies that in order to avoid posterior collapse, it is more effective to choose a value of M larger than the complexity of the decoder, rather than focusing on adjusting the penalty coefficients in the loss function.

In our experiments, posterior collapse can be readily observed if the KL divergence loss $\mathcal{L}_v$ is nearly zero during the early phase of training. However, in some cases, $\mathcal{L}_v$ will grow and then converge to a non-zero constant value as the number of epoch increases. This aligns with our previous analysis as longer training leads to a larger number M of latent variables $\boldsymbol{w}_{ij}$ seen by the network for each $\boldsymbol{v}_i$. Also, as mentioned above, another practical strategy would be to reduce the complexity of the decoder NN_d. In terms of the penalty coefficients (see Equations (36) and (37)), increasing λ_d, λ_r while decreasing λ_v promotes larger differences between $\boldsymbol{\mu}_i$ and $\mathbf{0}$, as well as between $\boldsymbol{\sigma}_i^2$ and $\mathbf{1}$. This rule, $\lambda_d, \lambda_r \gg \lambda_v$, applies globally to all our experiments (if the loss $\mathcal{L}_r$ is used, see Appendix C) to avoid posterior collapse.

3.3 Latent space sampling

As discussed in Section 2.3.1, a VAE forces the posterior distribution of $\boldsymbol{w}$ to be close to a standard normal, promoting latent spaces with concentrated support in $\boldsymbol{\mathcal{W}}$. However, in practice, the process $\{\boldsymbol{W}_{\boldsymbol{v}}\}_{\boldsymbol{v}\in\boldsymbol{\mathcal{V}}}$ is composed by the mixture of multiple uncorrelated multivariate Gaussian densities (one for each training example) and might contain regions with probability much lower than the standard normal. As we will discuss in Section 4, this may lead the decoder to generate incorrect inverse predictions when $\boldsymbol{w}$ comes from low-density posterior regions. We refer to these spurious predictions as *outliers* and dedicate the next sections to compare approaches to effectively sample from the latent space posterior.

3.3.1 Direct sampling from the prior

A straightforward approach to generate samples from the entire $\boldsymbol{\mathcal{W}}$ is to sample from the standard normal prior. This approach performs well for simple inverse problems, as shown in a few numerical experiments of Section 4. Nevertheless, as the complexity of the inverse problem increases, this straightforward approach tends to produce sub-optimal results. To address this limitation, we introduce alternative strategies in the following sections, which have proven to outperform the basic sampling method.

3.3.2 Predictor-Corrector (PC) sampling

An alternative approach to reduce the number of outliers is the *predictor-corrector* method (see Algorithm 4).

Algorithm 4 Predictor-corrector (PC) sampling.

1: For a given $\boldsymbol{y}^*$, sample $\boldsymbol{w}_{[0]}$ from $\mathcal{N}(\mathbf{0},\mathbf{I})$, or use a given set by other sampling method
2: Concatenate and decode to produce $\widehat{\boldsymbol{v}}_{[0]} = NN_d(\boldsymbol{w}_{[0]}, \boldsymbol{y}^*)$ ▷ Prediction step
3: **for** $r = 1, 2, \cdots, R$ **do** ▷ Correction loop
4: VAE encode: $[\boldsymbol{\mu}_{[r]}, \boldsymbol{\sigma}_{[r]}] = NN_v(\widehat{\boldsymbol{v}}_{[r-1]})$
5: Assign mean: $\boldsymbol{w}_{[r]} = \boldsymbol{\mu}_{[r]}$ ▷ Denoted as the operator: $NN_{v,\mu}(\cdot)$
6: Concatenate and decode: $\widehat{\boldsymbol{v}}_{[r]} = NN_d(\boldsymbol{w}_{[r]}, \boldsymbol{y}^*)$
7: **end for**
8: Output $\widehat{\boldsymbol{v}}_{[R]}$

It consists of an iterative approach expressed as

$$\boldsymbol{\beta}_{[r]} + \widehat{\boldsymbol{v}}_{[r]} = NN_d(\boldsymbol{w}_{[r]}, \boldsymbol{y}^*) = NN_d(NN_{v,\mu}(\widehat{\boldsymbol{v}}_{[r-1]}), \boldsymbol{y}^*),\ r = 1, \ldots, R\,,$$

where $\boldsymbol{\beta}_{[r]}$ is the prediction error.

Now assuming the term H is sufficiently small in (21) as a result of gradient-based optimization, then $\widehat{\boldsymbol{v}}_{[r]} \approx \widehat{\boldsymbol{v}}_{[r-1]}$, at least $\forall\, \widehat{\boldsymbol{v}}_{[r]}$ in the training set as r increases. Therefore, the operator $NN_d\big(NN_{v,\mu}(\cdot)\big)$ behaves as a *contraction*. In other words, a deterministic inVAErt network acts as a denoising autoencoder, but it learns to reduce the effects of latent space noise rather than noise in the input space.

In addition, the presence of potential overlap among latent density components $\boldsymbol{W}_{\boldsymbol{v}_i, i=1:N}$ can lead our iterative correction process to prioritize distributions with smaller support, and therefore to promote contractions towards a small subset of the entire training set. A latent space sample $\boldsymbol{w}_i$ belonging to the support of multiple density components from $\boldsymbol{W}_{\boldsymbol{v}_i, i=1:N}$ will tend to move towards the mean of the component with the highest density at $\boldsymbol{w}$, since such samples have been observed more often during training. This poses limitation on the flexibility of PC sampling to explore the global structure of the non-identifiable manifold (see, e.g., Figure 10d in Section 4.2). Thus, in practical applications, the trade-off between the number of outliers and maintaining an interpretable manifold structure can be achieved by varying the number R of iterations.

3.3.3 High-Density (HD) sampling

A second sampling method (see Algorithm 5) leverages the training data to draw more relevant latent samples, then ranks the samples based on the corresponding posterior density.

Algorithm 5 High-Density (HD) sampling.

Take a random subset of the training data input $\mathbf{V}^S \subset \mathbf{V} = \{\boldsymbol{v}_i\}_{i=1}^N$ of size S
for $\boldsymbol{v}_i$ in $\mathbf{V}^S$ **do**
    VAE encode: $[\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i] = NN_v(\boldsymbol{v}_i)$
    for $j = 1 : Q$ **do** ▷ Q: sub-sampling size
        Sample latent variable from the data-dependent distribution: $\boldsymbol{w}_{ij} \sim \mathcal{N}\big(\boldsymbol{\mu}_i, \mathbf{diag}(\boldsymbol{\sigma}_i^2)\big)$
        Record $[\boldsymbol{w}_{ij}, \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i]$
    end for
end for
Initialize the PDF matrix as zeros: $\boldsymbol{p} \in \mathbb{R}^{S\times Q}$ ▷ Since $\boldsymbol{w} \in \mathbb{R}^{S\times Q}$
for $i = 1 : S$ **do**
    Evaluate PDF and consider the overlapping: $\boldsymbol{p} \mathrel{+}= \mathcal{N}\big(\boldsymbol{w}; \boldsymbol{\mu}_i, \mathbf{diag}(\boldsymbol{\sigma}_i^2)\big)$
end for
Rank all $S \cdot Q$ samples based on their stacked PDF values and take samples from the top

It is important to note that the selection of the subset size S and the sub-sampling size Q require careful consideration to prevent the loss of important latent space features. Similar to PC sampling, HD sampling also tends to contract samples towards the means of highly-concentrated latent distributions. In practice, relative to the size of the entire dataset, we typically keep both S and Q small to attain more informative samples.

3.3.4 Sampling with normalizing flow

An additional sampling method considered in this paper uses normalizing flows to map the VAE latent space to a standard normal (we refer this approach as *NF sampling*). Similar approaches have been suggested in [7, 39], for improving VAE and manifold learning.

The learned Real-NVP based NF model, denoted as $NN_{f,\boldsymbol{w}}$, is conditioned on the learned posterior distribution $q(\boldsymbol{w}|\boldsymbol{v})$, which is a mixture of multivariate Gaussian densities. We show the benefit of NF sampling by an exaggerated case in Figure 4, where the transformation $NN_{f,\boldsymbol{w}}$ can effectively reduce the probability of selecting samples with high prior but low posterior density (see the blue dots in Figure 4).

Indeed, the NF sampling produces similar results to HD sampling (see Algorithm 5), has minimal reliance on hyperparameters (as opposed to S and Q in HD sampling), and improves with an increased amount of training data. In practice, like Algorithm 5, we first feed the trained VAE encoder NN_v with the input set $\mathbf{V}^S$ to obtain samples of $\boldsymbol{w}$ which correspond to inputs in the training set, then estimate their density with the new NF model $NN_{f,\boldsymbol{w}}$.

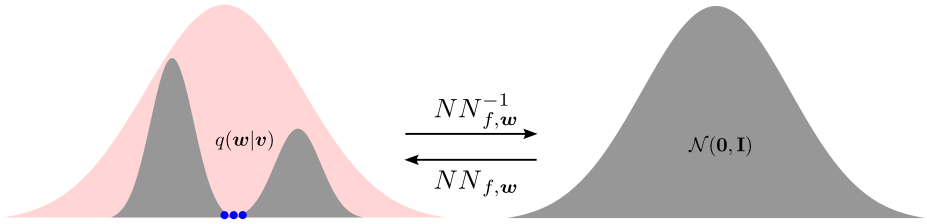


Figure 4: Diagram of the normalizing flow model built on posterior distribution $q(\boldsymbol{w}|\boldsymbol{v})$ of the latent variable $\boldsymbol{w}$.

4 Experiments

4.1 Underdetermined linear system

As a first example, we consider an under-determined linear system from $\mathbb{R}^3 \to \mathbb{R}^2$ defined as

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \pi & e & 0 \\ 0 & e & \pi \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \mathbf{F}\,\boldsymbol{v}. \tag{43}$$

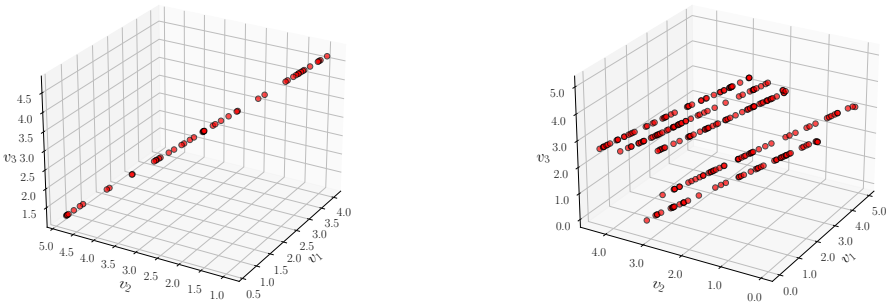
The linear map represented by the matrix $\mathbf{F}$ is surjective (on-to) but non-injective (one-to-one) such that we have a non-trivial one-dimensional kernel of the form

$$\text{Ker}(\mathbf{F}) \subset \mathbb{R} \approx c^*[0.5475278, -0.6327928, 0.5475278]^T, \tag{44}$$

where $c^* \in \mathbb{R}$ is an arbitrary constant and the kernel direction is normalized. As a result, any fixed $\boldsymbol{v}$ in $\mathbb{R}^3$ translating in the direction of $\text{Ker}(\mathbf{F})$ will leave the output $\boldsymbol{y}$ invariant. In another words, this system is non-identifiable along $\text{Ker}(\mathbf{F})$. This line represents the manifold $\boldsymbol{\mathcal{M}}_{\boldsymbol{y}}$ of non-identifiable parameters for this linear map $\mathbf{F}$, as discussed above.

Due to the simplicity of this example, we omit showing the performance of the emulator NN_e and density estimator NN_f, and focus on the VAE and decoder components NN_v and NN_d. We generate the dataset by letting: $\boldsymbol{v} \sim [\mathcal{U}(0,5)]^3$ and forward the exact model 10^4 times to gather the ground truth. In addition, a recommended set of hyperparameters for this example is reported in the Appendix.

First, we select a fixed $\boldsymbol{y}^*$ and reconstruct inputs in $\mathbb{R}^3$ by sampling a scalar w from the latent space. Due to a non-trivial null space, if $\boldsymbol{y}^* = \mathbf{F}\boldsymbol{v}^*$, any input of the form $\boldsymbol{v}^* + \text{Ker}(\mathbf{F})$ will map to the same $\boldsymbol{y}^*$. Therefore the reconstructed inputs are expected to lay on a straight line in $\mathbb{R}^3$ aligned with the direction of $\text{Ker}(\mathbf{F})$ (44), passing through $\boldsymbol{v}^*$. This behaviour is correctly reproduced as shown in Figure 5, where we either fix $\boldsymbol{y}^*$ at 1 value or 5 different values, sampled from NN_f. For each fixed $\boldsymbol{y}^*$, we draw 50 samples of the latent variable w from $\mathcal{N}(0,1)$ to formulate $\tilde{\boldsymbol{y}}^*$ and then pass it through the trained decoder (NN_d), resulting a series of points in $\boldsymbol{\mathcal{V}}$. A simple linear



(a) Decoded samples $\widehat{\boldsymbol{v}}$ for a fixed output $\boldsymbol{y}^*$. (b) Decoded samples $\widehat{\boldsymbol{v}}$ when considering 5 different outputs $\boldsymbol{y}^*$.

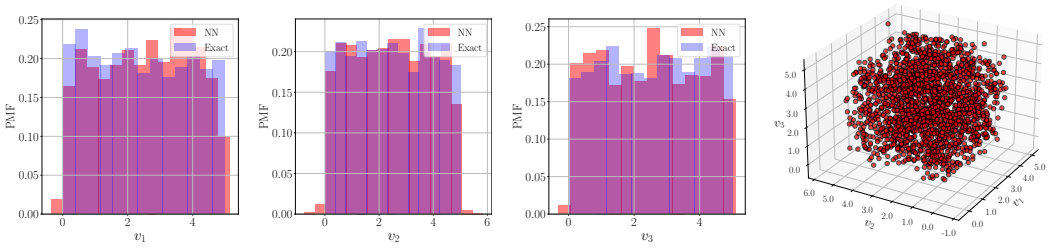
Figure 5: Model inversion results for the underdetermined linear system. We fix one or multiple outputs $\boldsymbol{y}^*$ and concatenate it with scalar samples w from the one-dimensional latent space.

regression through these locations provides a normalized direction vector (averaged of the 5 sets shown in Figure 5) equal to

$$[-0.549, 0.632, -0.547]^T$$

which is close to $\text{Ker}(\mathbf{F})$ (44).

Next we jointly sample from w and $\boldsymbol{y}$. This should pose no restrictions to the ability to recover all possible realizations in the input space $\boldsymbol{\mathcal{V}}$, since $\text{Ker}(\mathbf{F})$ as well as its orthogonal complement can be both reached. To test this, we draw 2000 random samples of $\boldsymbol{y}$ from the trained NN_f and w from $\mathcal{N}(0,1)$. We then use the trained decoder NN_d to determine 2000 values of the corresponding $\widehat{\boldsymbol{v}}$. From Figure 6, it can be observed that most of the predicted samples reside uniformly in the $[0,5]^3$ cube, despite the existence of a few outliers.



(a) Histogram of v_1. (b) Histogram of v_2. (c) Histogram of v_3. (d) Predicted samples of $\widehat{\boldsymbol{v}}$.

Figure 6: Model inversion results for the underdetermined linear system, when jointly sampling form w and $\boldsymbol{y}$. The initial uniform distribution of $\boldsymbol{v}$ is correctly recovered.

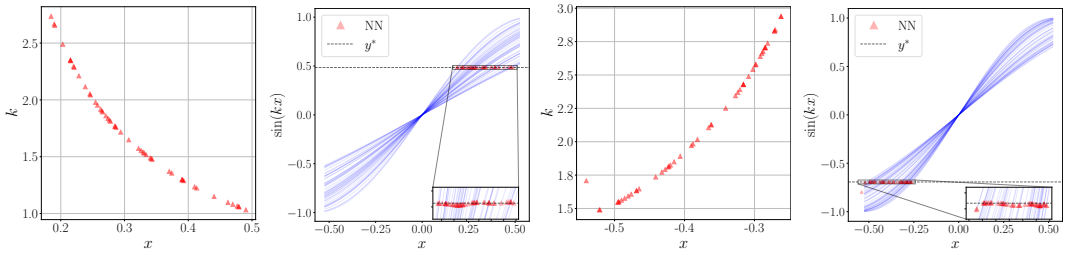
4.2 Nonlinear periodic map

We move to a nonlinear map $\mathcal{F} : \mathbb{R}^2 \to \mathbb{R}$:

$$y = \sin kx. \tag{45}$$

Initially, we consider $k \in [1, 3]$ and $x \in [-\frac{\pi}{6}, \frac{\pi}{6}]$ to avoid a periodic response. In this case $[k, x]^T = \boldsymbol{v} \in \boldsymbol{\mathcal{V}}$, and the forward map $y = \mathcal{F}(\boldsymbol{v})$ is not identifiable on the subspace $\boldsymbol{\mathcal{M}}_y \subset \boldsymbol{\mathcal{V}}$, where the product kx is constant. We again focus only on the inverse analysis task. The inputs k and x are uniformly sampled from their prior ranges for 10^4 times, and the optimal hyperparameters are reported in the Appendix.

As discussed for the previous example, we freeze $y = y^*$ at both positive and negative values and, for each fixed y^*, we pick 50 samples from the latent variable $w \sim \mathcal{N}(0, 1)$, concatenate and decode with NN_d. This results in the sample trajectories of $\widehat{\boldsymbol{v}}$ shown in Figure 7a and Figure 7c, respectively. These one-dimensional manifolds accurately capture the correct correlation between k and x, leading to the same $y = y^*$ as confirmed in Figure 7b and Figure 7d (the superimposed blue sine curves are based on the predicted $\widehat{k}$ values and a fixed sequence of x values. The red triangle marks the predicted $\widehat{x}$ value that should lead to $\sin \widehat{k}\widehat{x} = \widehat{y} \approx y^*$). We then sample together from



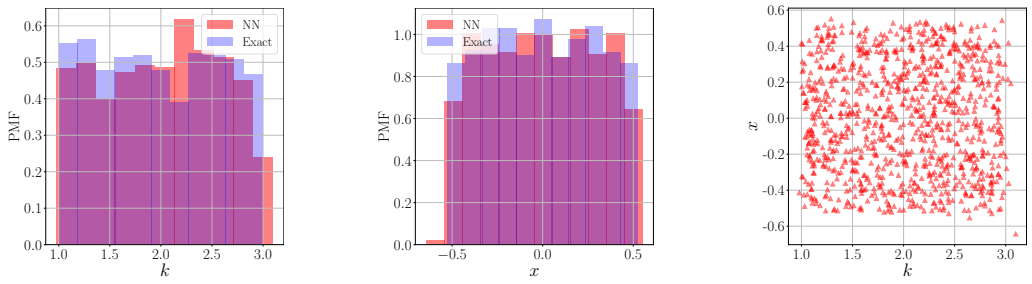
(a) Trajectory of the inversion samples $\widehat{\boldsymbol{v}}$ when fixing a $y^* > 0$. (b) Predicted system output from $\widehat{\boldsymbol{v}}$ when fixing a $y^* > 0$. (c) Trajectory of the inversion samples $\widehat{\boldsymbol{v}}$ when fixing a $y^* < 0$. (d) Predicted system output from $\widehat{\boldsymbol{v}}$ when fixing a $y^* < 0$.

Figure 7: Model inversion results for the sine wave model **without** periodicity. We fix y^* at both positive and negative values, sample w from $\mathcal{N}(0, 1)$, concatenate and decode. The inverse prediction $\widehat{\boldsymbol{v}}$ leads to system output predictions (NN) close to y^*, as expected.

w and y, recovering the initial distributions utilized during training data preparation, for k and x, as shown in Figure 8.

Next, we expand x from $[-\pi/6, \pi/6]$ to $[-\pi, \pi]$ and keep $k \in [1, 3]$. Consequently, the latent manifold is no longer $kx = \text{const}$ in this scenario due to periodicity. To prepare for training, we still generate 10^4 uniform samples from the given ranges while we figure out a more complicated network structure is required for achieving robust performance (please refer to Table 6 for recommended hyperparameters), compared to the above monotonic case.

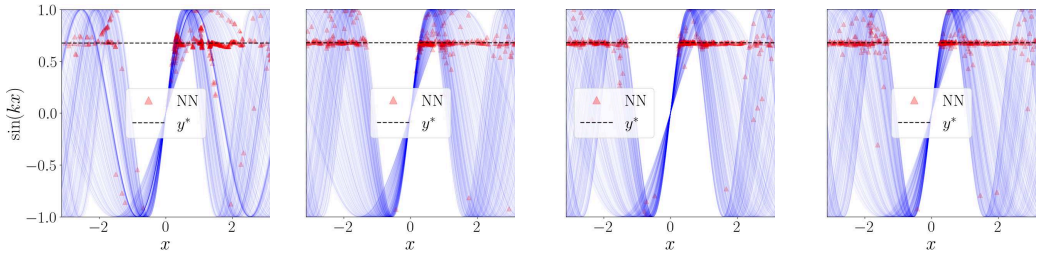
By construction, the latent space $\boldsymbol{\mathcal{W}}$ built by the VAE network should be one-dimensional since $\dim(\boldsymbol{\mathcal{V}}) = 2$ and $y \in \mathbb{R}$. However, for the current periodic system, we find the inverse prediction



(a) Histogram of k. (b) Histogram of x. (c) Predicted samples of $\widehat{\boldsymbol{v}}$.

Figure 8: Model inversion results for the sine wave model **without** periodicity: sample w and y together and the initial uniform distributions of k and x are recovered.

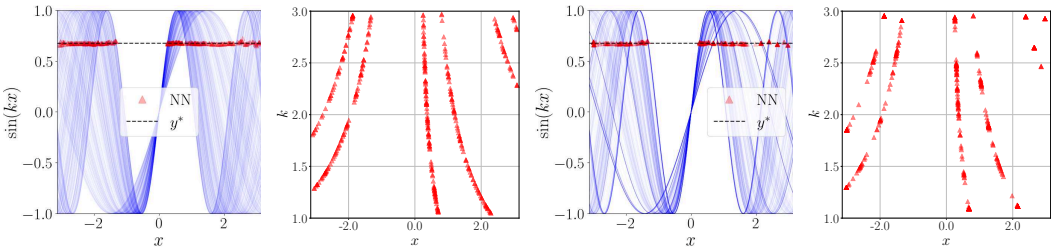
can be significantly improved if $\dim(\boldsymbol{\mathcal{W}})$ is increased beyond 1. We first compare inverse prediction results under $\dim(\boldsymbol{\mathcal{W}}) = 1, 2, 4, 8$ with respect to the same fixed $y^* = 0.676$. In Figure 9, we find the original 1D latent space produces the poorest result, while if $\dim(\boldsymbol{\mathcal{W}})$ increases beyond 1, the performance remains relatively consistent.



(a) $\dim(\boldsymbol{\mathcal{W}}) = 1$. (b) $\dim(\boldsymbol{\mathcal{W}}) = 2$. (c) $\dim(\boldsymbol{\mathcal{W}}) = 4$. (d) $\dim(\boldsymbol{\mathcal{W}}) = 8$.

Figure 9: Model inversion results for the sine wave model **with** periodicity. We fix $y^* = 0.676$ and draw 400 latent variable samples from the standard Gaussian spaces of dimension $1, 2, 4, 8$. Each blue curve is based on a predicted $\widehat{k}$ value and a fixed sequence of x. The red triangle marks the predicted $\widehat{x}$ value that should lead to $y^* \approx \sin \widehat{k}\widehat{x}$.

Next, we demonstrate how the three sampling methods discussed in Section 3.3 enhance our inverse predictions by removing the spurious outliers shown in Figure 9. We focus only on the case with $\dim(\boldsymbol{\mathcal{W}}) = 8$ and keep the sampling size as 400. We hypothesize that a more robust sampling method requires fewer samples to unveil the underlying structure of the non-identifiable manifold $\boldsymbol{\mathcal{M}}_{\boldsymbol{y}}$, though a larger sample size always leads to better visualizations.

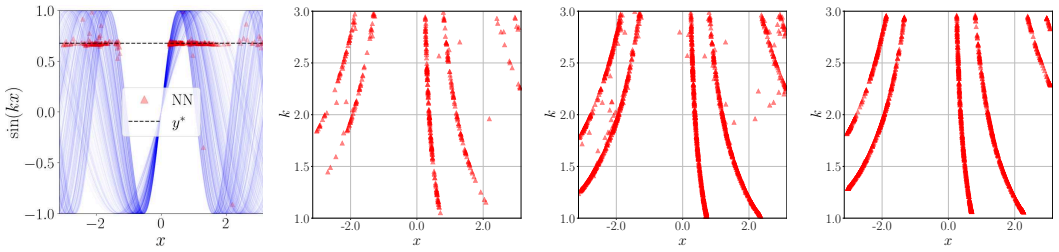


(a) Predicted system output: $R = 2$. (b) Inversion sample trajectory: $R = 2$. (c) Predicted system output: $R = 50$. (d) Inversion sample trajectory: $R = 50$.

Figure 10: Model inversion results for the sine wave model **with** periodicity and using PC sampling. The model output $y^* = 0.676$ is fixed and $\dim(\boldsymbol{\mathcal{W}}) = 8$. The results are shown for iteration number $R = 2$ and $R = 50$. Sample size: 400.

First, compared to Figure 9d, the number of outliers is significantly smaller with PC sampling applied. Furthermore, as the number of iteration R increases, details of the learned manifold gradually fade away, as the decoded samples are drawn towards the means of the posterior components with smaller support, as shown in Figures 10b and 10d. This confirms our analysis in Section 3.3.2.

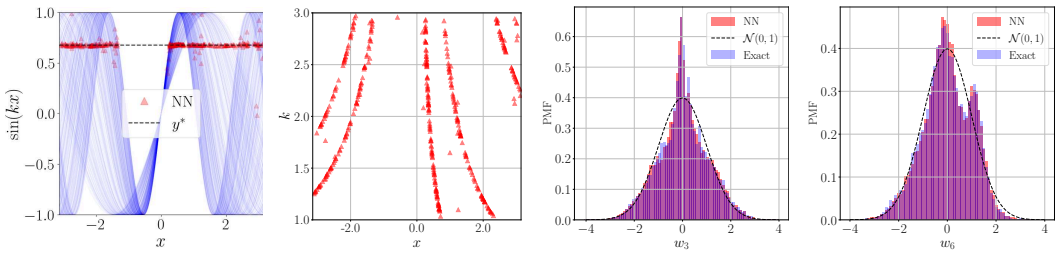
Moving to HD sampling, use of $S = 500, Q = 3$ can approximate the global structure of the non-identifiable manifold with 400 samples, even through several outliers are still visible and some details of the learned trajectories are missing (e.g., compare Figure 11b with Figure 10b). The missing details attribute to the density ranking mechanism of the HD sampling scheme, consistent with our discussion in Section 3.3.3. In this application, the performance of HD sampling can be improved just by utilizing more samples, as illustrated in Figure 11c. Finally, we show that the two sampling schemes (PC and HD) can be combined in Figure 11d.



(a) Predicted system output: $S = 500, Q = 3$. Sample size: 400.
(b) Inversion sample trajectory: $S = 500, Q = 3$. Sample size: 400.
(c) Inversion sample trajectory: $S = 500, Q = 3$. Sample size: 1500.
(d) Apply PC-sampling to denoise the results shown in Figure 11c, $R = 3$.

Figure 11: Model inversion results for the sine wave model **with** periodicity, using HD sampling. The model output $y^* = 0.676$ is fixed and $\dim(\boldsymbol{\mathcal{W}}) = 8$. The results are shown for subset size $S = 500$ and sub-sampling size $Q = 3$.

We then consider NF sampling. To do so, we first train $NN_{f,\boldsymbol{w}}$ based on the latent variable samples generated by the trained VAE encoder NN_v (Please refer to Table 7 for hyperparameter choices). The transformation associated with this flow should be simple if not trivial when the posterior $q(\boldsymbol{w}|\boldsymbol{v})$ is close (in terms of an appropriate statistical distance or divergence) to a standard normal distribution. However, this is not the case as shown in Figures 12c and 12d. The learned latent variable $\boldsymbol{w}$ displays spurious structures in two of its components, which we suspect to be associated with low-density posterior regions that would be incorrectly sampled according to a standard normal (see Section 3.3.4).



(a) Predicted system output.
(b) Inversion sample trajectory.
(c) Learned distribution of w_3 by $NN_{f,\boldsymbol{w}}$.
(d) Learned distribution of w_6 by $NN_{f,\boldsymbol{w}}$.

Figure 12: Model inversion results for the sine wave model **with** periodicity, using NF sampling. The model output $y^* = 0.676$ is fixed and $\dim(\boldsymbol{\mathcal{W}}) = 8$. Sample size: 400. Quantities in the histogram: Exact: data distribution of the latent variable samples generated by the trained VAE encoder NN_v. NN: Learned distribution by the NF sampler $NN_{f,\boldsymbol{w}}$.

In contrast to Figure 9d, we notice a reduction of the outliers in Figure 12a when using NF sampling. The remaining outliers, which may come from insufficient network training, or just due to the fact we are transforming a continuous latent space into disconnected branches, can be effectively removed

if we further apply the PC sampling method to denoise (e.g. see Figures 11c and 11d, results not shown for brevity).

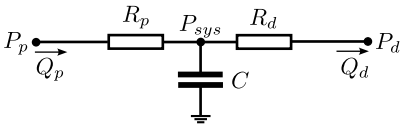
Finally, the comparison of Figures 10b, 11b and 12b indicates that NF sampling can adequately preserve the details of the non-identifiable manifold compared to HD sampling. Also due to their similar nature of drawing high-density posterior samples, we omit presenting results from HD sampling in the next sections.

4.3 Three-element Windkessel model

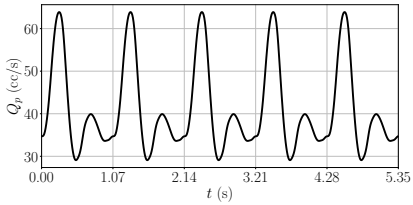
The three-element Windkessel model [48], provides one of the simplest models for the human arterial circulation and can be formulated as a RCR circuit through the hydrodynamic analogy (see Figure 13a). Despite its simplicity, the RCR model has extensive applications in data-driven physiological studies and is widely used to provide boundary conditions for three-dimensional hemodynamic models (see, e.g., [23, 59]). It is formulated through the following coupled algebraic and ordinary differential system of equations

$$\begin{cases} Q_p = \dfrac{P_p - P_{sys}}{R_p}, \\ Q_d = \dfrac{P_{sys} - P_d}{R_d}, \\ \dot{P}_{sys} = \dfrac{Q_p - Q_d}{C}, \end{cases} \tag{46}$$

where C is the overall systemic capacitance which represents vascular compliance. Two resistors, R_p and R_d are used to model the viscous friction in vessels and P_p, P_d and P_{sys} stand for the aortic (proximal) pressure, fixed distal pressure (see Table 1) and the systemic pressure, respectively. In addition, Q_d represents the distal flow rate, whereas the proximal flow rate data $Q_p(t)$ is assigned [23] (see Figure 13b).



(a) Schematic representation of an RCR circuit model.



(b) Proximal flow rate Q_p over 5 cardiac cycles.

Figure 13: Schematic representation of the RCR model and assigned inflow.

This simple circuit model is non-identifiable. The average proximal pressure $\bar{P}_p = (P_{p,\max} + P_{p,\min})/2$ depends only on the total systemic resistance $R_p + R_d$, rather than on each of these individual parameters. Thus, to keep the same $\bar{P}_p$, an increment of the proximal resistance R_p will cause a reduction of the distal resistance R_d, which also allows more flow $(Q_p - Q_d)$ exiting the capacitor, resulting in an increasing capacitance C to balance the system, which affects the *pulse* pressure, i.e., the difference between maximum and minimum proximal pressure. A nonlinear correlation thus exists between the capacitance, proximal and distal resistances when maximum and minimum proximal pressures are provided as data.

Rearranging equations (46) with respect to the variable of interest, $P_p(t)$, the aortic pressure, resulting in the following first-order, linear ODE

$$\begin{cases} \dot{P}_p = R_p \dot{Q}_p + \dfrac{Q_p}{C} - \dfrac{P_p - Q_p R_p - P_d}{C R_d}, \\ P_p(0) = 0. \end{cases} \tag{47}$$

When tuning boundary conditions in numerical hemodynamics, the diastolic and systolic pressures $P_{p,\min}$ and $P_{p,\max}$ are usually available. As a result, we consider the following map:

$$[P_{p,\max}, P_{p,\min}]^T = \mathcal{F}(\boldsymbol{v}), \tag{48}$$

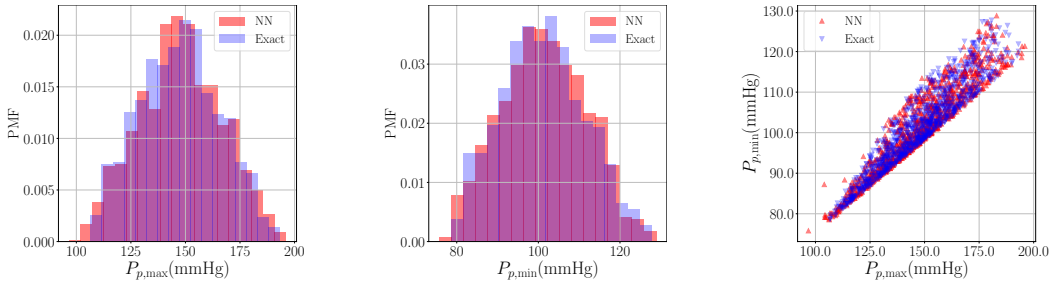
Cardiac cycle (t_c)	1.07 (s)
Distal pressure (P_d)	55 (mmHg)
Proximal resistance (R_p)	$[500, 1500]$ (Barye ·s/ml)
Distal resistance (R_d)	$[500, 1500]$ (Barye ·s/ml)
Capacitance (C)	$[1 \times 10^{-5}, 1 \times 10^{-4}]$ (ml/Barye)

Table 1: RCR model parameters and ranges.

where $\boldsymbol{v} = [R_p, R_d, C]^T$ and by construction, the latent space is assumed to be one-dimensional, i.e. $\dim(\boldsymbol{\mathcal{W}}) = 1$, enough for achieving robust inverse predictions.

To generate training data, we take uniform random samples of R_p, R_d and C from the ranges listed in Table 1 and solve the ODE 10^4 times using the fourth order Runge-Kutta time integrator (RK4). To achieve stable periodic solutions, we choose a time step size equal to $\Delta t = 0.01$ s and simulate the system up to 10 cardiac cycles (10.7 s), where $P_{p,\text{max}}$ and $P_{p,\text{min}}$ are extracted from the last three heart cycles. We then train the inVAErt network with the hyperparameter combination listed in the Appendix.

First, we discuss the learned distributions of the system outputs $P_{p,\text{max}}$ and $P_{p,\text{min}}$. The density estimator correctly learns the parameter correlations and ranges, as shown in Figure 14.



(a) Histogram of $P_{p,\text{max}}$. (b) Histogram of $P_{p,\text{min}}$. (c) $P_{p,\text{max}}$-$P_{p,\text{min}}$ correlation.

Figure 14: Distributions of parametric RCR model outputs as learned by NN_f.

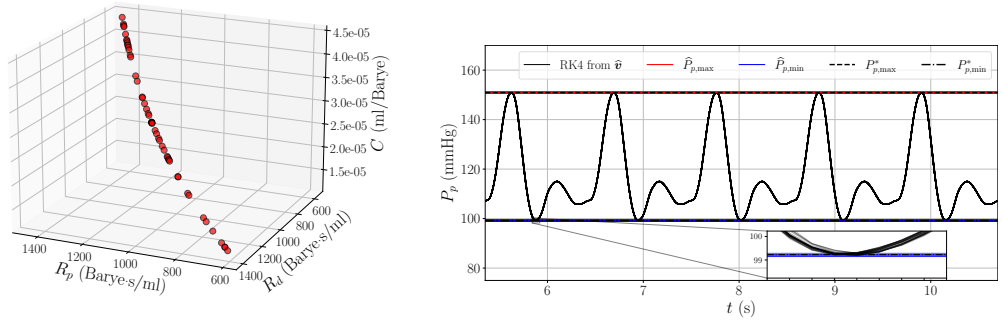
Next, by inverse analysis, we would like to determine all combinations of R_p, R_d and C which correspond to given systolic and diastolic distal pressures. To do so, we feed the trained decoder NN_d with a valid $[P^*_{p,\text{max}}, P^*_{p,\text{min}}]$ sampled from the trained NN_f and 50 w drawn from $\mathcal{N}(0, 1)$, resulting in the trajectory displayed in Figure 15a. Additionally, we plot the binary correlations of these predicted samples in Figure 16.

To confirm that the RCR parameters computed by the decoder correspond to the expected maximum and minimum pressures, we integrate in time the 50 resulting parameter combinations with RK4. The resulting periodic curves are displayed in Figure 15b. They are found to almost perfectly overlap with one another, all oscillating between the correct systolic and diastolic pressures. In addition, we can evaluate the performance of the trained emulator by examining the predicted $\widehat{P}_{p,\text{max}}$ and $\widehat{P}_{p,\text{min}}$ values and comparing them with those obtained from the exact RK4 integration in time. These predictions provide insight into the quality of the forward model evaluations.

Note that the process of obtaining correlated parameters from observations would normally require multiple optimization tasks, since each of these tasks would converge to a single parameter combination. Using the inVAErt network, this process is almost instantaneous and multiple parameter combinations are generated at the same time, providing a superior characterization of all possible solutions for this ill-posed inverse problem. Note also that any form of regularization would have only provided a partial characterization of the right answer, since there is no one right answer.

4.4 Lorenz oscillator

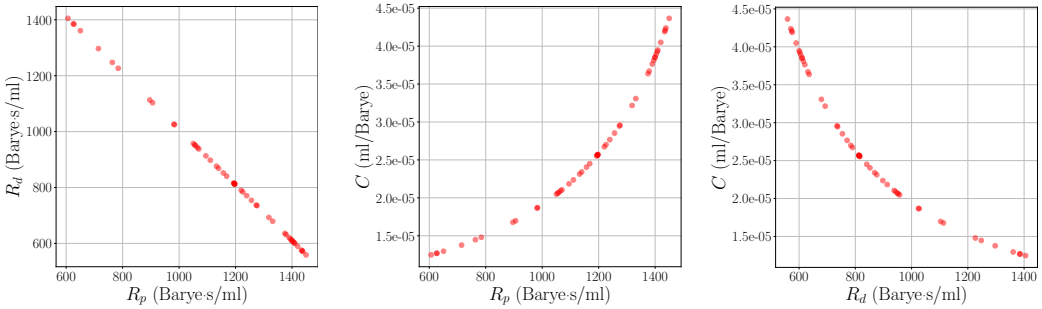
The Lorenz system describes the dynamics of atmospheric convection, where $x(t)$ relates to the rate of convection and $y(t)$, $z(t)$ models the horizontal and vertical temperature variations, respec-



(a) Predicted samples of $\widehat{\boldsymbol{v}}$.

(b) Predicted system output from $\widehat{\boldsymbol{v}}$.

Figure 15: Model inversion results for the parametric RCR system, setting $[P^*_{p,\max}, P^*_{p,\min}]$ and sampling from the latent space $\mathcal{W}$. The decoded parameters $\widehat{\boldsymbol{v}}$ leads to system output predictions (RK4) close to $[P^*_{p,\max}, P^*_{p,\min}]$. The trained forward model NN_e also provides accurate predictions $[\widehat{P}_{p,\max}, \widehat{P}_{p,\min}]$.



(a) R_p-R_d correlation.

(b) R_p-C correlation.

(c) R_d-C correlation.

Figure 16: Projected sample trajectories from Figure 15a when fixing $[P^*_{p,\max}, P^*_{p,\min}]$ and sample from $\mathcal{W}$.

tively [51]. It is formulated as a system of ordinary differential equations of the form

$$\begin{cases} \dot{x} = Pr(y - x), \\ \dot{y} = x(Ra - z) - y, \\ \dot{z} = xy - bz\ . \end{cases} \tag{49}$$

The parameters Pr, Ra are proportional to the Prandtl number and the Rayleigh number, respectively and b is a dependent geometric factor.

The forward map of this nonlinear ODE system is defined via ResNet-based emulator as:

$$\boldsymbol{y}(t) = NN_e(\boldsymbol{v}, \mathcal{D}_{\boldsymbol{v}}) + \boldsymbol{y}(t - \Delta t), \tag{50}$$

where $\boldsymbol{v} = [Pr, Ra, b, t]^T$ and $\boldsymbol{y}(t) = [x(t), y(t), z(t)]^T$, resulting in a one dimensional latent space $\boldsymbol{\mathcal{W}}$. The auxiliary dataset $\mathcal{D}_{\boldsymbol{v}}$, as described above, contains the time delayed solutions

$$\mathcal{D}_{\boldsymbol{v}} = \{\boldsymbol{y}(t - n_p\Delta t), \cdots, \boldsymbol{y}(t - 2\Delta t), \boldsymbol{y}(t - \Delta t)\}\ .$$

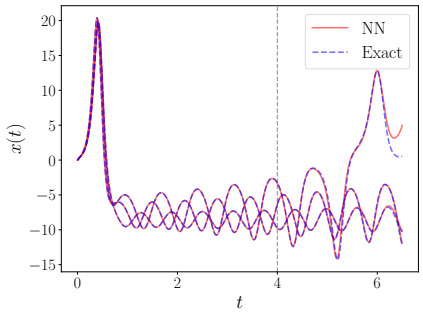
A training dataset is obtained by randomly sample the parameter vector $\boldsymbol{v}$ from the specified ranges listed in Table 2. Unlike previous examples, it is worth noting that we now consider the simulation ending time t as an additional random parameter, drawn from a discrete uniform distribution with an interval of $\Delta t = 5 \times 10^{-4}$ s. This interval aligns with the time step size used in the RK4 time integration. We solve the Lorenz system using 5000 sets of inputs $\boldsymbol{v}$ and take 30 time points at random from each simulation, leading to 1.5×10^5 total training samples. A suggested hyperparameter

Initial conditions (x_0, y_0, z_0)	0,1,0
Largest possible simulation time (T_f)	4 (s)
Prandtl number (Pr)	[8, 12]
Rayleigh number (Ra)	[26, 30]
Geometric factor (b)	[8/3-1, 8/3+1]

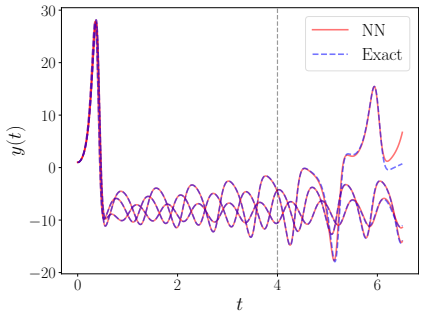
Table 2: Parameters of the Lorenz system.

choice is listed in the Appendix and additionally, in our experiments, we have observed that increasing the number of lagged steps n_p up to 10 leads to noticeable benefits for the emulator learning.

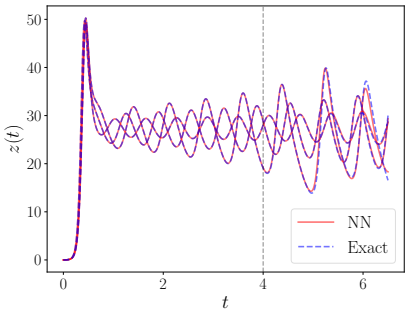
We first focus on the accuracy of the trained emulator. We pick three sets of random samples of Pr, Ra and b unseen during training and forward the trained encoder model NN_e up to 6.5 seconds. Note that the model only requires the first n_p exact steps as inputs, after which, the emulator outputs from previous time steps are used as inputs for successive steps (since the emulator, in this example, is designed to learn the *flow map*). For the prior ranges of Pr, Ra, b listed in Table 2, and up to 4 seconds simulation time, almost all solution trajectories converge towards one of the attractors with regular oscillatory patterns, although varying in frequency, magnitude and speed of convergence. Nevertheless, due to a relatively large Ra, the system has the potential to bifurcate towards another attractor much earlier in time, even though this event is rare within the selected prior ranges. Thus, as shown in Figure 17, an insufficient amount of training samples may lead to a sub-optimal performance in predicting rare dynamical responses.



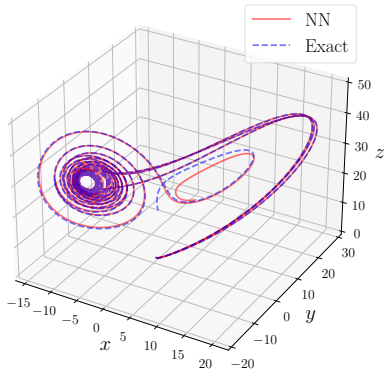
(a) Three examples of learned dynamical response for $x(t)$, superimposed to the exact Lorenz trajectory.



(b) Three examples of learned dynamical response for $y(t)$, superimposed to the exact Lorenz trajectory.



(c) Three examples of learned dynamical response for $z(t)$, superimposed to the exact Lorenz trajectory.

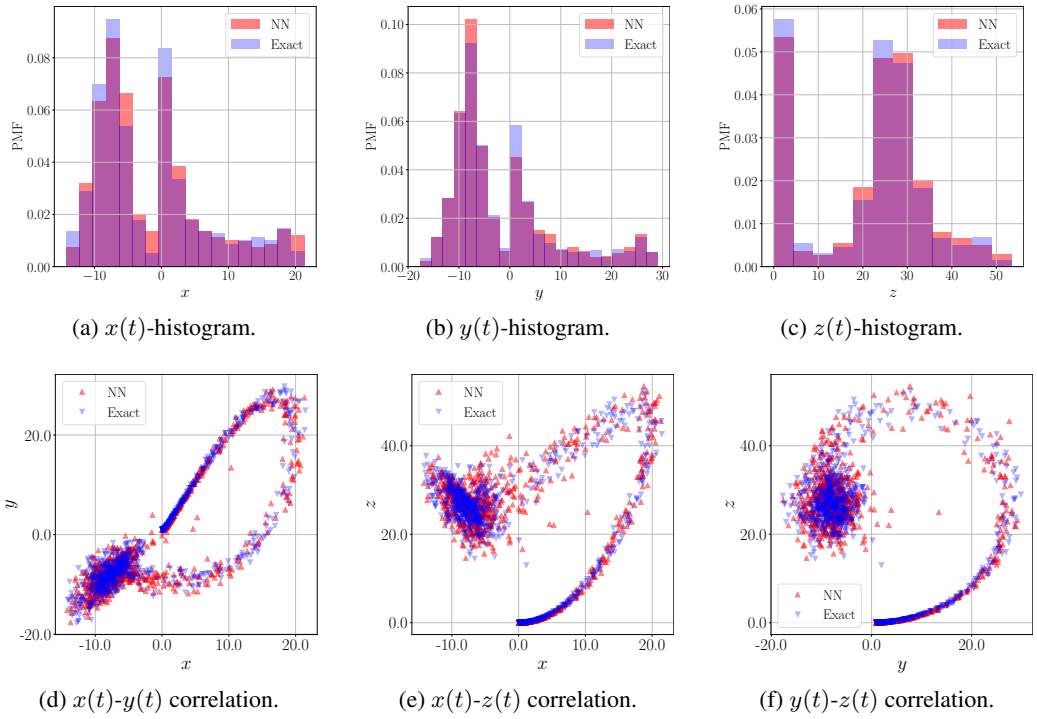


(d) Phase plots for the learned dynamical response and exact Lorenz trajectory.

Figure 17: Comparison between the dynamic response learned by the emulator NN_e and the exact solution computed with RK4 for the Lorenz system up to 6.5 seconds.

Next we find that the Real-NVP sampler NN_f effectively identifies whether a spatial location resides within the high-density regions, such as those around one of the attractors or in proximity to

the initial condition, as shown in Figure 18. However its accuracy is reduced at the tails, which corresponds to rare system trajectories.



(a) $x(t)$-histogram. (b) $y(t)$-histogram. (c) $z(t)$-histogram.

(d) $x(t)$-$y(t)$ correlation. (e) $x(t)$-$z(t)$ correlation. (f) $y(t)$-$z(t)$ correlation.

Figure 18: Density estimation of parametric Lorenz system outputs.

We then investigate the ability of the inVAErt network to solve inverse problems for the given parametric Lorenz system. From our definition of inputs and outputs in (50), we provide a predetermined system output $\boldsymbol{y}^*$, obtained by sampling from the trained Real-NVP model, and ask the inVAErt network to provide all parameters Pr, Ra, b and time t where the resulting solution trajectory should reach $\boldsymbol{y}^*$ at time t.

To do so, we fix $\boldsymbol{y}^* = [-3.4723, -8.9758, 26.2026]^T$, initially focusing on a state that the parametric Lorenz system can reach at an early time. We then draw 100 standard Gaussian samples for $\boldsymbol{w}$ and apply NN_d to the concatenated array $\widetilde{\boldsymbol{y}}^* = [\boldsymbol{w}, \boldsymbol{y}^*]$. To verify the accuracy, we forward the RK4 solver with each inverted sample $\widehat{\boldsymbol{v}} = [\widehat{Pr}, \widehat{Ra}, \widehat{b}, \widehat{t}]$ and terminate the simulation exactly at $t = \widehat{t}$.

Our definition of the forward model (50) implies an one-dimensional latent space $\boldsymbol{\mathcal{W}}$, but in practice, improved results can be obtained by increasing the latent space dimensionality. In this context, we investigated latent spaces $\boldsymbol{\mathcal{W}}$ with dimensions 2, 4, 6, and 8 and observed relatively consistent performances across these dimensions. Nevertheless, it is crucial to recognize that utilizing a lower-dimensional latent space can occasionally result in the loss of certain features when constructing non-identifiable input parameter manifolds.

For brevity, we only plot the best results in Figure 19 and Figure 20, obtained using a six-dimensional latent space. To evaluate how close the decoded parameters led to trajectories ending at the desired coordinates, we introduce a relative error measure ζ defined as

$$\zeta = \frac{\|\boldsymbol{y}^* - \widehat{\boldsymbol{y}}^*\|_2}{\|\boldsymbol{y}^*\|_2}, \tag{51}$$

where $\widehat{\boldsymbol{y}}^*$ is the RK4 numerical solution based on the inverse prediction $\widehat{\boldsymbol{v}}$. A histogram of ζ generated from all 100 inverse predictions is presented in Figure 20b, which reveals that almost all of these predictions yield a relative error less than 2%.

We also plot the learned correlations between the Prandtl number Pr and the other three input parameters in Figures 20c to 20e. The correlation plots presented here depict the projections from

the learned 4D non-identifiable manifold $\mathcal{M}_{\boldsymbol{y}^*} \subset \mathcal{V}$, associated with our fixed $\boldsymbol{y}^*$, onto 2D planes. From these plots, we notice a positive correlation between Pr-Ra and Pr-b, and negative for Pr-t.

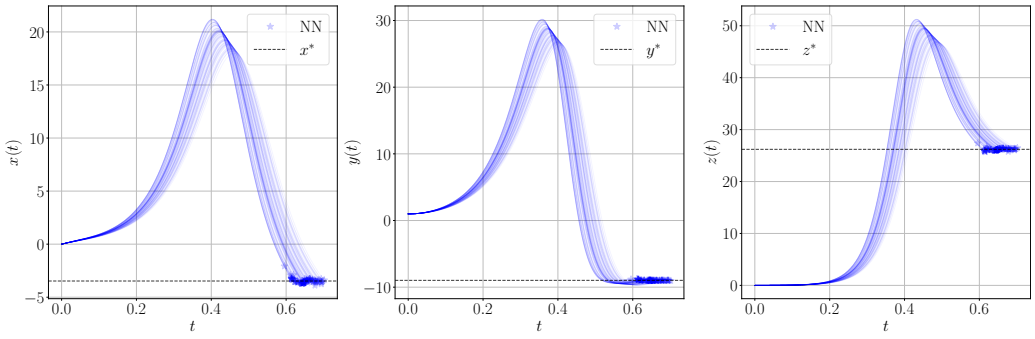
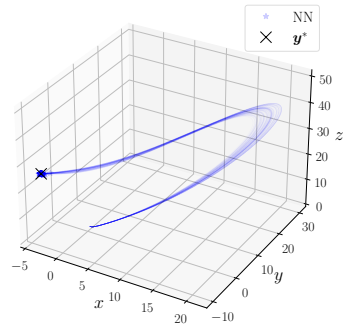
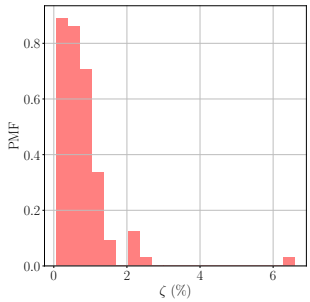


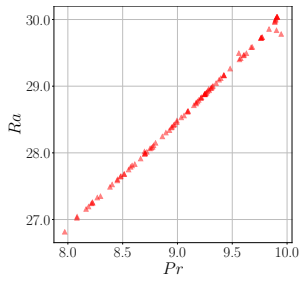
Figure 19: Model inversion results for the parametric Lorenz system. We fix $\boldsymbol{y}^* = [-3.4723, -8.9758, 26.2026]^T$ and sample $\boldsymbol{w}$ from a 6-dimensional standard Gaussian. Each RK4 solution trajectory generated from the decoded inputs $\widehat{\boldsymbol{v}}$ is plotted (NN) and a marker is added to the point computed at $t = \widehat{t}$.



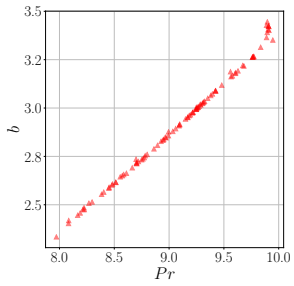
(a) Phase plot trajectories generated from $\widehat{\boldsymbol{v}}$.



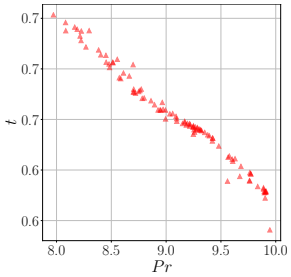
(b) Histogram of ζ.



(c) Pr-Ra correlation.



(d) Pr-b correlation.



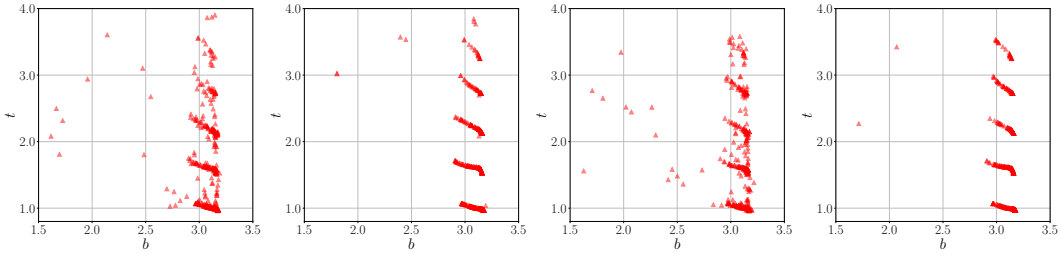
(e) Pr-t correlation.

Figure 20: Model inversion results for the parametric Lorenz system: fix $\boldsymbol{y}^* = [-3.4723, -8.9758, 26.2026]^T$ and sample $\boldsymbol{w}$ from a 6-dimensional standard Gaussian distribution. Phase plots, error measure and learned correlations between Pr and the other three parameters.

For the current fixed output $\boldsymbol{y}^*$, direct sampling of the latent variable $\boldsymbol{w}$ from a standard Gaussian performs relatively well. Therefore, we omit utilizing other sampling approaches mentioned in Section 3.3 to refine the inverse prediction. However, sampling from a standard normal behaves poorly if we significantly increase the complexity of the inverse problem by selecting $\boldsymbol{y}^* = [-11.5224, -10.3361, 30.7660]^T$, a state that can be revisited by a single system multiple times.

Like the periodic wave example studied in Section 4.2, the non-identifiable manifold induced by the new fixed $\boldsymbol{y}^*$ has disconnected components and contains outliers. To illustrate this, we first show

the learned correlations between the geometric factor b and time t in Figure 21, where the latent variables are generated using the sampling methods discussed in Section 3.3.



(a) $\mathcal{N}(\mathbf{0}, \mathbf{I})$ sampling. (b) PC sampling. (c) NF sampling. (d) NF+PC.

Figure 21: Model inversion results for the parametric Lorenz system: fix $\boldsymbol{y}^* = [-11.5224, -10.3361, 30.7660]^T$. Comparing b-t correlations using different latent variable sampling schemes. Sample size: 400, 6D latent space. Methods: sampling from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, PC sampling ($R = 4$), NF sampling (see Table 10), combined NF+PC sampling (apply PC sampling to inverse predictions $\widehat{\boldsymbol{v}}$ associated with Figure 21c, $R = 4$).

The results in Figure 21 suggest that the values of (b, t) for which the Lorenz systems pass through the selected spatial location consist of five disjoint regions. This is likely due to the presence of five instances in time where the given $\boldsymbol{y}^*$ can be reached within 4 seconds under a specific parameter combination, i.e. $[Pr, Ra, b]$. Again, as shown in Section 4.2, it is challenging for our inVAErt network to deform a continuous Gaussian latent space to disconnected sets that represent the non-identifiable manifold. Consequently, despite the concentration of samples around the correct locations, the classical VAE sampling scheme is still contaminated by numerous outliers (see Figure 21a). These outliers can be effectively reduced using PC sampling, or by combining PC and NF sampling (see Figure 21d). Besides, it is not surprising that NF sampling remains susceptible to outliers (see Figure 21c), as we observed the posterior $q(\boldsymbol{w}|\boldsymbol{v})$ in this case is almost the standard normal distribution. In other words, the outliers may not come from the previously mentioned low posterior density regions.

Next, we verify our inverse prediction by numerically integrating the Lorenz system using the samples of $\widehat{\boldsymbol{v}}$ associated with Figure 21d, obtaining a relative error ζ smaller than 5% in most cases (see Figure 22b). Finally, we would like to point out that we can also use the emulator NN_e learnt by the inVAErt network to verify whether a prediction $\widehat{\boldsymbol{v}}$ suggested by the decoder results in an acceptable error ζ, and deploy a rejection sampling approach to complement those discussed in Section 3.3. To verify the applicability of this approach, we forward NN_e with all 400 samples of $\widehat{\boldsymbol{v}}$ produced by the decoder NN_d and plot the endpoints, i.e. $t = \widehat{t}$, for each solution component in Figure 22. These predictions from the emulator, denoted as $\widehat{\boldsymbol{y}}$, match well with the RK4 solutions, and thus may replace $\widehat{\boldsymbol{y}}^*$ in equation (51), providing accurate approximations for ζ.

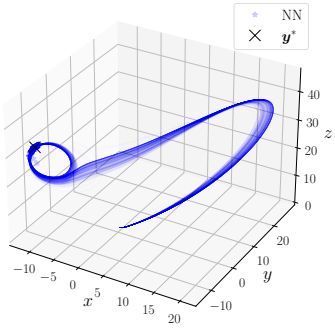
4.5 Reaction-diffusion PDE system

Finally, we consider applications to space- and time-dependent PDEs. To prepare the dataset, we utilize the reaction-diffusion solver from the scientific machine learning benchmark repository PDEBench [55]. The reaction-diffusion equations (52) model the evolution in space and time of two chemical compounds $\boldsymbol{c}(\boldsymbol{x}, t) = [c_1(\boldsymbol{x}, t), c_2(\boldsymbol{x}, t)]^T$ in the domain $\Omega = [-1, 1]^2$,

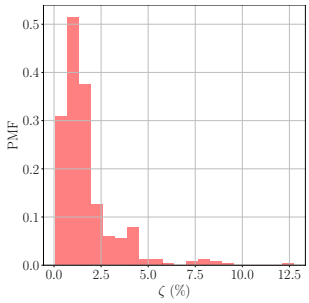
$$\begin{cases} \partial \boldsymbol{c}/\partial t = \mathbf{D}\Delta \boldsymbol{c} + \mathbf{R}(\boldsymbol{c}) & \text{in } \Omega \times (0, T], \\ \partial \boldsymbol{c}/\partial \boldsymbol{x} = \mathbf{0} & \text{on } \partial\Omega \times (0, T], \\ \boldsymbol{c}(\boldsymbol{x}, 0) = \boldsymbol{c}_0(\boldsymbol{x}) & \text{in } \Omega \text{ , at } t = 0, \end{cases} \quad \text{where } \mathbf{D} = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}, \tag{52}$$

where the nonlinear reaction function $\mathbf{R}(\boldsymbol{c})$ follows the Fitzhugh-Nagumo model [55] with reaction constant κ, and is expressed as

$$\mathbf{R}(\boldsymbol{c}) = \begin{bmatrix} c_1 - c_1^3 - \kappa - c_2 \\ c_1 - c_2 \end{bmatrix}. \tag{53}$$



(a) Phase plot trajectories generated from $\widehat{\boldsymbol{v}}$.



(b) Histogram of ζ.

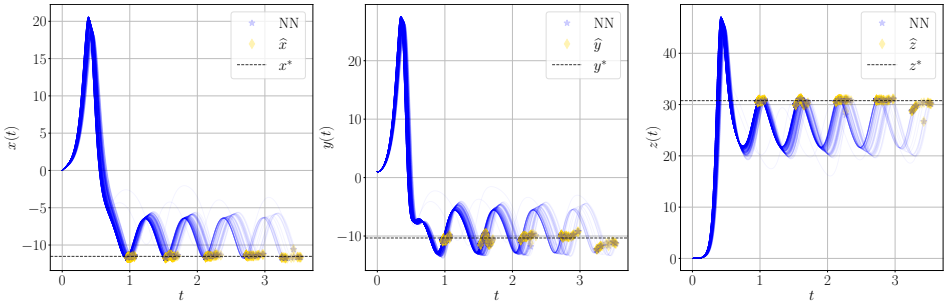


Figure 22: Model inversion results for the parametric Lorenz system: fix $\boldsymbol{y}^* = [-11.5224, -10.3361, 30.7660]^T$. NF sampling (see Table 10) is applied to generate 400 samples of $\boldsymbol{w}$, followed by PC sampling ($R = 4$) for denoising. Results corresponding to inversion samples of $\widehat{\boldsymbol{v}}$ associated with Figure 21d. Each RK4 solution trajectory from $\widehat{\boldsymbol{v}}$ is plotted (NN) and the end point is highlighted that corresponds to $t = \widehat{t}$. Endpoint predictions from the trained emulator NN_e is also plotted, denoted as $\widehat{\boldsymbol{y}} = [\widehat{x}, \widehat{y}, \widehat{z}]^T$.

This system describes the pattern formation phenomenon in biology and c_1, c_2 are usually referred as the enzyme activator and inhibitor, respectively [55]. PDEBENCH utilizes first order finite volumes (FV) in space and a 4-th order Runge-Kutta integrator in time to solve the above reaction-diffusion system, and we set $\boldsymbol{c}_0(\boldsymbol{x}) \sim \mathcal{N}([2, 2]^T, \mathbf{I})$, $\forall \boldsymbol{x} \in \Omega$ for initializations. Additional details on the simulations considered in this section are reported in Table 3.

Spatial discretization	32×32
Time step size (Δt)	0.005
Largest possible simulation time (T_f)	5
Diffusivity for c_1 (D_1)	$[2 \times 10^{-3}, 5 \times 10^{-3}]$
Diffusivity for c_2 (D_2)	$[2 \times 10^{-3}, 5 \times 10^{-3}]$
Reaction constant (κ)	$[2 \times 10^{-3}, 5 \times 10^{-3}]$

Table 3: Simulation parameters of 2D reaction-diffusion system.

The first-order finite volume method assumes constant solution within each grid cell, with space location identified through cell-center coordinates $\boldsymbol{x} = (x, y)$. We then consider a forward process modeled via a ResNet-based emulator as:

$$[c_1(x, y, t), c_2(x, y, t)]^T = NN_e(\boldsymbol{v}, \mathcal{D}_{\boldsymbol{v}}) + [c_1(x, y, t - \Delta t), c_2(x, y, t - \Delta t)]^T , \tag{54}$$

with dependent parameters $\boldsymbol{v} = [D_1, D_2, \kappa, t, x, y]^T$, and auxiliary data defined as

$$\mathcal{D}_{\boldsymbol{v}} = \{\boldsymbol{c}(x \pm \Delta x, y \pm \Delta y, t - \Delta t), \ldots, \boldsymbol{c}(x, y, t - n_p \Delta t), \ldots, \boldsymbol{c}(x, y, t - \Delta t)\},$$

containing solutions at a given cell "S" (see diagram in Figure 23) from the previous n_p time steps and the solutions at its neighbors from the last step. Eight neighbors are considered for all cells

by assuming the solution outside Ω is obtained by mirroring the solution within the domain (see Figure 23). To gather training data, we simulate the system using PDEBENCH [55] for 5000 times with uniform random samples of D_1, D_2 and κ. From each simulation, we randomly pick 10 cells at 5 different time instances to effectively manage the amount of training samples (2.5×10^5 data points in total) and to mitigate over-fitting.

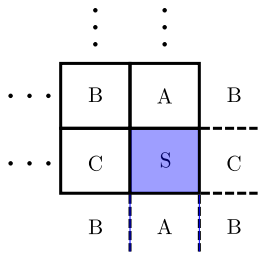
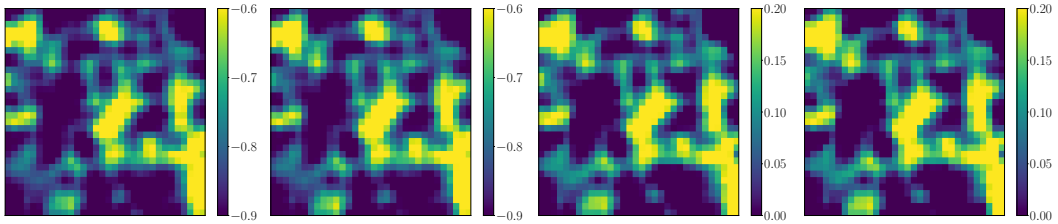


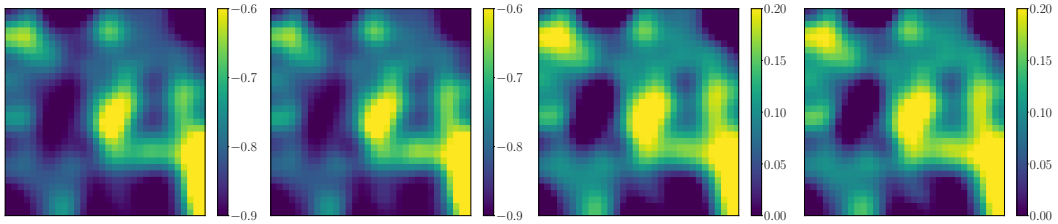
Figure 23: Symmetry condition used to gather auxiliary data $\mathcal{D}_{\boldsymbol{v}}$ at the boundary. Cell "S" here is at the bottom right corner of a two-dimensional discretized domain.

The emulator NN_e is trained by a residual network as discussed in Section 2.1, equation (3). We apply logarithmic transformations to the inputs D_1, D_2, κ since their magnitudes are much smaller compared to x, y, t, c_1, c_2. For a detailed list of hyperparameters, the interested reader is referred to the Appendix.

We first show the performance of the emulator NN_e. To do so, we pick two sets of parameters D_1, D_2, κ that correspond to a low-diffusive and a high-diffusive regime and plot the contours at $t = 2.0$ seconds in Figure 24. We also show the evolution of spatial-averaged, relative l^2-error of these two systems in Figure 25, up to 5 seconds.



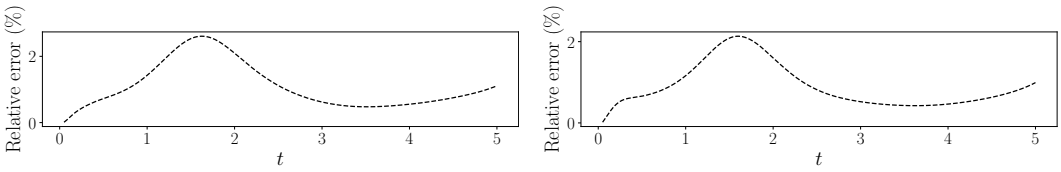
(a) Exact: c_1. (b) NN: c_1. (c) Exact: c_2. (d) NN: c_2.



(e) Exact: c_1. (f) NN: c_1. (g) Exact: c_2. (h) NN: c_2.

Figure 24: Comparison between the learned emulator dynamics (NN) at $T = 2.0$ seconds and reference numerical solutions (from the PDEBENCH package [55]) for the parametric reaction-diffusion system. Top: Low diffusion regime ($D_1 = D_2 = 2 \times 10^{-3}, \kappa = 3 \times 10^{-3}$). Bottom: High diffusion regime ($D_1 = D_2 = 5 \times 10^{-3}, \kappa = 3 \times 10^{-3}$). The coordinate of the left-bottom corner: (-1,-1).

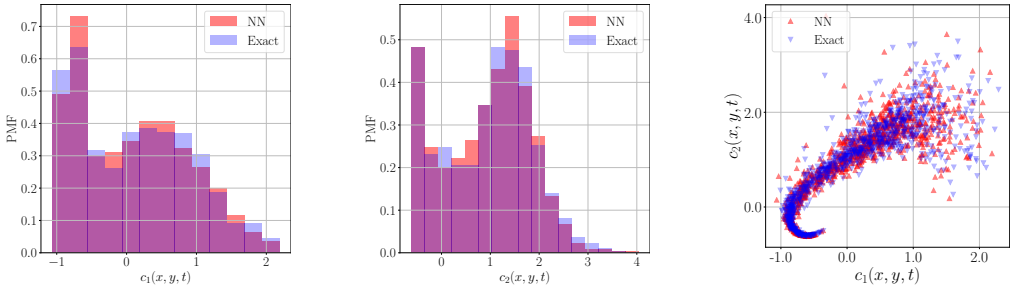
Next, Figure 26 illustrates the ability of the Real-NVP flow NN_f to learn the joint distribution of two system outputs $c_1(x, y, t)$ and $c_2(x, y, t)$. High-density areas are well captured by NN_f whereas some approximation is introduced for rare states in the tails, as shown in Figure 26c. During our experiments, we notice a strong correlation between those high-density regions and the close-to-steady-state behavior of the reaction-diffusion system. This brings our first inversion task: Find



(a) Error evolution of the low-diffusion system. (b) Error evolution of the high-diffusion system.

Figure 25: Spatial-averaged, relative l^2 prediction error of the two systems shown in Figure 24, up to $T = 5.0$ seconds.

all combinations of parameters D_1, D_2, κ, spatial locations x, y and time t where the parametric reaction-diffusion system reaches the selected state $\boldsymbol{c}^* = [c_1^*, c_2^*]^T$.



(a) $c_1(x, y, t)$-histogram. (b) $c_2(x, y, t)$-histogram. (c) c_1-c_2 correlation.

Figure 26: Generative model evaluation of the reaction-diffusion system outputs.

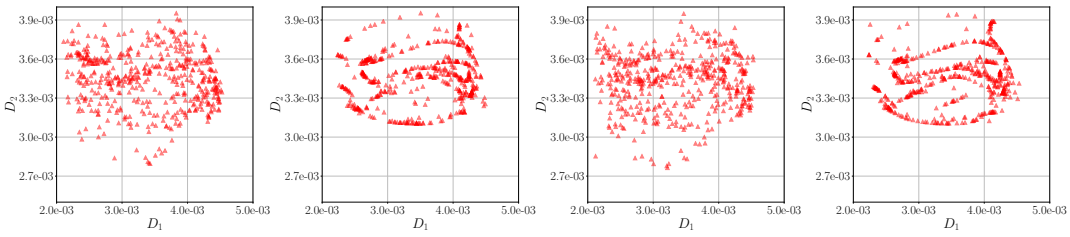
We select the state $\boldsymbol{c}^* = [-0.6616, -0.5964]^T$ sampled from the trained NN_f, which belongs to the high-density regions as illustrated in Figure 26c. To enrich the latent space representation, we set $\dim(\boldsymbol{\mathcal{W}}) = 8$, providing 4 additional dimensions compared to the difference between input and output dimensionality.

First, we utilize various sampling schemes discussed in Section 3.3 to test the trained decoder NN_d. For conciseness, we only plot the learned correlations between the two diffusivity coefficients D_1, D_2 and the spatial coordinates x, y in Figure 27. Like the Lorenz system, we found the posterior distribution $q(\boldsymbol{w}|\boldsymbol{v})$ closely resembles a standard Gaussian density, which explains the resulting similarity of $\mathcal{N}(\mathbf{0}, \mathbf{I})$ sampling and NF sampling. The PC sampling method, either applied to the $\mathcal{N}(\mathbf{0}, \mathbf{I})$ samples or those drawn from the trained $NN_{f,\boldsymbol{w}}$, exhibits promising potential in revealing unrecognizable latent structures polluted by spurious outliers (e.g. compare Figure 27a to Figure 27b).

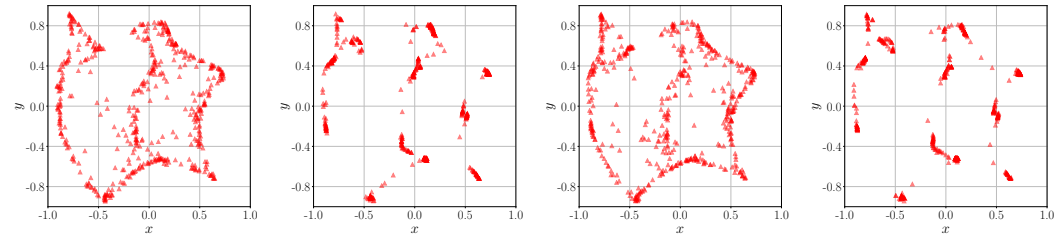
Next, in Figure 28, we verify that our parametric reaction-diffusion system is indeed non-identifiable under these inverse predictions. To realize this, we forward the FV-RK4 solver of PDEBENCH [55] with all 500 $\widehat{\boldsymbol{v}}$'s produced by the trained decoder NN_d, plus the PC sampling approach (i.e. inversion samples associated with Figures 27b and 27f). The simulation uses $\widehat{D}_1, \widehat{D}_2, \widehat{\kappa} \in \widehat{\boldsymbol{v}}$ as inputs and finishes at $t = \widehat{t} \in \widehat{\boldsymbol{v}}$. From Figures 28b and 28c, we see all solution trajectories of c_1, c_2, originated from various $(\widehat{x}, \widehat{y}) \in \widehat{\boldsymbol{v}}$, converge towards our prescribed values with a maximum relative error around 3% (i.e. see Figure 28a).

However, the accuracy of the proposed approach deteriorates when inverting an output $\boldsymbol{c}^*$ that is rarely observed during training. This is due to an insufficient characterization of the fiber $\boldsymbol{\mathcal{M}}_{\boldsymbol{c}^*}$ over $\boldsymbol{c}^*$ since the close-to-steady states outnumber the other possible states in the training dataset (e.g. Figure 26c). Besides increasing the training set, one could also use the exact forward model or the proposed emulator NN_e to reject decoded inputs, as discussed in Section 4.4.

In the end, we would like to leverage the inVAErt network to quickly answer relevant question on the inverse dynamics of the parametric reaction-diffusion system (52). Specifically, we would like to identify the locations in space where the chemical compounds c_1, c_2 may fall within some prescribed

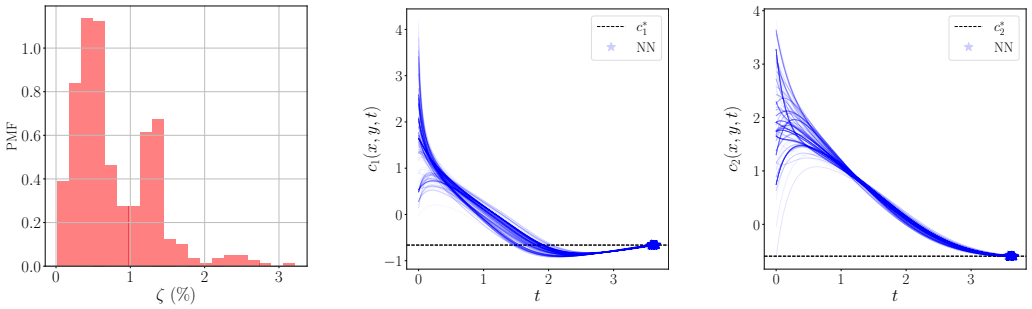


(a) $\mathcal{N}(\mathbf{0}, \mathbf{I})$ sampling. (b) PC sampling. (c) NF sampling. (d) NF+PC.



(e) $\mathcal{N}(\mathbf{0}, \mathbf{I})$ sampling. (f) PC sampling. (g) NF sampling. (h) NF+PC.

Figure 27: Model inversion results for the parametric reaction-diffusion system: fix $\boldsymbol{c}^* = [-0.6616, -0.5964]^T$. Comparing D_1-D_2, x-y correlations using different latent variable sampling schemes. Sample size: 500, 8D latent space. Methods: sampling from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, PC sampling ($R = 6$), NF sampling (see Table 12), combined NF+PC sampling (apply PC sampling to inverse predictions $\widehat{\boldsymbol{v}}$ associated with Figures 27c and 27g with $R = 6$).



(a) Histogram of ζ. (b) Verification of c_1^*. (c) Verification of c_2^*.

Figure 28: Model inversion results for the parametric reaction-diffusion system: fix $\boldsymbol{c}^* = [-0.6616, -0.5964]^T$. PC sampling ($R = 6$) is applied to generate 500 samples of $\boldsymbol{w}$. Results corresponding to inversion samples of $\widehat{\boldsymbol{v}}$ associated with Figures 27b and 27f. Each FV-RK4 solution trajectory from $\widehat{\boldsymbol{v}}$ is plotted (NN) and a marker added at the time instance when $t = \widehat{t}$.

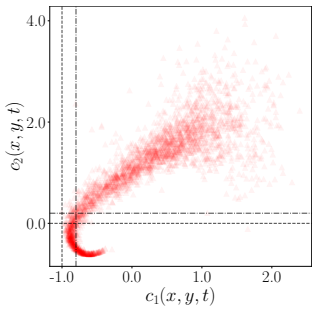
ranges. Answering this type of question represents a challenging inverse problem, but is well within reach for the proposed architecture.

First, suppose the system output state $\boldsymbol{c}^*$ is of particular interests if it belongs to an *active region* around the high-density areas, defined as:

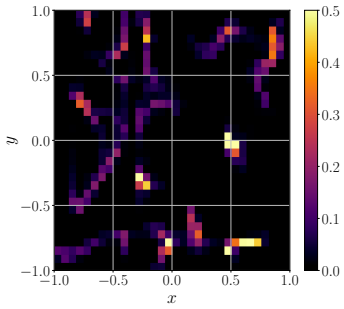
$$\mathcal{A} = \{\boldsymbol{c}^* | -1 \leq c_1^* \leq -0.8, 0.0 \leq c_2^* \leq 0.2\} .$$

To collect states within $\mathcal{A}$, we sample from the trained normalizing flow model NN_f and reject all samples outside of $\mathcal{A}$ (see Figure 29a). Then, using the trained decoder NN_d to invert each of the selected state, we obtain a sequence of inverse predictions $\widehat{\boldsymbol{v}}$. Next, we track each $(\widehat{x}, \widehat{y}) \in \widehat{\boldsymbol{v}}$ and accumulate the occurrence of its corresponding cell in space. This results in Figure 29b, where each cell's occurrence count is normalized by the maximum occurrence across all cells.

Finally, for verification, we take a small subset of all inverse predictions and forward the systems with the FV-RK4 solver [55]. Then, we check the resulting time series solution of c_1 and c_2 at locations ranked 1st, 4th, and 9th with respect to the normalized occurrence count shown in Figure 29b.



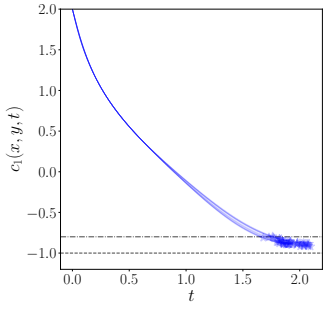
(a) The region $\mathcal{A}$ superimposed to the model outputs sampled from NN_f. We utilize a reduced sample size to enhance the clarity of the visualization.



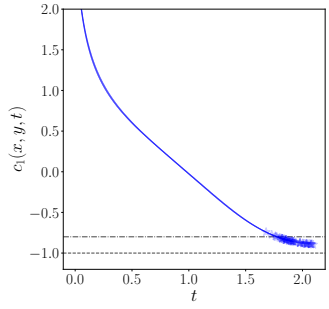
(b) Spatial locations where the output chemical compounds are compatible with the region $\mathcal{A}$. The maximum value is limited to 0.5 (instead of 1.0) for improved visualization.

Figure 29: Model inversion results for the parametric reaction-diffusion system: chemical compounds $\boldsymbol{c}^*$ from a selected region $\mathcal{A}$ (left) and corresponding spatial locations (right). Color scale: normalized occurrence of a spatial cell $(\widehat{x}, \widehat{y})$ in the decoded samples. Results are generated using PC sampling with $R = 6$, an eight-dimensional latent space, 1188/50000 $\boldsymbol{c}^*$ selected from $\mathcal{A}$, and 500 samples of $\boldsymbol{w}$ for each application of the decoder.

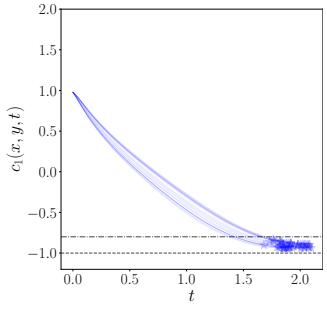
The results are reported in Figure 30, showing that most of the solution trajectories converge inside the region $\mathcal{A}$.



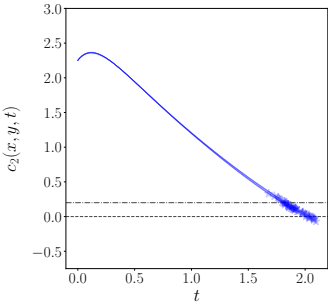
(a) c_1: 1st location.



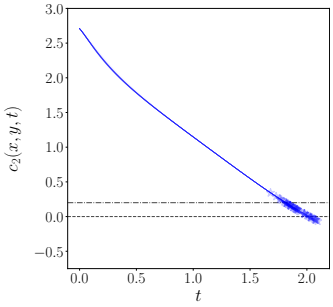
(b) c_1: 4th location.



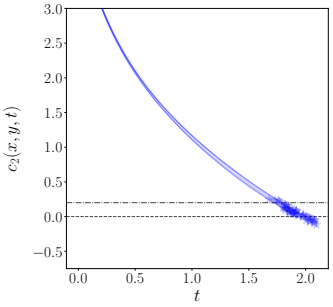
(c) c_1: 9th location.



(d) c_2: 1st location.



(e) c_2: 4th location.



(f) c_2: 9th location.

Figure 30: Model inversion results for the parametric reaction-diffusion system. Verification of three possible locations $(\widehat{x}, \widehat{y})$ associated with the 1st, 4th and 9th relative frequency among all inverse predictions $\widehat{\boldsymbol{v}}$. Each FV-RK4 solution trajectory from a small subset of all $\widehat{\boldsymbol{v}}$ is plotted and the predicted time point $t = \widehat{t}$ is marked.

5 Conclusion

In this paper we introduce inVAErt networks, a comprehensive framework for data-driven analysis and synthesis of parametric physical systems. An inVAErt network is designed to learn many different aspects of a map or differential model including an emulator for the forward deterministic response, a Real-NVP based density estimator for generating representative output samples, a decoder to quickly provide an approximation of the inverse output-to-input map, and a variational encoder which learns a compact latent space responsible for the lack of bijectivity between the input and output spaces.

We describe each component in detail and provide extensive numerical evidence on the performance of inVAErt networks with non-identifiable systems of varying complexity, including linear/nonlinear maps, dynamical systems, and spatio-temporal PDE systems. The current framework is also expected to be extended to more complex inverse problems involving inputs and outputs with significantly higher dimensionality, e.g. image reconstruction problems.

InVAErt networks can accommodate a wide range of data-driven emulators including dense, convolutional, recurrent, graph neural networks, and others. Compared to previous systems designed to simultaneously learn the forward and inverse maps for physical systems, inVAErt networks have the key property of providing a partition of the input space into non-identifiable manifolds and their identifiable complements. In the current study, manifolds in the input space are indicative of structural non-identifiability, as we focus on training with noise-free input/output pairs. In the presence of noise, an inVAErt can be extended to jointly analyze structural and practical non-identifiability.

We find that, in practice, the accuracy in capturing the inverse system response is affected by the selection of appropriate penalty coefficients in the loss function and by the ability to sample from the posterior density of the latent variables.

Once an inVAErt network is optimally trained from instances of the forward input-to-output map, inverse problems can be solved instantly, and multiple solutions can be determined at once, providing a much broader understanding of the physical response than unique solutions obtained by regularization, particularly when regularization is not strongly motivated in light of the underlying physical phenomena.

This work represents a demonstration of the capabilities of inVAErt networks and a first step to improve current understanding on the use of data-driven architectures for the analysis and synthesis of physical systems. A number of extensions will be the objective of future research, from applied studies to combination with PINNs or multifidelity approaches in order to minimize the number of examples needed during training.

Acknowledgements

GGT and DES were supported by a NSF CAREER award #1942662 (PI DES), a NSF CDS&E award #2104831 (University of Notre Dame PI DES) and used computational resources provided through the Center for Research Computing at the University of Notre Dame. CSL was partially supported by an Open Seed Fund between Pontificia Universidad Católica de Chile and the University of Notre Dame, by the grant Fondecyt 1211643, and by Centro Nacional de Inteligencia Artificial CENIA, FB210017, BASAL, ANID. DES would like to thank Marco Radeschi for the interesting discussions. The authors would also like to thank the two anonymous reviewers for their comments and suggestions that greatly contributed to improve the quality of this paper.

References

[1] M. Almaeen, Y. Alanazi, N. Sato, W. Melnitchouk, M. P. Kuchera, and Y. H. Li. Variational Autoencoder Inverse Mapper: An End-to-End Deep Learning Framework for Inverse Problems. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[2] M. Almaeen, Y. Alanazi, N. Sato, W. Melnitchouk, and Y. H. Li. Point Cloud-based Variational Autoencoder Inverse Mappers (PC-VAIM) - An Application on Quantum Chromody-

namics Global Analysis. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1151–1158, 2022.

[3] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. Pellegrini, R. Klessen, L. Maier-Hein, C. Rother, and U. Köthe. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*, 2018.

[4] S. Arridge, P. Maass, O. Öktem, and C. B. Schönlieb. Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174, 2019.

[5] M. Benning and M. Burger. Modern regularization methods for inverse problems. *Acta Numerica*, 27:1–111, 2018.

[6] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[7] J. Brehmer and K. Cranmer. Flows for simultaneous manifold learning and density estimation, 2020.

[8] L. H. Cao, T. O'Leary-Roseberry, P. K. Jha, J. T. Oden, and O. Ghattas. Residual-based error correction for neural operator accelerated infinite-dimensional bayesian inverse problems. *Journal of Computational Physics*, 486:112104, 2023.

[9] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. Variational Lossy Autoencoder, 2017.

[10] E. R. Cobian, J. D. Hauenstein, F. Liu, and D. E. Schiavazzi. Adaann: Adaptive annealing scheduler for probability density approximation. *International Journal for Uncertainty Quantification*, 13, 2023.

[11] S. L. Cotter, M. Dashti, and A. M. Stuart. Approximation of Bayesian inverse problems for PDEs. *SIAM journal on numerical analysis*, 48(1):322–345, 2010.

[12] K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

[13] M. Deistler, P. J. Goncalves, and J. H. Macke. Truncated proposals for scalable and hassle-free simulation-based inference. *Advances in Neural Information Processing Systems*, 35:23135–23149, 2022.

[14] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

[15] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[16] H. Fu, C. Y. Li, X. D. Liu, J. Gao, A. Celikyilmaz, and L. Carin. Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing, 2019.

[17] X. H. Fu, W. Z. Mao, L. B. Chang, and D. B. Xiu. Modeling unknown dynamical systems with hidden parameters. *Journal of Machine Learning for Modeling and Computing*, 3(3), 2022.

[18] N. Geneva and N. Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022.

[19] O. Ghattas and K. Willcox. Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica*, 30:445–554, 2021.

[20] H. Goh, S. Sheriffdeen, J. Wittmer, and T. Bui-Thanh. Solving Bayesian inverse problems via variational autoencoders. *arXiv preprint arXiv:1912.04212*, 2019.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[22] W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[23] K. Harrod, J. Rogers, J. Feinstein, A. Marsden, and D. Schiavazzi. Predictive modeling of secondary pulmonary hypertension in left ventricular diastolic dysfunction. *Frontiers in physiology*, page 654, 2021.

[24] S. Havrylov and I. Titov. Preventing Posterior Collapse with Levenshtein Variational Autoencoder, 2020.

[25] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[26] J. Kaipio and E. Somersalo. *Statistical and computational inverse problems*, volume 160. Springer Science & Business Media, 2006.

[27] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. F. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[28] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] D. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[30] D. Kingma, M. Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[31] A. Kirsch et al. *An introduction to the mathematical theory of inverse problems*, volume 120. Springer, 2011.

[32] B. T. Knapik, A. W. van der Vaart, and J. H. van Zanten. Bayesian inverse problems with Gaussian priors. *The Annals of Statistics*, 39(5), oct 2011.

[33] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.

[34] H. S. Li, J. Schwab, S. Antholzer, and M. Haltmeier. NETT: Solving inverse problems with deep neural networks. *Inverse Problems*, 36(6):065005, 2020.

[35] L. Lu, P. Z. Jin, G. F. Pang, Z. Q. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[36] J. Lucas, G. Tucker, R. B. Grosse, and M. Norouzi. Don't blame the ELBO! a linear vae perspective on posterior collapse. *Advances in Neural Information Processing Systems*, 32, 2019.

[37] Y. Marzouk, T. Moselhy, M. Parno, and A. Spantini. Sampling via measure transport: An introduction. *Handbook of uncertainty quantification*, 1:2, 2016.

[38] B. K. Miller, C. Weniger, and P. Forré. Contrastive neural ratio estimation. *Advances in Neural Information Processing Systems*, 35:3262–3278, 2022.

[39] R. Morrow and W. C. Chiu. Variational Autoencoders with Normalizing Flow Decoders, 2020.

[40] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22(1):2617–2680, 2021.

[41] G. Papamakarios, D. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 837–848. PMLR, 2019.

[42] L. Parussini, D. Venturi, P. Perdikaris, and G. E. Karniadakis. Multi-fidelity Gaussian process regression for prediction of random fields. *Journal of Computational Physics*, 336:36–50, 2017.

[43] T. Qin, K. L. Wu, and D. B. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

[44] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[45] A. Razavi, A. van den Oord, B. Poole, and O. Vinyals. Preventing Posterior Collapse with delta-VAEs, 2019.

[46] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[47] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[48] Y. Shi, P. Lawford, and R. Hose. Review of zero-D and 1-D models of blood flow in the cardiovascular system. *Biomedical engineering online*, 10:1–38, 2011.

[49] R. C. Smith. *Uncertainty quantification: theory, implementation, and applications*, volume 12. Siam, 2013.

[50] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[51] C. Sparrow. *The Lorenz equations: bifurcations, chaos, and strange attractors*, volume 41. Springer Science & Business Media, 2012.

[52] B. Stevens and T. Colonius. FiniteNet: A Fully Convolutional LSTM Network Architecture for Time-Dependent Partial Differential Equations, 2020.

[53] A. M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[54] D. J. Tait and T. Damoulas. Variational autoencoding of PDE inverse problems. *arXiv preprint arXiv:2006.15641*, 2020.

[55] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. PDEBench: An Extensive Benchmark for Scientific Machine Learning. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, 2022.

[56] G. G. Tong and D. E. Schiavazzi. Data-driven synchronization-avoiding algorithms in the explicit distributed structural analysis of soft tissue. *Computational Mechanics*, 71(3):453–479, 2023.

[57] A. Vadeboncoeur, Ö. D. Akyildiz, I. Kazlauskaite, M. Girolami, and F. Cirak. Deep probabilistic models for forward and inverse problems in parametric PDEs. *arXiv preprint arXiv:2208.04856*, 2022.

[58] T. Wang, P. Plechac, and J. Knap. Generative diffusion learning for parametric partial differential equations. *arXiv preprint arXiv:2305.14703*, 2023.

[59] Y. Wang, F. Liu, and D. Schiavazzi. Variational inference with NoFAS: Normalizing flow with adaptive surrogate for computationally expensive models. *Journal of Computational Physics*, 467:111454, 2022.

[60] Y. X. Wang, D. Blei, and J. P. Cunningham. Posterior collapse and latent variable non-identifiability. *Advances in Neural Information Processing Systems*, 34:5443–5455, 2021.

[61] H. Whitney. Differentiable manifolds. *Annals of Mathematics*, pages 645–680, 1936.

[62] L. Yang, X. H. Meng, and G. E. Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[63] L. Yang, D. K. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.

[64] S. J. Zhao, J. M. Song, and S. Ermon. InfoVAE: Information Maximizing Variational Autoencoders, 2018.

[65] W. H. Zhong and H. Meidani. PI-VAE: Physics-informed variational auto-encoder for stochastic differential equations. *Computer Methods in Applied Mechanics and Engineering*, 403:115664, 2023.

A Gâteaux derivative of loss function

In this appendix, we compute Gâteaux derivatives of the cost function T (21) term by term. Recall the expression $o(\alpha)$ denotes any function such that $\lim_{\alpha\to 0}\alpha^{-1}o(\alpha)=0$.

A.1 Derivatives of J

We first compute the partial derivative of J with respect to $\mathscr{D}$:

$$J(\mathscr{D}+\alpha\Delta\mathscr{D},\mathscr{V})=J(\mathscr{D},\mathscr{V})+\alpha\delta J(\mathscr{D},\mathscr{V};\Delta\mathscr{D})+o(\alpha)\,, \tag{55}$$

where

$$\delta J(\mathscr{D},\mathscr{V};\Delta\mathscr{D}):=\lim_{\alpha\to 0}\frac{J(\mathscr{D}+\alpha\Delta\mathscr{D},\mathscr{V})-J(\mathscr{D},\mathscr{V})}{\alpha}\,, \tag{56}$$

denotes the Gâteaux derivative, or *first variation*, of the functional J at $\mathscr{D}$ in the direction of $\Delta\mathscr{D}$ and $\alpha\in\mathbb{R}$. We then merge the expansion (20) with the definition (22), leading to

$$\begin{aligned}
J(\mathscr{D}+\alpha\Delta\mathscr{D},\mathscr{V}) &= \frac{1}{2\cdot N\cdot M}\sum_{i,j=1}^{N,M}\Big(\|\mathcal{F}(\mathscr{D}+\alpha\Delta\mathscr{D})(\boldsymbol{w}_{ij},\boldsymbol{y}_i)-\boldsymbol{y}_i\|_2^2\Big)\,,\\
&= \frac{1}{2\cdot N\cdot M}\sum_{i,j=1}^{N,M}\Big(\|\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)-\boldsymbol{y}_i\|_2^2\\
&\quad+2\alpha\big\langle\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)-\boldsymbol{y}_i,\nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)\Delta\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big\rangle\\
&\quad+\alpha^2\|\nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)\Delta\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\|_2^2\\
&\quad+2\alpha\big\langle\nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)\Delta\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i),o(\alpha)\big\rangle\\
&\quad+2\big\langle\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)-\boldsymbol{y}_i,o(\alpha)\big\rangle+\|o(\alpha)\|_2^2\Big)\,.
\end{aligned} \tag{57}$$

Neglecting the high order terms related to α^2 and $o(\alpha)$ and comparing with (55), we have

$$\begin{aligned}
\delta J(\mathscr{D},\mathscr{V};\Delta\mathscr{D}) &= \frac{1}{N\cdot M}\sum_{i,j=1}^{N,M}\big\langle\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)-\boldsymbol{y}_i,\nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)\Delta\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big\rangle\,,\\
&= \frac{1}{N\cdot M}\sum_{i,j=1}^{N,M}\big\langle\nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)^T\Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big)-\boldsymbol{y}_i\Big),\Delta\mathscr{D}(\boldsymbol{w}_{ij},\boldsymbol{y}_i)\big\rangle\,.
\end{aligned} \tag{58}$$

We then proceed to compute the partial derivative of J with respect to $\mathscr{V}$. This process is equivalent to deriving first-order conditions with respect to $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i$ through the VAE architecture. First,

consider the following expansion of J

$$
\begin{aligned}
J(\mathscr{D}, \mathscr{V} + \alpha\Delta\mathscr{V}) &= \frac{1}{2 \cdot N \cdot M} \sum_{i,j=1}^{N,M} \|\mathcal{F}\big(\mathscr{D}(\boldsymbol{\mu}_i + \alpha\Delta\boldsymbol{\mu}_i + (\boldsymbol{\sigma}_i + \alpha\Delta\boldsymbol{\sigma}_i) \odot \boldsymbol{\epsilon}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\|_2^2 \,, \\
&= J(\mathscr{D}, \mathscr{V}) \\
&+ \frac{\alpha}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i, \nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)\nabla\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\Delta\boldsymbol{\mu}_i \big\rangle \\
&+ \frac{\alpha}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i, \nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)\nabla\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\Delta\boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}_{ij} \big\rangle \\
&+ o(\alpha) \,,
\end{aligned} \tag{59}
$$

where $\nabla\mathscr{D}$ denotes the Jacobian matrix of $\mathscr{D}$ with respect to $\mathscr{V}$. Following similar procedure as (58), the Gâteaux derivative of J is then

$$
\begin{aligned}
\delta J(\mathscr{D}, \mathscr{V}; \Delta\mathscr{V}) &= \frac{1}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \nabla\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\Big), \Delta\boldsymbol{\mu}_i \big\rangle \\
&+ \frac{1}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \boldsymbol{\epsilon}_{ij} \odot \nabla\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \nabla\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big)^T \Big(\mathcal{F}\big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)\big) - \boldsymbol{y}_i\Big), \Delta\boldsymbol{\sigma}_i \big\rangle \,.
\end{aligned} \tag{60}
$$

A.2 Derivatives of H

Following the same procedure as above, we compute Gâteaux derivatives of the functional H with respect to $\mathscr{D}$ and $\mathscr{V}$ as:

$$
\delta H(\mathscr{D}, \mathscr{V}; \Delta\mathscr{D}) = \frac{1}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i, \Delta\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) \big\rangle \,. \tag{61}
$$

$$
\begin{aligned}
\delta H(\mathscr{D}, \mathscr{V}; \Delta\mathscr{V}) &= \frac{1}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \nabla\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big), \Delta\boldsymbol{\mu}_i \big\rangle \\
&+ \frac{1}{N \cdot M} \sum_{i,j=1}^{N,M} \big\langle \boldsymbol{\epsilon}_{ij} \odot \nabla\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i)^T \big(\mathscr{D}(\boldsymbol{w}_{ij}, \boldsymbol{y}_i) - \boldsymbol{v}_i\big), \Delta\boldsymbol{\sigma}_i \big\rangle \,.
\end{aligned} \tag{62}
$$

A.3 Derivatives of K

Since the functional K only depends on parameters of $\mathscr{V}$, it suffices to compute

$$
\begin{aligned}
K(\mathscr{V} + \alpha\Delta\mathscr{V}) &= K(\boldsymbol{\mu}_i + \alpha\Delta\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i + \alpha\Delta\boldsymbol{\sigma}_i) \,, \\
&= K(\mathscr{V}) + \frac{\partial K}{\partial \boldsymbol{\mu}_i}\Delta\boldsymbol{\mu}_i + \frac{\partial K}{\partial \boldsymbol{\sigma}_i}\Delta\boldsymbol{\sigma}_i + o(\alpha) \,, \\
&= K(\mathscr{V}) + \frac{\alpha}{N} \sum_{i=1}^{N} \big(\boldsymbol{\mu}_i\Delta\boldsymbol{\mu}_i + (\boldsymbol{\sigma}_i - \boldsymbol{\sigma}_i^{-1})\Delta\boldsymbol{\sigma}_i\big) + o(\alpha) \,,
\end{aligned} \tag{63}
$$

where $\boldsymbol{\sigma}_i^{-1}$ is to be interpreted componentwise. Hence, the Gâteaux derivative δK can be computed explicitly as:

$$
\delta K(\mathscr{V}; \Delta\mathscr{V}) = \frac{1}{N} \sum_{i=1}^{N} \big(\boldsymbol{\mu}_i\Delta\boldsymbol{\mu}_i + (\boldsymbol{\sigma}_i - \boldsymbol{\sigma}_i^{-1})\Delta\boldsymbol{\sigma}_i\big) \,. \tag{64}
$$

B Related work on posterior collapse

Posterior collapse is often attributed to the variational inference objective, e.g. ELBO (14) where the posterior distribution is drawn towards the prior, leading to the collapse phenomenon. To address this issue, one can either reduce decoder flexibility or decrease the impact of the KL loss [6, 64, 9, 24, 16]. However, Lucas et al. [36] challenge this perspective by comparing the ELBO objective to the MLE approach in linear VAEs. They suggest that the ELBO objective may not be the only reason for posterior collapse. Razavi et al. [45] propose an alternative approach that avoids modifying the ELBO objective. They introduce δ-VAEs, which are specifically designed to constrain the statistical distance between the learned posterior and the prior. Furthermore, Wang et al. [60] have shown that posterior collapses if and only the latent variable is not identifiable in the generative model. To mitigate this issue, they develop LIDVAE (Latent-IDentifiable VAE).

C Hyperparameter selection

This section outlines the optimal set of hyperparameters employed in each experiment. Unless otherwise specified, our training approach consists of mini-batch gradient descent with the Adam optimizer [28]. Besides, our learning rate η is set through the exponential scheduler $\eta(z) = \eta_0 \cdot \gamma^z$. For a given epoch z, the initial learning rate η_0 and the decay rate γ are treated as hyperparameters.

We also found that dataset normalization is particularly important to improve training accuracy. It helps us control the amount of spurious outliers during inference and significantly helps with inputs having different magnitudes. If not stated otherwise, we apply component-wise standardization to both the network inputs and outputs, i.e.

$$\bar{v}_i = \frac{v_i - \mu_{v_i}}{\sigma_{v_i}} \ , \ i = 1 : \dim(\boldsymbol{v}) \ ,$$

where μ_{v_i} and σ_{v_i} stand for the calculated mean and standard deviation of component v_i, from the entire dataset.

The MLP (Multi-Layer Perceptron) serves as the fundamental building block of our neural network. We use the notation: $[a, b, c]$ to represent a MLP, where a, b, c denote the number of neurons per layer, the number of hidden layers and the type of the activation function, respectively. We also highlight the use of Swish activation function `SiLU` [44] in a few numerical examples, as it has been shown to outperform other activation functions in our experiments.

For the Real-NVP sampler NN_f, we use the notation $[a, b, d, e]$ to specify the number of neurons per layer (a), the number of hidden layers (b), and the number of alternative coupling blocks (c). The Boolean variable e indicates whether batch normalization is used during training and our choice of activation functions aligns with the original Real-NVP literature [14].

Note that the three components of our inVAErt network, i.e., emulator NN_e, output density estimator NN_f, inference engine ($NN_v + NN_d$) can be trained independently (see Section 2.4), which enables us to apply distinct hyperparameters for achieving optimal performance. To make this more clear , we use the notation $\{A, B, C\}$ for each of the aforementioned components, respectively. In addition, we also include hyperparameter choices of the additional NF sampler $NN_{f,\boldsymbol{w}}$ (see Section 3.3.4), if it helps a certain numerical experiment by generating informative samples of latent variable $\boldsymbol{w}$.

Finally, although rare as our experiments suggest, it is worth mentioning that the phenomenon of posterior collapse may still occur with these recommended hyperparameters, since the training also implicitly depends on dataset, network initialization etc.

Data scaling	`True`
Initial learning rate η_0	$\{1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-2}\}$
Decay rate γ	$\{0.98, 0.98, 0.999\}$
Mini-Batch size	$\{128, 128, 64\}$
Parameters of NN_e	$[3, 2,$ `identity`$]$
Parameters of NN_f	$[6, 4, 4,$ `False`$]$
Parameters of NN_v	$[8, 4,$ `ReLU`$]$
Parameters of NN_d	$[10, 10,$ `SiLU`$]$
ℓ^2-weight decay rate	$\{0, 0, 1 \times 10^{-3}\}$
Loss penalty λ_v, λ_d	$1, 40$

Table 4: Hyperparameter choices of the simple linear system.

Data scaling	`True`
Initial learning rate η_0	$\{1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-2}\}$
Decay rate γ	$\{0.99, 0.99, 0.998\}$
Mini-Batch size	$\{128, 128, 128\}$
Parameters of NN_e	$[10, 4,$ `ReLU`$]$
Parameters of NN_f	$[10, 4, 4,$ `False`$]$
Parameters of NN_v	$[10, 4,$ `ReLU`$]$
Parameters of NN_d	$[10, 10,$ `SiLU`$]$
ℓ^2-weight decay rate	$\{0, 0, 1 \times 10^{-3}\}$
Loss penalty λ_v, λ_d	$1, 200$

Table 5: Hyperparameter choices of the simple nonlinear system **without** periodicity.

Data scaling	`True`
Initial learning rate η_0	$\{1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-3}\}$
Decay rate γ	$\{0.998, 0.998, 0.999\}$
Mini-Batch size	$\{64, 64, 32\}$
Parameters of NN_e	$[16, 8,$ `SiLU`$]$
Parameters of NN_f	$[10, 4, 4,$ `False`$]$
Parameters of NN_v	$[24, 8,$ `SiLU`$]$
Parameters of NN_d	$[48, 8,$ `SiLU`$]$
ℓ^2-weight decay rate	$\{0, 0, 0\}$
Loss penalty λ_v, λ_d, λ_r	$1, 200, 5$

Table 6: Hyperparameter choices of the simple nonlinear system **with** periodicity.

Data scaling	`True`
Subset size S	10000
Sub-sampling size Q	20
Mini-Batch size	1024
Initial learning rate η_0	5×10^{-3}
Decay rate γ	0.995
Parameters of $NN_{f,\boldsymbol{w}}$	$[10, 4, 6,$ `False`$]$
ℓ^2-weight decay rate	0

Table 7: Hyperparameter choices of the additional NF sampler of the simple nonlinear system **with** periodicity.

Data scaling	`True`
Initial learning rate η_0	$\{1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-2}\}$
Decay rate γ	$\{0.995, 0.995, 0.9992\}$
Mini-Batch size	$\{128, 128, 128\}$
Parameters of NN_e	$[10, 6, \texttt{ReLU}]$
Parameters of NN_f	$[10, 4, 6, \texttt{False}]$
Parameters of NN_v	$[10, 4, \texttt{ReLU}]$
Parameters of NN_d	$[10, 10, \texttt{SiLU}]$
ℓ^2-weight decay rate	$\{0, 0, 1 \times 10^{-3}\}$
Loss penalty λ_v, λ_d	$1, 400$

Table 8: Hyperparameter choices of the RCR system.

Data scaling	`False`
Initial learning rate η_0	$\{1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-3}\}$
Decay rate γ	$\{0.999, 0.998, 0.999\}$
Mini-Batch size	$\{512, 512, 512\}$
Parameters of NN_e	$[64, 15, \texttt{SiLU}]$
Parameters of NN_f	$[12, 4, 8, \texttt{False}]$
Parameters of NN_v	$[24, 8, \texttt{SiLU}]$
Parameters of NN_d	$[80, 15, \texttt{SiLU}]$
ℓ^2-weight decay rate	$\{0, 0, 0\}$
Loss penalty λ_v, λ_d, λ_r	$1, 200, 7$

Table 9: Hyperparameter choices of the Lorenz system.

Data scaling	`False`
Subset size S	30000
Sub-sampling size Q	15
Mini-Batch size	2048
Initial learning rate η_0	5×10^{-3}
Decay rate γ	0.995
Parameters of $NN_{f,\boldsymbol{w}}$	$[8, 4, 6, \texttt{False}]$
ℓ^2-weight decay rate	0

Table 10: Hyperparameter choices of the additional NF sampler of the Lorenz system.

Data scaling	`False`
Initial learning rate η_0	$\{1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-3}\}$
Decay rate γ	$\{0.999, 0.998, 0.998\}$
Mini-Batch size	$\{1024, 1024, 1024\}$
Parameters of NN_e	$[96, 12, \texttt{SiLU}]$
Parameters of NN_f	$[12, 4, 8, \texttt{False}]$
Parameters of NN_v	$[16, 8, \texttt{SiLU}]$
Parameters of NN_d	$[64, 8, \texttt{SiLU}]$
ℓ^2-weight decay rate	$\{0, 0, 0\}$
Loss penalty λ_v, λ_d, λ_r	$1, 200, 5$

Table 11: Hyperparameter choices of the reaction-diffusion system.

Data scaling	`False`
Subset size S	50000
Sub-sampling size Q	10
Mini-Batch size	2048
Initial learning rate η_0	1×10^{-2}
Decay rate γ	0.995
Parameters of $NN_{f,\boldsymbol{w}}$	$[10, 4, 6,$ `False`$]$
ℓ^2-weight decay rate	0

Table 12: Hyperparameter choices of the additional NF sampler of the reaction-diffusion system.