Approximate Bayesian Neural Operators: Uncertainty Quantification for Parametric PDEs

Anonymous authors
Paper under double-blind review

Abstract

Neural operators are a type of deep architecture that learns to solve (i.e. learns the nonlinear solution operator of) partial differential equations (PDEs). The current state of the art for these models does not provide explicit uncertainty quantification. This is arguably even more of a problem for this kind of tasks than elsewhere in machine learning, because the dynamical systems typically described by PDEs often exhibit subtle, multiscale structure that makes errors hard to spot by humans. In this work, we first provide a mathematically detailed Bayesian formulation of the "shallow" (linear) version of neural operators in the formalism of Gaussian processes. We then extend this analytic treatment to general deep neural ~~operators using~~ operators—specifically, graph neural operators—using approximate methods from Bayesian deep learning, enabling them to incorporate uncertainty quantification. As a result, our approach is able to identify cases, and provide structured uncertainty estimates, where the neural operator fails to predict well.

1 Introduction

Neural operators (Kovachki et al., 2023; Li et al., 2020b; 2021a; 2020a; 2021b) are ~~a deep learning architecture~~ deep learning architectures designed for reconstruction problems related to partial differential equations (PDEs). They approximate mappings between infinite-dimensional vector spaces of functions, such that – once trained – solutions of entire families of parametric PDEs can be represented by a single neural network. However, the learning process is subject to several sources of uncertainty, which can result in a potentially significant prediction error because of the nonlinear – and often nonintuitive – interactions of different stages of the approximation. The goal of this paper is to develop methods for estimating this error at a practically acceptable computational cost. This kind of functionality is urgently needed in this domain: Due to the intricate and often not intuitive nature of the dynamical systems described by PDEs, it can be hard for the human eye to detect prediction errors, even when they are large.

In this paper, we address this gap by developing an approximate Bayesian framework for neural operators – from a theoretical, and a computational point of view. We begin with a brief review of neural operators. Then, using linear, parametric PDEs as guiding examples, we show how their "shallow" (single-layer) base case allows for an analytic Bayesian treatment using the formalism of Gaussian processes (Rasmussen & Williams (2006)). This linear case, while primarily of theoretical interest, provides valuable insights and aims to make this model class more accessible to the Bayesian machine learning community. We then extend the theoretical analysis to the nonlinear deep case. Here, analytic treatments are no longer possible, so we fall back on approximations developed for Bayesian deep learning. Specifically, we focus on Laplace approximations (MacKay, 1992) which are easy to add post-hoc even to pretrained networks, and add only moderate computational cost relative to deep training without uncertainty quantification (Daxberger et al., 2021). Our experiments in Section 5 demonstrate that the resulting method effectively captures structure in the predictive error of graph neural operators, both in the over- and under-sampled regime. In ~~Section 3~~ Section 2 we discuss some theoretical background, and develop a probabilistic framework for neural operators in Section 3. We discuss related work in Section 4.

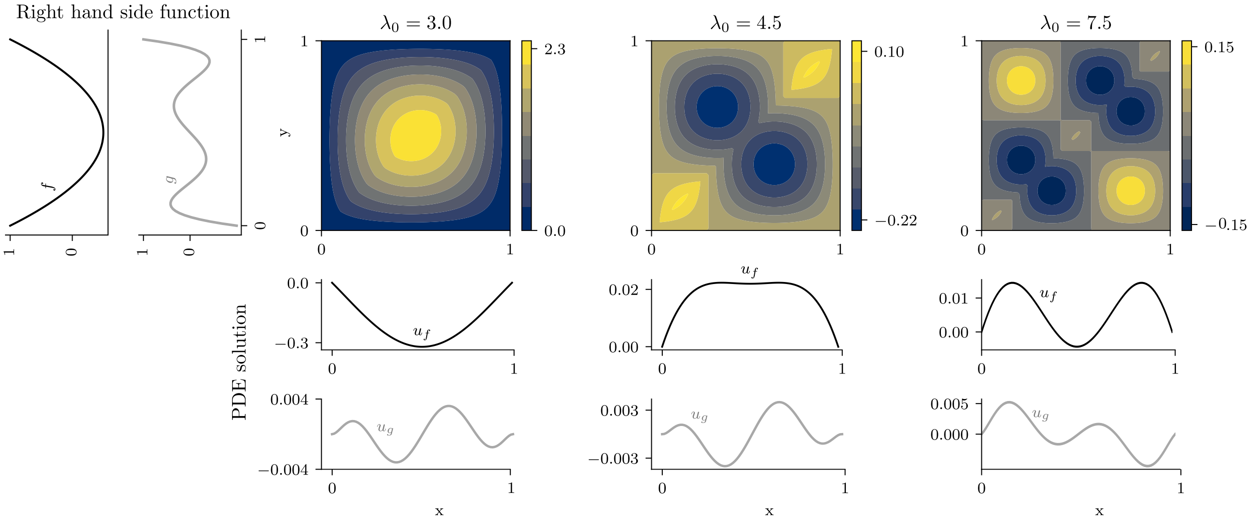


Figure 1: Green's functions in Equation (6) for different values of $\lambda_0 = \{3, 4.5, 7.5\}$. On the left, right-hand-side functions f, g for the PDE in Equation (5) and respective solutions u_f, u_g for the correspondent λ_0-value, computed through Equation (4).

2 Background

In this section, we examine how neural operators approximate solution operators for parametric PDEs through functional observations. If we fix one input of the solution operator, neural operators can be understood as effectively inverting the differential operator associated with the PDE. In this framework, the process of learning the operator becomes equivalent to reconstructing the Green's function, reducing the problem to a task of function approximation. This perspective, developed in Section 2.1, forms the basis for the Bayesian approach developed in Section 3.1. Subsequently, in Section 2.2, we outline the iterative structure of neural operators, their training ~~methodology, and their relationship~~ procedure, and how they relate to Green's functions.

2.1 PDEs And Green's Function

One of the main fields of applications of neural operators are ~~PDEs. In this work we consider the family~~ families of parametric PDEs of the form

$$\begin{aligned} (\mathcal{L}_\lambda u)(x) &= f(x), \quad && x \in D \\ u(x) &= 0, && x \in \partial D \end{aligned} \tag{1}$$

for some sufficiently well-behaved, bounded domain $D \subset \mathbb{R}^d$ with boundary ∂D (e.g. open, bounded D with Lipschitz boundary ∂D), where $U \ni u \colon D \to \mathbb{R}$, $F \ni f \colon D \to \mathbb{R}$, $\lambda \in \Lambda$, with U, F and Λ appropriate function spaces. The precise nature of those function spaces is not important for the remainder of this work. The function λ parametrises the differential operator $\mathcal{L}_\lambda$.

Equation (1) defines a solution operator

$$\mathcal{H} \colon \Lambda \times F \to U, \quad (\lambda, f) \mapsto u_{\lambda,f} \tag{2}$$

in the sense that $\mathcal{H}(\lambda, f)(x) = u_{\lambda,f}(x)$ solves the PDE for the given functions λ and f. Even though the PDE is linear, $\mathcal{H}$ is (possibly highly) nonlinear. ~~In particular, in this section we consider the case where λ is fixed, so the solution operator can be written as~~

~~$\mathcal{G} \colon f \mapsto u.$~~

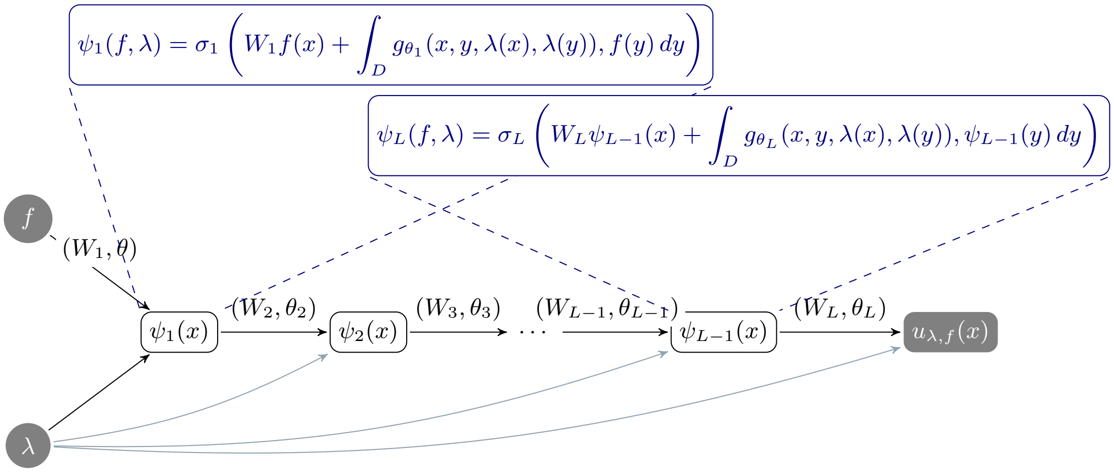

$\psi_1(f,\lambda)=\sigma_1\left(W_1 f(x)+\int_D g_{\theta_1}(x,y,\lambda(x),\lambda(y)),f(y)\,dy\right)$
$\psi_L(f,\lambda)=\sigma_L\left(W_L\psi_{L-1}(x)+\int_D g_{\theta_L}(x,y,\lambda(x),\lambda(y)),\psi_{L-1}(y)\,dy\right)$
f
(W_1,θ)
$\psi_1(x)$
(W_2,θ_2)
$\psi_2(x)$
(W_3,θ_3)
$\cdots$
(W_{L-1},θ_{L-1})
$\psi_{L-1}(x)$
(W_L,θ_L)
$u_{\lambda,f}(x)$
λ

Figure 2: Neural operator architecture NO_Θ. Each layer l computes a new function ψ_l, that contains the neural network g_θ in the integrand. Layer parameters are shown on the corresponding arrows.

~~The operator $\mathcal{G}$, like~~ The operator $\mathcal{H}$, is a map between function spaces. The idea behind neural operators is to approximate the operator ~~$\mathcal{G}$ (or~~ $\mathcal{H}$ ~~)~~ with a single neural network trained on function observations $\{f_i,u_i\}_{i=1}^{N}$. Thus, instead of approximating the solution of the PDE for only a fixed f or λ, neural operators directly infer the operator ~~$\mathcal{G}$.~~

$\mathcal{H}$. Numerically, the functions f and u are observed on a discretisation grid of the function domains.

In this subsection we are interested the particular case where λ is fixed, so the solution operator can be written as

$$\mathcal{G}\colon f\mapsto u. \tag{3}$$

This operator will be the one we want to approximate in the linear ("shallow") case in the next section. If the differential operator $\mathcal{L}_\lambda$ is linear, the map $\mathcal{G}$ inherits that linearity. Considering the operator in Equation (3) is ~~a key~~ an important step to understand the learning process of neural operators. In fact, observe how $\mathcal{G}$ is the inverse of the operator $\mathcal{L}_\lambda$. ~~The~~ In this simplified case where λ is fixed, the neural operator is therefore learning an operator, $\mathcal{G}$, through function observations $\{f_i,u_i\}_{i=1}^{N}$ that derive from the action of its inverse. In other words, during training, the neural operator is implicitly learning to invert the differential operator $\mathcal{L}_\lambda$. In particular, in the case where the differential operator is linear and admits a Green's function G, the solution of Equation (1) can be expressed through integration with the kernel G

$$u_\lambda(x)=\int_D G_\lambda(x,y)f(y)\,\mathrm{d}y. \tag{4}$$

Hence, learning the operator $\mathcal{G}$ is here equivalent to learn the function ~~G~~ G_λ, which means that an operator-learning task can be reduced to that of function-reconstruction. ~~The structure of neural operators in its one-layer case is inspired by the Green's solution formula for linear PDEs in Equation (4). We will examine their architecture, in the more general case, in the next section.~~

In the general analysis of linear PDEs (we refer to e.g. Evans (2010) for background on PDEs), the Green's function ~~$G(x,y)$~~ $G_\lambda(x,y)$ represents the impulse response of the linear operator $\mathcal{L}_\lambda$, that is ~~$\mathcal{L}_\lambda(G)(\cdot,y)=\delta(\cdot-y)$~~ $\mathcal{L}_\lambda(G_\lambda)(\cdot,y)=\delta(\cdot-y)$ for $y\in D$, where δ denotes the Dirac delta distribution. ~~Note how $\mathcal{L}_\lambda$ is a linearoperator, whereas the Green's function is usually nonlinear in~~ Despite $\mathcal{L}_\lambda$ being linear, the Green's function itself can be nonlinear in in either arguments. To visualize ~~the presented~~ these concepts, we consider the one-dimensional boundary value problem

$$\begin{aligned}\left(-\Delta-\lambda_0^2\,\mathrm{Id}\right)u(x)&=f(x),\quad x\in[0,1],\\ u(0)=u(1)&=0,\end{aligned} \tag{5}$$

~~that admits a~~ where in this case $\lambda_0 \in \mathbb{R}$ is a (scalar) parameter, and Id is the identity operator. It's Green's function~~in closed form,~~ , for $\lambda_0 \neq n\pi\ \forall n \in \mathbb{N}$, is given by

$$G_{\lambda_0}(x, y) := \frac{A + B}{\lambda_0 \sin(\lambda_0)} \tag{6}$$

where we abbreviated

$$A := H(y - x) \sin(\lambda_0 x) \sin(\lambda_0(1 - y)) \tag{7}$$
$$B := H(x - y) \sin(\lambda_0(1 - x)) \sin(\lambda_0 y), \tag{8}$$

and H denotes the Heaviside step function. Details of the Green's function derivation are in Appendix A . Equation (5) relates to Equation (1) in the sense that the differential operator $\mathcal{L}_{\lambda_0} = (-\Delta - \lambda_0^2 \,\mathrm{Id})$ is parametrised by λ_0. Figure 1 shows examples of Green's functions G_{λ_0} for different values of λ_0, ~~as well as the solutions computed through the formula in Equation (4), are depicted in Figure 1~~along with solutions computed via Equation (4).

~~Neural operator architecture NO_Θ. Each layer l computes a new function ψ_l, that contains the neural network g_θ in the integrand. Layer parameters are shown on the corresponding arrows. The input function f enters as an initialisation only in the first layer, while the function λ enters in g_θ at every ψ_l.~~

2.2 Overview of Neural ~~Operator Essentials~~Operators

~~Before formulating a Bayesian frameworkfor neural operators, we recall~~

Neural operators are neural-network-based architectures designed to approximate the general solution operator $\mathcal{H}$ defined in Equation (2) Before introducing our Bayesian framework, we briefly review their structure. A more thorough explanation of what follows can be found in the work by Kovachki et al. (2023); Li et al. (2020b; 2021a; 2020a; 2021b).

~~A neural operator is a neural network architecture designed to approximate the general solution operator $\mathcal{H}$ in Equation (2). For particular cases, such as the operator $\mathcal{G}$ in Equation (3) where λ is fixed, or for operators mapping $\lambda \mapsto u$ (where f is fixed), an analogous construction is straight forward.~~

Let $g_\theta : D \times D \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a neural network with parameters θ. Define the neural operator NO_Θ as a composition of $L \in \mathbb{N}$ layers

$$\begin{aligned} \mathrm{NO}_\Theta \colon &\Lambda \times F \to U, \\ &(\lambda, f) \mapsto (\psi_L \circ \psi_{L-1} \circ \ldots \circ \psi_1)(\lambda, f), \end{aligned} \tag{9}$$

where each layer

$$\psi_\ell : \Phi \to \Phi, \quad \ell = \{1, \ldots L\}, \tag{10}$$

is defined as a composition of (i) integrating the output of the previous layer against ~~g_θ~~g_{θ_ℓ}, and (ii) combining the integral with a linear component and an activation function σ,

$$\psi_\ell(\underline{g}h)(x) = \sigma\left(W_\ell \underline{g}h(x) + \int_D g_{\underline{\theta}\theta_\ell}(x, y, \lambda(x), \lambda(y))\underline{g}h(y)\, \mathrm{d}y \right). \tag{11}$$

The space Φ in Equation (10) is a vector space of ~~functions mapping from~~ real-valued functions on D~~to $\mathbb{R}$. The~~ , and the final layer of the neural operator maps into U, so $\psi_L \colon \Phi \to U$. In Equation (11), W_ℓ is a learnable linear operator (represented by a matrix after discretization), and g_{θ_ℓ} is the integral kernel in the ℓ-th layer. In practice, the integral cannot be computed in closed-form and a suitable quadrature formula needs to be employed (which turns the integral into a weighted sum of evaluations of the integrand; see e.g. Davis & Rabinowitz (2007)). The ~~parameters~~ parameter set Θ of NO_Θ ~~include the parameters θ of g_θ as well as the weights in each layer W_ℓ, i.e. $\Theta = \theta \cup \{W_\ell\}_{\ell=1}^L$~~is $\Theta = \{\theta_\ell \cup W_\ell\}_{\ell=1}^L$. Loosely speaking, one can think of this construction as a deep neural network (NO_Θ) that iteratively approximates the solution $u_{\lambda,f}$ ~~(see~~

~~Equation (2))~~ with linear transformations W_ℓ and nonlinear activation functions σ, and at every iteration (layer) employs another neural network (~~g_θ~~g_{θ_ℓ}). For a visualisation of NO_Θ see Figure 2.

~~This architecture is inspired by the process of solving linear PDEs with Green's functions: In the case where $L = 1$, $\lambda \equiv \lambda_0^2$, $\sigma = \mathrm{Id}$, and $W_1 = 0$, and we consider the mapping $\mathcal{G} \colon f \mapsto u$, the neural operator approximating $\mathcal{G}$ becomes~~

~~$\mathrm{NO}_\Theta(f) = \mathrm{NO}_\theta(f) = \int_D g_\theta(x, y) f(y) \,\mathrm{d}y.$~~

~~If g_θ is a sufficiently accurate approximation of the Green's function G_{λ_0} in Equation (6), Equation (15) is the solution formula of the PDE in Equation (5). In the next section we will provide a probabilistic formulation of the one layer architecture in Equation (15) that is based on the formalism of Gaussian processes.~~ Although the figure shows λ entering in each layer, in practice the kernel may encode λ in an initial "lifting" layer while using a final "projection" layer to map the function output back to the physical domain.

Note how NO_Θ approximates an operator. While, technically speaking, this means that its training and test set consist of functions, in the numerical computation, these functions need to be observed on some grid. Let $\{\lambda_1, ..., \lambda_N\} \times \{f_1, ..., f_M\}$ be a set of training inputs, each of which shall be observed on some mesh $\mathbb{X} := \{x_1, ..., x_K\}$. In total, that makes $NK \times MK = NMK^2$ training inputs. Without loss of generality, and for the sake of simple notation, assume that the solution of the PDE and the respective inputs are observed on the same mesh $\mathbb{X}$. Thus, we observe NM solutions $u_{11}, ..., u_{NM}$, i.e. NMK training outputs – one set of evaluations at $\mathbb{X}$ for each solution u_{nm} associated with (λ_n, f_m), $n = 1, ..., N$, $m = 1, ..., M$. Each of these outputs is a function that maps from D to $\mathbb{R}$, thus $u_{nm}(\mathbb{X}) \in \mathbb{R}^K$. The relation between inputs and outputs is

$$u_{nm} = \mathcal{H}(\lambda_n, f_m) \approx \mathrm{NO}_\Theta(\lambda_n, f_m). \tag{12}$$

While this equation is between functions, once discretised, it becomes an equation between vectors. To be able to optimise the parameters, we introduce the loss function

$$\mathcal{L} : \mathbb{R}^K \times \mathbb{R}^K \to [0, \infty). \tag{13}$$

The network parameters Θ are then computed by (approximately) solving the minimisation problem

$$\Theta^* = \arg\min_\Theta \sum_{n,m} \mathcal{L}(u_{nm}(\mathbb{X}), \mathrm{NO}_\Theta(\lambda_n, f_m)(\mathbb{X})), \tag{14}$$

where we used the above vectorised notation. This minimisation can be carried out with any of the optimisers popular in deep learning (see e.g. (Le et al., 2011)). Note that by approximating directly the solution operator $\mathcal{H}$, NO_Θ simultaneously learns the entire family of PDEs parametrised by f, λ without the need of re-training the network for a new λ or f. Considering that these new inputs samples can be out of distribution cases, which are notoriously harder to predict (Hendrycks & Gimpel, 2017), it is even more important to introduce uncertainty quantification for these architectures.

2.2.1 The One-Layer (Shallow) Case

A special, *shallow* version of the neural operator arises by setting $L = 1$, $\sigma \equiv \mathrm{Id}$, and $W_1 = 0$, with $\lambda \equiv \lambda_0^2$ fixed. In this simplified scenario, we focus on the operator $\mathcal{G} : f \mapsto u$, yielding

$$\mathrm{NO}_\Theta(f) = \mathrm{NO}_\theta^{\text{shallow}}(f) = \int_D g_\theta(x, y) f(y) \,\mathrm{d}y. \tag{15}$$

where $g_\theta := g_{\theta_1}$ is now the only learned integral kernel. If g_θ is a sufficiently accurate approximation of the Green's function G_{λ_0} in Equation (6), then Equation (15) essentially recovers the classical solution integral $\int_D G_{\lambda_0}(x, y)\, f(y)\, \mathrm{d}y$. Hence, the structure of neural operators in its one-layer (shallow) case is inspired by the Green's solution formula for linear PDEs (Equation (4)). In the next section, we provide a Gaussian process–based probabilistic perspective on this one-layer operator, which lays the groundwork for a more general Bayesian treatment of multi-layer (deep) neural operators.

3 Method

~~Here~~ In this section, we develop the Bayesian probabilistic framework for neural operators. Section 3.1 ~~explores~~ focuses on the special case of a one-layer ~~network, allowing~~ (shallow) network, where we can leverage Gaussian process regression to obtain an analytic non-parametric Bayesian treatment~~through a Gaussian process model~~. This setting provides not just a useable algorithm, but also an important conceptual base-case that is not prominently discussed in previous works on neural operators (including non-Bayesian ones). In Section 3.2, this "shallow" treatment is extended to the deep setting using a linearisation in form of the Laplace approximation, which again provides a Gaussian posterior distribution, albeit an approximate one.

3.1 Bayesian Neural Operators ~~and~~ In The Shallow Case With Gaussian ~~processes~~Processes

~~Consider the solution operator $\mathcal{G}\colon f \mapsto u$ of~~ We begin with the shallow neural operator $\mathrm{NO}_\theta^{\text{shallow}}$ introduced in Equation (15). In particular, we consider the *linear* PDE in Equation (5). In this ~~case $\mathcal{G}$ can be approximated with a one-layer neural operator, that in its single iteration computes the PDEsolution as the integral~~

~~$\mathrm{NO}_\theta = u_f(x) = \int_D g_\theta(x, y) f(y)\,\mathrm{d}y.$~~

~~As observed in Section 2.1, this "shallow" form of the neural operator is based on Green's solution formulas for linear PDEs.~~ setting, the PDE's solution operator $\mathcal{G}\colon f \mapsto u$ can be approximated via $\mathrm{NO}_\theta^{\text{shallow}}(f) = \int_D g_\theta(x, y)\, f(y)\,\mathrm{d}y$, where $g_\theta(x, y)$ plays the role of the Green's function $G(x, y)$. Since the considered linear PDE admits an analytic Green's function G (see Equation (6)), and since the only parameters of NO_Θ are the ones of the neural network g_θ ~~,~~ (i.e. $\Theta = \theta$~~, learning the operator $\mathcal{G}$ is here equivalent~~), learning $\mathcal{G}$ reduces to learning the function ~~G. Therefore, for this setting, one can reformulate the task of inferring the solution operator $\mathcal{G}\colon f \mapsto u$ (which maps between infinite-dimensional vector spaces of functions) as the inference problem of learning the function $G\colon \mathbb{R}^2 \to \mathbb{R}$~~$G\colon \mathbb{R}^2 \to \mathbb{R}$.

Formulating the Problem as GP Regression. In contrast to conventional GP regression, instead of ~~direct observations of G~~directly observing values of G, we only ~~have access to G through the integrals $u_n = \int_D G(x, y) f_n(y)\,\mathrm{d}y$ for every data point f_n, $n = 1, \ldots, N$. We define~~ observe integrals of G against various input functions. Specifically, for each training input function f_n, we observe

$$u_n(x) \;=\; \int_D G(x, y)\, f_n(y)\,\mathrm{d}y, \quad n = 1, \ldots, N.$$

Define the integral operator $\mathcal{A}_f = \mathcal{A}$ acting on G as ~~$\mathcal{A}G = \int_D G(\cdot, y) f(y)\,\mathrm{d}y = u(\cdot)$. Since $\mathcal{A}$~~

$$(\mathcal{A}G)(\cdot) \;=\; \int_D G(\cdot, y)\, f(y)\,\mathrm{d}y.$$

Because $\mathcal{A}$ is a linear operator in G, a Gaussian likelihood involving these observations (including the limit case of noise-free observations) ~~remains conjugate to~~ ensures conjugacy when we place a GP prior ~~and a Gaussian posterior can be computed in closed-form (Tanskanen et al., 2020; Longi et al., 2020).~~

~~Assume a Gaussian prior $G \sim \mathcal{GP}(\mu, k_\theta)$ with mean function $\mu\colon \mathbb{R}^2 \to \mathbb{R}$ and a parametrised kernel~~ over G. Concretely, suppose

$$G \;\sim\; \mathcal{GP}(\mu,\, k_\theta), \quad u \mid G \;\sim\; \mathcal{N}(\mathcal{A}G,\, \sigma^2),$$

where $\mu\colon \mathbb{R}^2 \to \mathbb{R}$ is the prior mean function ~~$k_\theta\colon \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$. Assuming $u \mid G \sim \mathcal{N}(\mathcal{A}G, \sigma^2)$~~and $k_\theta\colon \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ is the covariance kernel parameterized by θ. Because both the prior and the likelihood are Gaussian with a *linear* observation model, the posterior ~~distribution~~ over ~~G is a Gaussian process~~ G remains Gaussian (Tanskanen et al., 2020; Longi et al., 2020).

Posterior Mean and Covariance. The resulting posterior distribution over G is again a GP with mean and covariance:

$$\begin{aligned} \mathbb{E}[G] &= \ \mu \ + \ \mathcal{A}^* k_\theta \left(\mathcal{A}\mathcal{A}^* k_\theta + \sigma^2 \right)^{-1} (u - \mathcal{A}\mu), \\ \operatorname{Cov}(G) &= \ k_\theta \ - \ \mathcal{A}^* k_\theta \left(\mathcal{A}\mathcal{A}^* k_\theta + \sigma^2 \right)^{-1} \mathcal{A} k_\theta, \end{aligned} \tag{16}$$

where ~~$\mathcal{A}^*$~~ $\mathcal{A}^*$ is the adjoint ~~of $\mathcal{A}$. With the posterior distribution over~~ operator of $\mathcal{A}$. To see why, note that we have a standard linear–Gaussian model ($u_i = \mathcal{A}_{f_i} G + \sigma^2$), where observations u_i are obtained via the linear operator $\mathcal{A}$ acting on G ~~at hand we can compute~~ which yields a closed-form GP posterior (Rasmussen & Williams, 2006).

Interpretation and Extensions. This Gaussian posterior enables the usual suite of GP-based inference tools, such as computing uncertainty estimates on ~~the prediction, draw posterior samples, and exploit all the other properties of GP regression~~predictions and drawing posterior samples. Moreover, ~~the versatility of GPs allows to include prior information about G in the kernel k_θ. For example, the fact that Green's functions are symmetric, i.e.$G(x, y) = G(y, x)$, can be encoded in k_θ (Duvenaud (2014))~~prior domain knowledge about Green's functions (e.g., symmetry $G(x, y) = G(y, x)$) can be incorporated into the kernel k_θ (Duvenaud, 2014) . Since the solution u is a linear function of G, ~~the Gaussian posterior over G induces a GP over the solution u~~once G is learned, any new input function f^* can be mapped to a distribution over solutions u^*. That is, even in this simple "shallow" scenario, we obtain a probabilistic estimate over the ~~PDE solution. Moreover, since we learned the solution operator $\mathcal{G} \colon f \mapsto u$, we directly obtain an estimate of all the PDEsolutions for new right hand side functions f^*. In Section 5.1 we use this GP regression framework to learn the solution operator of~~ solution operator of th PDE. In Section 5.1, we demonstrate the use of this GP approach on Equation (5).

3.2 From ~~GP To NN~~Gaussian Processes to Neural Networks: Last-Layer Laplace Approximation~~On Neural Operators~~

While ~~we can directly use GP regression to obtain uncertainty estimates on PDE solutions for the~~ the GP-based approach from Section 3.1 provides an *exact* Bayesian treatment for the shallow (one-layer~~neural operator, this approach cannot be directly applied to deep~~) operator, it does not directly extend to *deep* neural operators, ~~which contain~~ whose non-linearities ~~. However~~break the linear–Gaussian framework. Instead, we can ~~use approximate inference techniques~~ adopt approximate inference methods from Bayesian deep learning to ~~obtain an approximation to~~ approximate the posterior distribution ~~over the weights $p(\Theta|\mathcal{D})$ with~~ $p(\Theta \mid \mathcal{D})$, where $\mathcal{D} = \{\lambda_n, f_m, u_{nm}\}$, ~~for~~ $n = 1, \dots, N$~~and~~ , $m = 1, \dots, M$ ~~. Since the computation of the true posterior is intractable, it is common to use a Gaussian approximation (MacKay, 1992; Blundell et al., 2015) .~~ are the training data, and Θ are the network parameters. In particular, we use the *Laplace approximation*, a relatively simple yet powerful approach to approximate the parameter's posterior distribution with a Gaussian (MacKay, 1992; Blundell et al., 2015).

Predictive Distribution. To make predictions ~~with the approximate posterior $q(\Theta)$,~~ at test inputs (λ_*, f_*), we need the predictive distribution

$$p(u_* \mid \mathrm{NO}_\Theta(\lambda_*, f_*), \mathcal{D}) \approx \int p(u_* \mid \mathrm{NO}_\Theta(\lambda_*, f_*)) q(\Theta) \, \mathrm{d}\Theta \tag{17}$$

~~for test functions (λ_*, f_*)~~where $q(\Theta) \approx p(\Theta \mid \mathcal{D})$ is the approximate posterior. In general, computing this predictive distribution requires ~~further approximation, such as the local linearisation~~ requires further approximation; for example, a local linearization of the neural network (Immer et al., 2020) ~~which results in~~ yields a Gaussian predictive distribution ~~for~~ under a Gaussian likelihood. ~~Alternatively, we can use a~~ A simpler yet often effective alternative is to focus on a *last-layer* Laplace approximation, ~~a relatively simple and early form of Bayesian deep learning (MacKay, 1992), on only the last layer of the network. This allows us to apply Laplace approximations to the intricate architecture of neural operators for efficient uncertainty quantification~~as we describe below.

Laplace Approximation. The Laplace approximation for neural networks ~~requires a~~ is built around the maximum a-posteriori (MAP) estimate ~~which is obtained by minimizing the loss $\mathcal{L}(\mathcal{D};\Theta)$~~ of Θ. Denote the regularized training loss as

$$\Theta_{\mathrm{MAP}} = \arg\min_{\Theta} \mathcal{L}(\mathcal{D};\Theta) = \arg\min_{\Theta} r(\Theta) + \sum_{n,m} \ell(\lambda_n, f_m, u_{nm}, \Theta).,\tag{18}$$

~~The empirical risk $\ell(\lambda_n, f_m, u_{nm}, \Theta)$~~ where ℓ corresponds to the negative ~~log likelihood~~ log-likelihood $-\log p(u_{nm} \mid \mathrm{NO}_\Theta(\lambda_n, f_m))$ and ~~the regularizer $r(\Theta)$ to the negative log prior distribution $-\log p(\Theta)$. The general idea of the Laplace approximation is to construct a local Gaussian approximation to the posterior $p(\Theta \mid \mathcal{D})$ by using a second order expansionof the loss $\mathcal{L}(\mathcal{D};\Theta)$ around Θ_{MAP}~~ $r(\Theta)$ is the negative log-prior. Then the MAP weights are

$$\Theta_{\mathrm{MAP}} \;=\; \arg\min_{\Theta} \mathcal{L}(\mathcal{D};\Theta).$$

Near Θ_{MAP}, we approximate $\mathcal{L}(\mathcal{D};\Theta)$ via a second-order Taylor expansion:

$$(\mathcal{D};\Theta) \approx_{\mathrm{MAP}} = \arg\min_{\Theta} \mathcal{L}(\mathcal{D};\Theta_{\mathrm{MAP}}) + \frac{1}{2}(\Theta - \Theta_{\mathrm{MAP}})^{\mathrm{T}} = \arg\min_{\Theta} \left(\nabla^2_{\Theta}\mathcal{L} r(\mathcal{D};\Theta)|_{\Theta_{\mathrm{MAP}}} + \sum_{n,m} \ell(\lambda_n, f_m, u_{nm}, \Theta - \Theta_{\mathrm{MAP}})\right),\tag{19}$$

~~where the first order~~ where the first-order term disappears at ~~Θ_{MAP}~~ Θ_{MAP}. Then the posterior approximation $q(\Theta)$ can be identified as a Gaussian centered at Θ_{MAP}, with a covariance corresponding to the local curvature:

$$q(\Theta) := \mathcal{N}(\Theta \mid \Theta_{\mathrm{MAP}}, (\nabla^2_{\Theta}\mathcal{L}(\mathcal{D};\Theta)|_{\Theta_{\mathrm{MAP}}})^{-1}).\tag{20}$$

~~That is, the covariance is~~ Hence, the approximate posterior is Gaussian, centered at Θ_{MAP}, with a covariance given by the inverse Hessian of the ~~regularized training loss (which is interpreted as an unnormalized negative log posterior) at the trained weights Θ_{MAP}~~ loss at that point.

~~A key practical advantage of this approach is that, since standard~~

Practical Advantages. Standard training of neural networks already identifies the local optimum Θ_{MAP}~~, the only~~ . Thus, the main additional cost is ~~to compute~~ computing the Hessian $\nabla^2_{\Theta}\mathcal{L}(\mathcal{D};\Theta)$ at ~~that point~~ Θ_{MAP}, once. ~~This also means the approximation can be computed~~ Moreover, this procedure can be done ~~*post-hoc*~~ *post hoc* ~~, for~~ on a pre-trained ~~networks~~ network, which implies that uncertainty quantification in the form of a Laplace approximation comes only at a very small computational overhead while also preserving the predictive power of the maximum a posteriori estimate.

~~As mentioned before, we can use the decomposition of the neural operator~~

Last-Layer Laplace in Neural Operators. To apply the Laplace method efficiently, one typically decomposes the network into a fixed feature map corresponding to the first $L-1$ layers and a last linear layer (Snoek et al., 2015). ~~This is particularly convenient in the case of the architecture considered by Li et al. (2020b) , since~~ In the graph neural operator by Li et al. (2020b) considered in this work, the last layer is ~~indeed linear . Due to the linearity in the weights of the last layer, the~~ linear in its weights. This linearity ensures that a Gaussian posterior on the last-layer weights induces a Gaussian distribution over the ~~function outputswill also be Gaussian~~ operator outputs. Hence, for a Gaussian likelihood the predictive distribution in Equation (17) can be computed in closed form by using the approximate posterior $q(\Theta)$. Note that this predictive distribution is equivalent to the one of a GP regression problem (Khan et al., 2019). ~~This directly connects the GP approach for the shallow~~ Conceptually, this connects the shallow GP approach to the deep case, although we are now not approximating the posterior over the parameters of the Green function, but over the weights of the last layer.

~~Kristiadi et al. (2020); Daxberger et al. (2021) showed~~ Recent work (Kristiadi et al., 2020; Daxberger et al., 2021) has shown that this approach achieves competitive performance on many common uncertainty quantification benchmarks compared to more recent alternatives – despite the low computational overhead. In ~~Section 5~~

~~, we empirically demonstrate that last-layer Laplace approximations effectively quantify uncertainty also in~~ Section 5, we demonstrate that the same methodology can be effectively combined with graph neural operator architectures ~~.~~ to provide predictive uncertainties for PDE solutions.

4 Related work

The interplay of (parametric) partial differential equation models (see Cohen & DeVore (2015) for a review) and deep learning has rapidly gained momentum in recent years. Broadly speaking, there are two approaches: learning the solution of a given PDE on the one hand, and learning the parameter-to-solution operator of a family of parametric PDEs on the other hand.

Conventional numerical PDE solvers (e.g. Ames (2014)) and physics-informed neural networks (PINNs) (Raissi et al., 2019; Sirignano & Spiliopoulos, 2018; Zhu et al., 2019) fall into the first category. In PINNs, the PDE solution is modelled as a neural network. The differential equation is then translated into an appropriate loss function, and an approximate PDE solution emerges from automatic differentiation and numerical optimisation. While the physics-informed neural network formulation extends naturally to PDE inverse problems (Raissi et al., 2019; Zhu et al., 2019), it brings with it some practical issues like hyperparameter-sensitivity and complicated loss landscapes (Wang et al., 2021; Sun et al., 2020). PINNs also need to be retrained once the parametrisation of the PDE (λ of f) changes.

As described in Section 2.2, neural operators do not face this issue because they learn the parameter-to-solution operator of a family of parametric PDEs (recall Equation (2)). Conceptualised by Lu et al. (2021), brought to the limelight by Bhattacharya et al. (2021); Nelsen & Stuart (2021); Li et al. (2020b;a; 2021a;b); Patel et al. (2021); Duvall et al. (2021); Kovachki et al. (2023), neural operators have since been extended into a range of architectures. These include graph neural operators (Li et al., 2020a), Fourier neural operators (FNOs) (Li et al., 2021a), multi-wavelet neural operators (Gupta et al., 2021), and physics-informed neural operators (Li et al., 2024), which integrate data and PDE constraints to simultaneously leverage observed data and governing equations in operator learning. For a comprehensive overview of neural operator architectures, we refer to Azizzadenesheli et al. (2024). Work on universal approximation results for neural operator architectures include Kovachki et al. (2023; 2021); Lanthaler et al. (2023). In this work, we focus on *graph neural operators* (Li et al., 2020b) for the experimental studies.

Despite these advances, ~~uncertainty quantification remains underexplored in the context of~~ *uncertainty quantification* remains relatively underexplored in neural operators. ~~Efforts in this direction~~ Recent efforts include Kumar et al. (2024), which ~~incorporate a Gaussian process prior with a mean function derived from~~ combine a Wavelet Neural Operator ~~, optimizing hyperparameters through~~ with a Gaussian process prior by optimizing hyperparameters via negative log-marginal likelihood~~minimization. Other Bayesian operator frameworks include Zou et al. (2024), which integrates Bayesian uncertainty into DeepONets, and Garg & Chakraborty (2022) , which employs~~ , and Zou et al. (2024), who propose a Bayesian extension of DeepONets. In addition, Garg & Chakraborty (2022) apply variational inference for uncertainty quantification~~. Kernel and Gaussian process frameworks for learning operators between function spaces have also been investigated by Batlle et al. (2024a) and Magnani et al. (2024) . Regarding non-neural network approaches for learning operators/PDEs, Gaussian process-based methods have been explored by Chen et al. (2021); Batlle et al. (2024b); Chen et al. (2024), whereas Boullé & Townsend (2022) focus on learning the~~ , while kernel- and GP-based operator-learning approaches (Batlle et al., 2024a; Magnani et al., 2024; Chen et al., 2021; Batlle et al., 2024b; Chen et al., 2024) address function-space mappings. Finally, Boullé & Townsend (2022) focus specifically on learning Green's ~~function associated with PDEs.~~ functions for PDEs. Our approach differs by providing an exact, GP-based formulation for the *shallow* (one-layer) operator under linear PDEs, and a *post-hoc* last-layer Laplace approximation for deep graph neural operators. Although Magnani et al. (2024) also employ Laplace approximations, their focus is on Fourier neural operators rather than graph-based architectures.

Uncertainty quantification is particularly critical in low-data regimes, where generating training data is computationally expensive due to the reliance on numerical PDE solutions. Bayesian methods offer a principled framework to address this challenge, providing insights into the reliability of predictions even when data is sparse. In the experiments below, we present an initial demonstration of the potential of uncertainty quantification for neural operators and discuss its potential implications for future developments in this field.

~~A principled approach to uncertainty quantification is generally provided by Bayesian deep learning. Besides the Laplace approximation, which has been discussed in Section 3.2, there are many more approximate Bayesian methods for inferring the neural networks ' weights. These~~ Outside the PDE context, approximate Bayesian treatments for neural networks include variational inference (Graves, 2011; Blundell et al., 2015; Khan et al., 2018; Zhang et al., 2018), Markov Chain Monte Carlo (Neal, 1996; Welling & Teh, 2011; Zhang et al., 2020), and heuristic methods (Gal & Ghahramani, 2016; Maddox et al., 2019). ~~Typically, they employ a Gaussian posterior approximation. One crucial advantage of the Laplace approximation over many of these methods is that it can be applied *post-hoc*, i.e. it is not only cheap but also preserves the estimate returned by the preceding~~ Most such approaches require either re-training or specialized sampling mechanisms, which can be computationally expensive and may alter the optimization process. The Laplace approximation (MacKay, 1992; Kristiadi et al., 2020; Daxberger et al., 2021) circumvents these downsides by approximating the posterior *around* a standard (non-Bayesian~~computation. In contrast, other methods require retraining the network, which can be expensive and may degrade predictive performance. Retraining often alters the optimization process, necessitating additional tuning and further increasing computational costs~~) pre-trained computation. As we demonstrate, this makes it especially appealing for neural operators where training can be time- and resource-intensive.

5 Experiments

~~Posterior distribution on G_{λ_0} for $\lambda_0 = 4.5$ (and ground truth) after $N = 3, 8$ observations $\{f_i\}_{i=1}^N$ with f_i shifted Legendre polynomials. The samples show the approximation's variance, which decreases when N increases.~~

In this section~~we exploit the theoretical analysis developed in ??~~, we apply the theoretical framework from Section 3 to construct Bayesian neural operators ~~delivering~~ that provide uncertainty estimates. We ~~use the analytic GP framework of Section 3.1to build a non-parametric Bayesian neural operator in the "shallow" case, then extend our method~~ begin with the *shallow* case, leveraging the exact Gaussian process formulation of Section 3.1, and then proceed to the deep ~~case. We reproduce the experiments on neural operators as carried out by Li et al. (2020b) to~~ setting. By replicating experiments from Li et al. (2020b), we show that we can effectively detect wrong predictions.

5.1 Uncertainty Quantification in the Shallow Case with GP regression

~~Consider the boundary value problem in Equation (5) for a fixed $\lambda_0 \in \mathbb{R}$. As discussed in Section 3.1, since the PDE is linear and admits the Green's function $G\colon \mathbb{R}^2 \to \mathbb{R}$ in Equation (6), inferring~~ We first consider the boundary-value problem from Equation (5) with a fixed parameter $\lambda_0 \in \mathbb{R}$. Since this linear PDE admits a Green's function $G\colon \mathbb{R}^2 \to \mathbb{R}$, learning the solution operator ~~$\mathcal{G}\colon f \mapsto u$ is equivalent to learning the function G given integral observations $\{f_i, u_i = \int_D G(\cdot, y) f_i(y)\,\mathrm{d}y\}_{i=1}^N$. Note that every observation point is a function, numerically observed on a grid $\mathbb{X} = \{x_1, \dots x_K\}$~~$\mathcal{G}\colon f \mapsto u$ reduces to estimating G from integral observations

$$\{(f_i,\ u_i = \int_D G(\cdot, y)\, f_i(y)\,\mathrm{d}y)\}_{i=1}^N.$$

Numerically, each right-hand side function f_i and the corresponding solution u_i are observed on an evenly spaced grid $\mathbb{X} = \{x_1, \dots x_K\}$. As training ~~points $\{f_i\}_{i=1}^N$ (right hand functions of the PDE in Equation (5))~~ functions $\{f_i\}$, we use the first ~~N Legendre polynomialsshifted on the interval $[0, 1]$ and observed on an evenly spaced grid $\mathbb{X} = \{x_1 = 0, \dots x_9 = 1\}$. We assume~~ N shifted Legendre polynomials, evaluated on $\mathbb{X}$. We then place a Gaussian prior $G \sim \mathcal{GP}(\mu, k)$ with a zero mean function μ and a kernel function $k\colon \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ that factorizes into the product $k((x_0, x_1), (y_0, y_1)) = k_1(x_0, y_0)k_2(x_1, y_1)$ where k_1 and k_2 are Matérn kernels with parameter $\nu = 2.5$. ~~To compute the integral operator $\mathcal{A}$ in Equation (16) we use numerical integration.~~ The integral operator $\mathcal{A}$ in Equation (16) is computed via numerical quadrature.

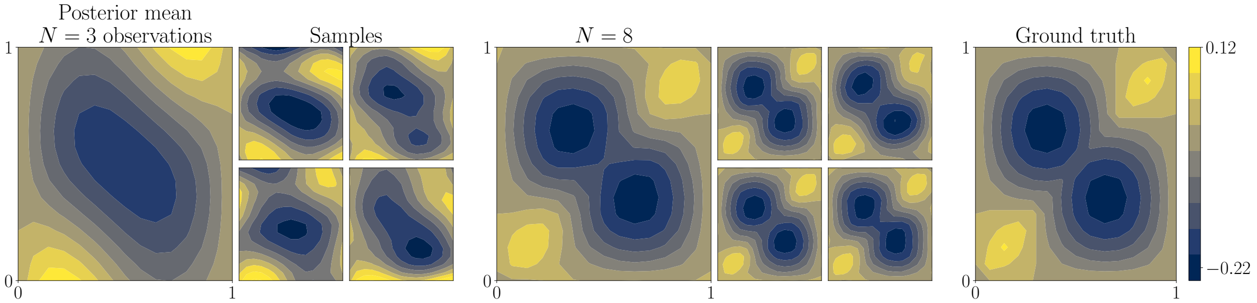


Figure 3: Posterior distribution on G_{λ_0} for $\lambda_0 = 4.5$ (and ground truth) after $N = 3, 8$ observations $\{f_i\}_{i=1}^{N}$ with f_i shifted Legendre polynomials. The samples show the approximation's variance, which decreases when N increases.

~~The posterior distribution over G, as inferred in Equation (16), is illustrated in Figure 3 . The figure shows the posterior distribution for G after $N = 3$ and $N = 8$ function observations~~

Figure 3 shows samples from the resulting posterior over G for $\lambda_0 = 4.5$ when $N = 3$ and $N = 8$. Samples from the posterior are used to visualize the posterior variance. For ~~$N = 3$, the samples exhibit high variability, corresponding to a high posterior variance and indicating that the approximation remains imprecise. In contrast, with $N = 8$ observations~~$N = 3$, the posterior variance is large, indicating a high degree of uncertainty. As N increases to 8, the posterior variance ~~is significantlyreduced, leading to a more accurate estimate of G~~diminishes significantly, yielding a closer approximation to the true Green's function. Since learning G corresponds to learning the inverse of the differential operator in Equation (5), the posterior distribution over G can be leveraged to obtain both an approximation of the solution and an associated error estimate for a new PDE with right-hand side function f^*.

5.2 Uncertainty Quantification in the Deep Case

~~In this section, we highlight the importance~~

We now showcase the role of uncertainty quantification ~~in the context of~~ for graph neural operators, ~~using~~ focusing on a second-order elliptic PDE~~as a representative example. Our results demonstrate that Bayesian~~. Our primary aim is to demonstrate how Bayesian graph neural operators can ~~effectively~~ identify regions of uncertainty in solution estimates and mitigate prediction errors in low-sampling regimes. ~~Although our experiments are limited to the linear case, these findings suggest that uncertainty quantification may play a critical role in extending neural operators to more complex, nonlinear settings.~~

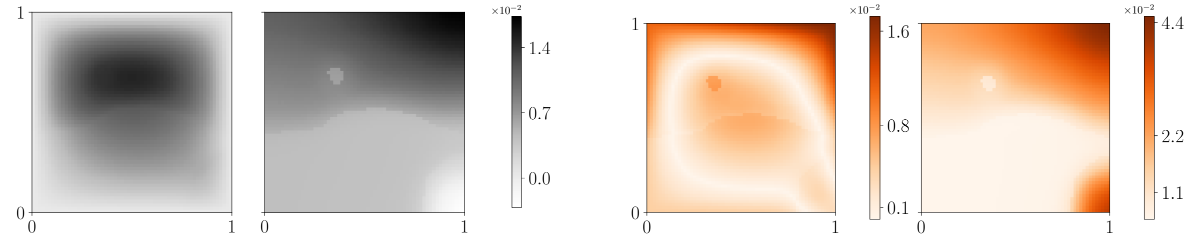
To recreate the results in Li et al. (2020b) we use their original code ~~.~~for graph-based neural operators[1] using message-passing layers (Kipf & Welling, 2016; Gilmer et al., 2017) with 64 hidden dimensions and ReLU activations. ~~As discussed in Section 3.2, our Bayesian framework computes Gaussian approximations of the posterior $p(\Theta \mid D)$ through Laplace approximations. For an efficient implementation of the~~ Training is performed via the Adam optimizer. We then apply our last-layer Laplace approximation ~~, we use the software~~ (as outlined in Section 3.2) post hoc, via the library introduced by Daxberger et al. (2021). ~~We use a last-layer Laplace approximation with~~ This method constructs a full generalized ~~Gauss-Newton~~ Gauss–Newton approximation (Schraudolph, 2002) of the Hessian ~~. There are two scalar hyperparameters,~~ of the training loss at the final-layer weights. Two scalar hyperparameters—the prior precision and ~~the observation noise. Both are tuned *post hoc* via~~ observation noise—are tuned *post hoc* by optimizing the log marginal likelihood ~~(Immer et al., 2021; Daxberger et al., 2021)~~(Immer et al., 2021; Daxberger et al., 2021).

We consider the second-order elliptic PDE examined in Li et al. (2020b), given by

$$\begin{aligned} -\nabla \cdot (\lambda(x) \nabla u(x)) &= f(x), \quad && x \in D \\ u(x) &= 0 && x \in \partial D \end{aligned} \tag{21}$$

---

[1] https://github.com/zongyi-li/graph-pde/graph-neural-operator



(a) Ground truth (**left**) and approximation (**right**). (b) Error (**left**) and standard deviation (**right**).

Figure 4: The Bayesian neural operator applied to the 2D Darcy flow problem in a low-data regime. The approximation is poor, and the predictive standard deviation highlights the areas of high error.

where $D = [0,1]^2$ is the unit square and $f \equiv 1$. The PDE in Equation (21) represents the steady state of a two dimensional Darcy flow and arises in several physical applications. Note that even though the PDE is linear, the parameter-to-solution operator $\mathcal{F}\colon \Lambda \to U,\ \lambda \mapsto u$ is not. The ~~nonlinear solution operator~~

$$\mathcal{F}\colon \Lambda \to U, \quad \lambda \mapsto u$$

~~is approximated with a type of~~ neural operator architecture ~~based on graph neural network structures~~ approximates $\mathcal{F}$ via a graph-based neural network (Kipf & Welling (2016)). In particular, for the computation of the integral in Equation (11), the domain D is discretised into a graph-structured data on which the message passing algorithm of Gilmer et al. (2017) is applied. In Section 5.3 we examine the case where only few data are available, while Section 5.4 addresses a high data regime.

5.3 Low-data Regime

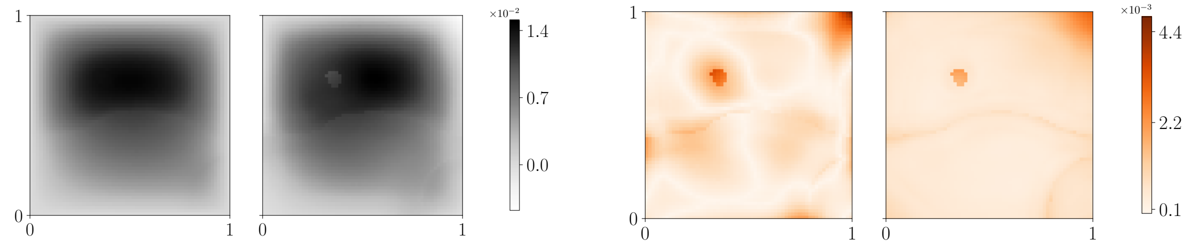
We begin by examining the case of sparse observation points on the unit square $D = [0,1]^2$, a common scenario in multi-scale dynamics described by PDEs, where data is often expensive to obtain. In such cases, the limited data can lead to inaccurate approximations, making it essential to quantify the uncertainty associated with predictions.

In particular, since the problem is relatively simple, we consider an extreme setting where we train on only two training functions and subsample only two points from a 16×16 grid for each. Figure 4 shows on a 61×61 grid that in this setting the NO fails to predict the solution well. As a consequence, our method exhibits low confidence (high predictive standard deviation) in the prediction, particularly in the areas of higher error. For readability, the plots use different color scales. This is due to the slight underconfidence of the Laplace approximation (in the scalar global parameter, not the local structure). Having measures such as the predictive standard deviation to determine whether the prediction should be trusted is of big practical benefit for many applications.

5.4 High-data Regime

The previous section examined a heavily under-sampled scenario, characterized by a limited amount of training data. While this setup may appear simplified, under-sampling is a common challenge in practical applications involving high-dimensional problems, where it is often infeasible to densely sample the domain with pre-computed PDE solutions. In this section, for completeness, we explore the opposite end of the spectrum—a highly over-sampled regime—and find that good and structured uncertainty quantification is nevertheless useful here.

Figure 5 shows results on a dense 61×61 grid, analogous to the previous one, trained on 100 densely evaluated 16×16 grid solutions. Note, that the model generalizes well from the smaller 16×16 grid used during training to the larger 61×61 grid for testing, as previously shown by Li et al. (2020b). Although the prediction error is generally of good quality (i.e. relative prediction errors are mostly below 10%), the trained network exhibits an artifact in one, sharply delineated region of the training domain. This is a common



(a) Ground truth (**left**) and approximation (**right**).

(b) Error (**left**) and standard deviation (**right**).

Figure 5: The Bayesian neural operator on the 2d Darcy flow problem in the high-data regime. The approximation is close to the ground truth. The regions of relatively high error, as well as their magnitude, are captured by the predictive standard deviation.

problem with the ReLU features in this architecture, which create piecewise linear predictive regions (Hein et al., 2019).

As the figure shows, the Laplace approximation is in fact able to identify and delineate this region well, and produce an effective, well-calibrated warning about its presence. It is important to note that this kind of functionality is only possible with the *structured* uncertainty produced by a Bayesian technique like the Laplace approximation – i.e. by an approximate posterior measure, rather than a global worst-case error bound.

6 Conclusions

We provided a theoretical Bayesian framework for neural operators. While these recently introduced architectures have demonstrated competitive performance compared to other numerical methods and shown promise in outperforming neural network-based approaches on large grids for certain tasks, they do not come with explicit uncertainty quantification. We developed an explicit analytic Bayesian treatment for the linear base-case, and illustrated how we can learn (the distribution over) solution operators through non-parametric GP regression. We provided an effective and efficient approximate Bayesian treatment for the full, deep case through the use of Laplace approximations. In experiments, our approach is able to quantify predictive uncertainty both in the sparsely and densely sampled regime. In the former, it produces structured uncertainty across the predictive domain. In the latter, it is able to precisely detect and delineate regions where the predictive estimate fails to approximate the true solution well. The code used to produce the results herein will be released with the final version of this paper.

If deep learning approaches to the simulation of dynamical systems are to fulfill their potential and be applied to serious, large-scale partial differential equations (including safety-critical and scientific applications), then uncertainty quantification as presented here has a crucial role to play in the prevention of accidental and potentially dangerous prediction errors.

References

William F Ames. *Numerical Methods for Partial Differential Equations.* Academic Press, 2014.

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, pp. 1–9, 2024.

Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496:112549, 2024a. ISSN 0021-9991.

Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496:112549, 2024b.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI Journal of computational mathematics*, 7:121–157, 2021.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, 2015.

Nicolas Boullé and Alex Townsend. Learning elliptic partial differential equations with randomized linear algebra. *Foundations of Computational Mathematics*, pp. 1–31, 2022.

Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew M Stuart. Solving and learning nonlinear PDEs with Gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.

Yifan Chen, Houman Owhadi, and Florian Schäfer. Sparse Cholesky factorization for solving nonlinear PDEs via Gaussian processes. *Mathematics of Computation*, 2024.

Albert Cohen and Ronald DeVore. Approximation of high-dimensional parametric PDEs. *Acta Numerica*, 24:1–159, 2015.

Philip J Davis and Philip Rabinowitz. *Methods of Numerical Integration*. Courier Corporation, 2007.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux – effortless Bayesian deep learning. In *NeurIPS*, 2021.

James Duvall, Karthik Duraisamy, and Shaowu Pan. Non-linear independent dual system (nids) for discretization-independent surrogate modeling over complex geometries. *arXiv:2109.07018*, 2021.

David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 2010.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

Shailesh Garg and Souvik Chakraborty. Variational bayes deep operator network: a data-driven bayesian solver for parametric differential equations. *arXiv preprint arXiv:2206.05655*, 2022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Alex Graves. Practical variational inference for neural networks. In *NeurIPS*, 2011.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. In *NeurIPS*, volume 34, 2021.

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, 2019.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *ICLR*, 2017.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of Bayesian neural networks via local linearization. In *AISTATS*, 2020.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021.

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in adam. In *ICML*, 2018.

Mohammad Emtiyaz Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into Gaussian processes. In *NeurIPS*, 2019.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for Fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *JMLR*, 24(89):1–97, 2023.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being Bayesian, even just a bit, fixes overconfidence in relu networks. In *ICML*, 2020.

Sawan Kumar, Rajdip Nayek, and Souvik Chakraborty. Neural operator induced Gaussian process framework for probabilistic solution of parametric partial differential equations. *arXiv preprint arXiv:2404.15618*, 2024.

Samuel Lanthaler, Zongyi Li, and Andrew M Stuart. The nonlocal neural operator: Universal approximation. *arXiv preprint arXiv:2304.13221*, 2023.

Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *ICML*, 2011.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *NeurIPS*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *ICLR*, 2021a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Markov neural operators for learning chaotic systems. *arXiv:2106.06898*, 2021b.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024.

Krista Longi, Chang Rajani, Tom Sillanpää, Joni Mäkinen, Timo Rauhala, Ari Salmi, Edward Hæggström, and Arto Klami. Sensor placement for spatial Gaussian processes with integral observations. In *UAI*, pp. 1009–1018. PMLR, 2020.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

David JC MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5): 720–736, 1992.

Wesley Maddox, T. Garipov, Pavel Izmailov, Dmitry P. Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *NeurIPS*, 2019.

Emilia Magnani, Marvin Pförtner, Tobias Weber, and Philipp Hennig. Linearization turns neural operators into function-valued Gaussian processes. *arXiv preprint arXiv:2406.05072*, 2024.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

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, 2021.

Ravi G Patel, Nathaniel A Trask, Mitchell A Wood, and Eric C Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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.

CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, January 2006.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian optimization using deep neural networks. In *ICML*, 2015.

Ivar Stakgold and Michael J Holst. *Green's functions and boundary value problems*. John Wiley & Sons, 2011.

Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

Ville Tanskanen, Krista Longi, and Arto Klami. Non-linearities in Gaussian processes with integral observations. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2020.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger B. Grosse. Noisy natural gradient as variational inference. In *ICML*, 2018.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient MCMC for Bayesian deep learning. In *ICLR*, 2020.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

Zongren Zou, Xuhui Meng, Apostolos F Psaros, and George E Karniadakis. Neuraluq: A comprehensive library for uncertainty quantification in neural differential equations and operators. *SIAM Review*, 66(1): 161–190, 2024.

A Appendix

A.1 Derivation of the Green's function for the one-dimensional Dirichlet problem

We consider the one-dimensional boundary value problem

$$\begin{aligned} \mathcal{L}_{\lambda_0} u\,(x) = \big(-\Delta - \lambda_0^2\,\mathrm{Id}\big)\,u(x) = \frac{d^2}{dx^2}u(x) \;-\; \lambda_0^2\,u(x) = f(x), \quad x \in [0,1], \\ u(0) = u(1) = 0, \end{aligned} \tag{22}$$

The *Green's function* $G_{\lambda_0}(x,y)$ solves, for each fixed $y \in [0,1]$,

$$\mathcal{L}_{\lambda_0}[G_{\lambda_0}(\,\cdot\,,y)](x) \;=\; \delta(x-y), \quad \text{with} \quad G_{\lambda_0}(0,y) \;=\; G_{\lambda_0}(1,y) \;=\; 0.$$

Step 1: Solve the homogeneous equation away from $x = y$. For $x \neq y$, the Dirac delta is zero, so G_{λ_0} satisfies the homogeneous problem

$$-\frac{d^2}{dx^2}G_{\lambda_0}(x,y) \;-\; \lambda_0^2\,G_{\lambda_0}(x,y) \;=\; 0.$$

Hence, for a fixed y,

$$G_{\lambda_0}(x,y) \;=\; \begin{cases} A(y)\,\sin\big(\lambda_0\,x\big) + B(y)\,\cos\big(\lambda_0\,x\big), & x < y, \\ C(y)\,\sin\big(\lambda_0\,x\big) + D(y)\,\cos\big(\lambda_0\,x\big), & x > y. \end{cases}$$

Imposing the boundary condition $G_{\lambda_0}(0,y) = 0$ implies $B(y) = 0$. Thus

$$G_{\lambda_0}(x,y) \;=\; \begin{cases} A(y)\,\sin\big(\lambda_0\,x\big), & x < y, \\ C(y)\,\sin\big(\lambda_0\,x\big) + D(y)\,\cos\big(\lambda_0\,x\big), & x > y. \end{cases} \tag{23}$$

The condition $G_{\lambda_0}(1,y) = 0$ then imposes

$$C(y)\,\sin(\lambda_0) + D(y)\,\cos(\lambda_0) \;=\; 0 \quad (\text{Condition 1}).$$

Step 2: Enforce continuity at $x = y$. Because G_{λ_0} itself must be continuous at $x = y$, we require

$$\lim_{x\to y^-} G_{\lambda_0}(x,y) \;=\; \lim_{x\to y^+} G_{\lambda_0}(x,y).$$

That is,

$$A(y)\,\sin\big(\lambda_0\,y\big) \;=\; C(y)\,\sin\big(\lambda_0\,y\big) + D(y)\,\cos\big(\lambda_0\,y\big) \quad (\text{Condition 2}).$$

Step 3: Impose the jump condition on the derivative. By integrating $\mathcal{L}_{\lambda_0}[G_{\lambda_0}(\cdot,y)](x) = \delta(x-y)$ across a small interval around $x = y$ we get

$$-\int_{y-\varepsilon}^{y+\varepsilon} G''_{\lambda_0}(x,y)\,dx \;=\; \int_{y-\varepsilon}^{y+\varepsilon} \delta(x-y)\,dx \;=\; 1.$$

Since

$$\int_{y-\varepsilon}^{y+\varepsilon} G''_{\lambda_0}(x,y)\,dx \;=\; G'_{\lambda_0}(y+\varepsilon,y) \;-\; G'_{\lambda_0}(y-\varepsilon,y),$$

we get $-\Big[G'_{\lambda_0}(y+\varepsilon,y) \;-\; G'_{\lambda_0}(y-\varepsilon,y)\Big] \;=\; 1$, hence

$$G'_{\lambda_0}(y^+,y) - G'_{\lambda_0}(y^-,y) \;=\; -1 \qquad (\text{Condition 3}).$$

Step 4: Solve for the coefficients. Solving equation 23 and Conditions 1,2,3 leads to the known closed-form expression for $\lambda_0 \neq n\pi$:

$$G_{\lambda_0}(x,y) \;=\; \frac{1}{\lambda_0\,\sin(\lambda_0)}\begin{cases}\sin\big(\lambda_0\,x\big)\,\sin\big(\lambda_0\,(1-y)\big), & x \leq y,\\ \sin\big(\lambda_0\,y\big)\,\sin\big(\lambda_0\,(1-x)\big), & x \geq y.\end{cases}$$

In the notation of Equation (6), one can equivalently write this piecewise definition in terms of the Heaviside step function H.

Step 5: Verify the non-degeneracy condition. If $\lambda_0 = n\pi$ for some $n \in \mathbb{N}$, then $\sin(\lambda_0) = 0$, and the above formula becomes singular. Indeed, in that case, the homogeneous problem with boundary conditions $u(0) = u(1) = 0$ has non-trivial solutions, which obstructs invertibility of $\mathcal{L}_{\lambda_0}$. Thus, the Green's function (and hence the unique solution) is well defined when $\lambda_0 \neq n\pi$.

This completes the derivation. For further details on Green's functions and partial differential equations, we refer e.g. to Stakgold & Holst (2011); Evans (2010).