Universal Functional Regression with Neural Operator Flows

Anonymous authors
Paper under double-blind review

Abstract

Regression on function spaces is typically limited to models with Gaussian process priors. We introduce the notion of universal functional regression, in which we aim to learn a prior distribution over non-Gaussian function spaces that remains mathematically tractable for functional regression. To do this, we develop Neural Operator Flows (OpFlow), an infinite-dimensional extension of normalizing flows. OpFlow is an invertible operator that maps the (potentially unknown) data function space into a Gaussian process, allowing for exact likelihood estimation of functional point evaluations. OpFlow enables robust and accurate uncertainty quantification via drawing posterior samples of the Gaussian process and subsequently mapping them into the data function space. We empirically study the performance of OpFlow on regression and generation tasks with data generated from Gaussian processes with known posterior forms and non-Gaussian processes, as well as real-world earthquake seismograms with an unknown closed-form distribution.

1 Introduction

The notion of inference on function spaces is essential to the physical sciences and engineering. In particular, it is often desirable to infer the values of a function everywhere in a physical domain given a sparse number of observation points. There are numerous types of problems in which functional regression plays an important role, such as inverse problems, time series forecasting, data imputation/assimilation. Functional regression problems can be particularly challenging for real world datasets because the underlying stochastic process is often unknown.

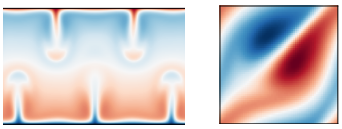
Much of the work on functional regression and inference has relied on Gaussian processes (GPs) (Rasmussen & Williams, 2006), a specific type of stochastic process in which any finite collection of points has a multivariate Gaussian distribution. Some of the earliest applications focused on analyzing geological data, such as the locations of valuable ore deposits, to identify where new deposits might be found (Chiles & Delfiner, 2012). GP regression (GPR) provides several advantages for functional inference including robustness and mathematical tractability for various problems. This has led to the use of GPR in an assortment of scientific and engineering fields, where precision and reliability in predictions and inferences can significantly impact outcomes (Deringer et al., 2021; Aigrain & Foreman-Mackey, 2023). Despite widespread adoption, the assumption of a GP prior for functional inference problems can be rather limiting, particularly in scenarios where the data exhibit heavy-tailed or multimodal distributions, e.g. financial time series or environmental modeling. This underscores the need for models with greater expressiveness, allowing for regression on general function spaces, with data arising from potentially unknown stochastic processes and distributions (Fig. 1). We refer to such a regression problem as universal functional regression (UFR).

We hypothesize that addressing the problem of UFR requires a model with two primary components. First, the model needs to be capable of learning priors over data that live in function spaces. Indeed, there has been much recent progress in machine learning on learning priors on function spaces (Rahman et al., 2022a; Lim et al., 2023; Seidman et al., 2023; Kerrigan et al., 2023; Pidstrigach et al., 2023; Baldassari et al., 2023; Hagemann et al., 2023; Franzese et al., 2023). Second, the models need a framework for performing functional regression with the learned function space priors and learned likelihood–a component that is critically missing from the previous works on pure generative models described above. This includes the ability to compute the likelihood of an input data sample, which would allow for posterior estimation over the entire physical

domain. Addressing these challenges not only expands the models available for functional regression but also enhances our capacity to extract meaningful insights from complex datasets.

In this study, our contributions are as follows. We develop a viable formulation for UFR with a novel learnable bijective operator, OpFlow. This operator extends the classical normalizing flow, which maps between finite dimensional vector spaces, to the function space setting. OpFlow consists of an invertible neural operator, an invertible map between function spaces, trained to map the data point-process distribution to a known and easy-to-sample GP distribution. Given point value samples of a data function, OpFlow allows for computing the likelihood of the observed values, a principled property a priori held by GP. Using this property, we formally define the problem setting for UFR and develop an approach for posterior estimation with Stochastic Gradient Langevin Dynamics (SGLD) (Welling & Teh, 2011). Finally, we demonstrate UFR with OpFlow on a suite of experiments that use both Gaussian and non-Gaussian processes.

2 Related work



(a) (b)

Figure 1: Examples of data from non-Gaussian processes. (a) Temperature field from Rayleigh Bénard convection problem. (b) Vorticity field from Naiver-Stokes Equation.

Neural Operators. Neural Operator is a new paradigm in deep learning for learning maps between infinite-dimensional function spaces. Unlike traditional neural networks that primarily work with fixed-dimensional vectors, neural operators are designed to operate on functions, making them inherently suitable for a wide range of scientific computing and engineering tasks involving partial differential equations (PDEs) (Kovachki et al., 2023; Li et al., 2020). A key feature of neural operators is their discretization agnostic (resolution invariant) property (Kovachki et al., 2023), which means that a neural operator can learn from data represented on various resolutions and predict outcomes on yet new resolutions. This property is particularly valuable in applications involving the natural sciences, PDEs, and complex physical simulations where functional data may come from variable discretizations of meshes or sensors. Within the family of neural operators, Fourier Neural Operator (FNO) (Li et al., 2021) stands out for its quasi-linear time complexity by defining the integral kernel in Fourier space and has been applied to many problems in engineering and science (Azizzadenesheli et al., 2024). In the conventional view of neural operators, provided discretization and point evaluation of functions resembles approximation of function for which increasing the resolution is connected to improved approximation error in the integration and differentiation operators (Liu-Schiaffini et al., 2024). The proposed view in the current work provides a new perspective for which the point evaluations are instead considered as points samples of functions that are associated probably rather than finite bases and finite resolution approximation of a continuous function. Therefore, increasing the resolution is seen as providing more information for regression rather than a way to reduce, e.g., the approximation error in the integral operators used in neural operations.

Function space generative modeling with Neural Operators. Several discretization invariant generative models on function spaces have been recently developed using Neural Operators (Li et al., 2020; Kovachki et al., 2023). These models include Generative Adversarial Neural Operator (GANO) (Rahman et al., 2022a), Denoising Diffusion models (DDO) (Lim et al., 2023) and Variational Autoencoding Neural Operator (VANO) (Seidman et al., 2023). To be specific, GANO is the first proposed infinite-dimensional generative model that generalizes GAN, and learns probability measures on function spaces. However, GANO training relies on an adversarial game, which can be slow (Goodfellow et al., 2014; Rahman et al., 2022a); DDO is a resolution-invariant diffusion generative model for function spaces, leveraging denoising score matching within Hilbert spaces and employing infinite-dimensional Langevin dynamics for sampling. The inference of DDO requires a diffusion process, as with the finite-dimensional score-based model, which can be time-consuming (Song et al., 2021; Lim et al., 2023); VANO offers a functional variational objective (Kingma & Welling, 2022; Seidman et al., 2023), mapping input functional data to a finite-dimensional latent space through an encoder, which may be a sub-optimal formulation due to the fixed dimensionality of the latent space being decoupled from the discretization of the input function data.

All of these aforementioned models have exhibited superior performance over their finite-dimensional counterparts for functional data by directly learning mappings between function spaces, which show great potential

in addressing the challenges in science and engineering problems (Shi et al., 2024a;b; Yang et al., 2021; Azizzadenesheli et al., 2024). Compared to these existing infinite-dimensional generative models, OpFlow shows advantages in fast training, fast inference, bijective architecture, and providing precise likelihood estimation as summarized in Table 1. These advantages of OpFlow can be explained by: (i) The training and inference processes of OpFlow only involve likelihood estimation, and don't require an adversarial game or a diffusion process. (ii) OpFlow directly generalizes normalizing flows to function space without finite-dimensional simplicity, and enables precise likelihood estimation for functional point evaluations on irregular grids. (iii) As an invertible operator, OpFlow can be useful for tasks that require the bijectivity between function spaces (Zhou et al., 2023; Furuya et al., 2023).

Table 1: Comparison of OpFlow with other infinite-dimensional generative models

Models	Fast training	Fast inference	Bijective architecture	Exact likelihood estimation
GANO	✗	✓	✗	✗
DDO	✗	✗	✗	✗
VANO	✓	✓	✗	✗
OpFlow [Ours]	✓	✓	✓	✓

Neural Processes. Neural Process (NP) (Garnelo et al., 2018) leverages neural networks to address the constraints associated with GPs, particularly the computational demands and the restrictive Gaussian assumptions for priors and posteriors (Dutordoir et al., 2022). While NP aims to model functional distributions, several fundamental flaws suggest NP might not be a good candidate for learning function data (Rahman et al., 2022a; Dupont et al., 2022). First, NP lacks true generative capability; it focuses on maximizing the likelihood of observed data points through an amortized inference approach, neglecting the metric space of the data. This approach can misinterpret functions sampled at different resolutions as distinct functions, undermining the NP's ability to learn from diverse function representations. As pointed out in prior works (Rahman et al., 2022a), if a dataset comprises mainly low-resolution functions alongside a single function with a somewhat high resolution, NP only tries to maximize the likelihood for the point evaluation of the high-resolution function during training and ignores information from all other lower-resolution functions. A detailed experiment to show the failure of NP for learning simple function data can be found in Appendix A.1 of (Rahman et al., 2022a) . What's more, NP relies on an encoder to map the input data pairs to finite-dimensional latent variables, which are assumed to have Gaussian distributions (Garnelo et al., 2018), and then project the finite-dimensional vector to an infinite-dimensional space; this approach results in diminished consistency at elevated resolutions, and thus disables NP to scale to large datasets (Dupont et al., 2022). The other fundamental limitation is the Bayesian framework of NP is also defined on a set of points, rather than functions themselves. This method results in the dilution of prior information as the number of evaluated points increases, further constraining NP's ability to generalize from function data.

Normalizing Flows. Normalizing flows are a class of flow-based finite-dimensional generative models, that usually composed of a sequence of invertible transformations (Kingma & Dhariwal, 2018; Dinh et al., 2017; Chen et al., 2019). By gradually transforming a simple probability distribution into a more complex target distribution with the invertible architecture, normalizing flows enable exact likelihood evaluation, direct sampling, and good interpretability (Kobyzev et al., 2021) with applications from image and audio generations (Kingma & Dhariwal, 2018; Ping et al., 2020), reinforcement learning (Mazoure et al., 2020) to Bayesian inference (Gao et al., 2021; Whang et al., 2021). Despite the advantages of normalizing flows, they still face challenges with high-dimensional data due to computational complexity, memory demands, scalability, and cannot be directly used for zero-shot super-resolution (Lugmayr et al., 2020). Moreover, traditional normalizing flows are defined on a finite-dimensional spaces, and constrained to evaluating likelihood on fixed-size, regular grids. This inherent limits their applications in Bayesian inverse problems, where the data may not conform to fixed and regular grids (Dashti & Stuart, 2017). These limitations underline the need for a new generation of normalizing flows designed on function space, providing a solution that could potentially transcend the limitations of traditional models.

Nonparametric Bayesian and Gaussian Process Regression. Nonparametric Bayesian models are defined on an infinite-dimensional parameter space, yet they can be evaluated using only a finite subset of

parameter dimensions to explain observed data samples (Orbanz & Teh, 2010). These models provide a flexible regression framework by offering analytically tractable posteriors and enable adjusting their complexity by allowing the model size to grow when more observations are available. Further, nonparametric Bayesian leverages a rich theoretical foundation of parametric Bayesian modeling and can be evaluated using common inference techniques like Markov Chain Monte Carlo (MCMC) and variational inference, etc. Despite the great advantages of the nonparametric Bayesian models, such models heavily depend on the chosen prior, which could restrict their adaptability and scalability in real-world complex scenarios (Barry, 1986; Wasserman, 2006). The emergence of learning-based Bayesian models, propelled by advancements in deep learning, offers a potential solution by marrying the flexibility and scalability of neural networks with Bayesian principles; these modern approaches learn directly from data rather than rely on the chosen priors.

Within the family of nonparmateric Bayesian models, GPR is distinguished for its widespread applications, characterized by its analytical posteriors within the Bayesian framework (Rasmussen & Williams, 2006). GPR offers a robust framework for capturing uncertainties in predictions given noisy observations. However, GPR assumes data can be modeled using a Gaussian process, both the prior and posterior should be Gaussian. Such assumption, while simplifying the analytical process, might not always align with the complex nature, thus the universal applicability of GPR is constrained by the diversity and complexity of real-world datasets.

3 Preliminaries

Invertible operators on function spaces. Consider two Polish function spaces, denoted $\mathcal{A}$ and $\mathcal{U}$, such that for any $a \in \mathcal{A}, a : \mathcal{D}_{\mathcal{A}} \to \mathbb{R}^{d_{\mathcal{A}}}$, and for any $u \in \mathcal{U}, u : \mathcal{D}_{\mathcal{U}} \to \mathbb{R}^{d_{\mathcal{U}}}$. Here, $\mathcal{U}$ represents the space of the data, whereas $\mathcal{A}$ represents the latent space. Let $\mathbb{P}_{\mathcal{U}}$ and $\mathbb{P}_{\mathcal{A}}$ be probability measures defined on $\mathcal{U}$ and $\mathcal{A}$, respectively. A finite set of observations $u|_D$ is collected on the discretized domain $D \subset \mathcal{D}_{\mathcal{U}}$, where D is the collection of co-location positions $\{x_i\}_{i=1}^{l}, x_i \in \mathcal{D}_{\mathcal{U}}$. Note that in practice, $u \in \mathcal{U}$ is usually not observable in its entirety over $\mathcal{D}_{\mathcal{U}}$, and is only observed on, e.g., given discretizations or meshes. For learning purposes, independent and identically distributed samples of $\{u_j\}_{j=1}^{m} \sim \mathbb{P}_{\mathcal{U}}$ are accessed on the discretization points, $\{D_j\}_{j=1}^{m}$, constituting a learning dataset of $\{u_j|_{D_j}\}_{j=1}^{m}$. The point evaluation of the function draws, i.e., $u \in \mathcal{U}$, can also be interpreted as a stochastic process where the process is discretely observed. The behavior of the stochastic process is characterized by the finite-dimensional distributions, such that for any discretization $D^{\dagger}$, $p(u|_{D^{\dagger}})$ denotes the probability distribution of points evaluated on a collection $D^{\dagger}$. In the UFR setting, each data sample u_j is provided on a specific collection of domain points D_j, and for the sake of simplicity of notation, we consider all the collected domain points to be the same and referred to by D.

In this work, we take samples of functions in $\mathcal{A}$, i.e., $a \in \mathcal{A}$ to be drawn from a known GP, and consequently, the evaluation of $a_j|_{D_{\mathcal{A}}}$ is a Gaussian random variable equipt with a Gaussian probability measure. Under these construction, and for a suitable Gaussian process on $\mathcal{A}$, we may establish an invertible (bijective) operator between $\mathcal{A}$ and $\mathcal{U}$[1]. We denote the forward operator as $\mathcal{G} : \mathcal{A} \to \mathcal{U}$, such that for any a_j, $\mathcal{G}(a_j) = u_j$, and its corresponding inverse operator $\mathcal{G}^{-1}$ as $\mathcal{F}$, where $\mathcal{F} = \mathcal{G}^{-1}$ and $\mathcal{F} : \mathcal{U} \to \mathcal{A}$, such that for any u_j, $\mathcal{F}(u_j) = a_j$. We aim to approximate $\mathcal{F}$ and correspondingly its inverse $\mathcal{G}$ with an invertible learnable model $\mathcal{F}_\theta \approx \mathcal{F}$, and $\mathcal{G}_\theta = {\mathcal{F}_\theta}^{-1}$, from a finite collection of observations $\{u_j|_D\}_{j=1}^{m}$. Finally, to have well-defined invertible maps between the two measures, we assume absolute continuity of measures on $\mathcal{A}$ and $\mathcal{U}$ with respect to each other. Given the GP nature of the measure on $\mathcal{A}$, such continuity is obtained by adding a small amount of GP noise to the measure on $\mathcal{U}$. A similar procedure is envisioned and incorporated by adding Gaussian noise to images in the prior normalizing flow in final dimensional spaces (Huang et al., 2020; Shorten & Khoshgoftaar, 2019).

4 Neural Operator Flow

4.1 Invertible Normalized Architecture

We introduce the Neural Operator Flow (OpFlow), an innovative framework that extends the principle of the finite-dimensional normalizing flow into the realm of infinite-dimensional function spaces. The architecture is shown schematically in Fig. 2. OpFlow retains the invertible structure of normalizing flows, while directly

[1]We only require homeomorphism between $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{U}}$ where the point alignment is through the underlying homeomorphic map, and use $D_{\mathcal{A}}$ to represent the discretized domain of $\mathcal{D}_{\mathcal{A}}$. For simplicity, we consider the case where $\mathcal{D}_{\mathcal{U}} = \mathcal{D}_{\mathcal{A}}$ and the trivial identity map is the homeomorphic map in our experimental study.

operating on function spaces. It allows for exact likelihood estimation for point estimated functions. OpFlow is composed of a sequence of layers, each containing the following parts:

Actnorm. Channel-wise normalization with trainable scale and bias parameters is often effective in facilitating affine transformations in normalizing flow neural network training (Kingma & Dhariwal, 2018). We implement a function space analog by representing the scale and bias as learnable constant-valued functions. Specifically, let $(v')^i$ denote the input function to the ith actnorm layer of OpFlow and let s^i_θ and b^i_θ represent the learnable constant-valued scale and bias functions, respectively, in this layer. The output of the actnorm layer is then given by $v^i = s^i_\theta \odot (v')^i + b^i_\theta$, where $\odot$ denotes pointwise multiplication in the function space.

Domain and codomain partitioning. The physical domain refers to the space in which the data resides, such as the time domain for 1D data or the spatial domain for 2D data. Conversely, the channel domain is defined as the codomain of the function data, such as the 3 channels in RGB images or temperature, velocity vector, pressure, and precipitation in weather forecast (Pathak et al., 2022; Hersbach et al., 2020). In normalizing flows, a bijective structure requires dividing the input domain into two segments (Dinh et al., 2017; Kingma & Dhariwal, 2018). Based on the method of splitting the data domain, we propose two distinct OpFlow architectures.

(i) Domain partitioning: In this architecture, we apply checkerboard masks to the physical domain, a technique widely used in existing normalizing flow models (Dinh et al., 2017; Kingma & Dhariwal, 2018) which we extend to continuous domains. Our experiments in the following sections reveal that the domain decomposed OpFlow is efficient and expressive with a minor downside in introducing checkerboard pattern artifacts in zero-shot super-resolution tasks.

(ii) Codomain partitioning: This approach partitions the data along the codomain of the input function data, while maintaining the integrity of the entire physical domain. We show that, unlike its domain decomposed counterpart, the codomain OpFlow does not produce artifacts in zero-shot super-resolution experiments. However, it exhibits a lower level of expressiveness compared to the domain decomposed OpFlow.

Affine coupling. Let v^i represent the input function data to the ith affine coupling layer of OpFlow. Then, we split v^i either along the physical domain or codomain, and we have two halves $h^i_1, h^i_2 = \mathbf{split}(v^i)$. Let $\mathcal{T}^i_\theta$ denote the ith affine coupling layer of OpFlow, where $\mathcal{T}^i_\theta$ is a FNO layer (Li et al., 2021), which ensures the model is resolution agnostic. Then, we have $\log(s^i), b^i = \mathcal{T}^i_\theta(h^i_1)$, where s^i and b^i are scale and shift functions, respectively, output by the ith affine coupling layer. From here, we update h^i_2 through $(h')^i_2 = s^i \odot h^i_2 + b^i$. Finally, we output the function $(v')^{i+1}$ for the next layer where $(v')^{i+1} = \mathbf{concat}(h^i_1, (h')^i_2)$. The **concat** (concatenation) operation corresponds to the reversing the split operation, and is either in the physical domain or codomain.

Inverse process. The inverse process for the $i$the layer of OpFlow is summarized in Algorithm 3. First, we split the input function $(v')^{i+1}$ with $h^i_1, (h')^i_2 = \mathbf{split}((v')^{i+1})$. Then the inverse process of the affine coupling layer also takes h^i_1 as input, and generates the scale and shift functions $\log(s^i), b^i = \mathcal{T}^i_\theta(h^i_1)$. Thus, both forward and inverse processes utilize the h^i_1 as input for the affine coupling layer, which implies that the scale and shift functions derived from the forward process and inverse process are identical from the affine coupling layer. Then we can reconstruct h^i_2 and v^i through $h^i_2 = ((h')^i_2 - b^i)/s^i$; $v^i = \mathbf{concat}(h^i_1, h^i_2)$. Finally, we have the inverse of the actnorm layer with $(v')^i = (v^i - b^i_\theta)/s^i_\theta$. Similar inverse processes of normalizing flows are described in (Kingma & Dhariwal, 2018; Dinh et al., 2017).

4.2 Training

Since OpFlow is a bijective operator, we only need to learn the inverse mapping $\mathcal{F}_\theta : \mathcal{U} \to \mathcal{A}$, as the forward operator is immediately available. We train OpFlow in a similar way to other infinite-dimensional generative models (Rahman et al., 2022a; Seidman et al., 2023; Lim et al., 2023) with special addition of domain alignment. The space of functions for $\mathcal{A}$ is taken to be drawn from a GP, which allows for exact likelihood estimation and efficient training of OpFlow by minimizing the negative log-likelihood. Since OpFlow is discretization agnostic (Rahman et al., 2022a; Kovachki et al., 2023), it can be trained on various discretizations of $\mathcal{D}_\mathcal{U}$ and be later applied to a new set of discretizations, and in the limit, to the whole of

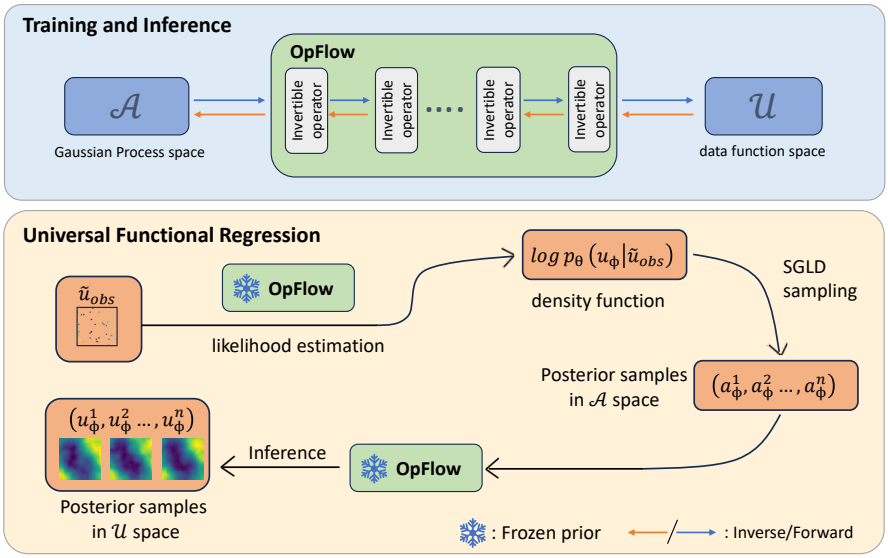


Figure 2: Model architecture of OpFlow, OpFlow is composed of a list of invertible operators. For the universal function regression task, OpFlow is the learnt prior, which is able to provide exact likehood estimation for function point evaluation. $\tilde{u}_{obs}$ is the noisy observation, u_ϕ is the posterior function of interest and $u_\phi = \mathcal{G}_\theta(a_\phi)$, where $\mathcal{G}_\theta$ is the learnt forward operator.

$\mathcal{D}_\mathcal{U}$ (Kovachki et al., 2023). We now define the training objective $\mathcal{L}$ for model training as follows,

$$\mathcal{L} = -\min_{\theta\in\Theta} \mathbb{E}_{u\sim\mathbb{P}_\mathcal{U}}[\log p_\theta(u|_D)], \quad where, \quad \log p_\theta(u|_D) = \log p(a|_{D_\mathcal{A}}) + \sum_{i=1}^{t} \log|\det(\frac{\partial(v^i|_{D^i_\mathcal{V}})}{\partial(v^{i-1}|_{D^{i-1}_\mathcal{V}})})|. \quad (1)$$

The inverse operator is composed of t invertible operator layers with $\mathcal{F}_\theta := \mathcal{F}^t_\theta \circ \mathcal{F}^{t-1}_\theta \cdots \mathcal{F}^0_\theta$, and gradually transforms v^0 to $v^1, \cdots v^{t-1}, v^t$, where $v^0 = u, v^t = a$ and the alignment naturally holds within the domains associated with $v^0, v^1, \cdots v^t$ as each layer of $\mathcal{F}_\theta$ is an invertible operator. $\mathcal{V}^i$ is the collection of positions for function v^i with $D^0_\mathcal{V} = D$, $D^t_\mathcal{V} = D_\mathcal{A}$, and each data point has its own discretization D [2]. The likelihood objective directly measures the distance between the probability measure $\mathbb{P}_\mathcal{A}$ and the push-forward probability measure $\mathcal{F}_\theta\sharp\mathbb{P}_\mathcal{U}$ of $\mathbb{P}_\mathcal{U}$, under the inverse operator $\mathcal{F}_\theta$.

Although the framework of OpFlow is well-defined, the negative log-likelihood for high-dimensional data may suffer from instabilities during training, leading to a failure to recover the true distribution.Through our experiments, we found that relying solely on likelihood objective in Eq 1 is insufficient for successfully training OpFlow. To address this, we introduce a regularization term, to ensure that OpFlow learns the true probability measure by helping to stabilize the training process and leading to faster convergence. Such regularization is inspired by the infinite-dimensional Wasserstein loss used in (Rahman et al., 2022a). Unlike the implicit Wasserstein loss provided by the discriminator in GANO, we could potentially have a closed-form expression of the infinite-dimensional Wasserstein loss in the context of OpFlow. This is due to the learning process mapping function data onto a well-understood GP. The inherent properties of Gaussian Processes facilitate measuring the distance between the learned probability measure and the true probability measure, which is explained into detail in the subsequent sections.

[2]Unlike finite-dimensional normalizing flows on fixed-size and regular grids, OpFlow can provide likelihood estimation $\log p_\theta(u|_D)$ on arbitrary grids D, which represents a potentially irregular discretization of $\mathcal{D}_\mathcal{U}$. To compute $\log p_\theta(u|_D)$, we need the homeomorphic mapping between $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{U}$ to find $D_\mathcal{A}$, corresponding to D. The density function in Eq. 1 holds for new points of evaluation. Our implementation of OpFlow incorporates FNO that requires inputs on regular grids. Consequently, although OpFlow is designed to handle functions for irregular grids, our specific implementation of OpFlow only allows for likelihood estimation on regular girds with arbitrary resolution.

A common way for measuring the distance between two probability measures is square 2-Wasserstein (Mallasto & Feragen, 2017) distance, which is defined as

$$W_2^2(\mathbb{P}_{\mathcal{A}}, \mathcal{F}_\theta \sharp \mathbb{P}_{\mathcal{U}}) = \inf_{\pi \in \Pi} \int_{\mathcal{A}\times\mathcal{A}} d_2^2(a_1, a_2) d\pi, \ (a_1, a_2) \in \mathcal{A} \times \mathcal{A}, \tag{2}$$

where Π is a space of joint measures defined on $\mathcal{A} \times \mathcal{A}$ such that for any $\pi \in \Pi$, the margin measures of $\pi(a_1, a_2)$ on the first and second arguments are $\mathbb{P}_{\mathcal{A}}$ and $\mathcal{F}_\theta \sharp \mathbb{P}_{\mathcal{U}}$, and d_2 is a metric induced by the L^2 norm. By construction, $\mathcal{A} \sim \mathcal{GP}$ and if sufficiently trained, $\mathcal{F}_\theta \sharp \mathcal{U} \sim \mathcal{GP}$. We can thus further simplify Eq. 2, which is named as F^2ID score by Rahman et al. (2022b), and is defined as:

$$W_2^2(\mathbb{P}_{\mathcal{A}}, \mathcal{F}_\theta \sharp \mathbb{P}_{\mathcal{U}}) = d_2^2(m_1, m_2) + Tr(\mathcal{K}_1 + \mathcal{K}_2 - 2(\mathcal{K}_1^{\frac{1}{2}} \mathcal{K}_2 \mathcal{K}_1^{\frac{1}{2}})^{\frac{1}{2}}). \tag{3}$$

for which $\mathbb{P}_{\mathcal{A}}$ is chosen to be $\mathcal{GP}_1(m_1, \mathcal{K}_1)$ and $\mathcal{GP}_2(m_2, \mathcal{K}_2)$ is the GP fit to the push-forward measure $\mathcal{F}_\theta \sharp \mathbb{P}_{\mathcal{U}}$. The $m_1, \mathcal{K}_1$ are the mean function and covariance operator of $\mathcal{GP}_1(m_1, \mathcal{K}_1)$, whereas $m_2, \mathcal{K}_2$ are the mean function and covariance operator of $\mathcal{GP}_2(m_2, \mathcal{K}_2)$. We note that it is equivalent to say a function $f \sim \mathcal{GP}(m, k)$ or $f \sim \mathcal{GP}(m, \mathcal{K})$, where k is the covariance function and $\mathcal{K}$ is the covariance operator (Mallasto & Feragen, 2017). We take Eq. 3 as the regularization term. Furthermore, we inject noise with negligible amplitude that sampled from $\mathcal{GP}_1$ to the training dataset $\{u_j\}_{j=1}^m$. In this way, we can have the inversion map to be well-define and stable. From here, we can define a second objective function.

$$\mathcal{L} = -\min_{\theta \in \Theta} \mathbb{E}_{u \sim \mathbb{P}_u}[\log p_\theta(u|_D)] + \lambda W_2^2(\mathbb{P}_{\mathcal{A}}|_D, \mathcal{F}_\theta \sharp \mathbb{P}_{\mathcal{U}}|_D), \tag{4}$$

where λ is the weight for the regularization term. For the collections of position $D_{\mathcal{A}} = \{y_i\}_{i=1}^l, y_i \in \mathcal{D}_{\mathcal{A}}$, we select a finite set of orthonormal basis $\{e_1, e_2, ..e_l\}$ on the $L^2(\mathcal{D}_{\mathcal{A}})$ space such that $\|e_i\|_2 = \frac{1}{\sqrt{l}}, i \in \{1, ..l\}$. Thus the relationship between the covariance operator and the covariance function form can be expressed as:

$$< \mathcal{K} e_i, e_j >= \int_{\mathcal{D}_{\mathcal{A}}} \frac{1}{\sqrt{l}} k(y, y_i) e_i(y) dy = \frac{1}{l} k(y_i, y_j) \tag{5}$$

Thus, given the chosen set of orthonormal basis functions, the covariance matrix (covariance function in matrix form) K associated with the covariance operator $\mathcal{K}$ follows the relationship:

$$K[i, j] = l < \mathcal{K} e_i, e_j >= k(y_i, y_j) \tag{6}$$

Where, $K[i, j]$ is the (i_{th}, j_{th}) component of the covariance matrix K. Detailed explanation of the chosen basis function can be found in the Appendix A.1 in (Rahman et al., 2022b). Thus the matrix form of the Eq. 3 can be expressed as follows,

$$\frac{1}{l}\|h_1 - h_2\|_2^2 + \frac{1}{l} Tr(K_1 + K_2 - 2(K_1^{\frac{1}{2}} K_2 K_1^{\frac{1}{2}})^{\frac{1}{2}}) = \frac{1}{l}(\|h_1 - h_2\|_2^2 + \|K_1^{\frac{1}{2}} - K_2^{\frac{1}{2}}\|_f^2) \tag{7}$$

and $h1, h2$ are the matrix forms of means m_1, m_2 in Eq. 3, K_1, K_2 are the covariance matrix, corresponding to $\mathcal{K}_1$ and $\mathcal{K}_2$. $\|\|_2^2$ is the square of L^2 norm and $\|\|_f^2$ is the square of Frobenius norm. For detailed proof of the simplification of the trace of covariance matrix for two Gaussian processes in Eq. 7, please refer to (Givens & Shortt, 1984). Equipped with the above preliminary knowledge, we introduce a two-step strategy for training as shown in Algorithm 1. In the first step, we use the objective defined in Eq. 4 with $\lambda > 0$, whereas in the second step, we fine-tune the model by setting $\lambda = 0$ and a smaller learning rate is preferred. For step one, even though we have the closed and simplified form of the matrix form of square 2-Wasserstein distance shown in Eq. 7, calculating the differentiable square root of the matrix is still time consuming and may be numerically unstable (Pleiss et al., 2020). Here, we directly calculate $\|K_1 - K_2\|_f^2$ instead of $\|K_1^{\frac{1}{2}} - K_2^{\frac{1}{2}}\|_f^2$, utilizing $\widehat{W}_2^2 = \frac{1}{l}(\|h_1 - h_2\|_2^2 + \|K_1 - K_2\|_f^2)$, and such approximation is acceptable due to the the regularization term only exists in the warmup phase and is not a tight preparation. In the appendix, we show an ablation study that demonstrates the need for the regularization in the warmup phase. Furthermore, Eq. 7 implies we normalize the Wasserstein distance and makes sure the square of L^2 norm and Frobenius norm remain finite as the number of discretization points increases.

Algorithm 1 OpFlow Training

Input: Training dataset $\{u_j|_D\}_{j=1}^m$; collection of positions $D = \{x_i, \cdots x_l\}$; mean vector h_1 and covariance matrix K_1; noise level γ; random noise samples $\{\nu_i\}_{i=1}^n \sim \mathcal{GP}_1$; regularization weight λ
Initialize the reverse OpFlow $\mathcal{F}_\theta$
while warmup **do**
  Random batch $\{u_i|_D\}_{i=1}^n$ from $\{u_j|_D\}_{j=1}^m$
  Inject $\mathcal{GP}$ noise: $\{u_i|_D\}_{i=1}^n = \{u_i|_D + \gamma \cdot \nu_i|_D\}_{i=1}^n$
  Construct h_2, K_2: $h_2 = \frac{1}{n}\sum_{i=1}^n \mathcal{F}_\theta(u_i|_D)$, $K_2 = cov(\{\mathcal{F}_\theta(u_i|_D)\}_{i=1}^n)$
  Calculate W_2^2 approximate regularization term: $\widehat{W}_2^2 = \frac{1}{l}(\|h_1 - h_2\|_2^2 + \|K_1 - K_2\|_f^2)$
  Compute the raining objective $\mathcal{L} = -\frac{1}{n}\sum_i^n \log p_\theta(u_i|_D) + \lambda \widehat{W}_2^2$
  Update $\mathcal{F}_\theta$ through gradient descent of $\mathcal{L}$
end while
while finetune **do**
  Random batch $\{u_i|_D\}_{i=1}^n$ from $\{u_j|_D\}_{j=1}^m$
  Inject $\mathcal{GP}$ noise: $\{u_i|_D\}_{i=1}^n = \{u_i|_D + \gamma \cdot \nu_i|_D\}_{i=1}^n$
  Computing training objective $\mathcal{L} = -\frac{1}{n}\sum_i^n \log p_\theta(u_i|_D)$
  Update $\mathcal{F}_\theta$ through gradient descent of $\mathcal{L}$
end while

4.3 Universal Functional Regression with OpFlow

Bayesian functional regression treats the mapping function itself as a stochastic process, placing a prior distribution over the mapping function space. This allows for predictions with uncertainty estimates throughout $\mathcal{D}$, most importantly away from the observation points. GPR places a GP prior on the mapping function space, a common choice because of mathematical tractability. However, there are many real world datasets and scenarios that are not well-represented by a GP (Kındap & Godsill, 2022). To address this, we outline an algorithm for UFR with OpFlow in Bayesian inference problems.

In the following we assume that OpFlow has been trained previously and sufficiently approximates the true stochastic process of the data. We thus have an invertible mapping between function spaces $\mathcal{U}$ and $\mathcal{A}$. We now can use the trained OpFlow as a learned prior distribution over the space of possible mapping functions, fixing the parameters of OpFlow for functional regression.

Similar to GPR, suppose we are given $\tilde{u}_{obs}$, pointwise and potentially noisy observations of a function u, where the observation points are at the discretization set $D \subset \mathcal{D}_{\mathcal{U}}$. We refer to the evaluations at D as u_{obs}, and the additive noise ϵ is the white noise with the law $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Note that this is based on the conventions in GPRs; otherwise, the noise ϵ can also be treated as a function with a GP from. One more time, similar to GPR, we aim to infer the function's values on a set of new points D' where $D \subset D'$ given the observation $\tilde{u}_{obs}$. The regression domain therefore is D' at which we refer to the inferred function's values as u_ϕ and $a_\phi = \mathcal{F}_\theta(u_\phi)$. Ergo, for the likelihood of the values on points in D', i.e., the likelihood of u_ϕ, we have,

$$\log p_\theta(u_\phi|\tilde{u}_{obs}) = \log p_\theta(\tilde{u}_{obs}|u_\phi) + \log p_\theta(u_\phi) - \log p_\theta(\tilde{u}_{obs}) \tag{8}$$

$$= \log p_\theta(\tilde{u}_{obs}|u_{\phi|D}) + \log p_\theta(u_\phi) - \log p_\theta(\tilde{u}_{obs}) \tag{9}$$

$$= -\frac{\|\tilde{u}_{obs} - u_{\phi|D}\|_2^2}{2\sigma^2} + \log p_\theta(u_\phi) - \log \sigma - \frac{1}{2} r \log(2\pi) - \log p_\theta(\tilde{u}_{obs}), \tag{10}$$

where $p_\theta(u|_D)$ denotes the probability of u evaluated on the collection of points D with the learned prior (OpFlow), and r is the cardinality of set D with $r = |D|$. Maximizing the above equation results in maximum *a posteriori* (MAP) estimate of $p_\theta(u_\phi|\tilde{u}_{obs})$. Since the last three terms in Eq.10 are constants, such maximization is carried using gradient descent focusing only on the first two terms. In the following, we refer to $\overline{u}_\phi$ as the MAP estimate given $\tilde{u}_{obs}$.

For sampling for the posterior in Eq. 10, we utilize the fact that OpFlow is a bijective operator and the target function u_ϕ uniquely defines a_ϕ in $\mathcal{A}$ space, therefore, drawing posterior samples of u_ϕ is equivalent to

drawing posterior samples of a_ϕ in Gaussian space where SGLD with Gaussian random field perturbation is relevant. Our algorithm for sampling from the $p_\theta(u_\phi|\tilde{u}_{obs})$ is described in Algorithm 2. We initialize $a_\phi|\tilde{u}_{obs}$ with the MAP estimate of $p_\theta(u_\phi|\tilde{u}_{obs})$, i.e., $\overline{u}_\phi$, and follow Langevin dynamics in $\mathcal{A}$ space with Gaussian structure to sample u_ϕ. This choice helps to stabilize the functional regression and further reduces the number of burn-in iterations needed in SGLD. The burn-in time b is chosen to warm up the induced Markov process in SGLD and is hyperparameter to our algorithm (Ahn, 2015). We initialize the SGLD with $a_\phi^0 = \mathcal{F}_\theta(\overline{u}_\phi)$ and $u_\phi^0 = \mathcal{G}_\theta(a_\phi^0) = \overline{u}_\phi$. Then, at each iteration, we draw a latent space posterior sample a_ϕ^t in $\mathcal{A}$ space, where a_ϕ^t is the tth sample of function a_ϕ, and harvest the true posterior samples u_ϕ^t by calling $\mathcal{G}_\theta(a_\phi^t)$. Since $p_\theta(a_\phi|\tilde{u}_{obs}) \sim \mathcal{GP}$, it is very efficient to sample from it, and we find that relatively little hyperparameter optimization is needed. Since the sequential samples are correlated, we only store samples every t_N time step, resulting in representative samples of the posterior without the need to store all the steps of SGLD (Riabiz et al., 2022).

Algorithm 2 Sample from posterior using SGLD

1: **Input and Parameters:** OpFlow models $\mathcal{G}_\theta, \mathcal{F}_\theta$, Langevin dynamics temperature T, $\mathcal{G}_\theta$, MAP $\overline{u}_\phi$.
2: **Initialization**: $a_\phi^0 = \mathcal{F}_\theta(\overline{u}_\phi)$ and $u_\phi^0 = \mathcal{G}(a_\phi^0)$,
3: **for** $t = 0, 1, 2, \ldots, N$ **do**
4: Compute gradient of the posterior: $\nabla_{a_\phi} \log p_\theta(u_\phi^t|\tilde{u}_{obs})$
5: Update a_ϕ^{t+1}: $a_\phi^{t+1} = a_\phi^t + \frac{\eta_t}{2}\nabla \log p_\theta(u_\phi^t|\tilde{u}_{obs}) + \sqrt{\eta_t T}\mathcal{N}(0, I)$
6: **if** $t \geq b$ **then**
7: Every t_N iterations: obtain new sample a_ϕ^{t+1}, $u_\phi^{t+1} = \mathcal{G}_\theta(a_\phi^{t+1})$
8: **end if**
9: **end for**

5 Experiments

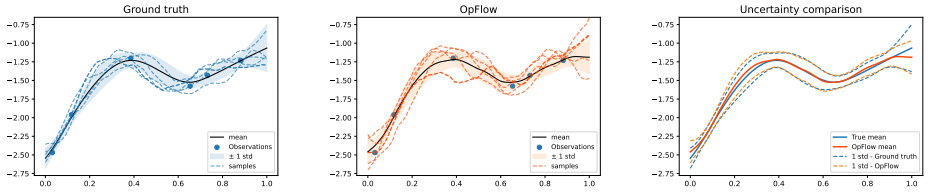
In this section, we aim to provide a comprehensive evaluation of OpFlow's capabilities for UFR, as well as generation tasks. We consider several ground truth datasets, composed of both Gaussian and non-Gaussian processes, as well as a real-world dataset of earthquake recordings with highly non-Gaussian characteristics.

5.1 Universal Functional Regression

Summary of parameters. Here, we show UFR experiments using the domain decomposed implementation of OpFlow, and leave the codomain OpFlow UFR experiments for the appendix. Table 2 lists the UFR hyperparameters used, as outlined in Algorithm 2. For training OpFlow, we take $\mathcal{A} \sim \mathcal{GP}()$ with a Matern covariance operator parameterized by length scale l and roughness ν. The mean function for the $\mathcal{GP}$ is set to zero over $\mathcal{D}$. The noise level (σ^2) of observations in Eq. 10 is assumed as 0.01 for all regression tasks.

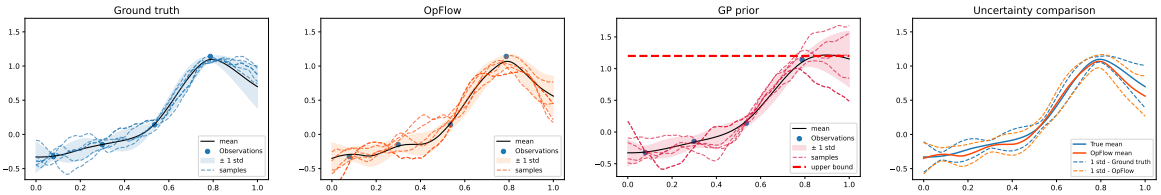
Gaussian Processes. This experiment aims to duplicate the results of classical GPR, where the posterior is known in closed form, but with OpFlow instead. We build a training dataset for $\mathcal{U} \sim \mathcal{GP}$, taking $l_\mathcal{U} = 0.5, \nu_\mathcal{U} = 1.5$ and the training dataset contains 30000 realizations. For the latent space GP, we take $l_\mathcal{A} = 0.1, \nu_\mathcal{A} = 0.5$.The resolution of $\mathcal{D}$ is set to 128 for the 1D regression experiment. Note that, the OpFlow is not explicitly notified with the mean and covariance of the data GP, and learns them using samples, while the GP ground truth is based on known parameters.

For performing the regression, we generate a single new realization of $\mathcal{U}$, and select observations at 6 random positions in $\mathcal{D}$. We then infer the posterior for u across the 128 positions, including noise-free estimations for the observed points. Fig. 3 displays the observation points and analytical solution for GP regression for this example, along with the posterior from OpFlow. In general, the OpFlow solution matches the analytical solution well; the mean predictions of OpFlow (equivalent to the MAP estimate for GP data) and its assessed uncertainties align closely with the ground truth. This experiment demonstrates OpFlow's ability to accurately capture the posterior distribution, for a scenario where the observations are from a Gaussian Process with an exact solution available.



(a) (b) (c)

Figure 3: OpFlow regression on GP data. (a) Ground truth GP regression with observed data and predicted samples (b) OpFlow regression with observed data and predicted samples. (c) Uncertainty comparison between true GP and OpFlow predictions.



(a) (b) (c) (d)

Figure 4: OpFlow regression on TGP data. (a) Ground truth TGP regression with observed data and predicted samples (b) OpFlow regression with observed data and predicted samples. (c) prior GP regression with observed data and predicted samples (c) Uncertainty comparison between true TGP and OpFlow predictions.

Truncated Gaussian Processes. Next, we apply OpFlow to a non-Gaussian functional regression problem. Specifically, we use the truncated Gaussian process (TGP) (Swiler et al., 2020) to construct a 1D training dataset. The TGP is analogous to the truncated Gaussian distribution, which bounds the functions to have amplitudes lying in a specific range. For the dataset, we take $l_{\mathcal{U}} = 0.5$ and $\nu_{\mathcal{U}} = 1.5$, and truncate the function amplitudes to the range $[-1.2, 1.2]$. For the latent space GP, we take $l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$. The resolution is set to 128. We conduct an experiment with only 4 random observation points, infer the function across all 128 positions. Unlike most non-Gaussian processes, where the true posterior is often intractable, the TGP framework, governed by parameters $l_{\mathcal{U}}, \nu_{\mathcal{U}}$ and known bounds, allows for true posterior estimation using rejection sampling. We consider a narrow tolerance value of ± 0.10 around each observation. The tolerance range can be analogous to but conceptually distinct from noise level (Robert et al., 1999; Swiler et al., 2020). For the training dataset, we use 30000 realizations of the TGP. For the posterior evaluation, we generate 1000 TGP posterior samples, and taken the calculated mean and uncertainty from the posterior samples as the ground truths.

Fig. 4 shows the UFR results for this dataset alongside the true TGP posterior. We see that OpFlow effectively captures the true non-Gaussian posterior, producing samples that are visually aligned with the ground truth. For comparison, we also show the results of applying GP regression to this dataset. Notice that the GP prior allows for non-zero probability outside of the truncation range, whereas the OpFlow prior learns these constraints, resulting in a much more accurate posterior. Fig. 4 reveals a slight deviation in OpFlow's mean prediction starting from $x = 0.8$, yet the uncertainty remains accurately represented. This discrepancy may be explained by the challenging nature of this task, which involves using only 4 observations. With more observations, OpFlow's predictions are expected to improve.

Gaussian random fields. We build a 2D dataset of Gaussian random fields having $l_{\mathcal{U}} = 0.5$ and $\nu_{\mathcal{U}} = 1.5$. For the latent space, we have a GP with $l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$. We set the resolution of the physical domain to 32×32 for the training data. We generate 20000 samples in total for training. This is another experiment with an analytical expression available for the true posterior. We generate one new realization of the data for functional regression, and choose observations at 32 random positions, as shown in Fig 5a. We then infer the possible outcomes over the entire 32×32 grid. Fig 5 shows that OpFlow reliably recovers the posterior

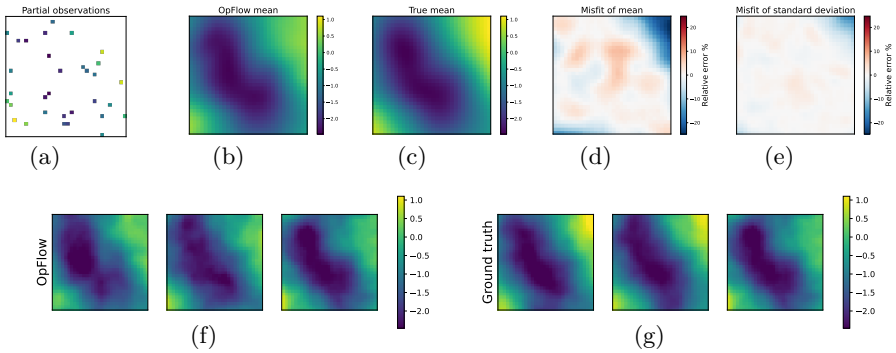


Figure 5: OpFlow regression on 32×32 GRF data. (a) 32 random observations. (b) Predicted mean from OpFlow. (c) Ground truth mean from GP regression. (d) Misfit of the predicted mean. (e) Misfit of predicted uncertainty. (f) Predicted samples from OpFlow. (g) Predicted samples from GP regression.

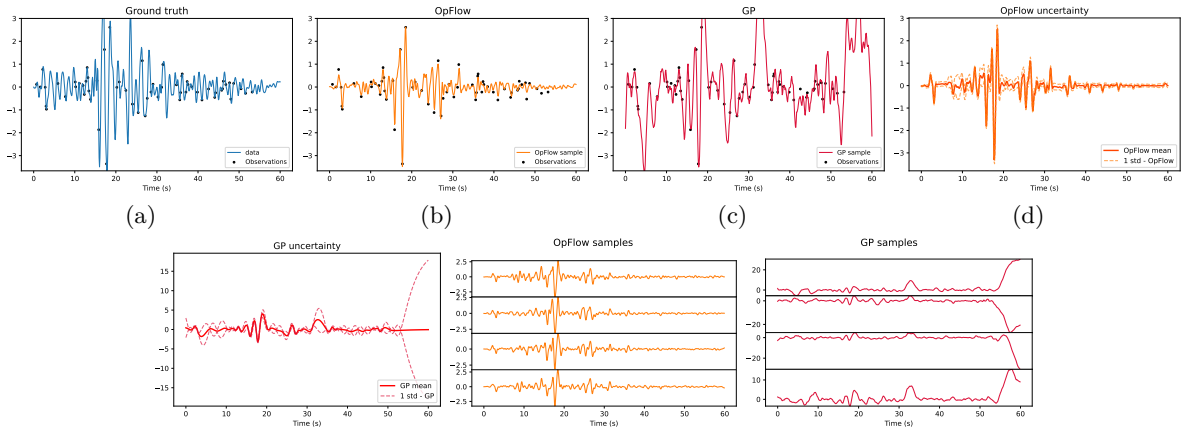
without assuming beforehand that the true data are from a Gaussian process. The error is non-uniform over the domain, the largest of which occurs in the data-sparse upper right corner. Fig 5 shows several samples of OpFlow's posterior, mirroring the true posterior variability produced by GPR. We show a second scenario for this same function in the appendix, but where the observations are concentrated along linear patterns (Fig 10).Similarly, OpFlow faithfully approximates the true posterior.

Seismic waveforms regression. In this experiment, OpFlow regression is applied to a real-world non-Gaussian dataset consisting of earthquake ground motion time series records from Japan Kiban–Kyoshin network (KiK-net). The raw data are provided by the Japanese agency, National Research Institute for Earth Science and Disaster Prevention (NIED, 2013). The KiK-net data encompasses 20,643 time series of ground velocity for earthquakes with magnitudes in the range 4.5 to 5, recorded between 1997 and 2017. All seismic signals were resampled at 10Hz and have a 60 second duration, resulting in a resolution of 600 for our analysis. The time series are aligned such that $t = 0$ coincides with the onset of shaking. A second-order Gaussian filter was used to attenuate high-frequency wiggles. We used $l_{\mathcal{A}} = 0.05, \nu_{\mathcal{U}} = 0.5$ for the latent Gaussian space $\mathcal{A}$. After training the OpFlow prior, we select a new (unseen) time series randomly for testing (Fig. 6a), and perform UFR over the entire domain with 60 randomly selected observation points, (Fig. 6). For comparison, we show the results for GP regression with parameters optimized via likelihood maximization. Fig 6b shows that a sample from OpFlow matches the actual data, with OpFlow accurately capturing key characteristics in the data that include various types of seismic waves arriving at different times in the time series (Shearer, 2019). Conversely, the standard GP model struggles with the complexity of the seismic data as shown in Fig. 6c. Further, Fig. 6d,e illustrate that OpFlow provides a reasonable estimation of the mean and uncertainty, whereas the GP model's uncertainty significantly increases after 50 seconds due to a lack of observations. Additional samples from OpFlow and GP (Fig. 6f,g) further highlight the superior performance of OpFlow for functional regression of seismic waveforms.

5.2 Generation Tasks

While our emphasis in this study is on functional regression, there is still value in evaluating the generative capabilities of OpFlow. Here, we take GANO as a baseline model as it is a generative neural operator with discretization invariance. The following study demonstrates that domain decomposed OpFlow provides comparable generation capabilities similar to prior works, such as GANO. The implementation of GANO used herein follows that of the paper (Rahman et al., 2022a). In the following experiments, the implementations of the GP for the Gaussian space $\mathcal{A}$ of GANO is adapted to optimize GANO's performance for each specific case. Unless otherwise specified, the default GP implementation utilize the Gaussian process package available in Scikit-learn (Pedregosa et al., 2011). More generation experiments of domain decomposed OpFlow and codomain OpFlow are provided in the appendix.

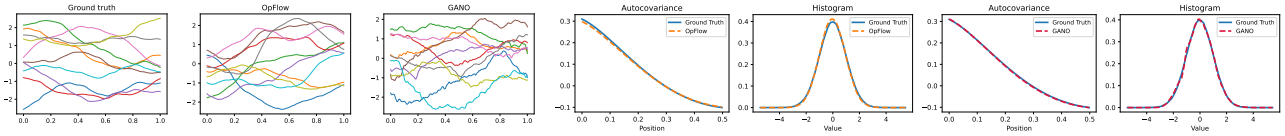
GP data. For both OpFlow and GANO, we set Matern kernel parameters with $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5$ for the observed data space. And $l_{\mathcal{A}} = 0.05, \nu_{\mathcal{A}} = 0.5$ for the latent Gaussian space of OpFlow, $\alpha = 1.5, \tau = 1.0$ for the latent Gaussian space of GANO, where α, τ are the coefficient parameter and inverse length scale



(a) (b) (c) (d) (e) (f) (g)

Figure 6: OpFlow regression on seismic waveform data. (a) Ground truth waveform. (b) One predicted sample from OpFlow. (c) One predicted sample from the best fitted GP. (d) Predicted mean and uncertainty from OpFlow. (e) Predicted mean and uncertainty from the best fitted GP. (f) Samples from OpFlow. (g) Samples from the best fitted GP.

parameter of the specific implementation of Matérn based Gaussian process used in GANO paper (Rahman et al., 2022a; Nelsen & Stuart, 2021). Data resolution used here is 256. Fig. 7 shows that OpFlow can generate realistic samples, virtually consistent with the true data. In contrast, GANO's samples exhibit small high frequency artifacts. Both OpFlow and GANO successfully recover the statistics with respect to the autovariance function and histogram of the point-wise value (amplitude distribution). We note that there is a significant difference in model complexity here, as OpFlow has 15M parameters whereas GANO has only 1.4M. The appendix shows a super-resolution experiment for this dataset.

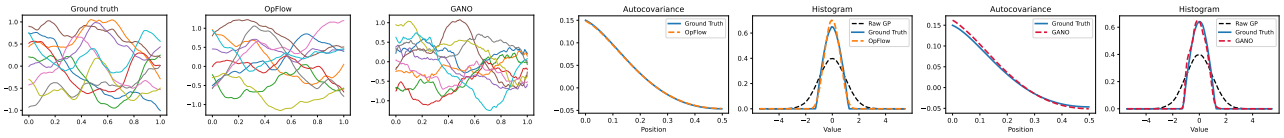


(a) (b) (c) (d) (e) (f) (g)

Figure 7: OpFlow generation performance on GP data. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from GANO. (g) Histogram of samples from GANO.

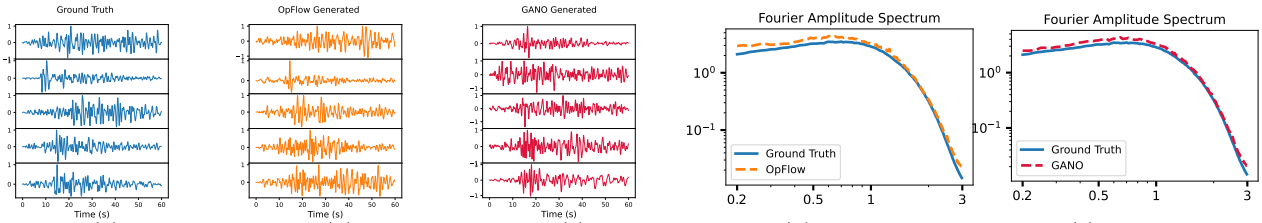
TGP data. We maintain the same parameter settings as used in the TGP functional regression experiment, with bounds set between -1.2 and 1.2, and Matern kernel parameters $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5$ for the observed space and $l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$ for the latent space of OpFlow, $\alpha = 1.5, \tau = 1.0$ for the latent Gaussian space of GANO. The only difference is that we increase the resolution to 256 from 128. Fig. 8 demonstrates that OpFlow is capable of generating realistic samples that closely mimic the ground truth samples, while GANO tends to produce samples that appear rougher in texture. Details on the super-resolution experiment for this task are available in the appendix.

Seismic Data. The dataset is the same as used in the functional regression section. However for this test, we no longer use a Gaussian filter to simplify the training dataset, which makes it more challenging to learn. The results are shown in Fig. 9. These figures demonstrate that both OpFlow and GANO can generate visually realistic seismic waveforms. For validation purposes, we examine the Fourier Amplitude Spectrum, a critical engineering metric (Shi et al., 2024b), in Fig. 9. The analysis shows that both models accurately learn the frequency content of the seismic velocity dataset, underscoring their potential utility in seismic hazard analysis and engineering design (Shi et al., 2024b).



(a) (b) (c) (d) (e) (f) (g)

Figure 8: OpFlow generation performance on TGP data. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from GANO. (g) Histogram of samples from GANO.



(a) (b) (c) (d) (e)

Figure 9: OpFlow generation performance on seismic velocity data from kik-net. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Fourier amplitude spectrum of samples from OpFlow. (e) Fourier amplitude spectrum of samples from GANO

6 Conclusion

In this paper, we have introduced Neural Operator Flows (OpFlow), an invertible infinite-dimensional generative model that allows for universal functional regression. OpFlow offers a learning-based solution for end-to-end universal functional regression, which has the potential to outperform traditional functional regression with Gaussian processes. This is in addition to OpFlow comparing favorably with other algorithms for generation tasks on function spaces. Looking ahead, OpFlow may allow for new avenues in analysis of functional data by learning directly from the data, rather than assuming the type of stochastic process beforehand.

References

Sungjin Ahn. *Stochastic gradient mcmc: Algorithms and applications*. University of California, Irvine, 2015. ISBN 1-339-12430-0.

Suzanne Aigrain and Daniel Foreman-Mackey. Gaussian Process Regression for Astronomical Time Series. *Annual Review of Astronomy and Astrophysics*, 61(1):329–371, 2023. doi: 10.1146/annurev-astro-052920-103508. URL `https://doi.org/10.1146/annurev-astro-052920-103508`. _eprint: https://doi.org/10.1146/annurev-astro-052920-103508.

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural Operators for Accelerating Scientific Simulations and Design, January 2024. URL `http://arxiv.org/abs/2309.15325`. arXiv:2309.15325 [physics].

Lorenzo Baldassari, Ali Siahkoohi, Josselin Garnier, Knut Solna, and Maarten V. de Hoop. Conditional score-based diffusion models for Bayesian inference in infinite dimensions. *Advances in Neural Information Processing Systems*, 36:24262–24290, December 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/4c79c359b3c5f077c0b955f93cb0f53e-Abstract-Conference.html`.

Daniel Barry. Nonparametric Bayesian Regression. *The Annals of Statistics*, 14(3):934–953, 1986. ISSN 0090-5364. URL `https://www.jstor.org/stable/3035551`. Publisher: Institute of Mathematical Statistics.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations, December 2019. URL `http://arxiv.org/abs/1806.07366`. arXiv:1806.07366 [cs, stat].

Jean-Paul Chiles and Pierre Delfiner. *Geostatistics: modeling spatial uncertainty*, volume 713. John Wiley & Sons, 2012. ISBN 0-470-18315-2.

Masoumeh Dashti and Andrew M. Stuart. The Bayesian Approach to Inverse Problems. In Roger Ghanem, David Higdon, and Houman Owhadi (eds.), *Handbook of Uncertainty Quantification*, pp. 311–428. Springer International Publishing, Cham, 2017. ISBN 978-3-319-12384-4 978-3-319-12385-1. doi: 10.1007/978-3-319-12385-1_7. URL http://link.springer.com/10.1007/978-3-319-12385-1_7.

Volker L. Deringer, Albert P. Bartók, Noam Bernstein, David M. Wilkins, Michele Ceriotti, and Gábor Csányi. Gaussian Process Regression for Materials and Molecules. *Chemical Reviews*, 121(16):10073–10141, August 2021. ISSN 0009-2665. doi: 10.1021/acs.chemrev.1c00022. URL https://doi.org/10.1021/acs.chemrev.1c00022. Publisher: American Chemical Society.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP, February 2017. URL http://arxiv.org/abs/1605.08803. arXiv:1605.08803 [cs, stat].

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. URL http://arxiv.org/abs/2010.11929. arXiv:2010.11929 [cs].

Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative Models as Distributions of Functions, February 2022. URL http://arxiv.org/abs/2102.04776. arXiv:2102.04776 [cs, stat].

Vincent Dutordoir, Alan Saul, Zoubin Ghahramani, and Fergus Simpson. Neural Diffusion Processes, June 2022. URL http://arxiv.org/abs/2206.03992. arXiv:2206.03992 [cs, stat].

Giulio Franzese, Giulio Corallo, Simone Rossi, Markus Heinonen, Maurizio Filippone, and Pietro Michiardi. Continuous-Time Functional Diffusion Processes, December 2023. URL http://arxiv.org/abs/2303.00800. arXiv:2303.00800 [cs, stat].

Takashi Furuya, Michael Puthawala, Matti Lassas, and Maarten V. de Hoop. Globally injective and bijective neural operators, June 2023. URL http://arxiv.org/abs/2306.03982. arXiv:2306.03982 [cs, stat].

Angela Gao, Jorge Castellanos, Yisong Yue, Zachary Ross, and Katherine Bouman. DeepGEM: Generalized Expectation-Maximization for Blind Inversion. In *Advances in Neural Information Processing Systems*, volume 34, pp. 11592–11603. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/606c90a06173d69682feb83037a68fec-Abstract.html.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural Processes, July 2018. URL http://arxiv.org/abs/1807.01622. arXiv:1807.01622 [cs, stat].

Clark R Givens and Rae Michael Shortt. A class of Wasserstein metrics for probability distributions. *Michigan Mathematical Journal*, 31(2):231–240, 1984. ISSN 0026-2285. Publisher: University of Michigan, Department of Mathematics.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. URL http://arxiv.org/abs/1406.2661. arXiv:1406.2661 [cs, stat].

Paul Hagemann, Sophie Mildenberger, Lars Ruthotto, Gabriele Steidl, and Nicole Tianjiao Yang. Multilevel Diffusion: Infinite Dimensional Score-Based Diffusion Models for Image Generation, March 2023. URL https://arxiv.org/abs/2303.04772v3.

Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, and Dinand Schepers. The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020. ISSN 0035-9009. Publisher: Wiley Online Library.

Chin-Wei Huang, Laurent Dinh, and Aaron Courville. Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models, February 2020. URL http://arxiv.org/abs/2002.07101. arXiv:2002.07101 [cs, stat].

Gavin Kerrigan, Justin Ley, and Padhraic Smyth. Diffusion Generative Models in Infinite Dimensions, February 2023. URL http://arxiv.org/abs/2212.00886. arXiv:2212.00886 [cs, stat].

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions, July 2018. URL http://arxiv.org/abs/1807.03039. arXiv:1807.03039 [cs, stat].

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022. URL http://arxiv.org/abs/1312.6114. arXiv:1312.6114 [cs, stat].

Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11): 3964–3979, November 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2020.2992934. URL http://arxiv.org/abs/1908.09257. arXiv:1908.09257 [cs, stat].

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces, April 2023. URL http://arxiv.org/abs/2108.08481. arXiv:2108.08481 [cs, math].

Yaman Kındap and Simon Godsill. Non-Gaussian Process Regression, September 2022. URL http://arxiv.org/abs/2209.03117. arXiv:2209.03117 [cs, stat].

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Graph Kernel Network for Partial Differential Equations, March 2020. URL http://arxiv.org/abs/2003.03485. arXiv:2003.03485 [cs, math, stat].

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, May 2021. URL http://arxiv.org/abs/2010.08895. arXiv:2010.08895 [cs, math].

Jae Hyun Lim, Nikola B. Kovachki, Ricardo Baptista, Christopher Beckham, Kamyar Azizzadenesheli, Jean Kossaifi, Vikram Voleti, Jiaming Song, Karsten Kreis, Jan Kautz, Christopher Pal, Arash Vahdat, and Anima Anandkumar. Score-based Diffusion Models in Function Space, November 2023. URL http://arxiv.org/abs/2302.07400. arXiv:2302.07400 [cs, math, stat].

Miguel Liu-Schiaffini, Julius Berner, Boris Bonev, Thorsten Kurth, Kamyar Azizzadenesheli, and Anima Anandkumar. Neural Operators with Localized Integral and Differential Kernels, February 2024. URL http://arxiv.org/abs/2402.16845. arXiv:2402.16845 [cs, math] version: 1.

Andreas Lugmayr, Martin Danelljan, Luc Van Gool, and Radu Timofte. SRFlow: Learning the Super-Resolution Space with Normalizing Flow, July 2020. URL http://arxiv.org/abs/2006.14200. arXiv:2006.14200 [cs, eess].

Anton Mallasto and Aasa Feragen. Learning from uncertain curves: The 2-Wasserstein metric for Gaussian processes. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/7a006957be65e608e863301eb98e1808-Abstract.html.

Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R. Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Proceedings of the Conference on Robot Learning*, pp. 430–444. PMLR, May 2020. URL https://proceedings.mlr.press/v100/mazoure20a.html. ISSN: 2640-3498.

Nicholas H. Nelsen and Andrew M. Stuart. The Random Feature Model for Input-Output Maps between Banach Spaces. *SIAM Journal on Scientific Computing*, 43(5):A3212–A3243, January 2021. ISSN 1064-8275, 1095-7197. doi: 10.1137/20M133957X. URL http://arxiv.org/abs/2005.10224. arXiv:2005.10224 [physics, stat].

NIED. Strong-Motion Seismograph Networks (K-NET, KiK-Net). *National Research Institute for Earth Science and Disaster Prevention*, 2013.

Peter Orbanz and Yee Whye Teh. Bayesian Nonparametric Models. *Encyclopedia of machine learning*, 1: 81–89, 2010.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, February 2022. URL `http://arxiv.org/abs/2202.11214`. arXiv:2202.11214 [physics].

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, and Vincent Dubourg. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011. ISSN 1532-4435. Publisher: JMLR. org.

Jakiw Pidstrigach, Youssef Marzouk, Sebastian Reich, and Sven Wang. Infinite-Dimensional Diffusion Models, February 2023. URL `https://arxiv.org/abs/2302.10130v2`.

Wei Ping, Kainan Peng, Kexin Zhao, and Zhao Song. WaveFlow: A Compact Flow-based Model for Raw Audio. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 7706–7716. PMLR, November 2020. URL `https://proceedings.mlr.press/v119/ping20a.html`. ISSN: 2640-3498.

Geoff Pleiss, Martin Jankowiak, David Eriksson, Anil Damle, and Jacob Gardner. Fast Matrix Square Roots with Applications to Gaussian Processes and Bayesian Optimization. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22268–22281. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/hash/fcf55a303b71b84d326fb1d06e332a26-Abstract.html`.

Shengju Qian, Hao Shao, Yi Zhu, Mu Li, and Jiaya Jia. Blending Anti-Aliasing into Vision Transformer. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5416–5429. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/2b3bf3eee2475e03885a110e9acaab61-Abstract.html`.

Md Ashiqur Rahman, Manuel A. Florez, Anima Anandkumar, Zachary E. Ross, and Kamyar Azizzadenesheli. Generative Adversarial Neural Operators, October 2022a. URL `http://arxiv.org/abs/2205.03017`. arXiv:2205.03017 [cs, math].

Md Ashiqur Rahman, Jasorsi Ghosh, Hrishikesh Viswanath, Kamyar Azizzadenesheli, and Aniket Bera. PaCMO: Partner Dependent Human Motion Generation in Dyadic Human Activity using Neural Operators, November 2022b. URL `http://arxiv.org/abs/2211.16210`. arXiv:2211.16210 [cs].

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2006. ISBN 978-0-262-18253-9. OCLC: ocm61285753.

Marina Riabiz, Wilson Ye Chen, Jon Cockayne, Pawel Swietach, Steven A. Niederer, Lester Mackey, and Chris. J. Oates. Optimal Thinning of MCMC Output. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(4):1059–1081, September 2022. ISSN 1369-7412, 1467-9868. doi: 10.1111/rssb.12503. URL `https://academic.oup.com/jrsssb/article/84/4/1059/7073269`.

Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

Jacob H. Seidman, Georgios Kissas, George J. Pappas, and Paris Perdikaris. Variational Autoencoding Neural Operators, February 2023. URL `http://arxiv.org/abs/2302.10351`. arXiv:2302.10351 [cs, stat].

Peter M. Shearer. *Introduction to Seismology*. Cambridge University Press, May 2019. ISBN 978-1-107-18447-3. Google-Books-ID: 08aVDwAAQBAJ.

Yaozhong Shi, Grigorios Lavrentiadis, Domniki Asimaki, Zach E. Ross, and Kamyar Azizzadenesheli. Development of Synthetic Ground-Motion Records through Generative Adversarial Neural Operators. pp. 105–113, February 2024a. doi: 10.1061/9780784485316.012. URL https://ascelibrary.org/doi/10.1061/9780784485316.012. Publisher: American Society of Civil Engineers.

Yaozhong Shi, Grigorios Lavrentiadis, Domniki Asimaki, Zachary E. Ross, and Kamyar Azizzadenesheli. Broadband Ground-Motion Synthesis via Generative Adversarial Neural Operators: Development and Validation. *Bulletin of the Seismological Society of America*, March 2024b. ISSN 0037-1106. doi: 10.1785/0120230207. URL https://doi.org/10.1785/0120230207.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60, July 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. SCORE-BASED GENERATIVE MODELING THROUGH STOCHASTIC DIFFERENTIAL EQUATIONS. 2021.

Laura Swiler, Mamikon Gulian, Ari Frankel, Cosmin Safta, and John Jakeman. A Survey of Constrained Gaussian Process Regression: Approaches and Implementation Challenges. *Journal of Machine Learning for Modeling and Computing*, 1(2):119–156, 2020. ISSN 2689-3967. doi: 10.1615/JMachLearnModelComput.2020035155. URL http://arxiv.org/abs/2006.09319. arXiv:2006.09319 [cs, math, stat].

Larry Wasserman. *All of nonparametric statistics*. Springer Science & Business Media, 2006. ISBN 0-387-30623-4.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. pp. 681–688. Citeseer, 2011.

Jay Whang, Erik Lindgren, and Alex Dimakis. Composing Normalizing Flows for Inverse Problems. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 11158–11169. PMLR, July 2021. URL https://proceedings.mlr.press/v139/whang21b.html. ISSN: 2640-3498.

Yan Yang, Angela F. Gao, Jorge C. Castellanos, Zachary E. Ross, Kamyar Azizzadenesheli, and Robert W. Clayton. Seismic Wave Propagation and Inversion with Neural Operators. *The Seismic Record*, 1(3):126–134, November 2021. ISSN 2694-4006. doi: 10.1785/0320210026. URL https://doi.org/10.1785/0320210026.

Qiongyi Zhou, Changde Du, Dan Li, Haibao Wang, Jian K. Liu, and Huiguang He. Neural Encoding and Decoding With a Flow-Based Invertible Generative Model. *IEEE Transactions on Cognitive and Developmental Systems*, 15(2):724–736, June 2023. ISSN 2379-8920, 2379-8939. doi: 10.1109/TCDS.2022.3176977. URL https://ieeexplore.ieee.org/document/9780264/.

A Appendix

A.1 Forward and Inverse processes of OpFlow

Algorithm 3 Forward and Inverse processes of OpFlow

Input and Parameters: Input function $(v')^i$ to the ith layer of OpFlow; ith actnorm layer parameterized by s^i_θ, b^i_θ; ith affine coupling layer $\mathcal{T}^i_\theta$
Forward process:
    Input function: $(v')^i$
    Actnorm layer: $v^i = s^i_\theta \odot (v')^i + b^i_\theta$
    Affine coupling layer: $h^i_1, h^i_2 = \textbf{split}(v^i)$; $\log(s^i), b^i = \mathcal{T}^i_\theta(h^i_1)$; $(h')^i_2 = s^i \odot h^i_2 + b^i$
    Output function: $(v')^{i+1} = \textbf{concat}(h^i_1, (h')^i_2)$
Inverse process:
    Input function: $(v')^{i+1}$
    Partition: $h^i_1, (h')^i_2 = \textbf{split}((v')^{i+1})$
    Inverse of the affine coupling layer: $\log(s^i), b^i = \mathcal{T}^i_\theta(h^i_1)$; $h^i_2 = ((h')^i_2 - b^i)/s^i$; $v^i = \textbf{concat}(h^i_1, h^i_2)$
    Inverse of the actnorm layer: $(v')^i = (v^i - b^i_\theta)/s^i_\theta$
    Output function:$(v')^i$

A.2 Discretization agnostic property of OpFlow

In this part, we first expand the definition of neural operators and elaborate on their discretization-invariant properties, then explain why OpFlow is also discretization invariant.

Given two separable Banach spaces $\mathcal{A}$ and $\mathcal{U}$, a neural operator (denoted as $\mathcal{G}_\theta$) learns mappings between two function spaces. $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$, involving input and output functions on bounded domain in real-valued spaces. Thus for an input function $a \in \mathcal{A}$, we have $\mathcal{G}_\theta(a) = u$, where $u \in \mathcal{U}$. The architecture of neural operators is usually composed of three parts (Kovachki et al., 2023) : (1) **Lifting:** Converts input functions $a : D \to \mathbb{R}^{d_a}$ into a higher-dimensional space $\mathbb{R}^{d_{v_0}}$ using a local operator, with $d_{v_0} > d_a$. (2) **Kernel Integration:** For $i = 0$ to $T - 1$, map hidden state v_i to the next v_{i+1} by a combination of local linear, non-local integral operator and a bias function. (3) **Projection:** The final hidden state v_T is projected to the output space $\mathbb{R}^{d_u}$ through a local operator, where $d_{v_T} > d_u$.

The lift and projection parts in neural operators are usually shallow neural networks, such as Multilayer Perceptrons. For ith kernel integration layer, we have the integral kernel operator $\mathcal{K}_i$ with corresponding kernel function $k_i \in C(\mathcal{D}_{i+1} \times \mathcal{D}_i; \mathbb{R}^{d_{v_{i+1}} \times d_{v_i}})$, where $v_{i+1}(y), v_i(x)$ are defined on domains $\mathcal{D}_{i+1}, \mathcal{D}_i$, respectively. Denote μ_i as the measure on $\mathcal{D}_i$, W_i as the local linear opeartor, b_i as the bias function. Then for the ith kernel integration layer, we have

$$v_{i+1}(y) = \int_{D_i} \kappa_i(x, y) v_i(x) d\mu_i(x) + W_i v_i(y) + b_i(y), \quad \forall y \in \mathcal{D}_{i+1} \tag{11}$$

Fourier Neural Operator (Li et al., 2021) simplifies Eq 11 using Fourier transform and achieve quasi-linear time complexity, which can be expressed as

$$v_{i+1} = \mathcal{F}^{-1}(R_i(\mathcal{F} v_i)) + W_i v_i + b_i \tag{12}$$

where R_i are learnable parameters defined in Fourier space associated with user-defined Fourier modes, $\mathcal{F}$ and $\mathcal{F}^{-1}$ are Fourier transformation and inverse Fourier transformation respectively. The calculation of Eq 11,12 for data with arbitrary discretization involves Riemann summation. For more details of calculation of the kernel integration, please refer to Kovachki et al. (2023); Li et al. (2021); Rahman et al. (2022a).

Furthermore, the formulation of OpFlow is for function spaces, all components and operations invovled in OpFlow are directly defined on function space. The likelihood computation in Eq. 1 can take place

for any discretization, and moreover, the 2-Wasserstein distance in Eq. 2 is derived for function spaces, which we show how to compute on given discretizations in Eq. 7. Therefore OpFlow is discretization agnostic. We further use a series of zero-shot super-resolution experiments in Appendix A.7 to demonstrate the discretization-invariant feature of OpFlow.

A.3 Parameters used in regression tasks

Table 2 shows the regression parameters (explained in Algorithm 2) for all regression experiments used in this paper. The learning rate is decayed exponentially, which starts from $\{\eta_t, t = 0\}$, ends with $\{\eta_t, t = N\}$

Table 2: Datasets and regression parameters

Datasets	total iterations (N)	burn-in iterations (b)	sample iterations (t_N)	temperature of noise (T)	initial learning rate ($\eta_t, t = 0$)	end learning rate ($\eta_t, t = N$)
GP	4e4	2e3	10	1	5e-3	4e-3
TGP	4e4	2e3	10	1	5e-3	4e-3
GRF	2e4	2e3	10	1	5e-3	4e-3
Seismic	4e4	2e3	10	1	1e-3	8e-4
Codomain GP	4e4	2e3	10	1	5e-5	4e-5

A.4 OpFlow regression on Gaussian random fields example

This is the second Gaussian random field regression example mentioned in Section 5.1. OpFlow faithfully approximates the true posterior given the observations are concentrated along linear patterns. The relative error is the absolute error normalized by the absolute max value of ground truth mean from GP regression.

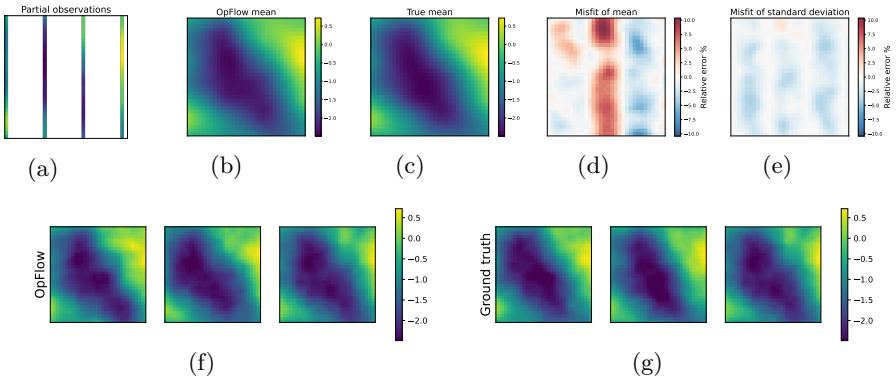


Figure 10: OpFlow regression on 32×32 GRF data. (a) 4 strips observations. (b) Predicted mean from OpFlow. (c) Ground truth mean from GP regression. (d) Misfit of the predicted mean. (e) Misfit of predicted uncertainty. (f) Predicted samples from OpFlow. (g) Predicted samples from GP regression.

A.5 More generation experiments with domain decomposed OpFlow

GRF data. For both OpFlow and GANO, we take the hyperparameters to be $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5$ for the observed space and $l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$ for the GP defined on latent Gaussian space $\mathcal{A}$. A resolution of 64×64 is chosen. Fig. 11 illustrates that OpFlow successfully generates realistic samples, closely resembling the ground truth in both autocovariance and amplitude distribution.

TGRF data. We take the hyperparameters $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5; l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$ for generating truncated 2D Gaussian random fields. The distribution is truncated between $[-2, 2]$, with a resolution of 64×64. From Figure 12, both OpFlow and GANO generate high quality samples. In Fig. 12, we also include the histogram of the GP training data before truncation, which shows that the TGRF data strongly deviates from Gaussian.

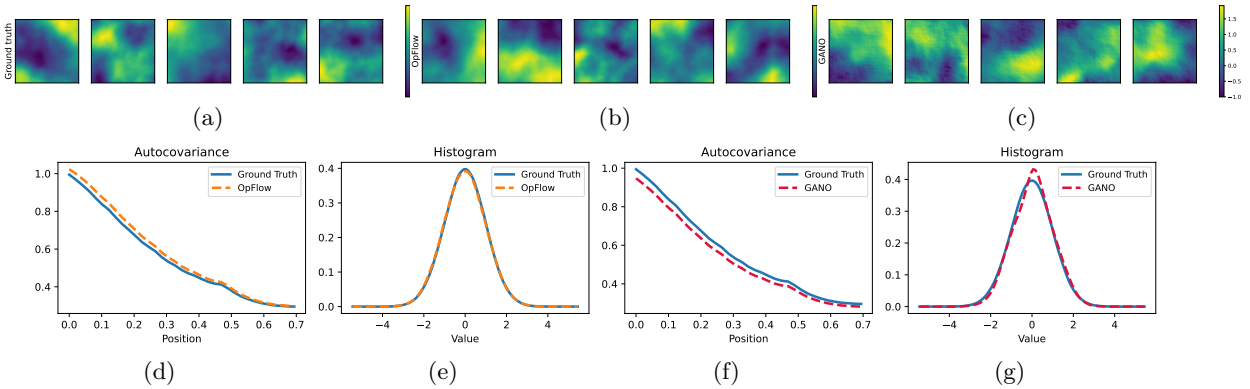


Figure 11: OpFlow generation performance on GRF data. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from GANO. (g) Histogram of samples from GANO.

While GANO exhibits some artifacts in the generated samples, it is worth noting the covariance function and amplitude histogram are learned reasonable well, indicating that the artifacts are only present at very high frequency.

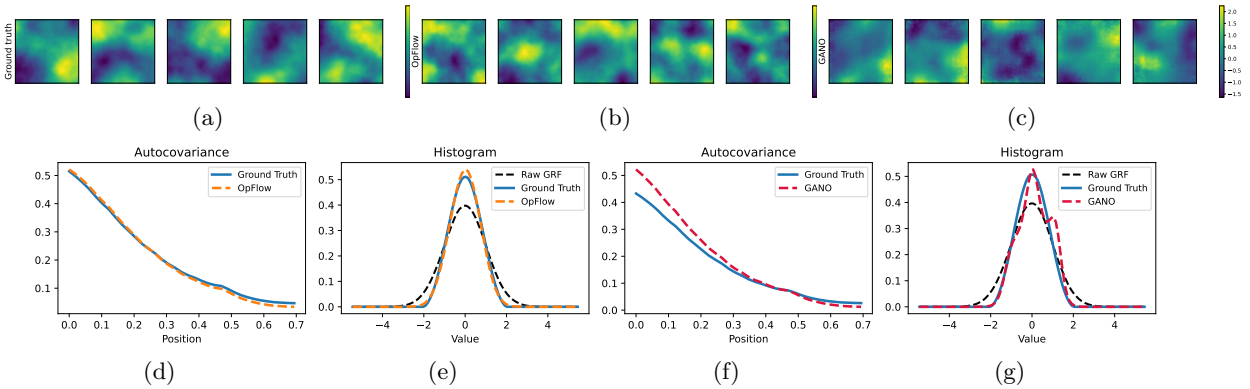


Figure 12: OpFlow generation performance on TGRF data. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from GANO. (g) Histogram of samples from GANO.

A.6 Generation with codomain OpFlow

For the codomain implementation of OpFlow, we will focus on super-resolution tasks, which do not suffer from the masking artifacts that occur with the domain decomposed setting. It is often the case that the function of interest is multivariate, i.e. the codomain spans more than one dimension. For instance, in the study of fluid dynamics, velocity fields are often linked with pressure fields. In these cases, there exists a natural partitioning among the co-domain for the affine coupling layers. However, in order to make codomain OpFlow applicable to univariate functions, we first randomly shuffle the training dataset $\{u_j|_D\}_{j=1}^m$, and partition the data into two equal parts $\{u_i|_D\}_{i=1}^{m_1}$ and $\{u'_i|_D\}_{i=1}^{m_1}$. We then concatenate these parts along the channel dimension to create a paired bivariate dataset for training, $\{(u_i, u'_i)|_D\}_{i=1}^{m_1}$.

Codomain GP data. We found that the codomain OpFlow requires different hyperparameters from the domain decomposed setting. Here we set $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 0.5; l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$, with a resolution of 256. In particular, these values suggest that codomain OpFlow has less expressivity than domain decomposed OpFlow, which we attribute to the need for codomain OpFlow to learn the joint probability measures, which is hard by nature. Given that both channels convey identical information, we present results from only the first channel for samples generated by codomain OpFlow. Fig. 13 demonstrates that both codomain

OpFlow and GANO effectively produce realistic samples, capturing both the autocovariance function and the histogram distribution accurately. Contrastingly, in the appendix, codomain OpFlow shows great performance in super-resolution task, potentially due to the channel-split operation preserving the physical domain's integrity. To the best of our knowledge, codomain OpFlow is the inaugural model within the normalizing flow framework to achieve true resolution invariance.

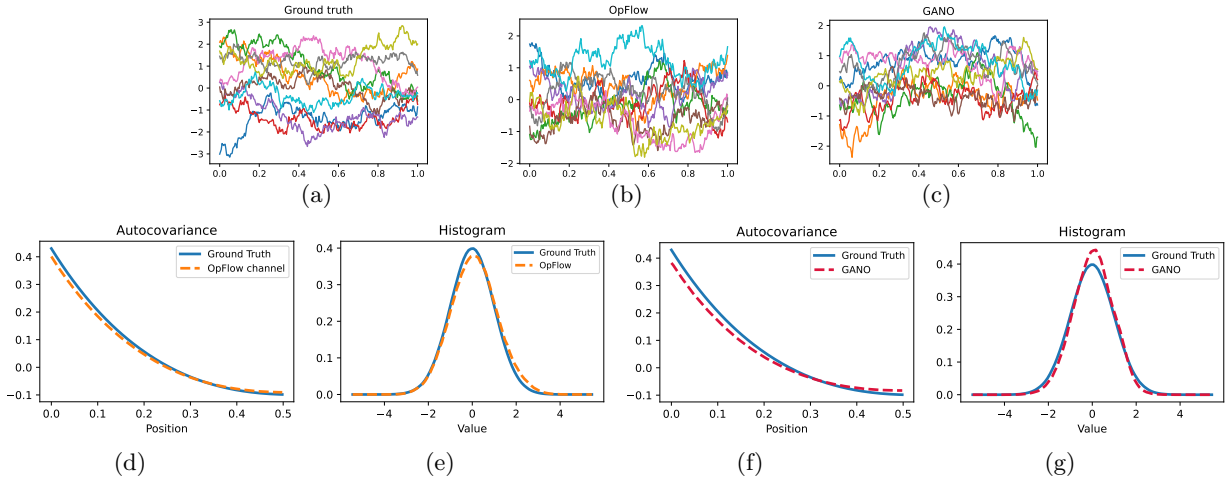


Figure 13: Codomain OpFlow generation performance on GP data. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from GANO. (g) Histogram of samples from GANO.

Codmain TGRF data. We set the hyperparameters to $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5$ and $l_{\mathcal{A}} = 0.2, \nu_{\mathcal{A}} = 1.5$, applying bounds of -2 and 2 to the data, with a resolution of 64×64. Fig. 14 demonstrates that, under these parameters, both OpFlow and GANO accurately capture the autocovariance function and histogram distribution. This experiment underscores codomain OpFlow's capability in effectively handling non-Gaussian 2D data. Additionally, super-resolution results included in the appendix reveal that codomain OpFlow is free from artifacts during super-resolution tasks, showcasing its robustness.

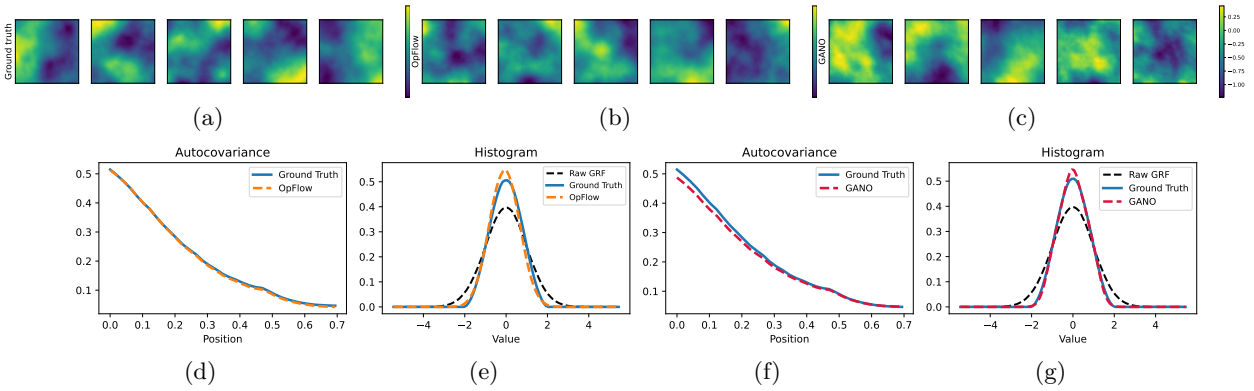
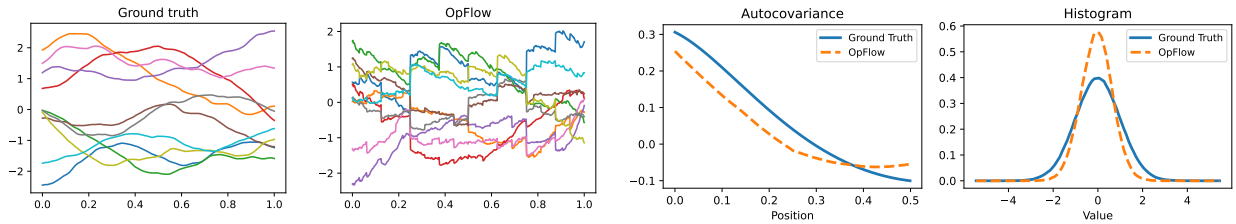


Figure 14: Codomain OpFlow generation performance on TGRF data. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Samples from GANO. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from GANO. (g) Histogram of samples from GANO

A.7 Zero-shot super-resolution generation experiments of OpFlow

Here we present zero-shot super-resolution results with domain-decomposed OpFlow and codomain OpFlow, corresponding to the experiments detailed in the *Generation Tasks* section. All statistics, such as the

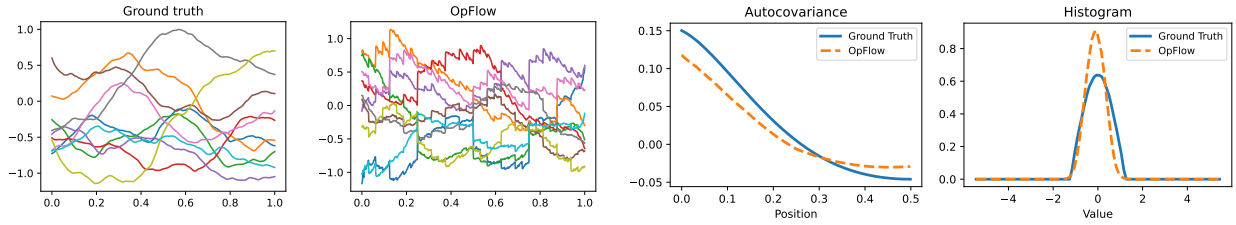
autocovariance function and the histogram of point-wise values, were calculated from 5,000 realizations. Fig. 15 depicts the domain-decomposed OpFlow, which was trained at a resolution of 256 and then evaluated at a resolution of 512. In Fig. 15(a) and (b), the super-resolution samples from domain-decomposed OpFlow display a jagged pattern and show high discontinuities, Additionally, Fig. 15(c) and (d) show that both the autocovariance function and the histogram function evaluated from the domain-decomposed OpFlow diverge from the ground truths. We attribute these artifacts to the domain decomposition operation, which splits the domain into smaller patches and can result in a loss of derivative continuity at the edges of these patches. This artifact is analogous to the jagged artifacts in vision transformers (ViTs), caused by the discontinuous patch-wise tokenization process (Dosovitskiy et al., 2021; Qian et al., 2021).



(a) (b) (c) (d)

Figure 15: Domain decomposed OpFlow super-resolution experiment on generating GP data with resolution of 512. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Autocovariance of samples from OpFlow. (d) Histogram of samples from OpFlow

The super-resolution study was extended to non-Gaussian data, with results illustrated in Fig. 16. Analysis of Fig. 16 leads us to a similar conclusion that the domain-decomposition operation introduces artifacts into the super-resolution outputs of OpFlow. This artifact is also evident in the GRF super-resolution experiment shown in Fig. 17. Here, the domain-decomposed OpFlow was trained at a resolution of 64×64 and evaluated at 128×128, resulting in observable checkerboard patterns in the 2D super-resolution outcomes. Furthermore, the statistical analysis of the samples generated by the domain-decomposed OpFlow, as depicted in Fig. 17, reveals a significant divergence from the ground truth.



(a) (b) (c) (d)

Figure 16: Domain decomposed OpFlow super-resolution experiment on generating TGP data with resolution of 512. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Autocovariance of samples from OpFlow. (d) Histogram of samples from OpFlow

However, in the super-resolution tasks using codomain OpFlow for both GP and non-GP experiments, as shown in Fig. 18 and 19, the jagged or checkerboard artifacts of zero-shot super-resolution performance in OpFlow are gone. This improvement is attributed to codomain OpFlow maintaining the integrity of the entire input domain. Yet, this advantage does come with its own set of challenges, notably an increased difficulty in the learning process, which requires more intensive tuning efforts and may yield results that are less expressive when compared to domain-decomposed OpFlow. Consequently, there exists a trade-off between stronger learning capability and achieving true resolution invariance when we use different partitioning methods of OpFlow.

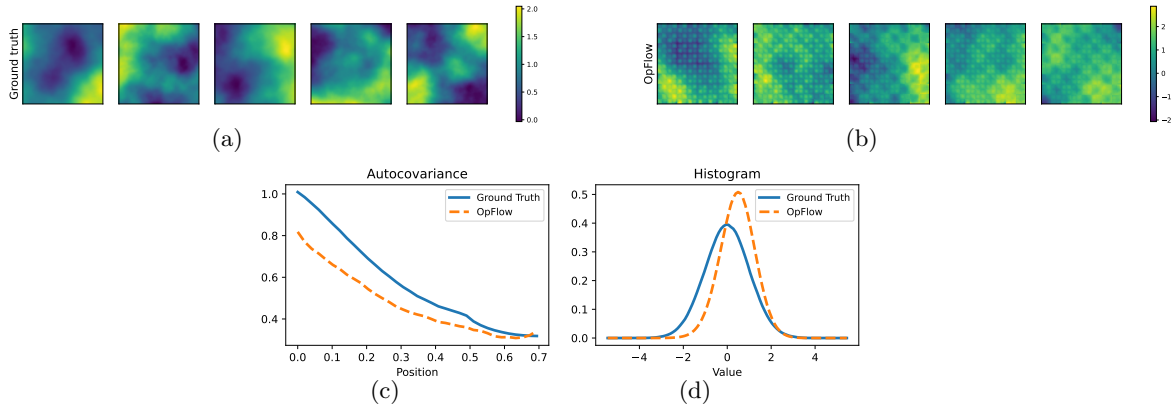


Figure 17: Domain decomposed OpFlow super-resolution experiment on generating GRF data with resolution of 128×128. (b) Samples from groud truth. (c) Samples from OpFlow. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow

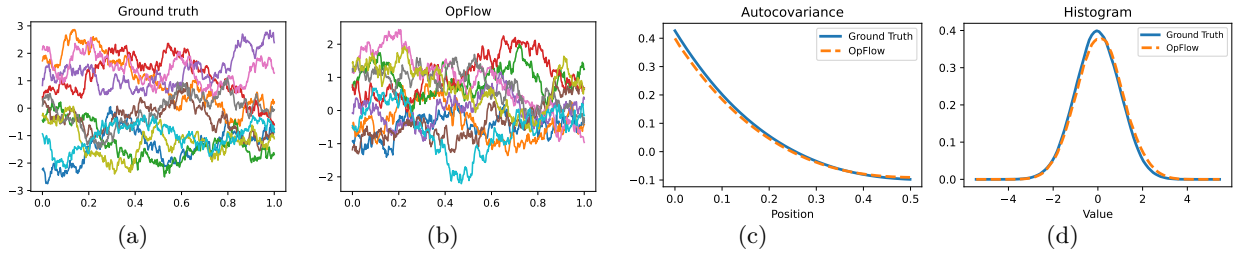


Figure 18: Codomain OpFlow super-resolution experiment on generating GP data with resolution of 512. (a) Samples from the ground truth. (b) Samples from OpFlow. (c) Autocovariance of samples from OpFlow. (d) Histogram of samples from OpFlow

A.8 Ablation study

In this ablation study, we investigate the role of 2-Wasserstein (W_2) regularization on the performance of the OpFlow, as detailed in Eq. 4. We set the hyperparameters to $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5$ for the observed space and $l_{\mathcal{A}} = 0.1, \nu_{\mathcal{A}} = 0.5$ for the Gaussian Process (GP) defined in the latent space $\mathcal{A}$, with a chosen resolution of 64×64. The domain-decomposed OpFlow model was trained under two conditions: with and without the incorporation of W_2 regularization during the warmup phase. According to the results depicted in Fig. 20, incorporating W_2 regularization significantly enhanced OpFlow's ability to accurately learn the statistical properties of the input function data. In contrast, the model trained without W_2 regularization demonstrated a notable deficiency in capturing the same statistics, underscoring the critical role of W_2 regularization in the model's performance

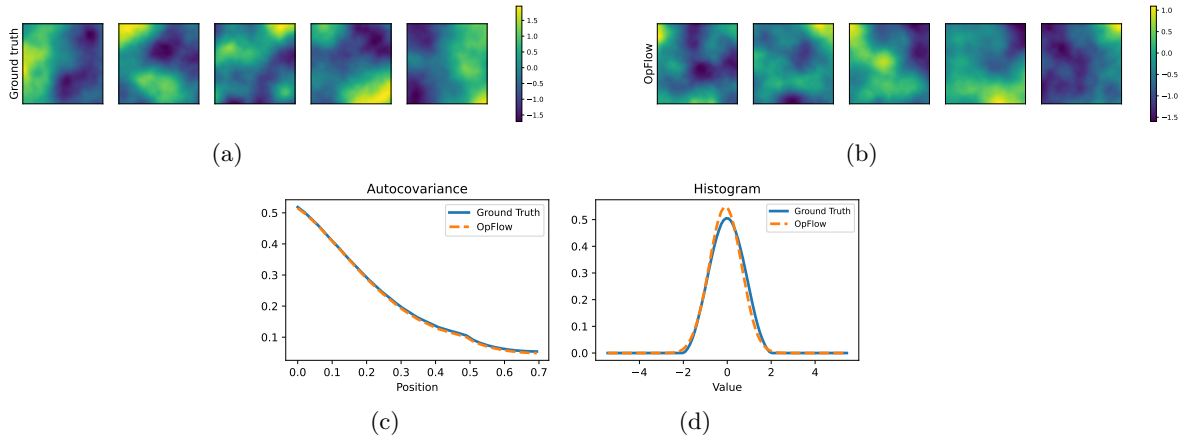


Figure 19: Codomain OpFlow super-resolution experiment on generating TGRF data with resolution of 128×128. (b) Samples from groud truth. (c) Samples from OpFlow. (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow

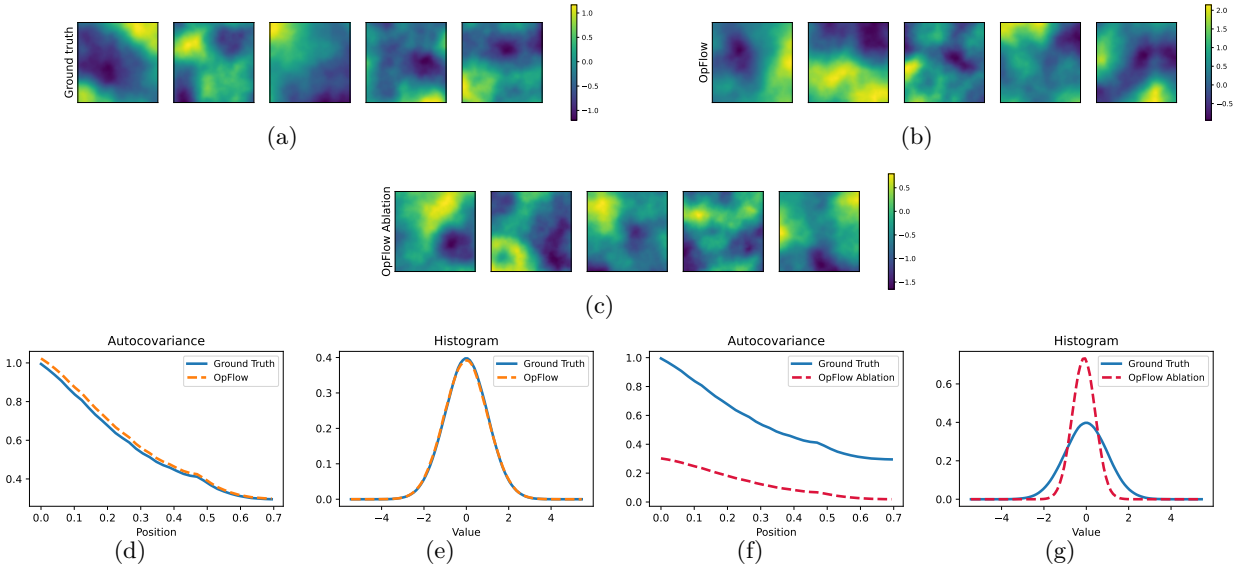
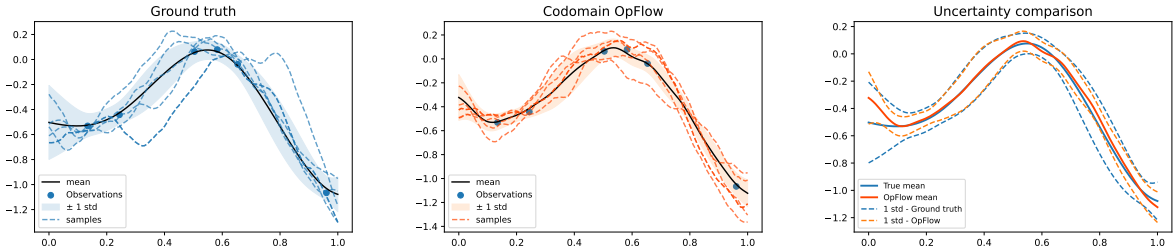


Figure 20: Ablation study of OpFlow generation performance on GRF data (a) samples from the ground truth. (b) Samples from OpFlow. (c) Samples from OpFlow-ablation (OpFlow trained without regularization). (d) Autocovariance of samples from OpFlow. (e) Histogram of samples from OpFlow. (f) Autocovariance of samples from OpFlow-ablation. (g) Histogram of samples from OpFlow-ablation

A.9 Regression with codomain OpFlow

For the training of the codomain OpFlow prior, we selected hyperparameters $l_{\mathcal{U}} = 0.5, \nu_{\mathcal{U}} = 1.5$ for the observed space and $l_{\mathcal{A}} = 0.2, \nu_{\mathcal{A}} = 1.5$ for the Gaussian Process (GP) in the latent space. In our regression task, a new realization of $u \in \mathcal{U}$ was generated, with observations taken at six randomly chosen positions within the domain $\mathcal{D}$. The goal was to infer the posterior distribution of u across 128 positions. Further details on regression parameters available in Table 2. Additionally, we only show the regression results from the first channel of OpFlow generated samples. As depicted in Fig. 21, codomain OpFlow accurately captures both the mean and uncertainty of the true posterior, as well as generating realistic posterior samples that resemble true posterior samples from GPR. However, as discussed in the previous sections, codomain OpFlow is not as effective as domain-decomposed OpFlow for learning the prior, which highly affects the regression performance and may account for the underestimated uncertainty after $x = 0.5$ on the x-axis. This

experiment confirms codomain OpFlow can also be used for URF, which demonstrates the robustness of UFR performance across different OpFlow architectures.



(a) (b) (c)

Figure 21: Codomain OpFlow regression on GP data. (a) Ground truth GP regression with observed data and predicted samples. (b) OpFlow regression with observed data and predicted samples. (c) Uncertainty comparison between true GP and OpFlow predictions.