

---

# PI-VAE: PHYSICS-INFORMED VARIATIONAL AUTO-ENCODER FOR STOCHASTIC DIFFERENTIAL EQUATIONS

---

A PREPRINT

**Weiheng Zhong**

Department of Civil and Environmental Engineering

University of Illinois at Urbana-Champaign

Champaign, Illinois

weiheng4@illinois.edu

**Hadi Meidani**

Department of Civil and Environmental Engineering

University of Illinois at Urbana-Champaign

Champaign, Illinois

meidani@illinois.edu

March 23, 2022

## ABSTRACT

We propose a new class of physics-informed neural networks, called physics-informed Variational Autoencoder (PI-VAE), to solve stochastic differential equations (SDEs) or inverse problems involving SDEs. In these problems the governing equations are known but only a limited number of measurements of system parameters are available. PI-VAE consists of a variational autoencoder (VAE), which generates samples of system variables and parameters. This generative model is integrated with the governing equations. In this integration, the derivatives of VAE outputs are readily calculated using automatic differentiation, and used in the physics-based loss term. In this work, the loss function is chosen to be the Maximum Mean Discrepancy (MMD) for improved performance, and neural network parameters are updated iteratively using the stochastic gradient descent algorithm. We first test the proposed method on approximating stochastic processes. Then we study three types of problems related to SDEs: forward and inverse problems together with mixed problems where system parameters and solutions are simultaneously calculated. The satisfactory accuracy and efficiency of the proposed method are numerically demonstrated in comparison with physics-informed generative adversarial network (PI-WGAN).

**Keywords** Physics-informed deep learning, stochastic differential equations, variational autoencoders

## 1 Introduction

Stochastic differential equations (SDEs) are differential equations that are parameterized by random variables and/or constrained by uncertain initial and boundary conditions. SDEs are used in various fields, such as mechanics, economics, biology, and physics [1]. Solving SDEs can be regarded as quantifying the effects that random inputs have on the system response or quantities of interest (QoI). Because of the stochastic nature of these problems, time-efficient numerical methods are needed. The more traditional solution approaches are Monte Carlo Simulation (MCS) and Generalized Polynomial Chaos (gPC) methods. Monte Carlo simulation (MCS) is straightforward and robust, but comes with high computational cost. In MCS, we sample realizations from random inputs, and for each input realization a deterministic differential equations is solved using numerical methods such as finite element, finite difference, and finite volume [2]. The statistics of QoIs can then be calculated using the obtained deterministic solutions. As another solution

approach, gPC methods consider a finite-term polynomial representation for the response, which can be calculated using collocation or Galerkin method [3, 4, 5, 6, 7]. However, gPC methods, especially in collocation-based form, suffer from the "curse of dimensionality".

Recently, models based on neural networks have received increasing attention as an effective approach to solve SDEs. In particular, physics-constrained, or physics-informed neural networks, have been widely studied. The idea of using physics-informed training for neural network solutions of differential equations was first introduced in 1990s, where neural networks solutions for initial/boundary value problems were developed in [8, 9, 10]. But it was only recently that it gained much attention, mainly due to advances in computing hardware and libraries, as an efficient solution for deterministic differential equations [11, 12, 13, 14]. More recently, ideas based on physics-informed neural networks have also been proposed for solving SDEs. In particular, a physics-informed deep residual network was used to solve random PDEs using physics-informed deep residual networks by constraining the neural network to satisfy governing equations in a strong or variational form [15]. A probabilistic formulation in the framework of Generative adversarial networks (GANs) to perform UQ of SDEs was presented in [16]. Also, physics laws have been incorporated into Deep operator networks (DeepONets) to learn the solution operator of parametric differential equations, which can predict the solution of a differential equation for any realization of the uncertain parameters [17]. These methods can effectively solve SDEs when the governing differential equations and its parameters are precisely known.

However, in many practical problems, we do not have access to exact analytical representations for random parameters of SDEs. A more common case is that we know precisely about the governing differential equations, but can only obtain information about the parameters using a limited number of sensors that capture scattered measurements (e.g., limited samples of soil permeability obtained from a subsurface region). To address this, Zhang et al. [18] proposed to solve SDEs by training physics-informed neural networks (PINNs) to learn the modal functions of the arbitrary polynomial chaos (aPC) expansion of the solution of the differential equation. In this paper, all the training data are collected by sparsely distributed fixed sensors. Inspired by [18], a Physics-informed Wasserstein Generative Adversarial Networks (PI-WGANs) was proposed for solving SDEs [19] and SPDEs [20], where sampled data from the parameters were used to train generators in Generative Adversarial Networks (GANs) which in turn produce samples of QoIs, with similar statistics.

Encoding physics laws, in the form of differential equations, into the framework of GANs successfully mitigates the "curse of dimensionality" and provides an accurate estimation. Nevertheless, GANs suffer from training instability because of simultaneous training of generator and discriminator, which sometimes leads to the problem of mode collapse [21]. Besides, as mentioned in [19], the computational cost of training GANs is much higher than training a feed-forward neural network. Our goal is then to seek a more stable and more efficient physics-informed learning method to solve SDEs. Similar to GANs, Variational Autoencoder (VAE) is another data generative model initially proposed in [22], and has proved effective for various tasks of reproducing synthetic data [23, 24, 25]. In this work, we seek to encode physics laws into VAE and propose Physics-Informed VAE (PI-VAE), which is a framework for solving SDEs. PI-VAE is easier for training because VAE has a simple end-to-end structure, similar to feed-forward neural networks.

The architecture of PI-VAE consists of an encoder structure and a physics-informed decoder structure, constructing a bi-directional mapping between latent space and high-dimensional input space. As the loss function, we use the Maximum Mean Discrepancy of the reconstructed data and training data in latent and high-dimensional input spaces over the fixed positions of the entire computational domain. The parameters of the encoder and the decoder are updated by gradient descent steps toward minimizing the loss function, using variants of the mini-batch gradient descent algorithm. Then the solution to the SDE is represented by the well-trained decoder.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce our data-driven problems that are solved in this paper. Section 3 briefly reviews distance measures between probability distributions and their applications in machine learning, and then introduces the main PI-VAE algorithms for approximating random processes and solving SDEs. Finally, a detailed analysis of performance evaluation of the proposed methods and conclusions are included in Sections 4 and 5.

## 2 Technical background

Let us consider the following SDE

$$\begin{aligned}\mathcal{N}_x[u(x, \omega), k(x, \omega)] &= f(x, \omega) \quad x \in \mathbf{D}, \quad \omega \in \Omega, \\ \mathcal{B}_x[u(x, \omega)] &= b(x, \omega), \quad x \in \Gamma,\end{aligned}\tag{1}$$

where  $\mathcal{N}_x$  is a general differential operator,  $\mathbf{D}$  is a  $d$ -dimensional physical domain in  $R^d$ ,  $x$  is a  $d$ -dimensional spatial coordinate,  $\Omega$  is a probability space, and  $\omega$  is a random event, and  $\mathcal{B}_x$  is a boundary condition operator acting on the domain boundary  $\Gamma$ . The coefficient  $k(x, \omega)$ , the forcing term  $f(x, \omega)$  and the boundary conditions  $b(x, \omega)$  can be considered to be random processes, and thus the solution  $u(x, \omega)$  will also be a random process.

Depending of the availability of measurement data from  $k(x, \omega)$ ,  $u(x, \omega)$ ,  $f(x, \omega)$  and  $b(x, \omega)$  we can two different problems, namely a forward problem which concerns the calculation of the solution to the SDE  $u(x, \omega)$  as the QoI given measurements of  $k(x, \omega)$ ,  $f(x, \omega)$  and  $b(x, \omega)$ , or an inverse problem which involves estimating the coefficient  $k(x, \omega)$  as the QoI given available measurements of  $u(x, \omega)$ ,  $f(x, \omega)$  and  $b(x, \omega)$ . Also, one can be dealing with ‘‘mixed’’ problems where only partial knowledge of  $k(x, \omega)$  and  $u(x, \omega)$  is available. This class includes a spectrum of problems depending on the number of sensors on  $k(x, \omega)$  versus  $u(x, \omega)$ , where by decreasing the number of sensors on  $k(x, \omega)$  and increasing the number of sensors on  $u(x, \omega)$ , the estimation problem gradually transforms from forward to a mixed and finally to an inverse problem [18, 19]. In this work, we also apply our proposed method to high-dimensional stochastic problems, where the uncertainty in the complex system properties or forcing terms can only be characterized by a high-dimensional stochastic space.

Our approach is based on the VAE framework which involves comparison between probability distributions. There are various distance measures that can be adopted for calculating the distance between two distributions. Our approach uses one of these measures, namely Maximum Mean Discrepancy (MMD). For comparison, we have also used other measures for probability distributions in the examples of Section 4. In this section, we briefly explain the used measures, which include a kernel-based and two Optimal Transport measures. We also introduce a brief background on VAEs and its improved variants that are proposed in the literature for better performance.

### 2.1 Measurement of distance between probability distributions

Optimal Transport (OT) theory was initially proposed to determine the optimal way to redistribute a mass to a new location, and was later used widely as a distance measure between distributions [26]. Specifically, the distance between two distributions is considered to be the effort of transporting a mass distributed according to a function  $P_X$  to another location with a different distribution  $P_Y$ . Kantorovich’s formulation of this problem is given by:

$$W_c(P_X, P_Y) := \inf_{\tau \in P(x \sim P_X, x \sim P_Y)} \mathbb{E}_{(X, Y) \sim \tau} [c(x, x)],$$

where  $c(X, Y)$  is any self-defined cost function to measure the cost of moving density from position of distribution  $P_X$  to the position of another distribution  $P_G$  and  $\tau$  is one of the joint distributions of  $(X, Y)$  whose marginals are  $P_X$  and  $P_G$ , respectively. Wasserstein distance is a special case of  $W_c$  where  $c(X, Y) = d^p(X, Y)$  for  $p \geq 1$ , where  $d(\cdot, \cdot)$  is a distance measure defined for realizations of variables  $X$  and  $Y$ . We call  $W_p$  as the  $p$ -Wasserstein distance, which is the  $p$ -th root of  $W_c$ . Particularly when  $p = 1$ , the 1-Wasserstein distance can be simplified as:

$$W_1(P_X, P_Y) := \sup_{f \in F_L} \mathbb{E}_{X \sim P_X} [f(X)] - \mathbb{E}_{Y \sim P_Y} [f(Y)]$$

where  $F_L$  is a class of bounded 1-Lipschitz functions on the metric space  $(X, d)$ .

Calculating Wasserstein distance is computationally demanding. As an algorithm to approximate the Wasserstein distance, the Sinkhorn Iteration was proposed, where the entropy regularized OT cost was calculated using discrete samples from the distribution. The resulting Sinkhorn divergence is a biased estimate of the Wasserstein distance [27]. It has been shown that Sinkhorn divergence can approximate the Wasserstein distance with satisfactory accuracy at a much smaller computational cost when appropriate hyper-parameters are selected [28].

Maximum Mean Discrepancy (MMD) has also been introduced to measure the distance between probability distributions [29]. In a data space  $\chi$ , given independent random variables  $x, x' \in \chi$  with distribution  $P$  and independent random variables  $y, y' \in \chi$  with distribution  $Q$ , the MMD between  $P$  and  $Q$  over  $\chi$  is defined as:

$$\text{MMD}_k^2(P, Q) := \mathbb{E}_{x, x'} [k(x, x')] + \mathbb{E}_{y, y'} [k(y, y')] - 2\mathbb{E}_{x, y} [k(x, y)],$$

where  $k$  is the kernel of a reproducing kernel Hilbert space (RKHS)  $H_k$  of functions on  $\mathcal{X}$ . We note that  $\text{MMD}_k(P, Q) = 0$  if and only if  $P = Q$ . When MMD is used as the loss function in stochastic gradient descent-based trainings, an unbiased U-statistic empirical estimator of MMD is calculated using discrete samples. This estimator for  $m$  realizations of distribution  $P$  denoted by  $\{x^i\}_{i=1}^m$  and  $n$  realizations of distribution  $Q$ , denoted by  $\{y^i\}_{i=1}^n$ , using kernel  $k$  is given by:

$$\text{MMD}_{k,U}^2(\{x^i\}_{i=1}^m, \{y^i\}_{i=1}^n) := \frac{1}{m(m-1)} \sum_{i=1}^m \sum_{j \neq i}^m k(x^{(i)}, x^{(j)}) + \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j \neq i}^n k(y^{(i)}, y^{(j)}) - \frac{2}{mn} \sum_{i=1}^m \sum_{j=1}^n k(x^{(i)}, y^{(j)}).$$

Since MMD is a kernel-based measurement, different choices of characteristic kernels will lead to different test powers for finite sample sizes [30]. But MMD is a reliable metric, its evaluations with various characteristic kernels have been shown to be consistent [29].

## 2.2 Variational autoencoders

Variational Autoencoders are neural network models that are used for efficient representation of complex data structures, e.g. for low-dimensional representation of high-dimensional data. The AE architecture consists of an encoder that maps the data into a latent space and a decoder that maps points from the latent space back into the original space. Specifically, let  $\mathcal{E} : X \rightarrow Z$  denote a parametric mapping from the input data space  $X$  to a latent space  $Z$  (e.g., a neural network encoder). Utilizing a technique often used in the theoretical physics community, known as Random Variable Transformation (RVT), the probability density function of the encoded samples  $z$  can be expressed in terms of  $\mathcal{E}$  and  $p_X$  by:

$$P_Z(z) = \int_X p_X(x) \delta(z - \mathcal{E}(x)) dx$$

where  $\delta$  denotes the Dirac distribution function. The main objective of the encoder is to encode the input data  $x \in X$  into latent variable  $z \in Z$  such that  $x$  can be recovered from the probability density function of the encoded samples,  $p_Z$ , following a prior distribution  $q_Z$ . Similar to traditional autoencoders, a decoder  $D : Z \rightarrow X$  is required to map the latent codes back to the input data space such that:

$$P_Y(y) = \int_X p_X(x) \delta(y - \mathcal{D}(\mathcal{E}(x))) dx$$

where  $y$  denotes the decoded samples. It is straightforward to see that when  $\mathcal{E} = \mathcal{D}^{-1}$ , the distribution of the decoder  $p_Y$  and the input distribution  $p_X$  are identical. Hence, the training of an autoencoder involves learning parameters of  $\mathcal{E}$  and  $\mathcal{D}$  by minimizing a loss functions which consists of dissimilarity measures between  $p_Y$  and  $p_X$  in input data space and between  $p_Z$  and  $q_Z$  in latent space. Defining and implementing the dissimilarity measure is an important design decision.

For comparison in the latent space, traditional VAE uses KL divergence as the dissimilarity measure. However, KL divergence only measures the information loss of approximating one distribution with the other one but can not measure the distance from a distribution to another distribution. Various alternatives exist to address this limitation. In Wasserstein autoencoder [31], the authors proposed a GAN-based method and maximum mean discrepancy (MMD)-based approach to measure the disparity of latent variables. In the Sinkhorn autoencoder [32], the Sinkhorn iteration was used to approximate the Wasserstein distance as the criterion to constrain the distribution of latent variables.

For comparison in the input data space, various measures can be used. In traditional variational autoencoders, the cross-entropy loss or mean square error (MSE) is used in the loss function [33]. The Wasserstein GAN [34] used the OT-based 1-Wasserstein distance to measure data similarity in the input data space and reformulated the loss function as a min-max optimization problem solved by adversarial training scheme. MMD is also used in [35], which replaces the discriminator of GAN with a two-sample test based on MMD. Even though there exists limited literature about autoencoders exploring other algorithms for constraining the distribution in input data space, inspired by the above two papers, replacing the current popular loss function with MMD in input data space may improve the performance of autoencoders.



### 3 Methodology

In this section, we propose two PI-VAE algorithms; one for approximating a stochastic process and one for solving an SDE. Although approximation of stochastic processes is not the primary objective of PI-VAE, numerical experiments on reconstructing the sample paths of a stochastic process can serve as preliminary tests of PI-VAE on stochastic processes. Following that, we discuss how a PI-VAE can be constructed and trained to solve forward problems, inverse problems, and mixed problems involving SDEs.

#### 3.1 Approximating stochastic processes with VAE

We first consider modeling a stochastic process  $f(x, \omega)$  on the domain  $\mathbf{D} \in \mathbb{R}^d$  using a limited number of measurements from scattered sensors on this process. Specifically, we place  $n_f$  sensors at locations  $\{x_i\}_{i=1}^{n_f}$  to collect "snapshots" of  $f(x, \omega)$ . Here, a "snapshot" represents simultaneous record of all sensors, so data in one snapshot correspond to same random event  $\omega$ . We collect a total of  $N$  snapshots of  $f(x, \omega)$ , each taken at one realization of  $\omega$  and at  $n_f$  different sensor locations:

$$\{F(\omega^{(j)})\}_{j=1}^N = \{(f(x_i, \omega^{(j)}))_{i=1}^{n_f}\}_{j=1}^N$$

The VAE framework consists of an encoder  $\mathcal{E}_\phi(\cdot)$ , which is a feed-forward DNN parameterized by  $\phi$  and a decoder  $\tilde{f}_\theta(\cdot)$ , which is a feed-forward DNN parameterized by  $\theta$ . The encoder takes sensor measurements  $F(\omega^{(j)})$  as input and returns the corresponding latent variable  $z_j = \mathcal{E}_\phi(F(\omega^{(j)}))$  as outputs. Then latent variables  $\{z_j\}$  and coordinates  $\{x_i\}$  will be then plugged into the decoder as inputs to calculate the reconstructions  $\tilde{f}_\theta(x_i, z^{(j)})$  as the final outputs. The encoder and decoder will be trained simultaneously in this end-to-end structure with variants of mini-batch gradient descent algorithms based on the following loss function:

$$L(\phi, \theta) = \sum_{i=1}^M \{\text{MMD}_{k_i, U}(z, \zeta) + \text{MMD}_{k_i, U}(f(x, \omega), \tilde{f}_\theta(x, z))\}.$$

The first term in this loss function represents the empirical MMD estimator between the sample distribution of latent variables  $z$  and that of random realizations  $\zeta$  drawn from the prior distribution, taken to be a normal distribution. The second term is the empirical MMD estimator between sensor measurements and reconstructed data. After the training, the output of a well-trained decoder,  $\tilde{f}_\theta(x, \zeta)$ , can be used to approximate the stochastic process  $f(x, \omega)$ . Both of the MMD estimators in this loss functions are evaluated at  $M$  different Gaussian kernels. The detailed algorithm is presented in Algorithm 1.

---

#### Algorithm 1: VAE for approximating stochastic process

---

Initialization: Set the number of training steps  $n_t$ , batch size  $N$ , Adam hyper-parameters  $\alpha, \beta_1, \beta_2$ , initial parameters for encoder and decoder  $\phi$  and  $\theta$ , kernels of MMD estimator  $\{k_i\}_{i=1}^M$ .

**for**  $i = 1, 2, \dots, n_t$  **do**

Sample  $N$  snapshots  $\{F(\omega^{(j)})\}_{j=1}^N$ .

Sample  $N$  random vector from Prior  $\{\zeta_j\}_{j=1}^N$ .

**for**  $j = 1, 2, \dots, n$  **do**

$z_j = \mathcal{E}_\phi(F(\omega^{(j)}))$

$\tilde{F}(\omega^{(j)}) = \tilde{f}_\theta(z_j)$

**end**

$L = \sum_i^M \text{MMD}_{k_i, U}(\{z_j\}_{j=1}^N, \{\zeta_j\}_{j=1}^N) + \text{MMD}_{k_i, U}(\{F(\omega^{(j)})\}_{j=1}^N, \{\tilde{F}(\omega^{(j)})\}_{j=1}^N)$

$\theta, \phi \leftarrow \text{Adam}(\nabla L, \theta, \phi, \alpha, \beta_1, \beta_2)$

**end**

---

#### 3.2 Solving SDEs with PI-VAE

Let us consider the following SDE

$$\mathcal{N}_x[u(x, \omega), k(x, \omega)] = f(x, \omega) \quad x \in \mathbf{D}, \quad \omega \in \mathbf{\Omega}, \quad (2)$$

$$\mathcal{B}_x[u(x, \omega)] = b(x, \omega), \quad x \in \Gamma, \quad (3)$$

defined in Section 2. The objective is to calculate the solution  $u(x, \omega)$  using measurements from scattered sensors which form snapshots of the three stochastic processes,  $k(x, \omega)$ ,  $f(x, \omega)$  and  $b(x, \omega)$ , at locations  $\{x_i^k\}_{i=1}^{n_k}$ ,  $\{x_i^f\}_{i=1}^{n_f}$  and  $\{x_i^b\}_{i=1}^{n_b}$ , where  $n_k$ ,  $n_u$ ,  $n_f$ , and  $n_b$  are the respective numbers of sensors. Here, one snapshot represents a simultaneous observation of all the sensors, and we assume that the data in one snapshot correspond to the same random event  $\omega$  in the random space  $\Omega$ , while  $\omega$  varies across different snapshots.

Suppose we have a group of  $N$  snapshots, denoted by  $\{S(\omega^{(j)})\}_{j=1}^N$ , defined as:

$$\begin{aligned} \{S(\omega^{(j)})\}_{j=1}^N &= \{K(\omega^{(j)}), F(\omega^{(j)}), B(\omega^{(j)})\}_{j=1}^N, \\ K(\omega^{(j)}) &= (k(x_i^k; \omega^{(j)}))_{i=1}^{n_k}, \\ F(\omega^{(j)}) &= (f(x_i^f; \omega^{(j)}))_{i=1}^{n_f}, \\ B(\omega^{(j)}) &= (b(x_i^b; \omega^{(j)}))_{i=1}^{n_b}. \end{aligned} \quad (4)$$

We assume that we always have a sufficient number of sensors for the forcing term  $f(x, \omega)$  because accurate estimation of the forcing term is critical in physics-based training.

The solution of the SDE is calculated by constructing the PI-VAE framework via the following three steps:

- Firstly, we use fully connected feed-forward neural networks as our encoder and decoders. An encoder  $\mathcal{E}_\phi$  maps input data to the latent variables  $z$ , i.e.  $z^{(j)} = \mathcal{E}_\phi(S(\omega^{(j)}))$ . Two independent decoders  $\tilde{k}_{\theta_k}(x, z)$  and  $\tilde{u}_{\theta_u}(x, z)$  approximate corresponding stochastic process  $k(x, \omega)$  and  $u(x, \omega)$  respectively by constructing a mapping from coordinates  $x$  and latent variable  $z$  to input data. We expect to generate approximate QoIs using the well-trained decoders after the training process.
- Second, inspired by physics-informed neural network for deterministic differential equations [11, 36], we encode the governing differential equation into the framework by applying operator  $\mathcal{N}_x$  and  $\mathcal{B}_x$  on the decoder outputs,  $\tilde{k}_{\theta_k}$  and  $\tilde{u}_{\theta_u}$ , to obtain approximations of the  $f(x, \omega)$  and  $b(x, \omega)$  in the governing SDE, i.e.:

$$\begin{aligned} \tilde{f}_{\theta_u, \theta_k}(x, z) &= \mathcal{N}_x[\tilde{u}_{\theta_u}(x, z), \tilde{k}_{\theta_k}(x, z)], \\ \tilde{b}_{\theta_u}(x, z) &= \mathcal{B}_x[\tilde{u}_{\theta_u}(x, z)]. \end{aligned}$$

Differentiations in  $\mathcal{N}_x$  and  $\mathcal{B}_x$  are carried out by automatic differentiation [37]. These physics-informed estimates  $\tilde{f}_{\theta_u, \theta_k}(x, z)$  and  $\tilde{b}_{\theta_u}(x, z)$  together with approximated response  $\tilde{u}_{\theta_u}(x, z)$  and system parameters  $\tilde{k}_{\theta_k}(x, z)$  constitutes the reconstructed snapshots  $\{\tilde{S}(z^{(j)})\}$ , i.e.

$$\begin{aligned} \{\tilde{S}(z^{(j)})\}_{j=1}^N &= \{\tilde{K}(z^{(j)}), \tilde{F}(z^{(j)}), \tilde{B}(z^{(j)})\}_{j=1}^N, \\ \tilde{K}(z^{(j)}) &= (\tilde{k}_{\theta_k}(x_i^k, z^{(j)}))_{i=1}^{n_k}, \\ \tilde{F}(z^{(j)}) &= (\tilde{f}_{\theta_u, \theta_k}(x_i^f, z^{(j)}))_{i=1}^{n_f}, \\ \tilde{B}(z^{(j)}) &= (\tilde{b}_{\theta_u}(x_i^b, z^{(j)}))_{i=1}^{n_b}, \end{aligned}$$

- In the third step, we formulate the loss function which consists of the reconstruction term and a regularization term. The reconstruction cost is calculated using the MMD between collected sensor measurements  $\{S(\omega^{(j)})\}_{j=1}^N$  and reconstructed samples  $\{\tilde{S}(z^{(j)})\}_{j=1}^N$ . The regularization term promotes the distribution of the latent encodings to be close to standard normal distribution. This term measures the MMD between latent variables  $z$  (obtained by encoding the measured snapshots) and random variables  $\zeta$  distributed according to the prior distribution, here chosen to be standard normal. Thus, the loss function is given by

$$L(\phi, \theta_u, \theta_k) = \sum_{i=1}^M \text{MMD}_{k_i, U}(\tilde{S}(z), S(\omega)) + \sum_{i=1}^M \text{MMD}_{k_i, U}(z, \zeta).$$

After training using this loss function, the decoders  $\tilde{u}_{\theta_u}(x; \zeta)$  and  $\tilde{k}_{\theta_k}(x; \zeta)$  is used to approximate the stochastic process  $u(x, \omega)$  and  $k(x, \omega)$  at various locations  $x$ . A detailed description of our training algorithm is provided in Algorithm 2 and Figure 1.

---

**Algorithm 2:** PI-VAE for solving SDEs (forward problems)

---

Initialization: Set the training steps  $n_t$ , batch size  $n$ , Adam hyper-parameters  $\alpha, \beta_1, \beta_2$ , initial values for encoder and decoder parameters  $\phi, \theta_u, \theta_k$ , and kernels of MMD estimator  $\{k_i\}_{i=1}^M$ .

**for**  $i = 1, 2, \dots, n_t$  **do**

    Sample  $N$  snapshots  $\{S(\omega^{(j)})\}_{j=1}^N$ .

    Sample  $N$  random vectors from the prior  $\{\zeta_j\}_{j=1}^N$ .

**for**  $j = 1, 2, \dots, N$  **do**

$z_j = \mathcal{E}_\phi(S(\omega^{(j)}))$

$\{\tilde{K}(z^{(j)}), \tilde{F}(z^{(j)}), \tilde{B}(z^{(j)})\} = \{k_{\theta_k}(x, z^{(j)}), \tilde{f}_{\theta_u, \theta_k}(x, z^{(j)}), \tilde{b}_{\theta_u}(x, z^{(j)})\}$

$\tilde{S}(z^{(j)}) = [\tilde{K}(z^{(j)}), \tilde{F}(z^{(j)}), \tilde{B}(z^{(j)})]$

**end**

$L = \sum_i^M \text{MMD}_{k_i, U}(\{\tilde{S}(z^{(j)})\}_{j=1}^N, \{S(\omega^{(j)})\}_{j=1}^N) + \text{MMD}_{k_i, U}(\{z^{(j)}\}_{j=1}^N, \{\zeta_j\}_{j=1}^N)$

$\phi, \theta_u, \theta_k \leftarrow \text{Adam}(\nabla L, \phi, \theta_u, \theta_k, \alpha, \beta_1, \beta_2)$

**end**

---

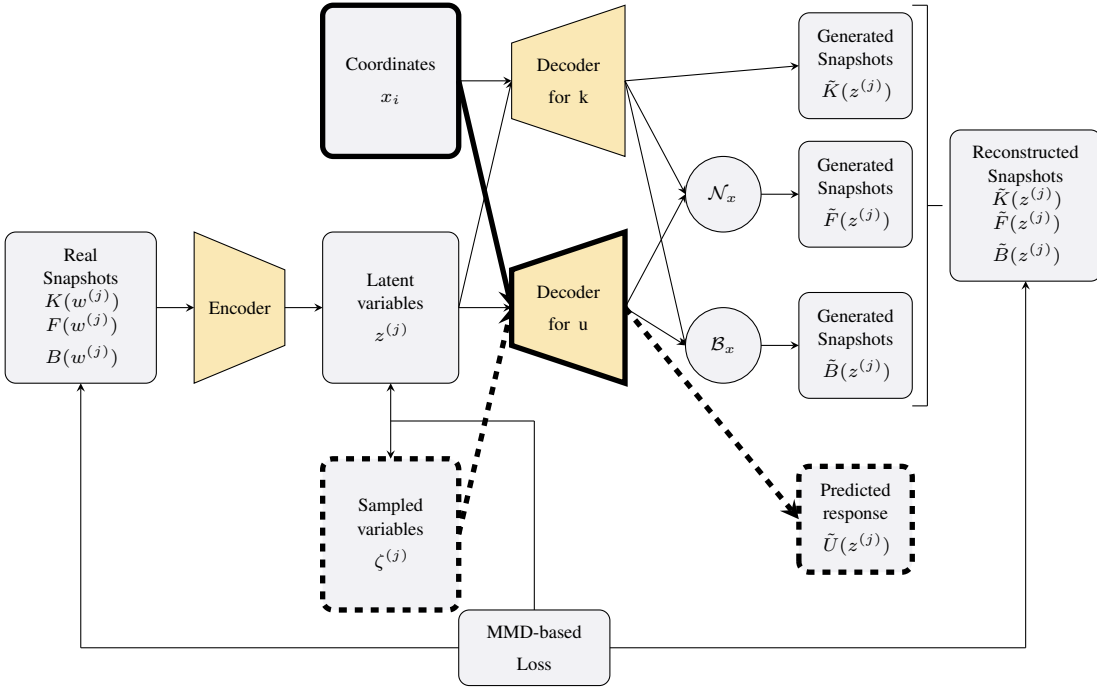


Figure 1: The architecture of the PI-VAE framework for solving forward problems. Thin lines show the parts used only during training; solid thick lines show the parts used both in training and test stages; dashed thick lines are the parts used only during test.

### 3.3 Solving inverse problems

In inverse problems, the goal is to calculate the unknown model parameters using measurement data from the response and boundary conditions. Following the formulation 6, we seek to estimate the random parameter  $k(x, \omega)$  using measurements from scattered sensors on these processes,  $u(x, \omega)$ ,  $f(x, \omega)$  and  $b(x, \omega)$ , at locations,  $\{x_i^u\}_{i=1}^{n_u}$ ,  $\{x_i^f\}_{i=1}^{n_f}$  and  $\{x_i^b\}_{i=1}^{n_b}$ , where  $n_u$ ,  $n_f$ , and  $n_b$  are the respective numbers of sensors. Since we have a physics-based loss which involves calculation of the residual, we also add a few measurements of  $k(x, \omega)$  at  $\{x_i^k\}_{i=1}^{n_k}$  with  $n_k \ll n_u$ . Suppose we have a group of  $N$  snapshots, denoted by  $\{S(\omega^{(j)})\}_{j=1}^N$ , defined as:

$$\begin{aligned} \{S(\omega^{(j)})\}_{j=1}^N &= \{K(\omega^{(j)}), U(\omega^{(j)}), F(\omega^{(j)}), B(\omega^{(j)})\}_{j=1}^N, \\ K(\omega^{(j)}) &= (k(x_i^k; \omega^{(j)}))_{i=1}^{n_k}, \\ U(\omega^{(j)}) &= (u(x_i^u; \omega^{(j)}))_{i=1}^{n_u}, \\ F(\omega^{(j)}) &= (f(x_i^f; \omega^{(j)}))_{i=1}^{n_f}, \\ B(\omega^{(j)}) &= (b(x_i^b; \omega^{(j)}))_{i=1}^{n_b}. \end{aligned} \quad (5)$$

We assume that we always have sufficient number of sensors for the forcing term  $f(x, \omega)$  because accurate estimation of the forcing term is critical in physics-based training. We take similar steps to the three steps discussed in the previous section and solve the inverse and mixed problems according to Algorithm 3 and Figure 2.

---

#### Algorithm 3: PI-VAE for solving mixed and inverse SDEs

---

Initialization: Set the training steps  $n_t$ , batch size  $n$ , Adam hyper-parameters  $\alpha, \beta_1, \beta_2$ , initial values for encoder and decoder parameters  $\phi, \theta_u, \theta_k$ , and kernels of MMD estimator  $\{k_i\}_{i=1}^M$ .

**for**  $i = 1, 2, \dots, n_t$  **do**

    Sample  $N$  snapshots  $\{S(\omega^{(j)})\}_{j=1}^N$ .

    Sample  $N$  random vectors from the prior  $\{\zeta_j\}_{j=1}^N$ .

**for**  $j = 1, 2, \dots, N$  **do**

$z_j = \mathcal{E}_\phi(S(\omega^{(j)}))$

$\{\tilde{K}(z^{(j)}), \tilde{F}(z^{(j)}), \tilde{U}(z^{(j)})\} = \{k_{\theta_k}(x, z^{(j)}), \tilde{f}_{\theta_u, \theta_k}(x, z^{(j)}), u_{\theta_u}(x, z^{(j)})\}$

$\tilde{S}(z^{(j)}) = [\tilde{K}(z^{(j)}), \tilde{F}(z^{(j)}), \tilde{U}(z^{(j)})]$

**end**

$L = \sum_i^M \{\tilde{S}(z^{(j)})\}_{j=1}^N, \{S(\omega^{(j)})\}_{j=1}^N) + \text{MMD}_{k_i, U}(\{z^{(j)}\}_{j=1}^N, \{\zeta^{(j)}\}_{j=1}^N)$

$\phi, \theta_u, \theta_k \leftarrow \text{Adam}(\nabla L, \phi, \theta_u, \theta_k, \alpha, \beta_1, \beta_2)$

**end**

---

## 4 Numerical results

In this section, we numerically evaluate how accurately the proposed VAE framework can approximate a stochastic process, and how well the proposed PI-VAE approach can solve SDEs. We compare our results with Wasserstein GAN with gradient penalty (WGAN-GP) and Physics-informed WGAN (PI-WGANs) [19].

All test cases commonly share the following default settings unless mentioned otherwise. We use the hyperbolic tangent function (tanh) as our activation function to ensure the smoothness in high-order derivatives of the developed neural networks. Every feed-forward network (i.e. encoders and decoders of VAE; generator and discriminator of WGAN) has four hidden layers of width 128. The latent space is chosen to be an independent multivariate standard Gaussian multivariate. Adam is the default optimizer with the following default hyper-parameters:  $\beta_1 = 0.5$  and  $\beta_2 = 0.9$ . For WGAN-GP, we set the hyper-parameters in the loss function  $\lambda=0.1$ ,  $n_d=5$  as default values similarly to [19]. For the hyper-parameter of Sinkhorn Iteration, we set the default value  $\epsilon=0.1$ . For the basic setting of MMD, we will use five radial basis function (RBF) kernels of different bandwidths for calculation. In all numerical experiments of approximating stochastic processes, the learning rate is 0.001 as the default value. For SDE cases, we consider special settings for the learning rate, which will be explained in the following sections. We collect all the training data by

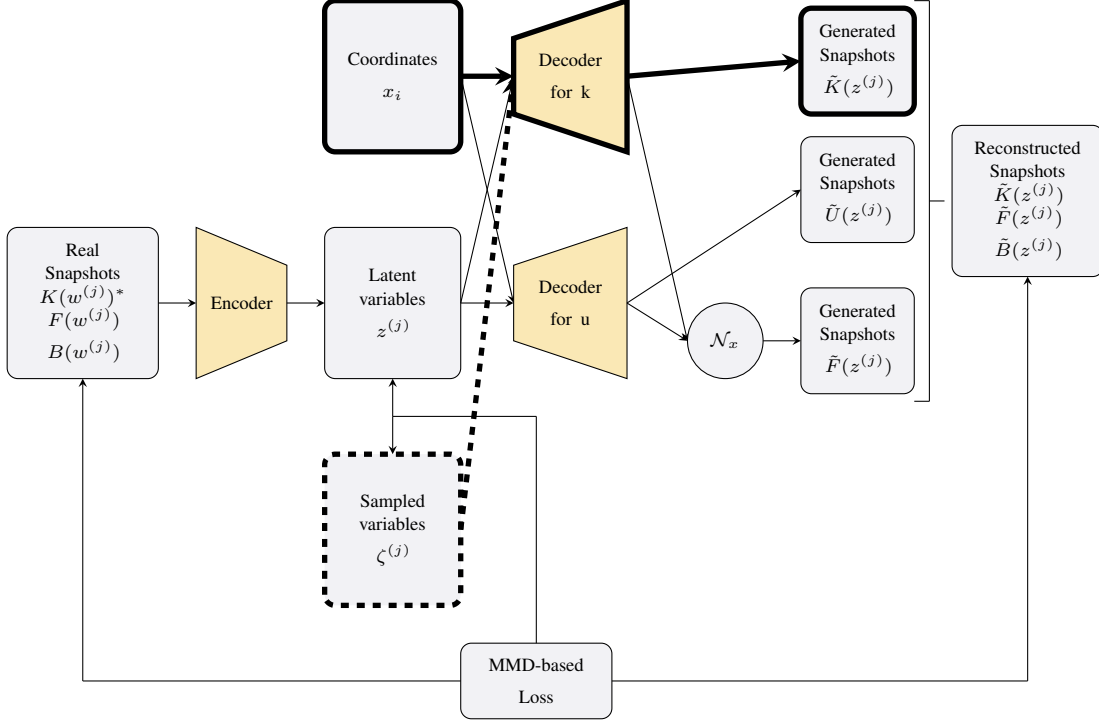


Figure 2: The architecture of the PI-VAE framework for solving mixed and inverse problems. Thin lines show the parts used only during training; solid thick lines show the parts used both in training and test stages; dashed thick lines are the parts used only during test. Note that only few samples from  $k$  is used in inverse and mixed problems.

numerically simulating sensor measurements of the stochastic processes based on Finite Difference Method and Monte Carlo Simulation.

## 4.1 Approximating stochastic processes

### 4.1.1 Processes with different kernels

We consider Gaussian Processes with zero mean and two different kernels: a squared exponential kernel and an exponential kernel, as follows

$$f_1(x) \sim \text{GP}(0, \exp(-\frac{1}{2l^2}(x-x')^2)), \quad x, x' \in [-1, 1],$$

$$f_2(x) \sim \text{GP}(0, \exp(-\frac{1}{2l}(x-x'))), \quad x, x' \in [-1, 1],$$

where  $l$  is the correlation length scale, which will be set to 0.2, 0.5, or 1. The number of sensors is 6 or 11. Sensors are uniformly placed over the domain  $[-1, 1]$ . In the following case, to show that our method can find the pattern from a limited number of scattered measurements, we will only use 2,000 snapshots for our training data as the default setting. Also, to ensure that the effect of batch size is considered, we will use a batch size of 1000 for our training, which is different from [19]. For each case, we run the code with different random seeds. Few sample paths of the stochastic processes are shown in Figure 3. As we can see in the figure, stochastic processes of the exponential kernel results in more locally correlated processes, which will be more challenging to be approximated.

For length scales of 1.0, 0.5, and 0.2, we train our VAE model and WGAN-GP for 500, 1000, and 1500 training epochs, respectively. After the training is finished, we will use 101 uniformly distributed coordinates over the domain for validation of the trained neural network. We pick 100 decoders at the last 100 training epochs and use them to reconstruct sample paths on the validation coordinates as reconstructed samples. The ground truth samples are collected by re-sampling of the stochastic processes. To validate the accuracy of the algorithms, in Figures 4 and 5, respectively for squared exponential and exponential kernels, we compare the reconstructed and ground truth sample paths in terms

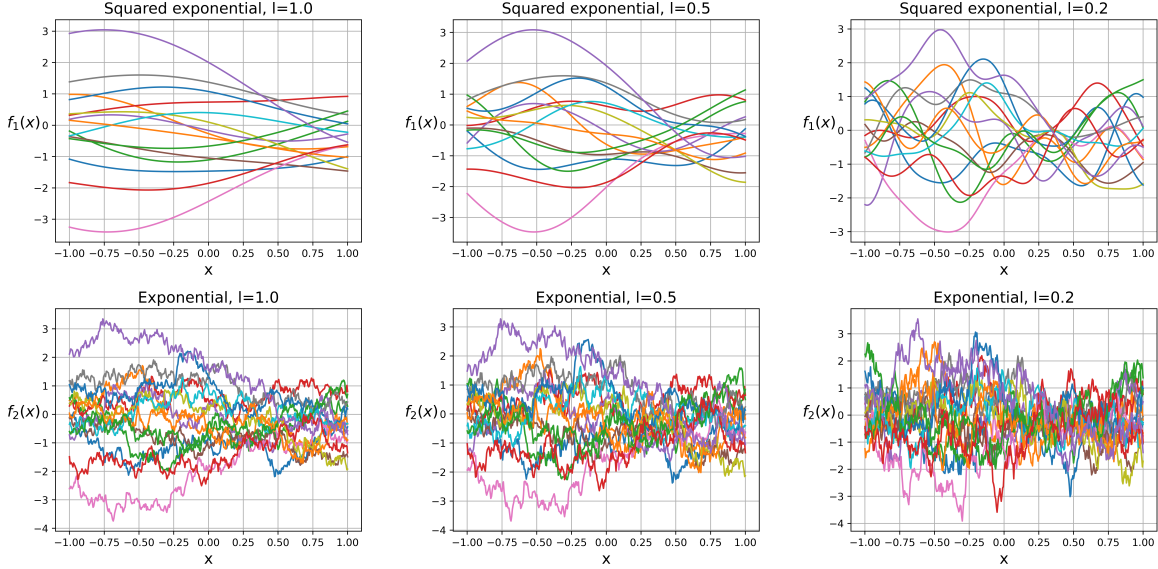


Figure 3: Sample paths of Gaussian processes of different kernel function and length scale. Sample paths of different correlation lengths:  $l = 1$  (left),  $0.5$  (middle), and  $0.2$  (right) and different kernels: squared exponential kernel (top) and exponential kernel (bottom) are shown.

of (1) the spectra of the sample paths, i.e., the eigenvalues of the covariance matrices, and (2) the Wasserstein distance  $\mathcal{W}_1$ .

In these figures, the  $\mathcal{W}_1$  distance between reconstructed samples and ground true samples are shown at different training epochs, and one can see the convergence after 500 training epochs. It can be seen that the Wasserstein distance for all the cases converge to approximately 1, except for exponential kernels with length scales of  $0.5$  and  $0.2$ , where we cannot achieve the same desired accuracy when we use only 6 sensors.

To compare the eigenvalues, as the ground truth, we calculate the eigenvalues of the analytical form of the covariance matrix of the stochastic processes. The eigenvalues of the reconstructed sample paths are calculated using 1,000 training sample paths, and as can be seen from the spectra of the covariance matrices of these process, our model can effectively reconstruct samples with similar spatial distributions. It should be noted that the approximation errors is larger for the more dominant eigenvectors. Also, the approximation errors are smaller in the squared exponential kernel case since these processes are less locally correlated, compared to processes with exponential kernels, and are easier to approximate. Similar justification can be used for the larger error observed in smaller length scales. Improvements in approximation errors can be seen when more sensors are used. In the cases of length scale =  $1.0$  and  $0.5$ , WGAN and our method can accurately reproduce the samples of covariance of similar spectral structures. However, for the smaller length scale of  $0.2$ , our VAE model has slightly better performance compared to WGAN-GP.

#### 4.1.2 Different choices of loss functions

As mentioned in Section 3.1, the VAE model can be built using different distance measures between probability distribution. In this section, we evaluate the performance of our proposed model when different distance measures are used, we now use different distance measures as the loss function in the training of (unconstrained) Gaussian processes and Gaussian processes constrained on the boundaries.

As mentioned before, the loss function of the VAE consists of dissimilarity measures in both latent space and input data space. In this work, we always use MMD as the dissimilarity measure in the latent space. In the high-dimensional data space, we consider three choices for the similarity measure: Mean Square Error (MSE), Sinkhorn Iteration and MMD, and assess the three corresponding models. Although cross entropy loss is also a popular loss function, it is recommended to adopt MSE as the loss function when the data of original space is not binary [22].

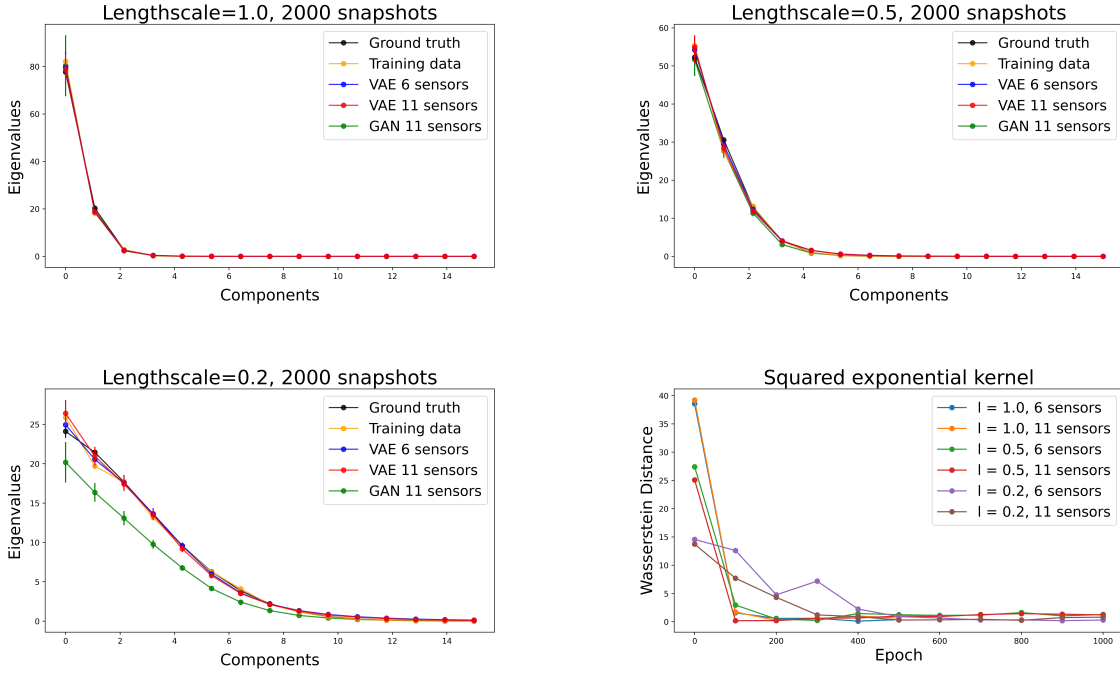


Figure 4: Results of using our method to approximate Gaussian Process of squared exponential kernel. The first three plots shows the spectral of the covariance matrix of the reconstructed samples and ground true samples. The last plot shows the similarity of reconstructed samples and ground true samples in terms of  $\mathcal{W}_1$  distance.

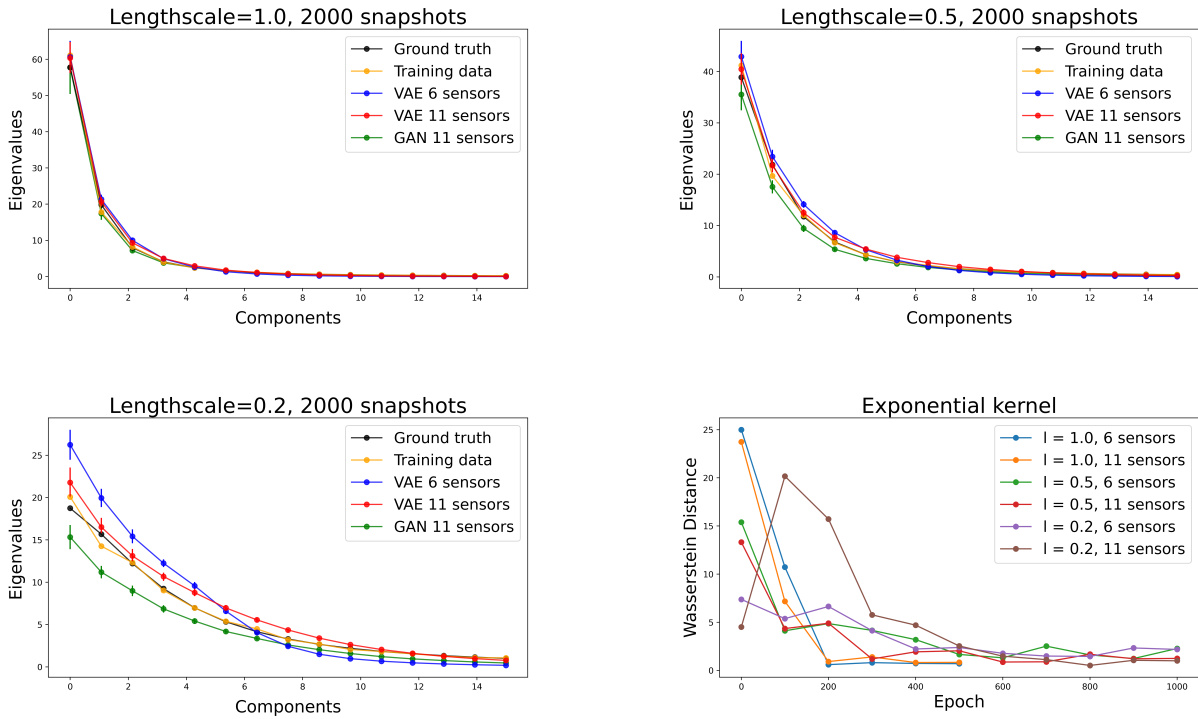


Figure 5: Results of using our method to approximate Gaussian Process of exponential kernel. The first three plots shows the spectral of the covariance matrix of the reconstructed samples and ground true samples. The last plot shows the similarity of reconstructed samples and ground true samples in terms of  $\mathcal{W}_1$  distance.

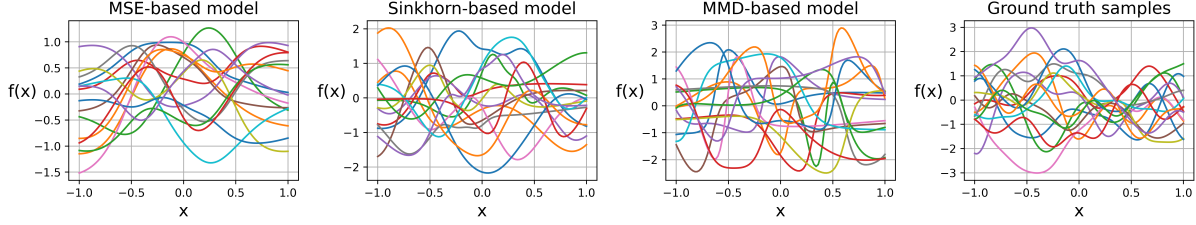


Figure 6: Reconstructed samples of models trained by different loss function which are MSE, Sinkhorn and MMD in input space. The ground truth samples are also shown for comparison.

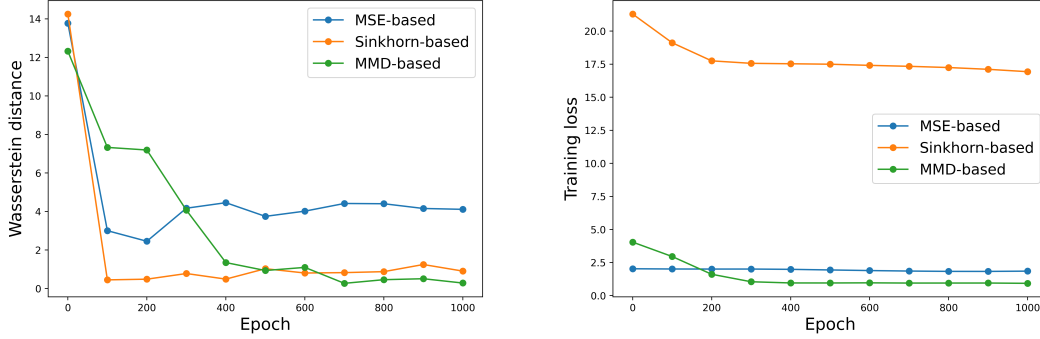


Figure 7: Wasserstein distance (left) and training loss (right) versus the training epoch of the three models using different training loss.

The reconstructed samples of models of different loss functions are shown in Figure 6. All these models can reproduce smooth sample paths, which means that functions represented by our trained model have good continuity. Compared with ground truth samples, samples of MSE loss are much more spatially concentrated than the other two models. Although samples from the model using Sinkhorn loss are similar to the training sample paths, the length scale of the sample paths is not as small as the training samples. Among these three models, the model with the MMD loss can reproduce samples that are most similar to the training data.

The following numerical results are in accordance with the results shown in Figure 7, which shows the training loss converges and that there is no over-fitting. But we observe that the Wasserstein distance from the MSE-based VAE model is significantly larger than that from the other two models, which indicates that the generated samples from the MSE-based VAE model does not match the distribution of input data statistically. Besides, the Wasserstein distance for the Sinkhorn-based model is slightly larger than the Wasserstein distance of the MMD-based model.

The spectra comparison in Figure 8 shows that the Sinkhorn-based model and MSE-based models fail to capture covariance information as accurately as the MMD-based model. The MSE-based model can not perform as well as the other two models. This can be due to the fact that the MSE is only verified to be effective when input data follows a Gaussian distribution with an approximately diagonal covariance [22]. For the case of highly correlated Gaussian data, it seems that MSE cannot give us a very promising results. However, it is surprising to observe that Sinkhorn model performs worse than MMD model because theoretically Sinkhorn divergence is closer to Wasserstein distance than MMD [32]. We can associate this to the suboptimal selection of the hyper-parameter of Sinkhorn algorithm.

One of the drawbacks of Sinkhorn Iteration is the difficulty in choosing an appropriate hyper-parameter, specifically the convergence threshold [38]. In our numerical experiments, we observed that the whole training will easily collapse when a small threshold is used, while a large threshold leads to inaccurate estimation of the distribution distance[38]. So, we note that the Sinkhorn-based model can be potentially improved with further fine-tuning of its hyper-parameter, but also underscore that the MMD-based model is the most advantageous since it is the most accurate model without any need for hyper-parameter fine-tuning.

Let us now approximate a stochastic process with a fixed boundary condition. Specifically, we consider  $f(x, \omega) = (x^2 - 1)g(x, \omega)$ , where  $g(x, \omega)$  is a Gaussian process. The mean and standard deviation of the generated samples are



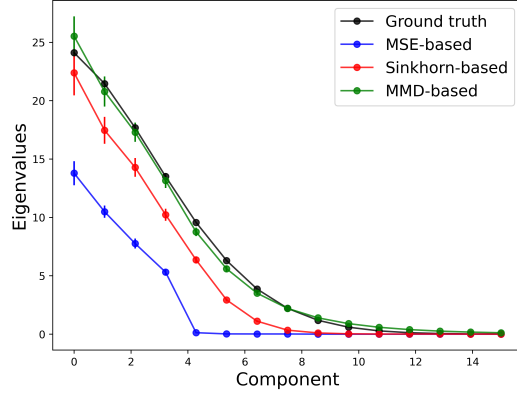


Figure 8: the spectral of the covariance matrix of the reconstructed samples of different models and ground truth samples

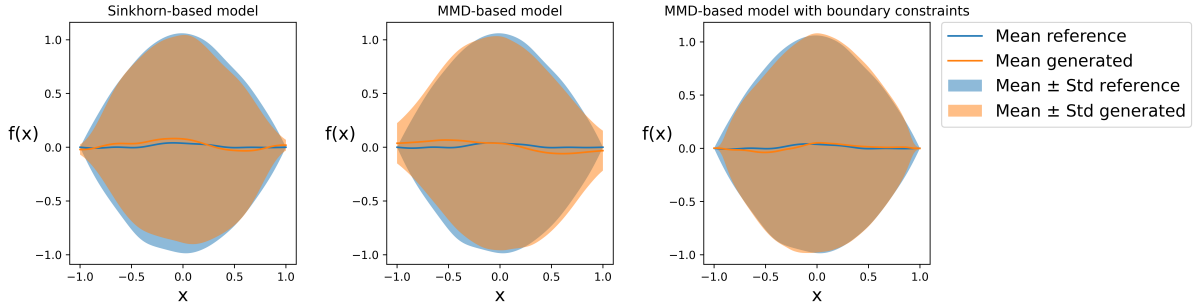


Figure 9: Mean and variance estimate of Sinkhorn-based model (left) and MMD-based model (mid) and the MMD-based model with MSE penalty on the boundary (right). Blue line represent the reference mean and blue shade area represents the reference standard deviation. In contrast, orange line and orange shade area represent the mean and std of the reconstructed samples, respectively.

compared for accuracy evaluation. We use 100 decoders and evaluate their average performance after 500 training epochs. It is observed that Sinkhorn-based model can approximate better on the boundary than MMD model, which indicates that Sinkhorn model can better detect the pattern of concentrated data distribution.

However, in some cases (e.g. solving SDEs), we will have an additional penalty term in the loss function to constrain the boundary condition. To increase the accuracy of our approximation on the boundary, we can add a penalty term on the boundary using MSE loss for MMD-based model. In our example, no trade-off is needed between different loss terms. As we can see in Figure 9, the approximation is much better than the vanilla MMD model.

### 4.1.3 Issue of mode collapse

The MMD measure calculates the distance between probability distributions using finite samples of reconstruction and input data. The number of samples will affect the accuracy of the estimation. Theoretically, the number of samples needed for MMD estimation is approximately proportional to the dimensionality of the data space [39]. This indicates that MMD estimation will not suffer from the "curse of dimensionality". In the numerical examples presented in this paper, the dimensions of the data are not large, allowing us to use regular mini-batch sizes to measure MMD. However, it is still necessary to assess the effect of batch size on the performance of the proposed method.

To this end, we show the calculated Wasserstein distance and the spectra of the covariance matrix in Figure 10 for different batch sizes. It can be seen the performance of our proposed model will decrease as the batch size in each training epoch decreases. This indicates that the reduction in the number of samples deteriorates the accuracy in MMD estimation. However, this performance decline in the covariance spectra is not significant except for the batch size of 10.

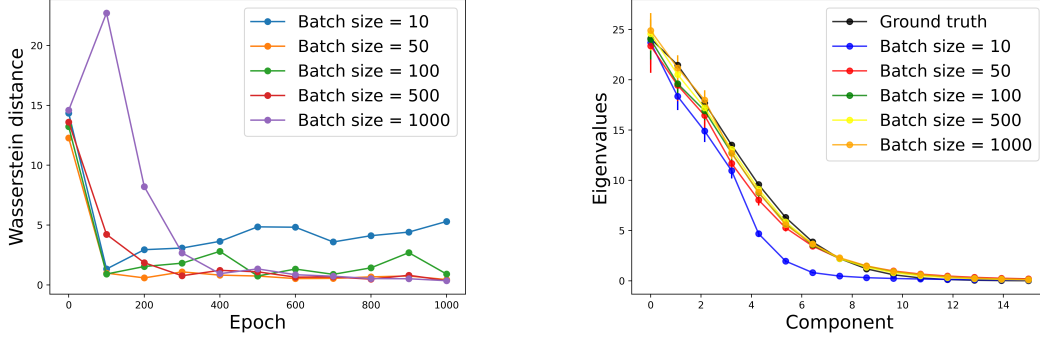


Figure 10: Wasserstein distance (left) of reconstructed samples and ground true samples and spectral of the covariance matrix (right) of the reconstructed samples of different batch size are shown.

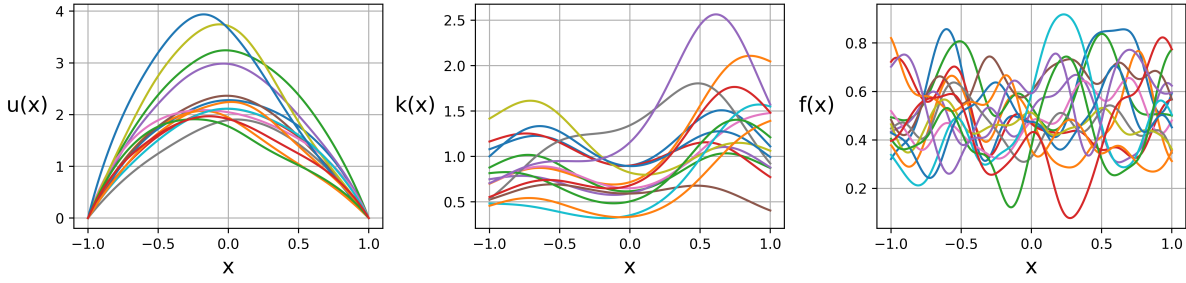


Figure 11: A few sample paths of  $k(x, \omega)$ ,  $u(x, \omega)$ , and  $f(x, \omega)$  used in the forward problem.

Similar observation can be made for the Wasserstein distance results, showing that our model is not sensitive to the common choices of batch size in low dimensional problems.

## 4.2 Forward problem

As a benchmark problem, similarly to [19], let us consider the following one-dimensional elliptic SDE

$$\begin{aligned}
 -\frac{1}{10} \frac{d}{dx} [k(x, \omega) \frac{d}{dx} u(x, \omega)] &= f(x, \omega), \quad x \in [-1, 1], \\
 u(-1) &= u(1) = 0,
 \end{aligned}
 \tag{6}$$

where on the domain boundary we impose homogeneous Dirichlet boundary conditions on  $u(x, \omega)$ . The independent stochastic processes  $k(x, \omega)$  and  $f(x, \omega)$  are given by

$$\begin{aligned}
 k(x) &= \exp\left[\frac{1}{5} \sin\left(\frac{3\pi}{2}(x+1)\right) + \hat{k}(x)\right], \\
 \hat{k}(x) &\sim \mathbf{GP}\left(0, \frac{4}{25} \exp(-(x-x')^2)\right), \\
 f(x) &\sim \mathbf{GP}\left(\frac{1}{2}, \frac{9}{400} \exp(-25(x-x')^2)\right),
 \end{aligned}
 \tag{7}$$

where  $k(x, \omega)$  is set to be strictly positive. In order to generate training data, we draw random sample paths of the stochastic processes  $k(x, \omega)$  and  $f(x, \omega)$ , based on which we use the finite difference method to solve the corresponding deterministic differential equation to obtain realizations of the solution  $u(x, \omega)$ , as the reference.

In order to train the neural network models, we use 17  $k$ -sensors, 21  $f$ -sensors, and 2  $u$ -sensors, at locations uniformly distributed over the computational domain. The sample paths of  $k(x, \omega)$ ,  $u(x, \omega)$ , and  $f(x, \omega)$  are shown in Figure 11.

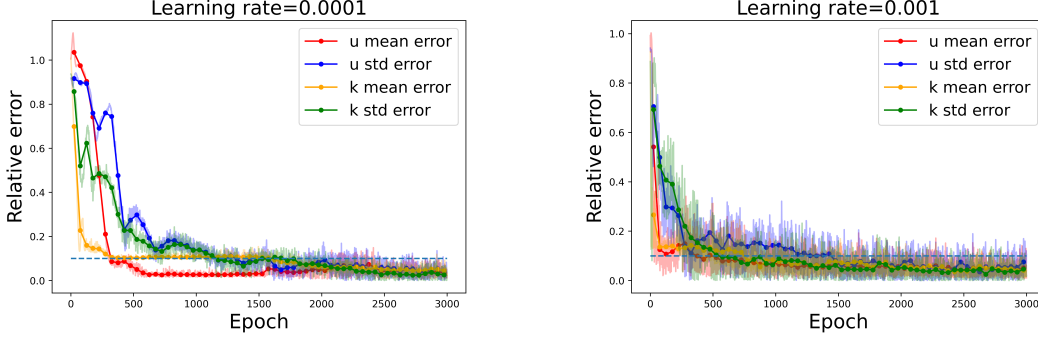


Figure 12: Relative error of QoIs of PI-WGAN trained using different learning rate of 0.0001 (left) and 0.001 (right).

Our main quantity of interest in this problem is the mean and standard deviation of  $u(x, \omega)$  on the validation coordinates, i.e.,  $\mu(x) = \mathbb{E}_w[u(x, \omega)]$  and  $\sigma^2(x) = \mathbb{E}_w[(u(x, \omega) - \mu(x))^2]$ . The number of validation points is 101. To study the influence of latent dimension, we fix the number of training snapshots to be 1000 and vary the latent dimension to be 2, 4, 10. Subsequently, we fix the latent dimension as 4 and vary the number of training snapshots to be 2000, 5000, and 10000 to study the influence of the number of training snapshots. During the training process, we keep the batch size to be 1000. One numerical solution of PI-WGAN using 10,000 snapshots with a four-dimensional latent space is obtained for comparison.

Due to the instability of GAN training, the hyper-parameters of the training scheme should be carefully chosen. To ensure a fair comparison between PI-WGANs and our PI-VAE method, we first need to select appropriate hyper-parameters for the PI-WGAN training. Most of the parameters are chosen following [19]. Although the default value of the learning rate is 0.001, we observed that using a small learning rate (i.e. 0.0001) will result in better convergence of PI-WGAN, as shown in Figure 12. The transparent curves show the exact error values, whereas the solid lines shown the error averaged over 50 training epochs for better comparison. We call the accuracy satisfactory, when average error is less than 0.1. In both these experiments, satisfactory accuracy is achieved after approximately 1500 epochs, but the smaller learning rate has resulted in a more stable performance. Based on these two numerical experiments, we choose 0.0001 as the learning rate for PI-WGAN training.

Now let us compare the performance of PI-VAE against PI-WGAN in terms of accuracy and efficiency. In Figure 13, we present the error of QoI versus the training epoch. In our experiments, we observed that only the training data size will affect the convergence and that different choices of dimension for the latent space has limited effects. It can be seen, that even though a much larger learning rate is used for PI-VAE training, the model performance remains stable after achieving satisfactory accuracy. Also, the training of PI-VAE is more stable than the training of PI-WGAN since no over-fitting or abnormal "bounce" of the error is detected in any of the numerical experiments.

It should be noted that the definition of an epoch is different in the training of PI-WGAN and PI-VAE. Thus, a more detailed efficiency comparison is presented in Table 1. It should be noted that the computation cost of each training epoch is different in PI-VAE and PI-WGAN models. In the procedure of GAN training, we need to perform  $n_d$  steps of optimizing discriminator parameters before a single update of generator parameters (with  $n_{de} = 5$  as the default value). This is because we need to maintain the discriminator to be near-optimal during the whole training [40]. However, VAE training is carried out similarly to the training of feed-forward neural networks. In other words, VAE training involves only one forward propagation and one backward propagation over the encoder and decoder parts. In contrast, GAN training involves six forward propagations over the generator and discriminator, five backward propagations over the discriminator, and one backward propagation over the generator.

Although the computation cost of backward will be higher, the computation cost of these two processes is at the same order of magnitude. Also, the architecture of the PI-VAE and PI-WGAN are similar, since we use the same neural network architecture between the encoder of PI-VAE and the discriminator of PI-WGAN, and also between the decoder of PI-VAE and the generator of PI-WGAN. As a result, the computational cost of one PI-WGAN training epoch is 4 to 5 times as large as the computational cost for one PI-VAE mini-batch training epoch.

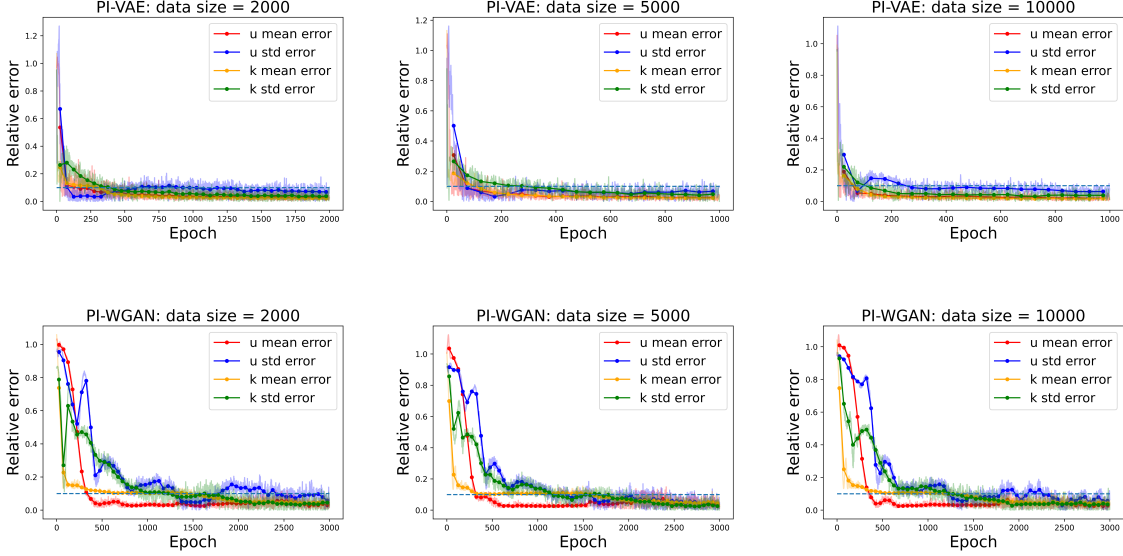


Figure 13: Relative error of QoIs versus training epoch of PI-VAE (top) and PI-WGANs (bottom) using different training data size 2000 (left), 5000(mid), and 10000(right) are shown.

Table 1: Comparison of converge time of different models(forward problem)

model	data size	converge epoch	time (sec)	model	data size	converge epoch	time	improvement
PI-VAE	2000	1300	338	PI-WGAN	2000	2500	1550	4.58x
PI-VAE	5000	400	260	PI-WGAN	5000	1800	1143	4.40x
PI-VAE	10000	300	390	PI-WGAN	10000	2400	1560	4.00x

Training is performed on an NVIDIA P100 GPU using the batch size of 1000. We need 0.62s for one training epoch of PI-WGAN while 0.13s for updating network parameters of PI-VAE using one mini-batch, which is consistent with our estimation. Based on the criterion mentioned before to evaluate the necessary training epoch of the model, we observe that at least 75% of the computational cost can be saved if PI-VAE is used instead of PI-WGAN to solve this SDE.

For accuracy evaluation, we use the approximation errors obtained in the last 100 training epochs as the criterion to evaluate the accuracy of the models. Figure 14 shows the mean and standard deviation (std) of the QoIs calculate for different models. It can be seen that even though PI-WGAN has better accuracy in estimating the standard deviation, it offers lower accuracy in estimating the mean. Among the PI-VAE models, by varying the dimension of latent space in PI-VAE-1 to PI-VAE-3, we observe that the accuracy in the standard deviation estimates increase if we increase the dimension of the latent space, since more latent variables allow for more information of the high-dimensional input data to be encoded. This accuracy improvement is not significant beyond the dimensionality of 4. This is because higher dimension latent space is more difficult to sample, leading to a discrepancy between random samples and the prior distribution of latent space. It can also be seen from the figure, by comparing PI-VAE-2, PI-VAE-4, and PI-VAE-5, that larger number of training data also contributes to higher accuracy, although the effect is not as significant as that seen for different latent space dimensions. As shown in Figure 14, the Monte Carlo Simulation (MCS) approach, which involves random sampling of the full trajectory, is the most accurate way to assess the statistics of QoIs, but it requires a large number of samples along the trajectory and is not practical in real-world applications.

Next, we use the trained PI-VAE-2 model to further evaluate the reproduced distribution for the process over the whole domain and also at one particular location. Specifically, in Figure 15 we show satisfactory accuracy in estimating the

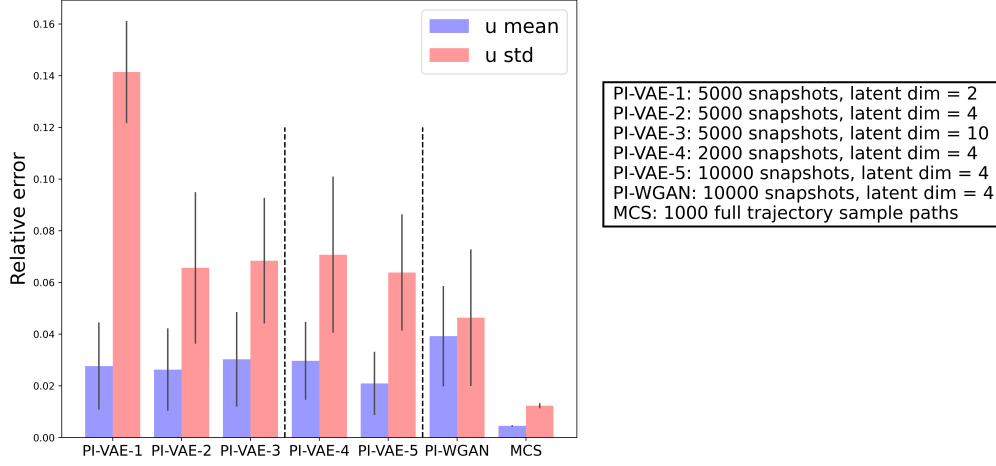


Figure 14: Relative error of QoIs of different model for the forward problem is shown in this figure. Effect of number of snapshots and latent dimension on the accuracy of our method are shown in case 1 to case 5. Results of PI-WGANs and full trajectory sample paths in case 6 and case 7 are shown for comparison.

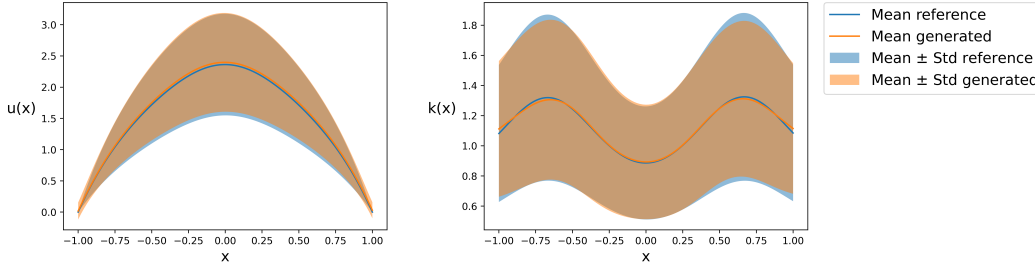


Figure 15: Mean and variance estimate of QoIs using trained model in case 2 are shown. Blue line represent the reference mean and blue shade area represents the reference standard deviation. In contrast, orange line and orange shade area represent the mean and std of the reconstructed samples, respectively.

mean response and the 68% confidence interval. The reference responses is calculated by obtaining another 10000 full trajectory sample paths.

Also, the stochastic response  $x = 0$  is selected as another QoI. The reconstructed response is obtained by drawing 10,000 samples of latent variables and a feeding them to the trained decoder to generate data points at  $x = 0$ . It can be seen in Figure 16, that the probability density functions (PDFs) of the reconstructed response is very close to that of the ground truth.

### 4.3 Inverse and mixed problems

In this section, using the SDE formulation of Equation 6, we evaluate the proposed PI-VAE method in solving a range of problems, from forward problems to inverse problems and mixed problems in between. We consider  $k(x, \omega)$  and  $f(x, \omega)$  to be independent processes, and we study the following four problems each with a different sensor placement scenario. The setting of all scenarios are shown in Table 2

We use 5,000 snapshots as training set with a batch size of 1000. We select the 100 models from the last 100 epochs of the training to generate the reconstructed sample paths. The mean and standard deviation of  $k(x, \omega)$  and  $u(x, \omega)$  are still selected as QoIs and are shown in Figure 17 for different trained models. In the case of mixed problems, by comparing the results from PI-WGAN-2 and PI-VAE-2, that our proposed PI-VAE method is slightly more accurate. In

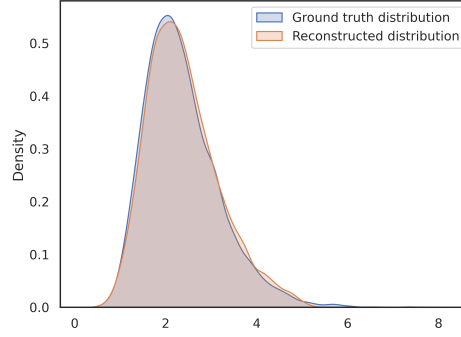


Figure 16: Probability density distribution of the reconstructed samples and ground true samples of  $u(x, \omega)$  at the coordinate of  $x = 0$  are shown.

Table 2: Number of sensors used in different cases

Case	Problem type	k-sensor	u-sensor non-boundary	u-sensor boundary	f-sensor
PI-VAE-1	Inverse problem	1	15	2	21
PI-VAE-2	Mixed problem	5	9	2	21
PI-VAE-3	Mixed problem	11	3	2	21
PI-VAE-4	Forward problem	17	0	2	21

solving inverse problem, by comparing the results of PI-WGAN-1 against PI-VAE-1, one can see that PI-VAE offers significantly higher accuracy levels at a much shorter training time. It can also be seen the errors in estimating  $k(x, \omega)$  statistics will decrease as the problem setup is changed from an inverse problem setting to a forward problem setting, by exploiting more information from  $k(x, \omega)$ .

Figure 18 shows approximation errors versus training epoch for the inverse problem (scenario 1) and a mixed problem (scenario 2) settings. It can be seen that PI-VAE consistently converges much faster than PI-WGAN in inverse and mixed problems. In these experiments, we observed over 80% of computational time saving when PI-VAE replaced PI-WGAN in solving these two cases.

#### 4.4 High-dimensional problems

In this section, we consider the same SDE formulation of Equation 6, now with a *high-dimensional* Gaussian process representation either for the coefficient term  $k(x, \omega)$  or forcing term  $f(x, \omega)$  and not both. This is to evaluate the ability of the PI-VAE method to solve the problems where there is an imbalance between the dimensionality of the coefficient and forcing terms. We do not analyze the case where both  $k(x, \omega)$  and  $f(x, \omega)$  are high-dimensional, as in those cases we to place more sensors to measure these processes. The stochastic processes are formulated as:

$$\begin{aligned}
 k(x) &= \exp\left[\frac{1}{5} \sin\left(\frac{3\pi}{2}(x+1)\right) + \hat{k}(x)\right], \\
 \hat{k}(x) &\sim \mathbf{GP}\left(0, \frac{4}{25} \exp\left(-\frac{1}{a^2}(x-x')^2\right)\right), \\
 f(x) &\sim \mathbf{GP}\left(\frac{1}{2}, \frac{9}{400} \exp\left(-\frac{1}{b^2}(x-x')^2\right)\right),
 \end{aligned} \tag{8}$$

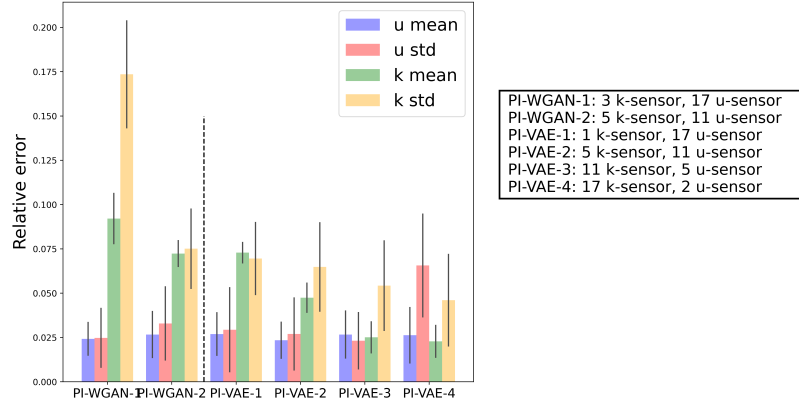


Figure 17: Relative error of QoIs of different model for the mixed problem and the inverse problem is shown in this figure. Results of PI-WGANs and in case 1 and case 2 are shown for comparison.

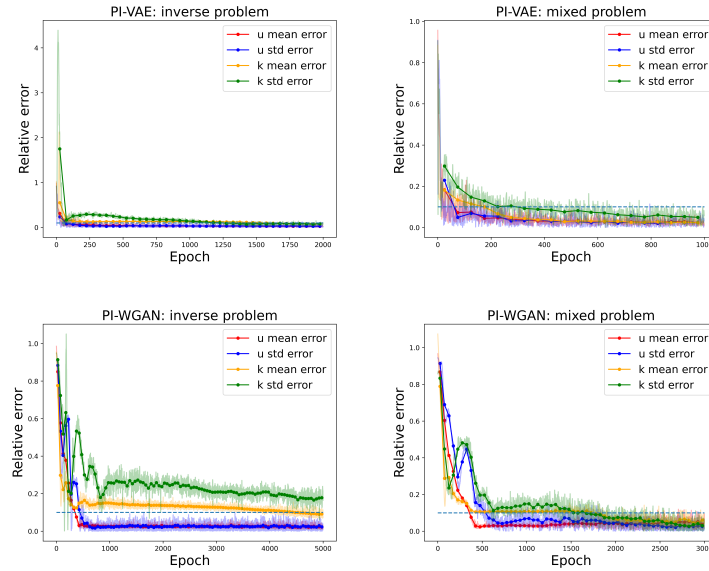


Figure 18: Relative error of QoIs versus training epoch of PI-VAE (top) and PI-WGANs (bottom) of different cases: inverse problem (left), mixed problem (right) are shown.

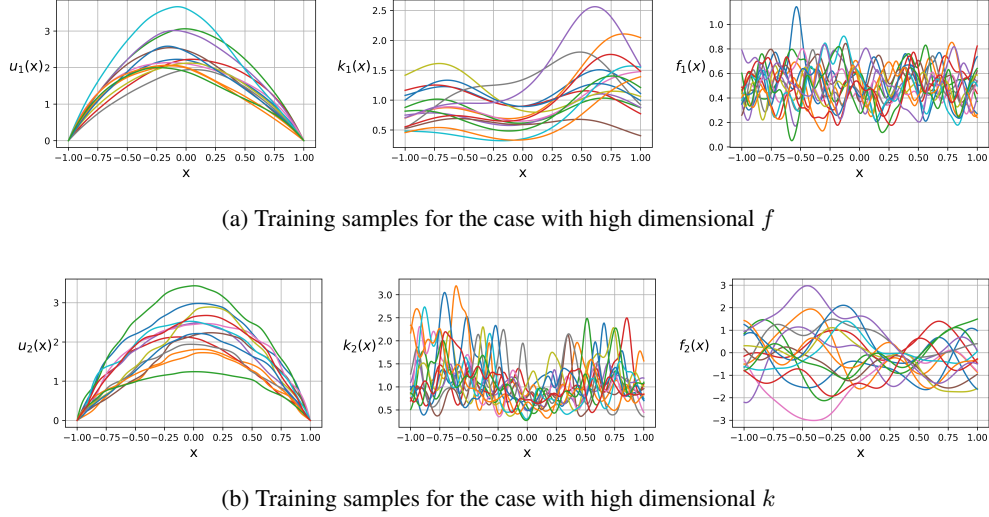


Figure 19: Samples of response  $u(x)$ ,  $k(x)$  and  $f(x)$  used in the training. The index 1 refers to the case where  $f$  has a high-dimensional representation and  $k$  a low dimensional representation. Index 2 refers to the case where the process  $k$  is high-dimensional and  $f$  is low-dimensional.

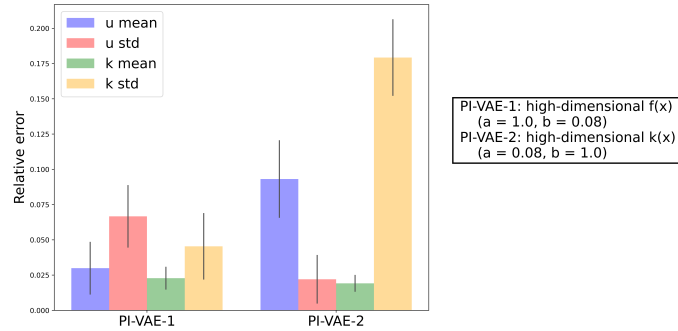


Figure 20: Relative error of QoIs of different model for the high-dimensional  $k(x, \omega)$  problems (left) and high-dimensional  $f(x, \omega)$  problems (right) is shown.

where  $a$  and  $b$  are the kernel length scales of the stochastic processes  $k(x, \omega)$  and  $f(x, \omega)$ , respectively, with the default values  $a = 1$  and  $b = 1$ . A process is transformed to a high-dimensional process by changing its length scale to the small value of 0.08. The sample paths of solutions  $u_1(x, \omega)$  and  $u_2(x, \omega)$ , respectively corresponding to high-dimensional  $f(x, \omega)$ ,  $k(x, \omega)$ , are shown in Figure 19 together with the sample paths of  $f(x, \omega)$ ,  $k(x, \omega)$ . The number of sensors for a high-dimensional processes, whether  $k(x, \omega)$  or  $f(x, \omega)$ , is set to be 51 to capture sufficient information. When a low dimensional process is concerned, the number of sensors remains the same as previous setting (i.e. 17  $k$ -sensors, 21  $f$ -sensors, and 2  $u$ -sensors on the boundary).

We use 5,000 snapshots as the training data, and we set the batch size to be 1,000. We train PI-VAE for 1,000 training epochs and train PI-WGAN for 5,000 training epochs, indicating that parameters of the decoder of PI-VAE and the generator of PI-WGAN go through the same number of parameter updates. It should be noted that solving the problem with high-dimensional  $k(x)$  is more difficult than that with high-dimensional  $f(x)$ , since the derivative information of  $k(x)$  is incorporated in the training loss. It can be seen in Figure 20 that we can obtain much higher accuracy in the case of high-dimensional  $f(x, \omega)$ . Also, the results from the high-dimensional  $f(x, \omega)$  case is even more accurate compared to the results from low dimensional processes, mainly because more sensor information is obtained from the external forcing term. Except for the variance estimation of  $k(x, \omega)$ , all QoIs satisfy the desired accuracy level.



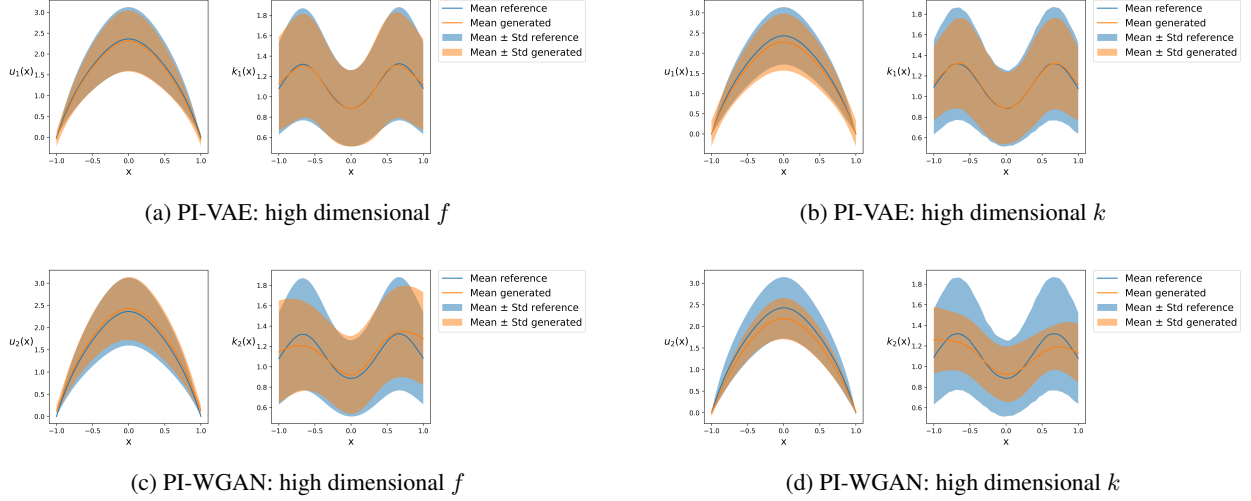


Figure 21: Mean and variance estimate of QoIs using trained model of PI-VAE (top) and PI-WGAN (bottom) for high-dimensional  $k(x, \omega)$  problems (left) and high-dimensional  $f(x, \omega)$  problems (right) are shown. Blue line represent the reference mean and blue shade area represents the reference standard deviation. In contrast, orange line and orange shade area represent the mean and std of the reconstructed samples, respectively.

In comparison with PI-WGAN, Figure 21 shows that PI-VAE offers significantly better performance than PI-WGAN, especially in the problem with high-dimensional  $k(x, \omega)$ . The disadvantage of PI-WGAN can be explained to be due to the gradient penalty [41] in the loss function that constrains the gradient magnitude of the trained neural networks. In the case of high-dimensional  $k(x, \omega)$ , if we constrain the norm of the gradient, we cannot approximate  $k(x, \omega)$  well enough even with sufficiently large number of  $k$ -sensors. In particular, we observed that the estimation error for  $k$  obtained from PI-VAE over the range of  $x$  is at most 18%, while PI-WGAN has estimation errors above 50%.

## 5 Conclusion

In this paper, we focused on solving problems governed by PDEs where the governing differential equation is known to us while information about the solution or the parameters of the differential equation are either unavailable, or partially available from sparse measurement sensors. We proposed PI-VAE; a new class of physics-informed Neural networks to efficiently solve these problems using inspirations from variational autoencoders. In comparison with PI-WGAN, which is another generative approach for solving similar problems, the presented results in this paper show that our method can obtain satisfactory accuracy with less training time in most of the studied cases. Furthermore, the presented numerical results showed our method was more effective in forward, inverse and mixed problems, and also problems where random parameters of the governing physics are represented by a high-dimensional stochastic model.

However, limitations exist in our method. First of all, the accuracy of the loss function in our training depends on the size of the mini-batch. In particular, in high-dimensional differential equations where we need to differentiate the neural network many times, the GPU constraint will require us to use a small batch size in the training. Also, we now assume that all sensors will give us accurate information while, in practice, measurement noises exist and should be accounted for. Finally, even though our method achieved better accuracy levels compared to PI-WGAN in all the cases, in a few of numerical experiments presented in the paper, the performance levels can still be improved in further research extensions. Future research directions also include quantifying the the sensor uncertainty as well as the modeling uncertainty into the framework.

## References

- [1] Carlos A Braumann. *Introduction to stochastic differential equations with applications to modelling in biology and finance*. John Wiley & Sons, 2019.

- [2] George Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.
- [3] Roger G Ghanem and Pol D Spanos. *Stochastic finite elements: a spectral approach*. Courier Corporation, 2003.
- [4] Dongbin Xiu and George Em Karniadakis. Modeling uncertainty in flow simulations via generalized polynomial chaos. *Journal of computational physics*, 187(1):137–167, 2003.
- [5] Howard C Elman, Christopher W Miller, Eric T Phipps, and Raymond S Tuminaro. Assessment of collocation and galerkin approaches to linear diffusion equations with random data. *International Journal for Uncertainty Quantification*, 1(1), 2011.
- [6] Nora Lüthen, Stefano Marelli, and Bruno Sudret. Sparse polynomial chaos expansions: Literature survey and benchmark. *SIAM/ASA Journal on Uncertainty Quantification*, 9(2):593–649, 2021.
- [7] Negin Alemazkoor and Hadi Meidani. Divide and conquer: An incremental sparsity promoting compressive sampling approach for polynomial chaos expansions. *Computer Methods in Applied Mechanics and Engineering*, 318:937–956, 2017.
- [8] MWMG Dissanayake and N Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.
- [9] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.
- [10] Dimitris C Psychogios and Lyle H Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511, 1992.
- [11] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.
- [12] E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.
- [13] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.
- [14] Samuel Rudy, Alessandro Alla, Steven L Brunton, and J Nathan Kutz. Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, 18(2):643–660, 2019.
- [15] Mohammad Amin Nabian and Hadi Meidani. A deep learning solution approach for high-dimensional random differential equations. *Probabilistic Engineering Mechanics*, 57:14–25, 2019.
- [16] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.
- [17] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deepnets. *arXiv preprint arXiv:2103.10974*, 2021.
- [18] Dongkun Zhang, Lu Lu, Ling Guo, and George Em Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.
- [19] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.
- [20] Liu Yang, Sean Treichler, Thorsten Kurth, Keno Fischer, David Barajas-Solano, Josh Romero, Valentin Churavy, Alexandre Tartakovsky, Michael Houston, Mr Prabhat, et al. Highly-ccalable, physics-informed gans for learning solutions of stochastic pdes. In *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, pages 1–11. IEEE, 2019.
- [21] Vandana Kushwaha, GC Nandi, et al. Study of prevention of mode collapse in generative adversarial network (gan). In *2020 IEEE 4th Conference on Information & Communication Technology (CICT)*, pages 1–6. IEEE, 2020.
- [22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

- [23] Merlijn Blaauw and Jordi Bonada. Modeling and transforming speech using variational autoencoders. *Morgan N, editor. Interspeech 2016; 2016 Sep 8-12; San Francisco, CA.[place unknown]: ISCA; 2016. p. 1770-4., 2016.*
- [24] Jay A Hennig, Akash Umakantha, and Ryan C Williamson. A classifying variational autoencoder with application to polyphonic music generation. *arXiv preprint arXiv:1711.07050*, 2017.
- [25] Ivan P Yamshchikov and Alexey Tikhonov. Music generation with variational recurrent autoencoder supported by history. *SN Applied Sciences*, 2(12):1–7, 2020.
- [26] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.
- [27] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013.
- [28] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.
- [29] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.
- [30] Danica J Sutherland, Hsiao-Yu Tung, Heiko Strathmann, Soumyajit De, Aaditya Ramdas, Alex Smola, and Arthur Gretton. Generative models and model criticism via optimized maximum mean discrepancy. *arXiv preprint arXiv:1611.04488*, 2016.
- [31] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schölkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.
- [32] Giorgio Patrini, Rianne van den Berg, Patrick Forre, Marcello Carioni, Samarth Bhargav, Max Welling, Tim Genewein, and Frank Nielsen. Sinkhorn autoencoders. In *Uncertainty in Artificial Intelligence*, pages 733–743. PMLR, 2020.
- [33] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.
- [34] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.
- [35] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. *arXiv preprint arXiv:1705.08584*, 2017.
- [36] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. 2017.
- [37] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- [38] Yujia Xie, Xiangfeng Wang, Ruijia Wang, and Hongyuan Zha. A fast proximal point method for computing exact wasserstein distance. In *Uncertainty in Artificial Intelligence*, pages 433–453. PMLR, 2020.
- [39] Sashank Reddi, Aaditya Ramdas, Barnabás Póczos, Aarti Singh, and Larry Wasserman. On the high dimensional power of a linear-time two sample test under mean-shift alternatives. In *Artificial Intelligence and Statistics*, pages 772–780. PMLR, 2015.
- [40] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [41] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.