Random Grid Neural Processes for Parametric Partial Differential Equations

Arnaud Vadeboncoeur [1]   **Ieva Kazlauskaite** [1]   **Yanni Papandreou** [2]   **Fehmi Cirak** [1]   **Mark Girolami** [1 3]

Ömer Deniz Akyildiz [2]

Abstract

We introduce a new class of spatially stochastic physics and data informed deep latent models for parametric partial differential equations (PDEs) which operate through scalable variational neural processes. We achieve this by assigning probability measures to the spatial domain, which allows us to treat collocation grids probabilistically as random variables to be marginalised out. Adapting this spatial statistics view, we solve forward and inverse problems for parametric PDEs in a way that leads to the construction of Gaussian process models of solution fields. The implementation of these random grids poses a unique set of challenges for inverse physics informed deep learning frameworks and we propose a new architecture called Grid Invariant Convolutional Networks (GICNets) to overcome these challenges. We further show how to incorporate noisy data in a principled manner into our physics informed model to improve predictions for problems where data may be available but whose measurement location does not coincide with any fixed mesh or grid. The proposed method is tested on a nonlinear Poisson problem, Burgers equation, and Navier-Stokes equations, and we provide extensive numerical comparisons. We demonstrate significant computational advantages over current physics informed neural learning methods for parametric PDEs while improving the predictive capabilities and flexibility of these models.

[1]Department of Engineering, University of Cambridge, Trumpington St, Cambridge CB2 1PZ. [2]Department of Mathematics, Imperial College London, Exhibition Rd, South Kensington, London SW7 2AZ, United Kingdom. [3]The Alan Turing Institute, British Library, 96 Euston Rd, London NW1 2DB, United Kingdom.. Correspondence to: Arnaud Vadeboncoeur <av537@cam.ac.uk>.

1. Introduction

Partial differential equations (PDEs) are of central importance in natural sciences and engineering as models for describing physical phenomena. Numerical solvers of these equations have received immense attention over the decades with the development of methods such as finite differences, finite elements, spectral methods, to name a few (Quarteroni & Valli, 2008). The main computational problems pertinent to PDE modelling can be categorised into two main classes: The task of obtaining a solution to a PDE for a given set of parameters (*forward problem*) and the task of recovering model parameters from solution fields or observations of solution fields (*inverse problem*) (Belov, 2012; Stuart, 2010). Solving each of these problems efficiently and accurately remains an open problem in the field of scientific computing. Both problems are of fundamental importance in scientific and engineering applications and creating efficient and scalable methods for solving these problems could have large ramifications for engineering practice. Recently, physics informed machine learning (ML) (Raissi et al., 2019) offers a novel perspective on solving these types of problems for parametric PDEs and has been shown to be advantageous to train large networks to learn parametric PDEs for ranges of parameters (Lu et al., 2021a; Li et al., 2020). These models can then be deployed to solve PDEs in real-time. The central objective of this paper is to introduce accurate and uncertainty aware physics informed probabilistic deep learning methods that go beyond *fixed grid* approaches to parametric PDEs, allowing for a synthesis with *noisy* data measured on arbitrary grids for forward and inverse problems.

1.1. Contributions

In this paper, we propose a new framework for jointly learning probabilistic mappings of forward and inverse problems of parametric PDEs using ideas from spatial statistics (Ripley, 2005; Cressie & Moores, 2021). This new approach connects neural Gaussian process models of PDE solution fields with physical parameters through conditioning on random domain partitions. More precisely, we propose (1) a physics driven variational inference framework based on random grids; (2) new kernels for the learning of Gaussian random fields; (3) a new grid invariant architecture to enable learning through random collocation. We demonstrate our

methods on three PDEs of increasing complexity. The first is a 1D nonlinear Poisson PDE where we learn the mapping of a spectral representation of a diffusion field to the solution field for a range of forcing conditions. The second PDE is the spatio-temporal Burgers equation learned for a range of diffusion and non-linearity coefficients and of parameterized initial conditions. The third PDE is the incompressible Navier-Stokes lid-driven cavity flow problem where we map density and viscosity coefficients to spatio-temporal 3D solution fields. Furthermore, we demonstrate how to correctly incorporate sparse, *noisy* observations of sample solution fields to improve the predictive capabilities of the model. We furthermore compare our proposed method to existing methods such as Physics Driven Deep Latent Variable Models (PDDLVM) (Vadeboncoeur et al., 2022), modified DeepONets (Lu et al., 2021a), and Physics-Informed Parametric Fourier Feature Networks (Wang et al., 2021a) which we adapt to solve both forward and inverse problems as described in (Zhao et al., 2022).

1.2. Related Work

In this section, we review relevant work in the literature. We focus our review on machine learning methods for inference in parametric PDEs.

Supervised and Semi-Supervised Operator Learning: These methods for learning differential operators are based either entirely or partially on *noiseless* PDE solution datasets created with classical numerical methods such as FEM or spectral methods. Such methods include Fourier Neural Operators (Li et al., 2020; Fanaskov & Oseledets, 2022; Tripura & Chakraborty, 2023). Other methods which use a combination of pre-computed solutions from classical solvers and physics informed losses include Physics Informed Neural Operators and DeepONets (Li et al., 2021; Lu et al., 2021a). Although these methods have been shown to be effective in certain scenarios, they fundamentally rely on classical solvers in order to learn, and so are bound to their natural limitations for learning new problems. The limitations include the need to recompute models from scratch for new parameter instances and rely on CPU-driven operations that do not parallelize easily. Furthermore, these methods generally focus on learning the operator from one function space to another and are typically evaluated on fixed grids. This is restrictive in cases where the domain geometry, boundary, and initial conditions cannot be easily defined in a function space form. Instead, we focus our work on PDEs parameterized by sets of scalar coefficients. The parametric representation overcomes many of the limitations outlined in Lu et al. (2022); Tang et al. (2023).

Physics Informed Variational Autoencoders: Several methods have been proposed to adapt the popular Variational Autoencoder (VAE) (Welling & Kingma, 2014) framework to physics problems. Such methods include physics-informed VAEs (Zhong & Meidani, 2023), physics integrated VAEs (Takeishi & Kalousis, 2021), autoencoding PDEs (Tait & Damoulas, 2020), and physics-informed dynamical VAEs (Glyn-Davies et al., 2022). While these methods rely on variational inference like our approach, they aim at solving a fundamentally different problem as they relate the observable space to a *discovered* physical latent space. In contrast, we relate solution fields to physical latent spaces and may use some data from the observation space to enhance these mappings but we do not directly map parameters to the observation space (but rather to the solution space). Furthermore, we are interested in methods that can perform inference in the absence of data and are only supplemented/improved by data.

Neural Processes: Neural processes are a new class of deep probabilistic regression tools (Garnelo et al., 2018b). Neural and conditional neural processes (Markou et al., 2022; Garnelo et al., 2018a) output Gaussian uncertainty estimates around predictions similar to Gaussian processes, but they do not suffer from the same training scalability issues. These have partially been adapted to physics problems in Yang & Perdikaris (2019b;a), however, they are essentially data-driven rather than physics driven methods.

Methods related to the probabilistic modelling of single instances of solution fields are pertinent to the proposed methodology, such as (Long et al., 2022; Tronarp et al., 2022). Further methods such as (Lu et al., 2021b) compute inverse problems by posing an optimisation problem. Other works parameterize physics informed Gaussian processes (Pang et al., 2019; Long et al., 2022; Zhang et al., 2022; Chen et al., 2021). These methods are not adapted to parametric PDE scenarios as described in Bhattacharya et al. (2021). We also note the work of Ardizzone et al. (2018) which uses invertible networks to solve inverse problems and the methods used for discovering dynamics, e.g., Raissi & Karniadakis (2018); Brunton et al. (2016). However, these methods have different objectives than solving classic inverse problems and focus on hidden dynamics and discovering coordinate transformations. Other works (Chiu et al., 2022) have explored the use of sets of random collocation points drawn for every residual evaluation, but this was studied for single solution field instances, not parametric PDEs, and only for the forward problem.

2. Background

2.1. Physics Informed ML for Forward Problems

We formulate the nonlinear parametric PDE of interest as

$$\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x) = 0, \quad x \in \Omega, \tag{1}$$

$$\mathcal{B}_{\mathbf{w}}(u)(x) = 0, \quad x \in \partial\Omega, \tag{2}$$

where $\Omega \subset \mathbb{R}^d$, $\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}$ is a nonlinear differential operator, $\mathbf{z}$ is a set of parameters for which we solve the inverse problem, $\mathbf{w}$ is an extra set of model parameters for which we would like to learn the forward and inverse maps[1]. Similarly, $\mathcal{B}_{\mathbf{w}}$ is the boundary operator. For the forward problem, consider the problem of learning a forward parametric emulator $f_\alpha : \mathbf{z}, \mathbf{w}, x \to u(x)$ where α denotes the parameters of the emulator. This can be informally formulated as the approximation problem of finding $\alpha_\star$ s.t.

$$\mathcal{G}_{\mathbf{z}}^{\mathbf{w}} \circ f_{\alpha_\star}(\mathbf{z}, \mathbf{w})(x) \approx 0, \tag{3}$$

given the boundary conditions, following (1). This can be formulated into an optimization problem or can inform a probabilistic model (Kaltenbach et al., 2023; Gao et al., 2022; Zhao et al., 2022). In the context of PDEs, the residual is defined as $r = \mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x)$. When $r(x) = 0$ then $u(x)$ is the solution to the PDE (1).

Another important aspect to consider is the boundary conditions. In general, there are many ways to enforce boundary conditions (Raissi et al., 2019). In this work, we use the hard enforcement method. We can enforce hard boundary conditions for Dirichlet problems (Rao et al., 2021; Sukumar & Srivastava, 2022) with a linear transformation of the solution field

$$u(x) = B(x) + D(x)N(x), \tag{4}$$

where $B(x)$ is an arbitrary function that satisfies the boundary conditions and $D(x) = 0$ for $x \in \partial\Omega$, and $D(x) \neq 0$ for $x \in \Omega$. Other formulations exist for mixed enforcement of boundary conditions as well as for more complicated boundary conditions.

2.2. Physics Informed ML for Inverse Problems

We now consider the problem of fitting a deterministic parametric inverse emulator denoted as $h_\beta : u(x), \mathbf{w} \to \mathbf{z}$ where we seek $\beta_\star$ s.t.

$$h_{\beta_\star}(f_{\alpha_\star}(\mathbf{z}, \mathbf{w}, \mathbf{X}), \mathbf{w}) \approx \mathbf{z}, \tag{5}$$

for any subset $\mathbf{X}$ of the domain Ω. Similarly to the forward case, this can be done using an optimization or a probabilistic formulation (Vadeboncoeur et al., 2022). We note that the inverse emulator approximation problem proposed here relies on a trained forward emulator and is free from classical numerical solvers. In what follows, we develop a comprehensive probabilistic framework to tackle such problems in a principled way.

3. Model Derivation

In this section, we derive the training objective for two cases of our model: physics informed model in Sec. 3.1

[1]The inverse problem is defined here w.r.t. $\mathbf{z}$, not $\mathbf{w}$.

and physics and data informed model in Sec. 3.2. In Appendix A.3 we derive a model for incorporating observations of nonlinear transformations of solution fields and noisy input parameters e.g. measurements of drag coefficients.

3.1. Physics Informed Probabilistic Framework

Function Space Model: We begin the derivation of the physics informed probabilistic model by introducing all distributions of interest with a random field over our solution field. Our hierarchical probabilistic model is defined as

$$r|u, \mathbf{z}, \mathbf{w} \sim \mathcal{GP}(\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x), k_r(x, x'; u, \mathbf{z}, \mathbf{w})), \tag{6}$$

$$\mathbf{z}|u, \mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_\beta(u), \boldsymbol{\Sigma}_\beta(u, \mathbf{w})), \tag{7}$$

$$u \sim \mathcal{GP}(\mu_u(x), k_u(x, x')), \tag{8}$$

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w}}, \boldsymbol{\Sigma}_{\mathbf{w}}). \tag{9}$$

In this model, Eq. (6) defines a probability distribution over the residual and informs the model with physics. Eq. (7) is the *inverse emulator* to be learned during training for the inverse map. Finally, (8) and (9) define the priors over u and $\mathbf{w}$. Leveraging the GP view allows us to choose from several possible choices of kernels for our different distributions. Some of these include

$$k_\theta(x, x') = \epsilon_\theta \delta_{x,x'}, \tag{10}$$

$$k_\theta(x, x') = \lambda_\theta(x)\delta_{x,x'}, \tag{11}$$

$$k_\theta(x, x') = \lambda_\theta(x)\delta_{x,x'} + \langle V_\theta(x), V_\theta(x')\rangle. \tag{12}$$

Here θ denotes learnable functions or parameters. The kernels in 10–12 correspond to a fixed white noise process, a heteroscedastic white noise process (diagonal kernel), and a degenerate deep low-rank covariance matrix (also known as the left Gram matrix (Rahman et al., 2022; Williams & Rasmussen, 2006)), respectively. We elaborate on (12) in Appendix B. These kernels are chosen for their favorable scalability properties.

Function Space Variational Family: We next introduce the variational family in function space (which will be discretized later). For this, we define

$$u|\mathbf{z}, \mathbf{w} \sim \mathcal{GP}(\mu_\alpha(x; \mathbf{z}, \mathbf{w}), k_\alpha(x, x'; \mathbf{z}, \mathbf{w})), \tag{13}$$

$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}}, \boldsymbol{\Sigma}_{\mathbf{z}}) \tag{14}$$

where (13) is a flexible variational family parameterized by a neural network to learn the *forward emulator*. Eq. (14) define the prior in the variational family while the variational distribution over $\mathbf{w}$ is the same as the prior in Eq. (9).

Discretization through Conditioning: To obtain a tractable algorithm, we condition all distributions on a set of N (grid) points $\mathbf{X} \subset \Omega$ in the domain of the PDE. We

achieve this by assigning a joint probability measure on $\Omega^{\otimes N}$ which we denote as $p(\mathbf{X})$. Sampling from this measure and conditioning on the sample effectively discretizes the Gaussian processes. Given the sample, we can discretize our distributions through conditioning on $\mathbf{X}$ as in Cressie & Moores (2021) and convert distributions defined on the function space into multivariate normal distributions (MVN) (Rudner et al., 2021). More precisely, through conditioning on $\mathbf{X}$, we can convert the infinite dimensional model given in (6)–(9) into a conditional finite-dimensional model where

$$p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = \mathcal{N}(\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(\mathbf{u}), K_r(\mathbf{X}, \mathbf{X}; \mathbf{u}, \mathbf{z}, \mathbf{w})), \quad (15)$$

$$p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}_\beta(\mathbf{u}, \mathbf{X}, \mathbf{w}), \boldsymbol{\Sigma}_\beta(\mathbf{u}, \mathbf{X}, \mathbf{w})), \quad (16)$$

$$p(\mathbf{u}|\mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}_u(\mathbf{X}), K_u(\mathbf{X}, \mathbf{X})) \quad (17)$$

where $\mathbf{u} = u(\mathbf{X})$, and $\mathbf{r} \in \mathbb{R}^N$ is the discretized residual (a vector). For the conditional residual (15) we choose a white noise kernel (10) and the mean function is given by the PDE evaluated at locations $\mathbf{X}$. The conditional distribution for $\mathbf{z}$ (16) has mean and covariance given as the output of a neural network with learnable β parameters. We call this network the "β-Net" and it relates the probability density of the $\mathbf{z}$ parameter to the solution field $\mathbf{u}$ and the parameters $\mathbf{w}$ given a partition of the domain. The distribution (17) is chosen in practice to be uninformative so that the model predictions are purely influenced by the PDE residual. The prior on $p(\mathbf{w})$ is the same as (9) since it is independent of the grid. Finally, our joint model can be factorized as

$$p(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = \quad (18)$$
$$p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})p(\mathbf{w})p(\mathbf{X}).$$

From this joint model we are interested in the marginal residual which can be obtained by integrating out all other variables (including the grid) as

$$p(\mathbf{r}) = \int p(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) \, d\mathbf{u} \, d\mathbf{z} \, d\mathbf{w} \, d\mathbf{X}. \quad (19)$$

We know that the PDE is solved if $\mathbf{r} = \mathbf{0}$, hence our aim will be to maximize the marginal probability $p(\mathbf{r} = \mathbf{0})$. For this, we discretize our variational approximation in (13)–(14) by conditioning on the grid $\mathbf{X}$. In addition, we can convert the infinite dimensional variational approximation in (13) to a finite dimensional one by conditioning on $\mathbf{X}$ as

$$q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) = \mathcal{N}(\mu_\alpha(\mathbf{X}; \mathbf{z}, \mathbf{w}), K_\alpha(\mathbf{X}, \mathbf{X}; \mathbf{z}, \mathbf{w})) \quad (20)$$

where (20) is a conditional neural process with α learnable weights that relates the probability density of the solution field $\mathbf{u}$ to the two sets of parameters $\mathbf{z}$ and $\mathbf{w}$. This neural process is given by the α-Net which is the forward emulator. In the experiments we alternate between a heteroskedastic white noise kernel (11), and a low-rank kernel (12). Finally, our joint variational approximation can be written as

$$q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})p(\mathbf{w})p(\mathbf{X}). \quad (21)$$

We then set the virtual observable $\mathbf{r} = \mathbf{0}$ (Rixner & Koutsourelakis, 2021; Vadeboncoeur et al., 2022). Using Jensen's inequality we write out the evidence lower bound

$$\log p(\mathbf{r} = \mathbf{0}) \geq \mathcal{F}(\alpha, \beta) \quad (22)$$
$$= \int \log \frac{p(\mathbf{r} = \mathbf{0}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})}$$
$$\times q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})p(\mathbf{w})p(\mathbf{X}) \, d\mathbf{u} \, d\mathbf{z} \, d\mathbf{w} \, d\mathbf{X}.$$

The most important distinction between this objective and all other methods known to us is the marginalization of the spatio-temporal domain through conditioning on random partitions of the domain. This objective can be computed as an expectation of the form

$$\mathcal{F}(\alpha, \beta) = \quad (23)$$
$$\mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r} = \mathbf{0}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})} \right].$$

This expectation can be approximated with Monte Carlo integration. Every Monte Carlo sample evaluation requires a newly sampled set of collocation points akin to methods of variational inference in function space (Burt et al., 2020; Sun et al., 2018). In Alg. 1 we write out the procedure for training a physics informed RGNP. A similar algorithm is used for the data and physics case where we then add a mini-batched data likelihood term in $\mathcal{F}^N(\alpha, \beta)$. The RGNP update starts in the latent space and maps a sampled parameter into the solution field, from this proposed solution field we compute a physics residual and possibly a data likelihood, and then map this proposed solution field back to the parameters space. Because of this fundamental difference in the construction of the ELBO, our method can work in the complete absence of solution field data. Furthermore, our method generates solution fields of PDEs in function space, i.e. they can be evaluated anywhere in the domain.

3.2. Physics and Data Informed Model Derivation

When developing physics emulators to be used in practice, we may have noisy observation of real world physics behaviours. Data of this kind can be of great use when developing better calibrated and more accurate models (Zhong & Meidani, 2023; Takeishi & Kalousis, 2021). In the current state of the art there is a lack of methods that map parameters to solution fields adjusted with data in a statistically principled manner. Many methods which do incorporate data in a Bayesian manner then map parameters to the observation space, which may not be the desired output. In this section we derive a model for direct observation of a noisy solution field with deterministic inputs. The Bayesian approach is to pose a model of the form

$$\mathbf{y}_D = G(\mathbf{z}, \mathbf{w}, \mathbf{X}) + \sigma_n \mathbf{e}, \quad \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (24)$$

where $\mathbf{y}_D$ is the data and σ_n is the observational noise, and G is the mapping from parameters to solution field $\mathbf{u}$. For our method we pose this mapping $G(\mathbf{z}, \mathbf{w}, \mathbf{X})$ to be our stochastic forward model given the α-Net,

$$\mathbf{y}_D^i = \boldsymbol{\mu}_\alpha(\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) \tag{25}$$
$$+ \mathbf{K}_\alpha(\mathbf{X}_D^i, \mathbf{X}_D^i; \mathbf{z}_D^i, \mathbf{w}_D^i)^{\frac{1}{2}} \mathbf{e}_2 + \sigma_n \mathbf{e}_1$$

where $\mathbf{e}_1, \mathbf{e}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and i indexes the observation set. In our framework we then jointly learn the forward model through the α-Net while adjusting the predictions to match the observed noisy dataset taking into account the relevant uncertainties. Our full joint model can then be written as

$$\log p(\mathbf{r} = \mathbf{0}, \mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \mathcal{F}(\alpha, \beta) \tag{26}$$
$$= \sum_i^N \log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i)$$
$$+ \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right].$$

The lower bound on the marginal likelihood of the zero residual and the observed data requires the evaluation of the entire dataset at every iteration. For computational efficiency, we can replace the summation over the entire dataset with a mini-batch approximation as

$$\mathcal{F}(\alpha, \beta) \approx \frac{N}{|M|} \sum_{i \in M} \log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) \tag{27}$$
$$+ \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right].$$

The first term of the ELBO can be efficiently evaluated and the second term can be approximated using a finite sample size through a Monte Carlo approximation of the expectation.

3.3. Grid Invariant Inversion Networks

Central to this new framework is the use of random collocation grids which are sampled every iteration. This poses unique challenges for the inversion network. Convolutional neural networks (CNNs) require information to be given on a fixed uniform grid and are thus unsuitable for our task. A natural solution for passing information originating on random grids to a CNN would be to use a kernel interpolation method such as the Nadaraya–Watson kernel estimator (Cai, 2001) to interpolate points at random locations to fixed input locations. However, to capture information from different length scales, a very fine projection grid is required. Such grids are computationally expensive and scale as $\mathcal{O}((nm)^d)$, where m is the number of points on the projection grid. Borrowing ideas from Li et al. (2020) and kernel feature space methods (Scholkopf et al., 1999), we develop a scalable alternative to fine interpolation grids. We first project the

---

Algorithm 1 Pseudocode for RGNP

Initialise: α_0, β_0, T (number of iterations), N (number of Monte Carlo samples), and choose $p(\mathbf{z})$, $p(\mathbf{w})$, $p(\mathbf{X})$.
for $t = 1, \ldots, T$ **do**
  for $i = 1, \ldots, N$ **do**
    Sample $\mathbf{X}^{(i)} \sim p(\mathbf{X})$
    Sample $\mathbf{z}^{(i)} \sim p(\mathbf{z})$
    Sample $\mathbf{w}^{(i)} \sim p(\mathbf{w})$
    Sample $\mathbf{u}^{(i)} \sim q_{\alpha_{t-1}}(\mathbf{u}|\mathbf{z}^{(i)}, \mathbf{w}^{(i)}, \mathbf{X}^{(i)})$
  end for
  Compute $\mathcal{F}^N(\alpha, \beta)$ using Monte-Carlo.
  $(\alpha_t, \beta_t) \leftarrow \text{ADAM}(\alpha_{t-1}, \beta_{t-1}, \mathcal{F}^N(\alpha, \beta))$
end for

---

spatially dependent inputs $u(x) \in \mathbb{R}^{d_s}$ to a learned higher dimensional space through a small fully connected neural network $P(u(x), x) = v(x)$ where $v(x) \in \mathbb{R}^{d_v}$ and $x \in \Omega \subset \mathbb{R}^d$. We then project each dimension of $v(x)$ at random locations x onto its own fixed location coarse grid x^* through the Nadaraya-Watson estimator

$$I_k(v(x), x^*) = \frac{\sum_i \phi_k(x^*, x_i) v(x_i)}{\sum_i \phi_k(x^*, x_i)}, \tag{28}$$

where each of the d_v kernels has its own learnable length scale initialized at several different orders of magnitude. A visual representation of this can be seen in the Appendix, Fig. 5. Each coarse grid can then represent information at different lengthscales and the complexity grows as $\mathcal{O}(d_v(nm')^d)$ where m' is the new grid density of the interpolation grid which can now be chosen to be arbitrarily coarse and in practice is chosen as coarse as a 10 points/dimension lattice. We can choose the kernel $\phi_k(\cdot, \cdot)$ to be any number of distance measuring functions with learnable parameters such as RBF, Matern, etc. We then define the grid invariant convolutional network (GICNet) as

$$G(u, \mathbf{w}, x) = \text{Conv}(\cdot, \mathbf{w}) \circ I_{1:d_v}(\cdot, x^*) \circ P(\cdot, x) \circ u(x)$$

where the projection layer $P(u(x), x)$, the interpolation layer $I(v(x), x)$ along with the kernel $\phi(x^*, x)$ lengthscale and the convolutional and fully connected layers are jointly trained. Other architectures than a convolutional network can be used for mapping the d_v intermediary function representations to the output PDE parameters. Architectures of interest include Fourier Neural Operator layers (Li et al., 2020) and Wavelet Neural Operator layers (Tripura & Chakraborty, 2023) as well as fully connected layers. Each of these may bring certain information processing advantages for PDE inversion networks to be explored in future work.

4. Experiments

In this section, we test our method on a series of PDEs and compare our method to alternative approaches. We outline

the setup for the three parameterized PDEs used for comparisons, namely a 1D nonlinear Poisson problem, the Burgers equation, and a non-stationary lid-driven cavity flow Navier-Stokes problem. The chosen testing metrics are the mean normalized squared error (see Appendix, (99)) averaged over 1000 independent samples of $\mathbf{z}, \mathbf{w}$ drawn from their priors (100 samples in the Navier-Stokes examples) solved using FEniCS (Logg et al., 2012). We set the ϵ_r value in the residual kernel to a value of 10^{-2} other than for the Burgers example as explained in Sec. 4.1.2.

4.1. The PDEs

In this section, we describe the PDEs used in the testing of the method along with the relevant boundary conditions and their parametrizations.

4.1.1. NONLINEAR POISSON 1D

The first testing setup is a nonlinear 1D Poisson problem. The variable $\mathbf{z}$ which we want to learn in the inverse problem are coefficients to a Chebyshev expansion describing the diffusion field. The variable $\mathbf{w}$ over which we are marginalizing is a scalar representing a constant forcing over the domain. We write out the PDE $\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x)$ is given as

$$\frac{\partial}{\partial x}\left(k(u,x)\frac{\partial u(x)}{\partial x}\right) - w = 0, \qquad (29)$$

$$k(u,x) = \log\left(1 + \exp\left(u(x)\sum_{i=0}^{n_z} z_i\phi_i(x)\right)\right) + 0.1.$$

for $\Omega = [-1, 1]$, where the Dirichlet boundary conditions are $x(-1) = 0$, $x(1) = 0$. Furthermore, we enforce boundary conditions in an exact manner with,

$$\bar{\mathbf{u}} = B(\mathbf{X}) + D(\mathbf{X})\mathbf{u}, \quad \mathbf{u} \sim q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) \qquad (30)$$

where $B(x)$ captures the boundary conditions. The prior distributions are $p(z_i) = \mathcal{U}(-1, 1)$ and $p(w) = \mathcal{U}(1, 2)$.

4.1.2. BURGERS EQUATION

The second PDE used to test the method is the parametric Burgers equation with 2 spatio-temporal dimensions (one space, one time). Here the $\mathbf{z}$ parameters control the scaling of the nonlinear and the diffusive term and the $\mathbf{w}$ scalar modifies the parametric initial conditions. This allows us to learn forward and inverse mappings for a continuum of initial conditions of the given form. The PDE $\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x)$ is defined as

$$\frac{\partial u(x,t)}{\partial t} + z_1 u(x,t)\frac{\partial u(x,t)}{\partial x} - z_0\frac{\partial^2 u(x,t)}{\partial x^2} = 0, \quad (31)$$

and the boundary and initial conditions are

$$u(-1,t) = u(1,t) = 0, \qquad (32)$$

$$u(x,0) = \sin(2\pi wx)\sin(\pi x), \qquad (33)$$

for a set domain given as $\Omega[-1, 1], \quad 0 \le t \le 1$ (Duffin, 2022). The priors are: $p(z_0) = \mathcal{U}(10^{-2}, 10^{-1})$, $p(z_1) = \mathcal{U}(0.5, 1)$, $p(w) = \mathcal{U}(0.5, 2)$. As in the nonlinear Poisson example we enforce boundary conditions in an exact manner with (30). In this example, we make the standard deviation of the residual kernel value (ϵ_θ in (10)) a learnable parameter which drastically increases stability while maintaining accuracy. It converges to a value in the range of 10^{-1}.

4.1.3. NAVIER-STOKES EQUATIONS

The final PDE used to test the method is the incompressible Navier-Stokes non-stationary lid-driven cavity flow. This is a setup for studying fundamental aspects of confined fluid-flows (Botella & Peyret, 1998). The solution field is a 3-dim. vector field defined as the velocity and pressure $[u, p]^\top$, where $u = [u_1, u_2]^\top$ are the horizontal and vertical velocities. We do not define a $\mathbf{w}$ variable for these experiments. The equations for $\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x)$ describing the dynamics are given by

$$z_1\frac{\partial u}{\partial t} + z_1 u \cdot \nabla u + \nabla p - z_2\nabla^2 u = 0, \qquad (34)$$

$$\nabla \cdot u = 0 \qquad (35)$$

over a square domain, $\Omega(x, y, t)$ between 0 and 1. We define a non-stationary boundary condition on the top edge of the square domain as

$$u_1(x,1,t) = (1 - (2x - 1)^6)t, \qquad (36)$$

and all other boundary conditions of u_1, u_2 are zero. The α and β distributions are then defined as

$$u_1, u_2, p|\mathbf{z} \sim \mathcal{GP}(\mu_\alpha(x; \mathbf{z}), k_\alpha(x, x'; \mathbf{z})), \qquad (37)$$

$$\mathbf{z}|u_1, u_2, p \sim \mathcal{N}(\boldsymbol{\mu}_\beta(u_1, u_2, p), \boldsymbol{\Sigma}_\beta(u_1, u_2, p)). \qquad (38)$$

The z_1 variable corresponds to the fluid density and the z_2 parameter corresponds to the dynamic viscosity. Here $p(z_1) = \mathcal{U}(0.8, 1)$ and $p(z_2) = \mathcal{U}(0.1, 1)$. For this example, we enforce exact boundary conditions with (30) but enforce a soft divergence as an extra residual appended in the residual vector (scaled $10\times$).

4.2. Comparisons of Physics Informed Model

In this section we describe the physics informed comparisons for the nonlinear Poisson and Burgers equation. We compare our diagonal and low-rank covariance methods (RGNP-D, RGNP-LR) to a fixed grid PDDLVM (Vadeboncoeur et al., 2022), a Fourier Feature Net (FFNet) forward emulator (Wang et al., 2021a) with a GICNet inversion network on a fixed grid, and a physics informed DeepONet

---

[2]We used a reduced batched size of 5 collocation sets because of memory constraints. The same number of gradient updates were used hence the similar run-times for the 2.5k and 10k examples.

Table 1. Comparisons of Physics Informed Models

METHOD	N. COLL	MNSE u	MNSE $\mathbf{z}$	u IN 2σ	$\mathbf{z}$ IN 2σ	TIME
NL POISSON 1D						
RGNP-D	30	$\mathbf{9.21 \cdot 10^{-5} \pm 8.66 \cdot 10^{-4}}$	$1.48 \cdot 10^{-2} \pm 2.41 \cdot 10^{-2}$	**95.6%**	99.7%	6.31
RGNP-LR $\alpha\, 2, \beta\, 1$	30	$2.63 \cdot 10^{-4} \pm 1.88 \cdot 10^{-3}$	$6.72 \cdot 10^{-2} \pm 5.96 \cdot 10^{-2}$	94.7%	68.1%	6.19
PDDLVM	30	$1.69 \cdot 10^{-4} \pm 1.36 \cdot 10^{-3}$	$\mathbf{8.10 \cdot 10^{-3} \pm 1.36 \cdot 10^{-3}}$	92.3%	**95.2%**	4.83
FFNET & GICNET	30	$1.90 \cdot 10^{-4} \pm 2.04 \cdot 10^{-3}$	$6.75 \cdot 10^{-1} \pm 1.57 \cdot 10^{-0}$	–	–	5.06
DEEPONETS & K-I.	30	$6.62 \cdot 10^{-4} \pm 3.41 \cdot 10^{-3}$	$5.58 \cdot 10^{-2} \pm 4.62 \cdot 10^{-2}$	–	–	3.31
DEEPONETS & K-I.	100	$6.62 \cdot 10^{-4} \pm 3.95 \cdot 10^{-3}$	$3.66 \cdot 10^{-2} \pm 2.48 \cdot 10^{-2}$	–	–	5.36
DEEPONETS & K-I.	300	$7.69 \cdot 10^{-4} \pm 3.90 \cdot 10^{-3}$	$2.58 \cdot 10^{-2} \pm 2.58 \cdot 10^{-2}$	–	–	11.34
BURGERS						
RGNP-D	225	$1.63 \cdot 10^{-4} \pm 1.58 \cdot 10^{-4}$	$9.05 \cdot 10^{-3} \pm 1.73 \cdot 10^{-2}$	99.9%	**100.0%**	73.56
RGNP-LR $\alpha\, 2, \beta\, 0$	225	$\mathbf{9.51 \cdot 10^{-5} \pm 9.73 \cdot 10^{-5}}$	$\mathbf{8.04 \cdot 10^{-3} \pm 1.66 \cdot 10^{-2}}$	99.9%	**100.0%**	73.45
PDDLVM	225	$2.70 \cdot 10^{-1} \pm 2.12 \cdot 10^{-1}$	$5.32 \cdot 10^{-2} \pm 6.49 \cdot 10^{-2}$	**97.7%**	63.0%	59.02
FFNET & GICNET	225	$7.11 \cdot 10^{-1} \pm 1.70 \cdot 10^{-1}$	$1.19 \cdot 10^{-1} \pm 1.75 \cdot 10^{-1}$	–	–	66.45
DEEPONETS & K-I.	225	$9.44 \cdot 10^{-1} \pm 4.68 \cdot 10^{-1}$	$8.66 \cdot 10^{-2} \pm 1.45 \cdot 10^{-1}$	–	–	36.00
DEEPONETS & K-I.	900	$4.16 \cdot 10^{-1} \pm 2.07 \cdot 10^{-1}$	$9.79 \cdot 10^{-1} \pm 7.15 \cdot 10^{-1}$	–	–	105.61
DEEPONETS & K-I.	2.5K	$9.65 \cdot 10^{-2} \pm 1.05 \cdot 10^{-1}$	$5.04 \cdot 10^{-1} \pm 2.32 \cdot 10^{-1}$	–	–	281.14
DEEPONETS & K-I.[2]	10K	$3.71 \cdot 10^{-3} \pm 8.22 \cdot 10^{-3}$	$2.01 \cdot 10^{-2} \pm 5.35 \cdot 10^{-2}$	–	–	284.90

Table 2. Comparisons of Physics and Noisy Data Informed Models

METHOD	N. COLL	MNSE u	MNSE $\mathbf{z}$	u IN 2σ	$\mathbf{z}$ IN 2σ	TIME
NL POISSON 1D						
RGNP-D	30	$\mathbf{1.59 \cdot 10^{-5} \pm 5.26 \cdot 10^{-5}}$	$\mathbf{5.45 \cdot 10^{-3} \pm 6.25 \cdot 10^{-3}}$	97.2%	99.7%	6.21
RGNP-LR $\alpha\, 2, \beta\, 1$	30	$3.56 \cdot 10^{-5} \pm 2.48 \cdot 10^{-4}$	$3.43 \cdot 10^{-2} \pm 7.40 \cdot 10^{-2}$	98.6%	62.9%	6.47
PDDLVM	30	$1.67 \cdot 10^{-5} \pm \mathbf{3.60 \cdot 10^{-5}}$	$7.26 \cdot 10^{-3} \pm 7.23 \cdot 10^{-3}$	**95.3%**	**97.6%**	4.23
FFNET & GICNET	30	$2.85 \cdot 10^{-4} \pm 8.76 \cdot 10^{-4}$	$4.36 \cdot 10^{-1} \pm 5.76 \cdot 10^{-1}$	–	–	5.59
DEEPONETS & K-I.	30	$1.55 \cdot 10^{-4} \pm 2.77 \cdot 10^{-4}$	$9.70 \cdot 10^{-2} \pm 8.99 \cdot 10^{-2}$	–	–	3.94
DEEPONETS & K-I.	100	$2.17 \cdot 10^{-4} \pm 2.86 \cdot 10^{-4}$	$1.29 \cdot 10^{-1} \pm 1.52 \cdot 10^{-1}$	–	–	5.67
DEEPONETS & K-I.	300	$3.27 \cdot 10^{-4} \pm 4.44 \cdot 10^{-4}$	$6.97 \cdot 10^{-2} \pm 4.56 \cdot 10^{-1}$	–	–	11.70

(Wang et al., 2021b) with a test-time kernel interpolation (K-I) layer with a convolutional neural net on a fixed grid. Methods other than ours rely on creating a dataset of 1k input pairs of $\mathbf{z}$, $\mathbf{w}$ variables for the Poisson problem, and 10k samples for the Burgers example. Table 1 summarizes the results for all methods for the two PDE setups. We report the MNSE as in (99) and its standard deviation over the 1000 testing samples. We also report the percentage of predictions within 2σ of the ground truth solution and the total training times in minutes. Test time inference is in the order of $10^{-2} - 10^{-3}$ seconds. The column labeled "N. COLL" denotes the number of collocation points used at every residual computation step. We use a batch size of 50 samples for the Poisson problem, and a batch size of 25 for the Burgers equation (this is reduced to a batch size of 5 for the 10k collocation DeepONet & kernel-interpolation case because of excessive memory consumption). We train the Poisson problem for 20k gradient updates and we train the Burgers setup for 80k gradient update steps. All learning is

done using the Adam optimizer (Kingma et al., 2015) with a decaying learning rate. In Fig. 1 we show an example output from the α-Net for the solution field of the nonlinear Poisson 1D problem. In Fig. 2 we show example outputs for a solution field for the diagonal model from the α-Net. The main reason for the gain in performance of the probabilistic model is due to the stochastic treatment of the collocation points. By treating the collection of collocation points as random variables to be marginalized through Monte Carlo integration we obtain domain averaged residuals which proves to be advantageous when learning parametric physics.

4.3. Comparison of Physics & Data Informed Model

In this section, we show the results from the physics and data informed model compared to the nonlinear Poisson 1D problem. The setups are similar to the previous section but now the loss for the proposed model is given by (27) and we incorporate noisy observation of solution fields in the inference. We use 1k noisy sample solutions measured at

Table 3. Results for Physics informed models applied to Navier-Stokes lid-driven cavity flow

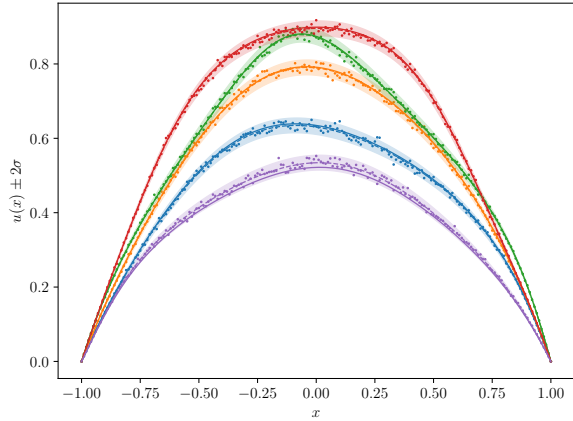| METHOD	N. COLL	MNSE u_1 MNSE u_2	MNSE $\mathbf{z}$	u_1 IN 2σ u_2 IN 2σ	$\mathbf{z}$ IN 2σ	TIME
RGNP-D	512	$2.37 \times 10^{-2} \pm 1.16 \times 10^{-2}$ $3.59 \times 10^{-2} \pm 1.55 \times 10^{-2}$	$6.12 \times 10^{-3} \pm 6.00 \times 10^{-3}$	38.3% 49.8%	66.5%	297.6
RGNP-LR $\alpha\,2, \beta\,0$	512	$2.39 \times 10^{-2} \pm 1.16 \times 10^{-2}$ $3.62 \times 10^{-2} \pm 1.54 \times 10^{-2}$	$4.07 \times 10^{-3} \pm 3.98 \times 10^{-3}$	50.4% 58.0%	66.5%	306.2

Figure 1. Sample results from the Nonlinear Poisson 1D setup for the diagonal model given by the $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})$ distribution. Here, the solid line is the FE solution, the dashed line is the mean estimate, and the scatter points are the random samples; the shaded area is the 2σ uncertainty.

60 locations with a noise standard deviation $\sigma_n = 0.05$. In Fig 6 we show 5 samples of observed solution fields. The algorithm only sees the scatter points, not the solid ground-truth line. The losses for the other methods against which we compare are modified to include a data-fit term as in Wang et al. (2021b) to have a trade-off between fitting the differential operator and the observations. When incorporating data from noisy observations of solution fields, it is of crucial importance to have an inference scheme capable of correctly characterizing the uncertainty inherent in the observations (Kennedy & O'Hagan, 2001) to maximise the accuracy of the inference. A deterministic formulation is not able to take account of the noise in the observations and thus produces a single estimate of the parameters that might over-fit to the noise in the observations.

4.4. Navier-Stokes Lid Driven Cavity Flow

We apply our method to a challenging parametric Navier-Stokes problem. We define the parametric differential operator and boundary conditions for a time-dependent lid-driven cavity flow example using the incompressible Navier-Stokes
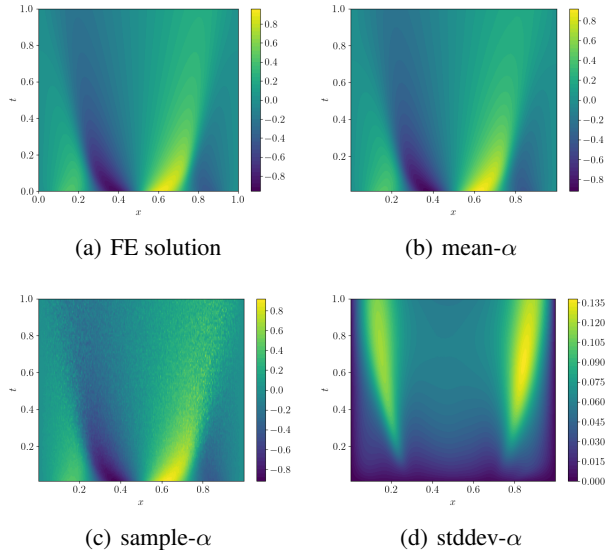

(a) FE solution   (b) mean-α

(c) sample-α (d) stddev-α

Figure 2. Sample result from the α-Net with a diagonal kernel for the Burgers equation. We show the FE solution, the mean estimate, the standard deviation field, and a sample from $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})$.

equations. Here the parameters correspond to the dynamic viscosity and the fluid density. In Table 1 we compute these quantities for 100 independent samples of z and w drawn from the priors. All models are run for 100k gradient update steps. We note that obtaining reliable and accurate reference solutions to Navier-Stokes problems is very challenging.

5. Conclusion

In this paper, we propose a new framework of random collocation neural processes for solving forward and inverse parametric physics problems. Our method leverages spatial statistics, variational inference, neural processes, and a proposed grid-invariant convolutional network to solve forward and inverse problems in a probabilistically coherent manner. Our method is physics informed at training time and can incorporate noisy observations of solutions fields from arbitrary grids in a statistically principled way. We test our method on a nonlinear Poisson problem, the Burgers equation, and the incompressible Navier-Stokes equations.
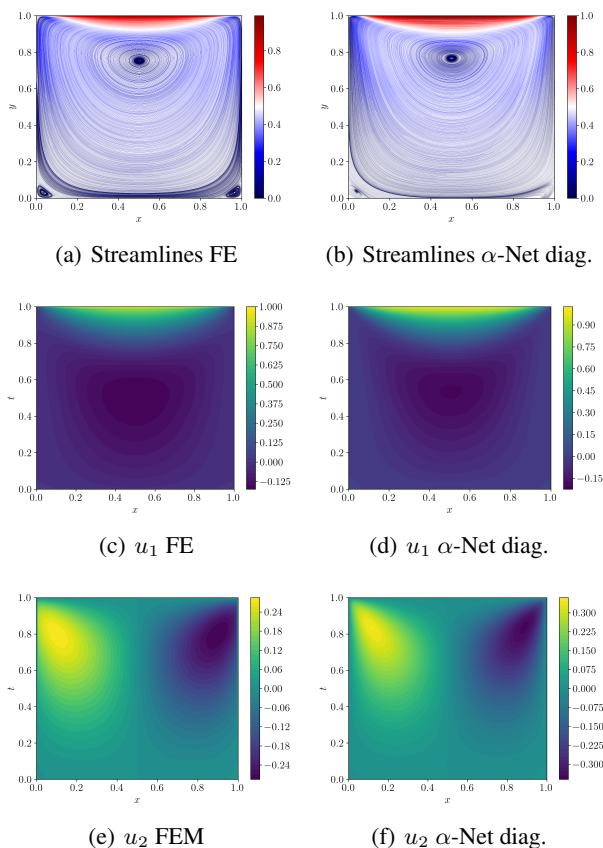
(a) Streamlines FE

(b) Streamlines α-Net diag.

(c) u_1 FE

(d) u_1 α-Net diag.

(e) u_2 FEM

(f) u_2 α-Net diag.

Figure 3. A comparison of the mean solution fields of the α-Net with a diagonal covariance and the FE solution.

We further compare our method with a series of alternative physics and data-informed methods. We find our method is highly competitive with other approaches in terms of accuracy, uncertainty quantification, and compute time. This strongly supports the probabilistic treatment of collocation grids for the physics-informed solution of parametric PDEs.

The uncertainty captured by the proposed methodology reflects the confidence of the model with respect to the given solution fields and parameters. When using these models in practice, the application expert can then use this uncertainty to gauge the reliability of the predictions from the model, an important feature due to the black-box nature of deep learning models. In essence, a practitioner has information helping them assess their confidence in the accuracy of the given solution fields and whether more training is required or if they should use a different solution method outright. Furthermore, the inverse problems are often ill-posed and a range of parameters may yield the observed solution fields; a uncertainty quantification framework can capture this while a deterministic approach cannot.

The performance of our approach on the Navier-Stokes ex-

amples points to some limitations in using uniform distributions over domains for sampling the collocation points. The distributions $p(\mathbf{X})$ can be adapted to sample more points close to boundaries to better capture complex boundary effects; this can be implemented within our framework. Divergence enforcing (Richter-Powell et al., 2022) methods could potentially also be used to accelerate convergence. Chebyshev network architectures could also be investigated (Tang et al., 2023) as an alternative architecture for faster and more regularized residual computation. Further possible extensions include the incorporation of CAN-PINN collocation methods (Chiu et al., 2022). The GP formulation of our method implies that other GP methods such as deep kernels (Wilson et al., 2016) and sparse GPs could be leveraged (Snelson & Ghahramani, 2005; Titsias, 2009). We could also make use of variational weak forms in the residual (Kharazmi et al., 2019) to lower the differentiability order of the PDEs and potentially increase learning stability. The generality of the proposed framework implies that it can be readily extended to incorporate many of the newest advances in physics informed machine learning.

Acknowledgements

References

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

Belov, Y. Y. Inverse problems for partial differential equations. In *Inverse Problems for Partial Differential Equations*. De Gruyter, 2012.

Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI journal of computational mathematics*, 7, 2021. doi: 10.5802/smai-jcm.74.

Botella, O. and Peyret, R. Benchmark spectral results on the lid-driven cavity flow. *Computers & Fluids*, 27(4): 421–433, 1998.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Burt, D. R., Ober, S. W., Garriga-Alonso, A., and van der Wilk, M. Understanding variational inference in function-space. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2020.

Cai, Z. Weighted nadaraya–watson regression estimation. *Statistics & probability letters*, 51(3):307–318, 2001.

Chen, Y., Hosseini, B., Owhadi, H., and Stuart, A. M. Solving and learning nonlinear PDEs with Gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.

Chiu, P.-H., Wong, J. C., Ooi, C., Dao, M. H., and Ong, Y.-S. Can-pinn: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method. *Computer Methods in Applied Mechanics and Engineering*, 395:114909, 2022.

Cressie, N. and Moores, M. T. Spatial statistics. *arXiv preprint arXiv:2105.07216*, 2021.

Dehaene, D. and Brossard, R. Re-parameterizing VAEs for stability. *arXiv preprint arXiv:2106.13739*, 2021.

Ding, J. and Zhou, A. Eigenvalues of rank-one updated matrices with some applications. *Applied Mathematics Letters*, 20(12):1223–1226, 2007.

Duffin, C. Statistical finite element methods for nonlinear PDEs. 2022.

Fanaskov, V. and Oseledets, I. Spectral neural operators. *arXiv preprint arXiv:2205.10573*, 2022.

Gao, H., Zahr, M. J., and Wang, J.-X. Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. Conditional neural processes. In *International Conference on Machine Learning*, pp. 1704–1713. PMLR, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Glyn-Davies, A., Duffin, C., Akyildiz, Ö. D., and Girolami, M. Φ-DVAE: Learning Physically Interpretable Representations with Nonlinear Filtering. *arXiv preprint arXiv:2209.15609*, 2022.

Kaltenbach, S., Perdikaris, P., and Koutsourelakis, P.-S. Semi-supervised invertible neural operators for bayesian inverse problems. *Computational Mechanics*, pp. 1–20, 2023.

Kennedy, M. C. and O'Hagan, A. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

Kharazmi, E., Zhang, Z., and Karniadakis, G. E. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

Kingma, D. P., Ba, J., Bengio, Y., and LeCun, Y. 3rd international conference on learning representations. *ICLR, San Diego*, 2015.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

Logg, A., Mardal, K.-A., and Wells, G. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

Long, D., Wang, Z., Krishnapriyan, A., Kirby, R., Zhe, S., and Mahoney, M. AutoIP: A united framework to integrate physics into Gaussian processes. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 14210–14222. PMLR, 17–23 Jul 2022.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. Physics-informed neural networks with

hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021b.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Markou, S., Requeima, J., Bruinsma, W., Vaughan, A., and Turner, R. E. Practical conditional neural process via tractable dependent predictions. In *International Conference on Learning Representations*, 2022.

Pang, G., Yang, L., and Karniadakis, G. E. Neural-net-induced Gaussian process regression for function approximation and PDE solution. *Journal of Computational Physics*, 384:270–288, 2019.

Petersen, K. B., Pedersen, M. S., et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

Quarteroni, A. and Valli, A. *Numerical approximation of partial differential equations*. Springer, 2008.

Rahman, S., Johnson, V. E., and Rao, S. S. Using the left gram matrix to cluster high dimensional data. *arXiv preprint arXiv:2202.08236*, 2022.

Raissi, M. and Karniadakis, G. E. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

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

Rao, C., Sun, H., and Liu, Y. Physics-informed deep learning for computational elastodynamics without labeled data. *Journal of Engineering Mechanics*, 147(8): 04021043, 2021.

Richter-Powell, J., Lipman, Y., and Chen, R. T. Neural conservation laws: A divergence-free perspective. In *Advances in Neural Information Processing Systems*, 2022.

Ripley, B. D. *Spatial statistics*. John Wiley & Sons, 2005.

Rixner, M. and Koutsourelakis, P.-S. A probabilistic generative model for semi-supervised training of coarse-grained surrogates and enforcing physical constraints through virtual observables. *Journal of Computational Physics*, 434: 110218, 2021.

Rudner, T. G., Chen, Z., Teh, Y. W., and Gal, Y. Tractable function-space variational inference in bayesian neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Scholkopf, B., Mika, S., Burges, C. J., Knirsch, P., Muller, K.-R., Ratsch, G., and Smola, A. J. Input space versus feature space in kernel-based methods. *IEEE transactions on neural networks*, 10(5):1000–1017, 1999.

Snelson, E. and Ghahramani, Z. Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18, 2005.

Stuart, A. M. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010. doi: 10.1017/S0962492910000061.

Sukumar, N. and Srivastava, A. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

Sun, S., Zhang, G., Shi, J., and Grosse, R. Functional variational bayesian neural networks. In *International Conference on Learning Representations*, 2018.

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

Takeishi, N. and Kalousis, A. Physics-integrated variational autoencoders for robust and interpretable generative modeling. *Advances in Neural Information Processing Systems*, 34:14809–14821, 2021.

Tang, S., Feng, X., Wu, W., and Xu, H. Physics-informed neural networks combined with polynomial interpolation to solve nonlinear partial differential equations. *Computers & Mathematics with Applications*, 132:48–62, 2023.

Titsias, M. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial intelligence and statistics*, pp. 567–574. PMLR, 2009.

Tripura, T. and Chakraborty, S. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.

Tronarp, F., Bosch, N., and Hennig, P. Fenrir: Physics-enhanced regression for initial value problems. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 21776–21794. PMLR, 17–23 Jul 2022.

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

Wang, S., Wang, H., and Perdikaris, P. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021a.

Wang, S., Wang, H., and Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40): eabi8605, 2021b.

Welling, M. and Kingma, D. P. Auto-encoding variational bayes. In *ICLR*, 2014.

Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial intelligence and statistics*, pp. 370–378. PMLR, 2016.

Yang, Y. and Perdikaris, P. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019a.

Yang, Y. and Perdikaris, P. Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems. *Computational Mechanics*, 64(2):417–434, 2019b.

Zhang, J., Zhang, S., and Lin, G. PAGP: A physics-assisted Gaussian process framework with active learning for forward and inverse problems of partial differential equations. *arXiv preprint arXiv:2204.02583*, 2022.

Zhao, Q., Lindell, D. B., and Wetzstein, G. Learning to solve PDE-constrained inverse problems with graph networks. In *ICML 2nd AI for Science Workshop*, 2022.

Zhong, W. and Meidani, H. PI-VAE: Physics-informed variational auto-encoder for stochastic differential equations. *Computer Methods in Applied Mechanics and Engineering*, 403:115664, 2023. ISSN 0045-7825.

A. Physics & Data Models

In this section, we write out the full derivation for the physics and data informed models. We include an extra derivation for a model where we observe a nonlinear transformation of the solution field and noisy parameter observations. This results in an additional lower bound. We first derive the model for maximizing the marginal likelihood of the residual and observational data. We then derive the tractable model shown in the main paper and the model for nonlinear solution field observations.

A.1. General Physics & Data Informed Model

We begin by presenting the overall framework for maximizing the marginal likelihood of the residual and the data. We outline the observational model in the subsequent subsections as this depends on the nature of the data we are dealing with. Firstly, as shown in the paper for the physics informed model we write out a joint distribution over all variables of interest to maximize a marginal likelihood. The new marginal likelihood is for the residual and the data observation conditioned on the dataset inputs for these observations. We factorize the model in a similar fashion as the physics informed model including a data likelihood term. For brevity, we begin with the already discretized distributions. We have

$$p(\mathbf{r}, \mathbf{y}_D, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X} | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) = p(\mathbf{r} | \mathbf{u}, \mathbf{z}, \mathbf{w}) p(\mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u} | \mathbf{X}) p(\mathbf{w}) p(\mathbf{X}). \tag{39}$$

We then write out the variational approximation to the joint over the latent variables as

$$q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z}) p(\mathbf{w}) p(\mathbf{X}). \tag{40}$$

Here $\mathbf{y}_D, \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D$ denotes the observations of the solution field, the associated physical parameter, the extra model parameter, and the observation locations of the $\mathbf{y}_D$ data respectively. We can then write out the evidence lower-bound on the marginal likelihood using Jensen's inequality

$$\log p(\mathbf{r} = \mathbf{0}, \mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \mathcal{F}(\alpha, \beta) \tag{41}$$

$$\mathcal{F}(\alpha, \beta) = \int \log \frac{p(\mathbf{r}, \mathbf{y}_D, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X} | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D)}{q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})} q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) \, \mathrm{d}\mathbf{u} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\mathbf{w} \, \mathrm{d}\mathbf{X}. \tag{42}$$

Rewriting the integration for an expectation we obtain

$$\mathcal{F}(\alpha, \beta) = \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log p(\mathbf{r}, \mathbf{y}_D, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X} | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) - \log q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) \right]. \tag{43}$$

We then factorize out the likelihood term from the expectation as this term does not depend on the integration variables,

$$\mathcal{F}(\alpha, \beta) = \log p(\mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log p(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) - \log q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) \right]. \tag{44}$$

We note that the form of the likelihood term depends on the nature of the observation model.

A.2. Physics and Noisy Data Informed Model

We now derive the full ELBO for an observation model where we observe noisy solution fields at point locations. We pose our observation model as

$$\mathbf{y}_D^i = G(\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) + \sigma_n \mathbf{e}_1. \tag{45}$$

In our case, we learn the observation operator $G(\cdot)$ is $q_\alpha(\mathbf{u} | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i)$ which yields a mean and a covariance. We can write this out as the resulting observation model explicitly as

$$\mathbf{y}_D^i = \boldsymbol{\mu}_\alpha(\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) + \mathbf{K}_\alpha(\mathbf{X}_D^i, \mathbf{X}_D^i; \mathbf{z}_D^i, \mathbf{w}_D^i)^{\frac{1}{2}} \mathbf{e}_2 + \sigma_n \mathbf{e}_1, \tag{46}$$

where $\mathbf{e}_1, \mathbf{e}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This corresponds to a product of normal distributions of the form

$$p(\mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) = \prod_{i=0}^{N} \mathcal{N}(\mathbf{y}_D^i; \boldsymbol{\mu}_\alpha(\mathbf{X}^i, \mathbf{z}_D^i, \mathbf{w}_D^i), \bar{\boldsymbol{\Sigma}}_\alpha(\mathbf{X}^i, \mathbf{z}_D^i, \mathbf{w}_D^i)), \tag{47}$$

with a new covariance that takes into account the iid Gaussian noise along with the covariance coming from the α-distribution,

$$\bar{\mathbf{\Sigma}}_\alpha(\mathbf{X}, \mathbf{z}, \mathbf{w}) = \mathbf{K}_\alpha(\mathbf{X}, \mathbf{X}; \mathbf{z}, \mathbf{w}) + \sigma_n^2 \mathbf{I}. \tag{48}$$

We will then approximate the likelihood function over the entire dataset with a mini-batch version which we write out as

$$\log p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \approx \frac{N}{|M|} \sum_{i \in M} \log \mathcal{N}(\mathbf{y}_D^i; \boldsymbol{\mu}_\alpha(\mathbf{X}^i, \mathbf{z}_D^i, \mathbf{w}_D^i), \bar{\mathbf{\Sigma}}_\alpha(\mathbf{X}^i, \mathbf{z}_D^i, \mathbf{w}_D^i)). \tag{49}$$

Bringing all things together from Sec. A.1 we obtain the mini-batched ELBO

$$\log p(\mathbf{r}, \mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \mathcal{F}(\alpha, \beta) \tag{50}$$

$$= \sum_i^N \log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right] \tag{51}$$

$$\approx \frac{N}{|M|} \sum_{i \in M} \log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right]. \tag{52}$$

This objective is a lower-bound on the marginal likelihood of the observational data and the physics residual. This model relies on direct noisy observations of the solution field of interest along with knowing (or deterministically estimating) the generating parameters of the PDE yielding the observations.

A.3. Physics and Indirect Noisy Observations and Noisy Parameters Informed Model

In this section we derive a model that is not included in the main paper that deals with indirect observations of the solution field and noisy measurements of the source parameters of the PDE. In this case, we deal with a general setting where we indirectly observe the solution field through some other known nonlinear mapping. One of the more salient examples of such a scenario is the noisy measurement of drag coefficients given from a fluid flow. Here the PDE describes the fluid velocity, and from a velocity field, we can compute the drag coefficient. Drag coefficients can also be easier to measure than direct solution fields.

For such a model where the parameters are also noisily measured (or estimated probabilistically through a Bayesian inverse problem), we write out the full observation model as

$$\mathbf{y}_D^i = g\left(G(\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) \right) + \mathbf{e}_{\mathbf{y}_D^i}, \quad \mathbf{e}_{\mathbf{y}_D^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{y}_D^i}^2 \mathbf{I}), \tag{53}$$

$$\mathbf{z}_D^i = \tilde{\mathbf{z}} + \mathbf{e}_{\mathbf{z}^i}, \quad \mathbf{e}_{\mathbf{z}^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{z}^i}^2 \mathbf{I}), \tag{54}$$

$$\mathbf{w}_D^i = \tilde{\mathbf{w}} + \mathbf{e}_{\mathbf{w}^i}, \quad \mathbf{e}_{\mathbf{w}^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{w}^i}^2 \mathbf{I}), \tag{55}$$

$$\mathbf{X}_D^i = \tilde{\mathbf{X}} + \mathbf{e}_{\mathbf{X}^i}, \quad \mathbf{e}_{\mathbf{X}^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{X}^i}^2 \mathbf{I}). \tag{56}$$

where $G(\cdot)$ is $q_\alpha(\mathbf{u}|\tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}})$. This yields a collection of Gaussian distributions which we write out as

$$p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) = \mathcal{N}(\mathbf{y}_D^i|g(\tilde{\mathbf{u}}), \epsilon_{\mathbf{y}_D^i}^2 \mathbf{I}), \tag{57}$$

$$p(\tilde{\mathbf{z}}|\mathbf{z}_D^i) = \mathcal{N}(\tilde{\mathbf{z}}|\mathbf{z}_D^i, \epsilon_{\mathbf{z}_D^i}^2 \mathbf{I}), \tag{58}$$

$$p(\tilde{\mathbf{w}}|\mathbf{w}_D^i) = \mathcal{N}(\tilde{\mathbf{w}}|\mathbf{z}_D^i, \epsilon_{\mathbf{w}_D^i}^2 \mathbf{I}), \tag{59}$$

$$p(\tilde{\mathbf{X}}|\mathbf{X}_D^i) = \mathcal{N}(\tilde{\mathbf{X}}|\mathbf{X}_D^i, \epsilon_{\mathbf{X}_D^i}^2 \mathbf{I}). \tag{60}$$

The $\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}$ variables are not directly observed. We can then write out the likelihood term of the observation data as

$$p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) = \prod_i^N p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i), \tag{61}$$

$$\log p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) = \sum_i^N \log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i), \tag{62}$$

$$\log p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \approx \frac{N}{|M|} \sum_{i \in M} \log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i). \tag{63}$$

We then marginalize out the data variables to obtain the marginal likelihood of the data in terms of the available distributions,

$$\log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) = \log \int p(\mathbf{y}_D^i, \tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}^i) \, d\tilde{\mathbf{u}} \, d\tilde{\mathbf{z}} \, d\tilde{\mathbf{w}} \, d\tilde{\mathbf{X}}, \tag{64}$$

$$\log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) = \log \int p(\mathbf{y}_D^i|\tilde{\mathbf{u}})q_\alpha(\tilde{\mathbf{u}}|\tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}})p(\tilde{\mathbf{z}}|\mathbf{z}_D^i)p(\tilde{\mathbf{w}}|\mathbf{w}_D^i)p(\tilde{\mathbf{X}}|\mathbf{X}_D^i) \, d\tilde{\mathbf{u}} \, d\tilde{\mathbf{z}} \, d\tilde{\mathbf{w}} \, d\tilde{\mathbf{X}}. \tag{65}$$

We then use Jensen's inequality to obtain a lower bound on this marginal likelihood in terms of log distributions for computational convenience and apply the same mini-batching as before,

$$\log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) \geq \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} \left[\log p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) \right], \tag{66}$$

$$\log p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \sum_i^N \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} \left[\log p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) \right], \tag{67}$$

$$\approx \frac{N}{|M|} \sum_{i \in M} \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} \left[\log p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) \right]. \tag{68}$$

We can then use the previously derived result from Sec. A.1 to obtain the complete ELBO on the marginal likelihood of the data and the residual,

$$\log p(\mathbf{r}, \mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \mathcal{F}(\alpha, \beta) \tag{69}$$

$$= \sum_i^N \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} \left[\log p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) \right] + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})} \right], \tag{70}$$

$$\approx \frac{N}{|M|} \sum_{i \in M} \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} \left[\log p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) \right] + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})} \right]. \tag{71}$$

With this model, the method can be generalized to incorporate data from multiple sources that are not just direct observations of the solution field. This also opens the doors to the possibility of learning missing dynamics not described in the PDEs in an experimentally feasible way.

B. Low-Rank Covariance Matrix

In this section, we discuss in detail how we can use a low-rank covariance matrix. The low-rank kernel is expressed as

$$k(x, x') = \lambda(x)\delta_{x, x'} + \langle V(x), V(x') \rangle, \tag{72}$$

where $V(x) \in \mathbb{R}^l$ and l denotes the column dimension of the rectangular matrices resulting from this kernel. Here we show that the left Gram matrix $\mathbf{V}\mathbf{V}^\top$ results from this kernel

$$(\mathbf{V}\mathbf{V}^\top)_{ij} = \sum_l \mathbf{V}_{il}\mathbf{V}_{jl}, \tag{73}$$

$$= \sum_l V(x_i)_l V(x_j)_l, \tag{74}$$

$$= \langle V(x_i), V(x_j) \rangle. \tag{75}$$

Expanding out the definition of the kernel for a collection $\mathbf{X}, \mathbf{X}'$ of points we obtain

$$K(\mathbf{X}, \mathbf{X}) = \mathbf{\Lambda} + \mathbf{V}\mathbf{V}^\top, \tag{76}$$

$$\mathbf{\Lambda} = \text{diag}(\boldsymbol{\lambda}), \tag{77}$$

$$\mathbf{V} \in \mathbb{R}^{n \times l}, \tag{78}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^n$ and $\mathbf{V} \in \mathbb{R}^{n \times l}$. We now show how we can sample and evaluate the log-density efficiently without ever constructing the full $n \times n$ matrix.

B.1. Sampling the Low-Rank Covariance

Sampling from a typical dense covariance matrix requires a Cholesky decomposition of the Matrix, an $O(n^3)$ operation. However, with the special structure of the low-rank kernel in (76) we can sample efficiently with

$$\boldsymbol{\epsilon} = \boldsymbol{\lambda}^{1/2} \odot \boldsymbol{\epsilon}_\lambda + \mathbf{V}\boldsymbol{\epsilon}_V, \tag{79}$$

$$\boldsymbol{\epsilon}_\lambda \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n) \in \mathbb{R}^n, \tag{80}$$

$$\boldsymbol{\epsilon}_V \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_l) \in \mathbb{R}^l. \tag{81}$$

B.2. Evaluating the Log-Density

Evaluating the log-density requires two main operations that are computationally expensive: solving a linear system of equations, and evaluating a log determinant. Usually, both of these two operations can be computed with a Cholesky decomposition which is $O(n^3)$. We show how we can evaluate the log density more efficiently by using the form of (76). Firstly we write out the log density of an MVN,

$$\log \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{n}{2}(2\pi) - \frac{1}{2}\log(|K(\mathbf{X},\mathbf{X})|) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top K(\mathbf{X},\mathbf{X})^{-1}(\mathbf{x}-\boldsymbol{\mu}). \tag{82}$$

We first look at how to efficiently solve the linear system by making use of the Woodbury (Petersen et al., 2008) identity,

$$K(\mathbf{X},\mathbf{X})^{-1}\mathbf{a} = \boldsymbol{\Lambda}^{-1}\mathbf{a} - \boldsymbol{\Lambda}^{-1}\mathbf{V}(\mathbf{I}_l + \mathbf{V}^\top\boldsymbol{\Lambda}^{-1}\mathbf{V})^{-1}\mathbf{V}^\top\boldsymbol{\Lambda}^{-1}\mathbf{a}, \tag{83}$$

where $\mathbf{a} = \mathbf{x} - \boldsymbol{\mu}$. To compute this efficiently we make use of

$$(\boldsymbol{\Lambda}^{-1} \cdot \mathbf{a})_i = \lambda_i^{-1}\mathbf{a}_i, \tag{84}$$

$$(\boldsymbol{\Lambda}^{-1} \cdot \mathbf{V})_{ij} = \lambda_i^{-1}\mathbf{V}_{ij}. \tag{85}$$

The largest matrix is $l \times n$ and requires inverting a $l \times l$ matrix where $l \ll n$. We then look at how to efficiently compute the determinant of the low-rank covariance matrix. Using the matrix determinant lemma (Ding & Zhou, 2007) we have

$$|K(\mathbf{X},\mathbf{X})| = \det(\boldsymbol{\Lambda} + \mathbf{V}\mathbf{V}^\top) = \det(\mathbf{I}_l + \mathbf{V}^\top\boldsymbol{\Lambda}^{-1}\mathbf{V})\prod_i \lambda_i. \tag{86}$$

This only requires taking the determinant of a $l \times l$ matrix. This can be done by computing the Cholesky of the inner matrix as,

$$L = \text{chol}(\mathbf{I}_l + \mathbf{V}^\top\boldsymbol{\Lambda}^{-1}\mathbf{V}), \tag{87}$$

and computing the log determinant as,

$$\log|\boldsymbol{\Sigma}| = 2\text{Tr}(\log(L)). \tag{88}$$

To form the covariance matrix in (76) from the neural network output of the α-Net we have

$$N_\alpha(\mathbf{z},\mathbf{w},\mathbf{X}_{1:n}) = \mathbf{O}_{1:n}, \quad \mathbf{o}_i \in \mathbb{R}^{2+l}, \tag{89}$$

where $N_\alpha(\cdot)$ is a the α-Net and

$$\mathbf{O}_{1:n,1} = \boldsymbol{\mu}_\alpha \in \mathbb{R}^n, \tag{90}$$

$$\boldsymbol{\sigma}_T(\mathbf{O}_{1:n,2}) = \boldsymbol{\lambda}_\alpha \in \mathbb{R}^n, \tag{91}$$

$$\mathbf{O}_{1:n,3:3+l} = \mathbf{V}_\alpha \in \mathbb{R}^{n \times l}. \tag{92}$$

and $\boldsymbol{\sigma}_T(\cdot)$ is an exp-linear function (Dehaene & Brossard, 2021) constraining the value to be in a set positive interval typically chosen to be $[10^{-5}, 1]$. In Sec. 4.4 we construct distributions for a vector field, in which case the distribution can be written simply as

$$q_\alpha(\mathbf{u}^1,\ldots,\mathbf{u}^d|\mathbf{z},\mathbf{w},\boldsymbol{\omega}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\alpha^1(\ldots)\\ \vdots \\ \boldsymbol{\mu}_\alpha^d(\ldots)\end{bmatrix}; \text{diag}\begin{bmatrix}\boldsymbol{\lambda}_\alpha^1(\ldots)\\ \vdots \\ \boldsymbol{\lambda}_\alpha^d(\ldots)\end{bmatrix} + \begin{bmatrix}\mathbf{V}_\alpha^1(\ldots)\\ \vdots \\ \mathbf{V}_\alpha^d(\ldots)\end{bmatrix}\begin{bmatrix}\mathbf{V}_\alpha^1(\ldots)\\ \vdots \\ \mathbf{V}_\alpha^d(\ldots)\end{bmatrix}^\top\right). \tag{93}$$

We note that the columns of $\mathbf{V}$ must be unique, else the matrix will be singular.

C. Distribution of Normalised Squared Errors

In this section we show boxplots of the 1000 log NSEs samples for RGNP-D (corresponding to row 1 of table 1) and DeepONet (trained with 300 collocation points, corresponding to row 7 of Table 1. - Poisson) for the nonlinear Poisson example.
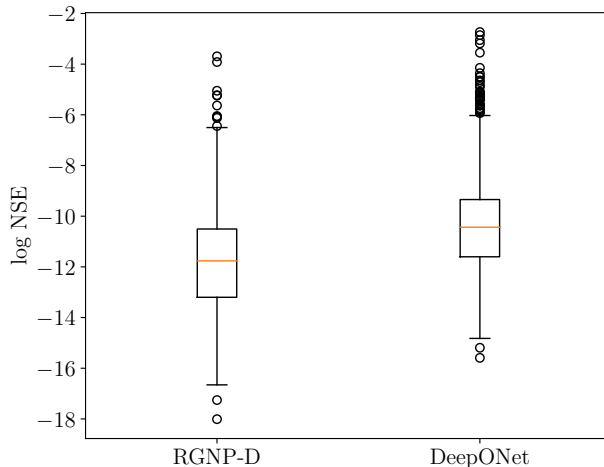

Figure 4. Box plot of log NSE for RGNP-D and DeepONet (300 N. Coll.) for the nonlinear Poisson example.

In Fig. 4 we see the spread of the samples in the log domain. The outliers as exponentially far from the mean.

D. Experiments

The experiments were all run using TensorFlow. All experiments were conducted with "swish" activation functions (Ramachandran et al., 2017). The inversion networks are implemented with 1D, 2D, and alternated 1D & 2D convolutional networks for the Nonlinear Poisson, Burgers, and Navier-Stokes problems, respectively. In Fig. 6 we show 5 example data samples. The scatter points are the data passed to the algorithm. A visual representation of the GICNet interpolation layers can be seen in Fig. 5.

D.1. Nonlinear Poisson

The $D(x)$ function used to enforce the Dirichlet boundary conditions for this problem is

$$D(x) = \cos\left(\frac{x\pi}{2}\right). \tag{94}$$

Architecture Details:

- number of hidden layers: 7

- number of neurons / hidden layer: 300

- activation: swish

- GICNet channel dimension: 20

- GICNet point/dim lattice: 20

We generate a dataset of 1000 evaluations for parameters drawn from the priors as the ground truth. It took 12.0 minutes to generate 1000 solutions using FEniCS.
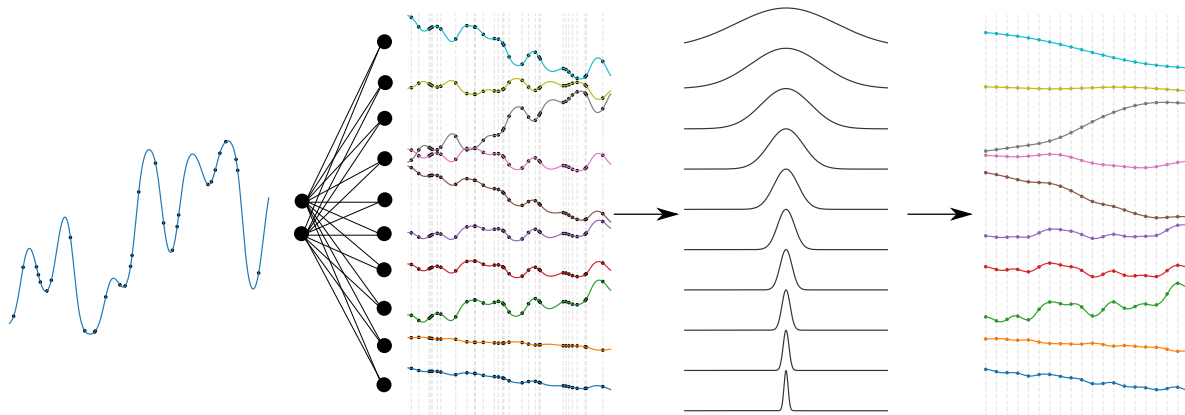
Figure 5. A visual representation of the lifting of a signal sampled at random locations into a higher dimensional feature space where it is then interpolated with various kernels onto a fixed grid. This rich feature space evaluated at a fixed location is then passed to a convolutional neural network with a matching channel dimension.

D.2. Burgers

For the DeepONet, FNN, and GICNet we train forward net with mean absolute residual as squared mean residual is too unstable. We have 10,000 training latent parameters. For DeepONet we shorten the number of collocation evaluations for 100×100 grid as we reduce the batch size but keep the same number of iterations. Generating the 1000 sample dataset using FEniCS takes 33.5 mins. The $D(x, t)$ function used to enforce the Dirichlet boundary conditions for this problem is

$$D(x, t) = \sin(x\pi) t. \tag{95}$$

Architecture Details:

- number of hidden layers: 8

- number of neurons / hidden layer: 400

- activation: swish

- GICNet channel dimension: 20

- GICNet point/dim lattice: 10

D.3. Navier-Stokes

The $D(x, t)$ function used to enforce the Dirichlet boundary conditions for this problem is
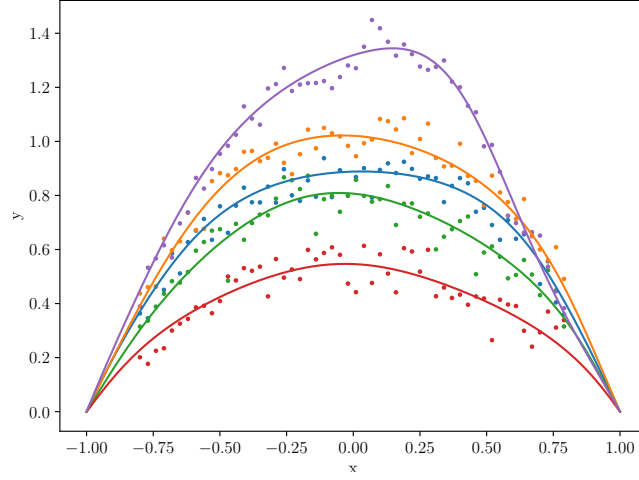
$$D(x, y, t) = \sin(x\pi)\sin(y\pi)t. \tag{96}$$

18

Figure 6. Five example data samples. We show the true solution given by the solver in a solid line, and the noisy scatter data $\mathbf{y}_D$.

The full Navier-Stokes equations can be written more explicitly as

$$z_0 \frac{\partial u_1}{\partial t} + z_0 \left(u_1 \frac{\partial u_1}{\partial x} + u_2 \frac{\partial u_1}{\partial y} \right) + \frac{\partial p}{\partial x} - z_1 \left(\frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} \right) = 0,$$

$$z_0 \frac{\partial u_2}{\partial t} + z_0 \left(u_1 \frac{\partial u_2}{\partial x} + u_2 \frac{\partial u_2}{\partial y} \right) + \frac{\partial p}{\partial y} - z_1 \left(\frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} \right) = 0,$$

$$\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} = 0. \tag{97}$$

The fully defined boundary and initial conditions are

$$u_1(x, 1, t) = (1 - (2x - 1)^6)t, \tag{98}$$
$$u_1(0, y, t) = u_1(1, y, t) = u_1(x, 0, t) = 0,$$
$$u_2(0, y, t) = u_2(1, y, t) = u_2(x, 0, t) = u_2(x, 1, t) = 0,$$
$$u_1(x, y, 0) = u_2(x, y, 0) = 0,$$
$$p(0, 0, t) = 0.$$

Architecture Details:

- number of hidden layers: 10

- number of neurons / hidden layer: 200

- activation: swish

- GICNet channel dimension: 30

- GICNet point/dim lattice: 10

D.4. MNSE

The expression used for defining the MNSE used to test the methods is

$$\text{MNSE}(x, x^*) = \frac{1}{N} \sum_i^N \frac{\|x - x^*\|_2^2}{\|x^*\|_2^2}, \tag{99}$$

where x is the output of the method and x^* is the ground truth.

19

D.5. Hardware

All experiments were run on an AMD Ryzen 9 5950X CPU (16 cores, 32 virtual) with 128GB memory and a Nvidia RTX 3090 (24GB VRAM) GPU. The GPU memory usage was limited to 6GB for the nonlinear Poisson problems, 10GB for the Burgers examples, and 20GB for the Navier-Stokes examples.