Computer Methods in Applied Mechanics and Engineering 434 (2025) 117581

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Operator learning with Gaussian processes

Carlos Mora [a], Amin Yousefpour [a], Shirin Hosseinmardi [a], Houman Owhadi [b], Ramin Bostanabad [a,*]

[a] Department of Mechanical and Aerospace Engineering, University of California, Irvine, CA, United States of America
[b] Computing and Mathematical Sciences, Caltech, Pasadena, CA, United States of America

ARTICLE INFO

Dataset link: https://github.com/Bostanabad-Research-Group/GP-for-Operator-Learning

Keywords:
Operator learning
Gaussian processes
Neural operators
Zero-shot learning
Optimal recovery

ABSTRACT

Operator learning focuses on approximating mappings $\mathcal{G}^\dagger : \mathcal{U} \to \mathcal{V}$ between infinite-dimensional spaces of functions, such as $u : \Omega_u \to \mathbb{R}$ and $v : \Omega_v \to \mathbb{R}$. This makes it particularly suitable for solving parametric nonlinear partial differential equations (PDEs). Recent advancements in machine learning (ML) have brought operator learning to the forefront of research. While most progress in this area has been driven by variants of deep neural networks (NNs), recent studies have demonstrated that Gaussian process (GP)/kernel-based methods can also be competitive. These methods offer advantages in terms of interpretability and provide theoretical and computational guarantees. In this article, we introduce a hybrid GP/NN-based framework for operator learning, leveraging the strengths of both deep neural networks and kernel methods. Instead of directly approximating the function-valued operator $\mathcal{G}^\dagger$, we use a GP to approximate its associated real-valued bilinear form $\widetilde{\mathcal{G}}^\dagger : \mathcal{U} \times \mathcal{V}^* \to \mathbb{R}$. This bilinear form is defined by the dual pairing $\widetilde{\mathcal{G}}^\dagger(u, \varphi) := [\varphi, \mathcal{G}^\dagger(u)]$, which allows us to recover the operator $\mathcal{G}^\dagger$ through $\mathcal{G}^\dagger(u)(y) = \widetilde{\mathcal{G}}^\dagger(u, \delta_y)$. The mean function of the GP can be set to zero or parameterized by a neural operator and for each setting we develop a robust and scalable training mechanism based on maximum likelihood estimation (MLE) that can optionally leverage the physics involved. Numerical benchmarks demonstrate our method's scope, scalability, efficiency, and robustness; showing that (1) it enhances the performance of a base neural operator by using it as the mean function of a GP, and (2) it enables the construction of zero-shot data-driven models that can make accurate predictions without any prior training. Additionally, our framework (a) naturally extends to cases where $\mathcal{G}^\dagger : \mathcal{U} \to \prod_{s=1}^{S} \mathcal{V}^s$ maps into a vector of functions, and (b) benefits from computational speed-ups achieved through product kernel structures and Kronecker product matrix representations of the underlying kernel matrices.[1]

1. Introduction

Operator learning naturally arises in many applications such as solving partial differential equation (PDE) systems [1], statistical description and modeling of random functions [2,3], mechanistic reduced order modeling [4], speech inversion and sound recognition [5], or emulation of expensive simulations [6–8]. Recent advancements in machine learning (ML) have pushed operator learning to the forefront of research in both academia and industry. While the vast majority of the developments on this topic leverage variants of deep neural networks (NNs) [9–12], kernel-based methods [13] and feature-map-based methods [14] have also recently been shown to be competitive. In this paper, we propose a hybrid GP/NN-based framework for approximating mappings

* Corresponding author.
E-mail address: Raminb@uci.edu (R. Bostanabad).

[1] GitHub repository: https://github.com/Bostanabad-Research-Group/GP-for-Operator-Learning

https://doi.org/10.1016/j.cma.2024.117581
Received 6 September 2024; Received in revised form 22 October 2024; Accepted 15 November 2024
Available online 26 November 2024

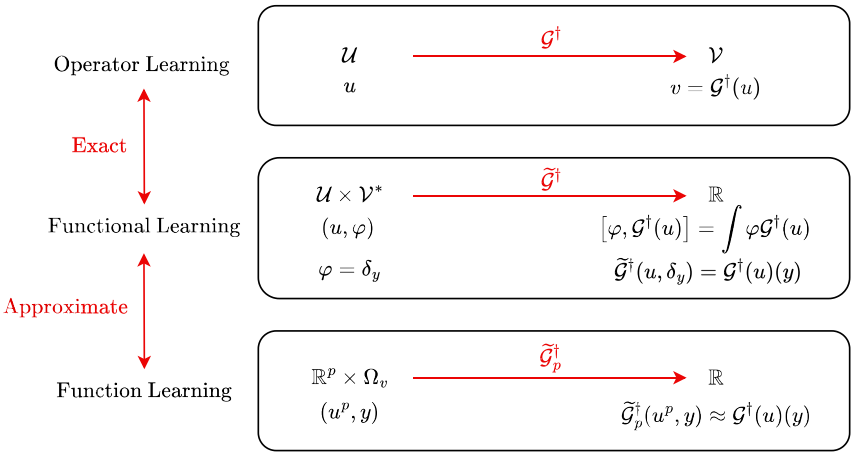


Fig. 1. Diagram of our framework for operator learning: We convert the operator learning problem to a regression one which can be solved via GPs. For multi-output operator learning, we use multi-response or multi-task GPs.

between infinite-dimensional function spaces. Our framework provides a first of its kind mechanism for simultaneously leveraging the strengths of both deep NNs and kernels for operator learning.

1.1. Description of the operator learning problem

Let $\mathcal{U}$ and $\mathcal{V}$ denote two separable infinite-dimensional Banach spaces of continuous functions and define the nonlinear operator $\mathcal{G}^\dagger$ as:

$$\mathcal{G}^\dagger : \mathcal{U} \to \mathcal{V}. \tag{1}$$

Furthermore, let $\phi : \mathcal{U} \to \mathbb{R}^p$ and $\psi : \mathcal{V} \to \mathbb{R}^q$ be two bounded linear observation operators, that is:

$$\phi : u \mapsto u^p := \left[u(x_1), \ldots, u(x_p)\right]^T \quad \text{and} \quad \psi : v \mapsto v^q := \left[v(y_1), \ldots, v(y_q)\right]^T \tag{2}$$

where $\{x_j\}_{j=1}^p$ and $\{y_j\}_{j=1}^q$ denote the location of the points where the functions u and v are observed in their respective domains Ω_u and Ω_v. Our goal is to approximate $\mathcal{G}^\dagger$ using N pairs of input–output observations, i.e., we aim to approximate $\mathcal{G}^\dagger$ given $\mathcal{D} := \{\phi(u_i), \psi(v_i)\}_{i=1}^N$ where $\{u_i, v_i\}_{i=1}^N$ are elements from $\mathcal{U} \times \mathcal{V}$.

1.2. Summary of the proposed approach

We propose to cast the operator learning problem in Eq. (1) as the following equivalent functional learning problem:

$$\widetilde{\mathcal{G}}^\dagger : \mathcal{U} \times \mathcal{V}^* \to \mathbb{R} \tag{3}$$

where $\mathcal{V}^*$ is the dual space of $\mathcal{V}$, consisting of all continuous linear functionals from $\mathcal{V}$ to $\mathbb{R}$. For any pair $(u, \varphi) \in \mathcal{U} \times \mathcal{V}^*$, $\widetilde{\mathcal{G}}^\dagger(u, \varphi)$ returns the scalar value $[\varphi, \mathcal{G}^\dagger(u)]$ defined as the dual pairing between φ and $\mathcal{G}^\dagger(u)$. If we specifically choose $\varphi = \delta_y$, where δ_y is the delta functional which evaluates the output function at a specific point $y \in \Omega_v \subset \mathbb{R}^d$, then we can easily retrieve the desired operator $\mathcal{G}^\dagger$ given that $\mathcal{G}^\dagger(u)(y) = \widetilde{\mathcal{G}}^\dagger(u, \delta_y)$.

In practice, (1) we only have access to a discretization of u, i.e., $\phi(u) \in \mathbb{R}^p$, and (2) we are only interested in pointwise evaluations $\mathcal{G}^\dagger(u)(y)$ with $y \in \Omega_v$. Hence, the functional learning problem can be reduced to identifying an operator:

$$\widetilde{\mathcal{G}}_p^\dagger : \mathbb{R}^p \times \Omega_v \to \mathbb{R}. \tag{4}$$

such that $\widetilde{\mathcal{G}}_p^\dagger(\phi(u), y) \approx \mathcal{G}^\dagger(u)(y)$.

Since the mapping in Eq. (4) is between finite-dimensional Euclidean spaces, it can be numerically approximated via classical NN/GP-based regression techniques. This regression-based approach differs from other kernel-based methods such as [13] that directly approximate the operator rather than its dual action on $\mathcal{V}^*$. The overall articulation of our regression-based approach for operator learning is schematically shown in Fig. 1 which also applies to other well-known techniques such as DeepONet [10]. In this paper, we approximate the mapping $\widetilde{\mathcal{G}}_p^\dagger$ in Eq. (4) via GPs whose parameters are estimated either in a purely data-driven manner, or via both data and physics. In the former case, we use maximum likelihood estimation (MLE) for parameter optimization and set the mean function to be either zero or a deep NN. In the latter case, we rely on GPs whose mean functions are represented via neural operators (e.g., FNO or DeepONet) whose parameters are estimated via a weighted combination of MLE and mean squared error (MSE).

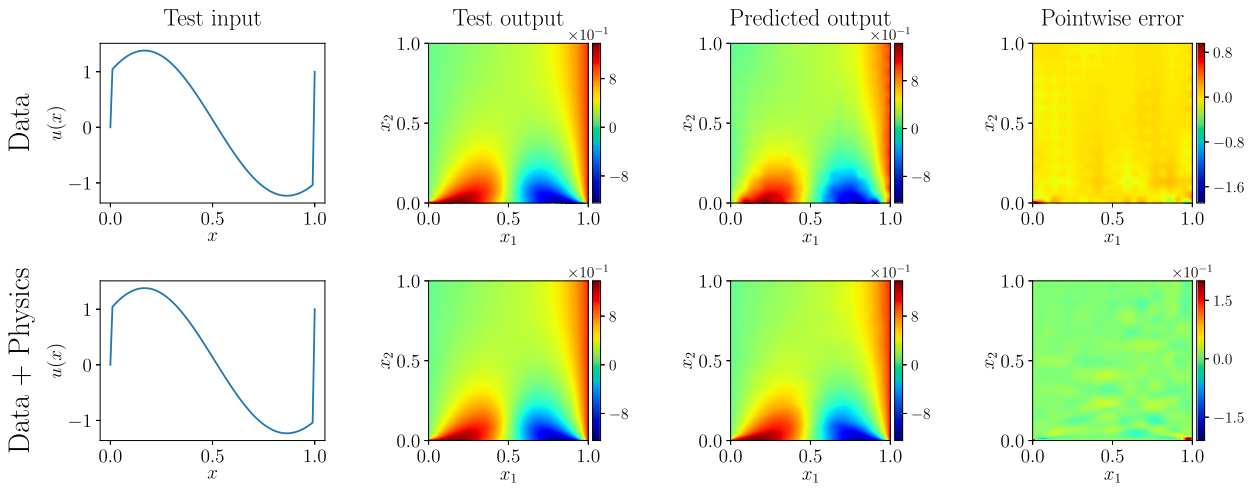


Fig. 2. Data-driven and physics-informed operator learning for the Burgers' problem with Dirichlet BCs: The training dataset has 400 pairs of $\{u_i, v_i\}$ that satisfy Eq. (5). The observation operators ϕ and ψ sample u_i and v_i at $p = 100$ and $q = 12^2$ collocation points, respectively. It can be observed that leveraging the physics reduces the prediction error. A DeepONet is used as the mean function of the GP in both data-driven and physics-informed cases.

As in [13], the diagram in Fig. 1 can be made commutative by introducing two reconstruction (optimal recovery) operators associated with ϕ and ψ that map the observations in $\mathbb{R}^p$ and $\mathbb{R}^q$ back to $\mathcal{U}$ and $\mathcal{V}$, respectively. Since ϕ and ψ are bounded and linear, their elements are in $\mathcal{U}^*$ and $\mathcal{V}^*$ which are the dual spaces of $\mathcal{U}$ and $\mathcal{V}$, respectively. Assuming $\mathcal{U}$ and $\mathcal{V}$ are reproducing kernel Hilbert spaces (RKHSs) and the elements of the observation operators are linearly independent, the reconstruction (optimal recovery) operators can be obtained via the usual representer theorem as detailed in [13].

1.3. Illustrative example

To contextualize the above abstract definitions, we consider the Burgers' equation with Dirichlet boundary conditions (BCs):

$$\begin{aligned} &v_t + vv_x - \zeta v_{xx} = 0, && \forall x \in (0,1), && t \in (0,1] \\ &v = 0, && \forall x = 0, && t \in [0,1] \\ &v = 1, && \forall x = 1, && t \in [0,1] \\ &v = u, && \forall x \in [0,1], && t = 0, \end{aligned} \tag{5}$$

where x and t denote space and time, v is the PDE solution, u denotes the initial condition (IC), and $\zeta = 0.1$ is the kinematic viscosity. We denote the $(0,1)^2$ domain and its boundaries where IC and BCs are specified by Ω, $\partial\Omega_t$, and $\partial\Omega_x$, respectively. The goal in this problem is to learn the nonlinear operator $\mathcal{G}^\dagger$ that maps u to v, i.e., to the solution field in Ω. To approximate $\mathcal{G}^\dagger$, we consider two scenarios:

- **Data-driven operator learning:** We take the linear operators ϕ and ψ defined in Section 1.1 to be the function evaluations at a set of p and q collocation points (CPs) on $\partial\Omega_t$ and in Ω, respectively. This choice provides the training dataset $\mathcal{D} := \{\phi(u_i), \psi(v_i)\}_{i=1}^N$ where $\phi(u_i)$ and $\psi(v_i)$ are of size p and q, respectively, and the pair $\{u_i, v_i\}$ satisfies the PDE system in Eq. (5). In this scenario, we follow the diagram in Fig. 1 and approximate $\mathcal{G}^\dagger$ via $\widetilde{\mathcal{G}}_p^\dagger$ using $\mathcal{D}$.
- **Physics-informed operator learning:** In this scenario, we leverage the PDE system in Eq. (5) as well as $\mathcal{D}$ to approximate $\mathcal{G}^\dagger$. That is, we require the N_{pi} samples $\mathcal{D}_{pi} := \{u_i, \widehat{v}_i\}_{i=1}^{N_{pi}}$, where the elements of $\widehat{v}_i$ are predicted by $\widetilde{\mathcal{G}}_p^\dagger$, to satisfy the PDE system in Eq. (5).

We set $N = 400, p = 100, q = 12^2, N_{pi} = 50$, and consider ϕ and ψ to be pointwise function evaluations. The GPs' training mechanisms and the experimental setup for both scenarios are detailed in Sections 2 and 3, respectively. We provide example train and test samples along with our predictions in Fig. 2. It is observed that (1) the predictions achieve small relative $L2$ errors which are 7.35% and 1.65% in the data and physics-informed cases, respectively, and (2) leveraging the physics increases the approximation accuracy.

1.4. Review of related literature

Operator learning aims at approximating mappings between infinite-dimensional function spaces and is particularly well suited for solving parametric nonlinear PDEs. Extensive research over the past few decades has explored operator learning in the context of methods such as polynomial chaos expansion [15–17] or the stochastic finite element method (FEM) [18–20]. These methods

have been widely applied to solving stochastic and parametric PDEs. In essence, they represent both the input function and the PDE solution via a set of suitable basis functions, and then learn a high-dimensional mapping between the coefficients of these two sets of bases. Similarly, GP and numerical homogenization methods have been shown to be data efficient and achieve competitive performance for learning/compressing the solution operator of linear elliptic PDEs with rough coefficients [21–23].

Neural Operators: Fueled by their success in many scientific applications and the advancements in hardware and software, deep NNs are increasingly used for operator approximation. One of the main distinguishing factors among different methods is the network architecture which may be designed via multi-layer perceptrons, convolution layers, attention mechanism, skip connections, graphs, or many other alternatives [9,10,12,24]. We review some of the most prominent methods below and refer the interested reader to [25–27] for more information.

Following the universal approximation theorem for operators [28], a standard DeepONet [29] approximates an operator via a deep NN that combines two sub-networks that encode the input function (branch net) and the spatiotemporal coordinates (trunk net). Since their inception, DeepONets have been extended in a few directions and a particular one that is related to our work is physics-informed DeepONet [30] which aim to reduce the reliance of the network on large training datasets by regularizing the training process based on the known PDE system.

Integral neural operators, first introduced in [24,31], express the solution operator as an integral operator of Green's function. Their network primarily consist of lifting, iterative kernel integration, and projection layers. To accelerate the kernel evaluations and increase efficiency, FNO [11] was developed where the integral kernel is parameterized in the Fourier space where the fast Fourier transform (FFT) and its inverse are used to accelerate the computations. Similar to DeepONets, FNO has been extended in multiple directions to better capture spatial signal behaviors [32] or to build a physics-informed neural operator (PINO) that leverages the physics [33] to reduce the reliance on training data.

To build more flexible neural operators that accommodate irregular geometries and sampling grids, [34] introduces a coordinate-based model that leverages implicit neural representations to encode functions into low-dimensional latent spaces and, in turn, infer mappings between function representations in the latent space. Another example is GNOT [35] which is a transformer-based model that leverages attention and geometric gating mechanisms to handle irregular data grids and multiple input functions. Multi-grid techniques which are ubiquitously used in numerical solvers have also been combined with deep NNs to increase the accuracy and flexibility of neural operators [36–38].

Operator-valued Kernels: Functional data analysis (FDA) [39] also deals with learning mappings between function spaces and has been extensively studied in the statistics literature as a natural extension of multi-response (aka multi-task or multi-output function) learning problems [40–44]. In this context, the focus has primary been on the use of RKHS theory which naturally enables the construction of optimal recovery maps between function spaces via the representer formula [45–47]. In the context of learning PDE operators, recently, [13] introduced a framework based on the theory of operator-valued RKHS and GPs where the reconstruction maps (corresponding to the observation operators) as well as the finite-dimensional map connecting u^p and v^q are all formulated as optimal recovery problems whose solution can be identified by the representer formula. Motivated by this work and FNO, kernel neural operators [48] are recently developed that use parameterized, closed-form, finitely-smooth, and compactly-supported kernels with trainable sparsity parameters in integral operators. A more technical review of [49] and its relation to this article is provided in Section 2.4.

GP Regression: The mapping in Eq. (4) is between finite-dimensional Euclidean spaces and hence can be numerically approximated via GPs which have long been used for probabilistic regression [50–55]. Over the past few years, GPs have attracted some attention for solving nonlinear PDE systems. The majority of existing works [56–58] employ zero-mean GPs and with this choice, solving a PDE system amounts to designing the GP's kernel whose hyperparameters are obtained via either MLE or a regularized MLE where the penalty term quantifies the GP's error in satisfying the PDE system. Using zero-mean GPs, [46] casts solving nonlinear PDEs as an optimal recovery problem whose loss function is derived based on the PDE system and aims to estimate the solution at a finite number of interior nodes in the domain. Once these values are estimated, the PDE solution is approximated anywhere in the domain via kernel regression. The scalability of this approach can be increased using sparse GPs [58–60]. GPs with non-zero means have been employed for solving PDEs in a data-driven manner [61] but [62] introduced the first robust mechanism for solving nonlinear PDE systems via NN-mean GPs for both forward and inverse problems. This latter work serves as a foundation for the current article.

1.5. Contributions and article outline

We summarize our main contributions as follows:

- We introduce a hybrid GP/NN-based operator learning framework. Our approach incorporates both zero-mean and NN-mean GPs both of which provide very competitive performance. The latter option, in particular, leverages the strengths of both kernel methods and neural operators such as DeepONet and FNO.
- We consider both data-driven and physics-informed operator learning scenarios. In the former case, we use MLE for parameter optimization and set the mean function to be either zero or a neural operator. In the latter case, we only use NN-mean GPs whose parameters are estimated via a weighted combination of MLE and PDE residuals.
- We address the computational costs and ill-conditioning issues associated with the covariance matrix of GPs by (1) leveraging specific kernel structures and (2) proper initialization of the kernel parameters that are either trained (for zero-mean GPs) or frozen (for NN-mean GPs).

- We introduce the first *zero-shot* mechanism for operator learning based on zero-mean GPs whose kernel parameters are properly initialized.
- We test our approach on single- and multi-output operator learning problems and demonstrate how the prediction accuracy is affected by (1) replacing MSE via MLE in data-driven operator learning, (2) augmenting MLE with physics in the case of NN-mean GPs, and (3) using the trained models for extrapolation.

The rest of the paper is organized as follows. We introduce our framework in Section 2 where we consider both data-driven and physics-informed scenarios in Sections 2.1 and 2.2, respectively. Details on the initialization of the kernel parameters, stability of the covariance matrix, and inference costs are also included in Section 2. We compare our approach against competing methods in Section 3 using single- and multi-output operators with and without incorporating physics into the operator learning problem. Concluding remarks and future research directions are provided in Section 4.

2. Proposed framework for operator learning

We first introduce our approach in the context of data-driven operator learning in Section 2.1 and then consider the physics-informed version in Section 2.2 to enable augmenting the training data with physical laws. The data-driven scenario in Section 2.1 can leverage either zero-mean or non-zero mean GPs while the physics-informed one in Section 2.2 relies on non-zero mean GPs. To ensure numerical stability and computational efficiency, in Section 2.3 we study the kernel parameters and provide general guidance for properly initializing them and potentially updating them in zero-mean and NN-mean GPs. We conclude this section by comparing our approach to the optimal recovery-based approach of [13] and then commenting on its inference complexity in Sections 2.4 and 2.5, respectively.

2.1. Data-driven operator learning

2.1.1. Single-output operators

In Section 1.2 we transformed the operator learning problem into an equivalent regression one with the goal of learning $\widetilde{\mathcal{G}}_p^{\dagger} : \mathbb{R}^n \times \Omega_v \to \mathbb{R}$. To approximate $\widetilde{\mathcal{G}}_p^{\dagger}$ we start by placing a GP prior on it:

$$\widetilde{\mathcal{G}}_p^{\dagger} \sim GP\left(m(\phi(u), y; \theta), c\left(\left[\phi(u), y\right], \left[\phi(u'), y'\right]; \beta, \sigma^2\right)\right) \tag{6}$$

where $\phi(u)$ is the linear observation operator defined in Eq. (2), $m(\phi(u), y; \theta)$ is the prior mean function parameterized by θ, and $c\left(\left[\phi(u), y\right], \left[\phi(u'), y'\right]; \beta, \sigma^2\right)$ is a kernel with hyperparameters β and σ^2. We elaborate on the choice of the mean function later in this section. As for the kernel, there are many choices available but in this paper we employ the Gaussian or Matérn covariance functions defined as:

$$c(x, x'; \beta, \sigma^2) = \sigma^2 \exp\left\{-\left(x - x'\right)^T \operatorname{diag}(\beta)\left(x - x'\right)\right\} = \sigma^2 \exp\left\{-\sum_{i=1}^{d_x} \beta_i (x_i - x_i')^2\right\} \tag{7a}$$

$$c(x, x'; \beta, \sigma^2) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\left(x - x'\right)^T \operatorname{diag}(\beta)\left(x - x'\right)\right)^{\nu} K_{\nu}\left(\sqrt{2\nu}\left(x - x'\right)^T \operatorname{diag}(\beta)\left(x - x'\right)\right) \tag{7b}$$

where x denotes the feature vector with length d_x, β are the length-scale parameters, σ^2 is the process variance, K_{ν} is the modified Bessel function of the second kind, and Γ is the gamma function. It is common practice [63,64] to restrict $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$ since these values lead to simplified expressions of K_{ν} and hence a substantial decrease in the computational cost. In addition, ν is not typically optimized but rather chosen a priori to avoid evaluating repeatedly K_{ν}.

The inductive bias that the kernels in Eq. (7) encode into the learning process is that close-by inputs x and x' have similar (i.e., correlated) output values. The degree of this correlation also depends on β whose magnitude quantifies the pace at which the correlations die out as the distance between x and x' increases.

Predicting $\mathcal{G}^{\dagger}(u^*)(y^*)$ for an unseen input function u^* at unseen output location y^* requires the estimation of θ, σ^2, and β. To this end, we combine the training data $\mathcal{D} := \{\phi(u_i), \psi(v_i)\}_{i=1}^{N}$ with maximum likelihood estimation. To accommodate our regression problem, we reorganize $\mathcal{D}$ as $\{(\phi(u_i), y_j), [\psi(v_i)]_j\}_{i,j=1}^{N,q}$ where $\phi(u_i) \in \mathbb{R}^p$ and $y_j \in \Omega_v$ denote an input pair whose corresponding scalar output is $[\psi(v_i)]_j \in \mathbb{R}$. For notational convenience, we denote the inputs and outputs of this dataset by $\boldsymbol{X} = \left\{\phi(u_i), y_j\right\}_{i,j=1}^{N,q}$ and $\boldsymbol{v} = \left\{[\psi(v_i)]_j\right\}_{i,j=1}^{N,q}$, respectively. The MLE process involves the following maximization problem:

$$\left[\hat{\theta}, \hat{\beta}, \hat{\sigma}^2\right] = \underset{\theta \in \Theta, \beta \in \mathcal{B}, \sigma^2 \in \mathbb{R}^+}{\operatorname{argmax}} \quad (2\pi)^{-Nq/2} |\boldsymbol{C}|^{-\frac{1}{2}} \exp\left\{\frac{-1}{2}(\boldsymbol{v} - \boldsymbol{m})^T \boldsymbol{C}^{-1}(\boldsymbol{v} - \boldsymbol{m})\right\}, \tag{8}$$

or equivalently:

$$\left[\hat{\theta}, \hat{\beta}, \hat{\sigma}^2\right] = \underset{\theta \in \Theta, \beta \in \mathcal{B}, \sigma^2 \in \mathbb{R}^+}{\operatorname{argmin}} \quad \frac{1}{2}\log|\boldsymbol{C}| + \frac{1}{2}(\boldsymbol{v} - \boldsymbol{m})^T \boldsymbol{C}^{-1}(\boldsymbol{v} - \boldsymbol{m}), \tag{9}$$

where $\boldsymbol{C} = c\left(\boldsymbol{X}, \boldsymbol{X}; \beta, \sigma^2\right)$ is the $Nq \times Nq$ symmetric positive-definite covariance matrix, $\boldsymbol{m} = m(\boldsymbol{X}; \theta)$ is the $Nq \times 1$ vector of mean function evaluations at $\boldsymbol{X}$, Θ and $\mathcal{B}$ denote the search spaces for θ and β, respectively, and $\log|\boldsymbol{C}|$ denotes the log-determinant of $\boldsymbol{C}$. The $\boldsymbol{v} - \boldsymbol{m}$ vector quantifies the residuals of the mean function in reproducing the training outputs and is discussed further in Section 2.1.3.

The minimization problem in Eq. (9) is almost always solved with a gradient-based optimization technique where the parameters are first initialized and then iteratively updated until a convergence metric (e.g., maximum number of iterations or function evaluations) is met. The overall computational cost of this iterative process is dominated by the repeated inversion of the covariance matrix $\boldsymbol{C}$. Specifically, the computational complexity associated with $\boldsymbol{C}^{-1}$ is $\mathcal{O}\left((Nq)^3\right)$ which poses a major challenge in operator learning problems where N is typically in the order of thousands. Even with relatively small N, inverting $\boldsymbol{C}$ can still be very costly for high-resolution data where q is large.

To reduce the computational costs of inverting $\boldsymbol{C}$, we exploit the structure of the data where we assume that the discretization of u and the locations where v is observed are fixed across the N training samples. This assumption is quite reasonable in many operator learning problems and allows us to leverage the Kronecker product properties to bypass the construction and inversion of $\boldsymbol{C}$. To this end, we adopt a separable kernel of the form:

$$c([\phi(u), y], [\phi(u'), y']; \beta, \sigma^2) = c_\phi(\phi(u), \phi(u'); \beta_\phi, \sigma_\phi^2) c_y(y, y'; \beta_y, \sigma_y^2) \tag{10}$$

where we use Eq. (7) for both kernels on the right-hand side. Without loss of generality, we assume $\sigma_y^2 = 1$ hereafter.[1] Using this formulation and the assumption on the data structure we rewrite the covariance matrix in Eq. (9) as:

$$\boldsymbol{C} = \boldsymbol{C}_\phi \otimes \boldsymbol{C}_y, \tag{11}$$

where $\boldsymbol{C}_\phi = c_\phi(\boldsymbol{U}, \boldsymbol{U}; \beta_\phi, \sigma_\phi^2)$ and $\boldsymbol{C}_y = c_y(\boldsymbol{Y}, \boldsymbol{Y}; \beta_y)$ with $\boldsymbol{U} = \left\{\phi(u_1), \dots, \phi(u_N)\right\}$ and $\boldsymbol{Y} = \left\{y_1, \dots, y_q\right\}$, $\beta = \left[\beta_\phi, \beta_y\right]$. In this equation, $\boldsymbol{C}_\phi$ and $\boldsymbol{C}_y$ model the correlations among the discretized input functions and the locations where the output function is observed, respectively. Using the Kronecker product in Eq. (11) we can simplify three of the computationally expensive operations in Eq. (9):

$$\boldsymbol{C}^{-1} = \boldsymbol{C}_\phi^{-1} \otimes \boldsymbol{C}_y^{-1}, \tag{12a}$$

$$\boldsymbol{C}^{-1}(\boldsymbol{v} - \boldsymbol{m}) = \mathrm{vec}\left(\boldsymbol{C}_y^{-1}(\boldsymbol{V} - \boldsymbol{M})\boldsymbol{C}_\phi^{-1}\right) \tag{12b}$$

$$|\boldsymbol{C}| = \left|\boldsymbol{C}_\phi\right|^q \left|\boldsymbol{C}_y\right|^N, \tag{12c}$$

where $\boldsymbol{V}$ and $\boldsymbol{M}$ are formed by reshaping $\boldsymbol{v}$ and $\boldsymbol{m}$ to $q \times N$ matrices, respectively, and $\mathrm{vec}(\cdot)$ denotes the vectorization operation. Using Eq. (12) we now rewrite Eq. (9) as:

$$\begin{aligned}\left[\hat{\theta}, \hat{\beta}, \hat{\sigma}_\phi^2\right] &= \underset{\theta\in\Theta, \beta\in\mathcal{B}, \sigma_\phi^2\in\mathbb{R}^+}{\operatorname{argmin}} \quad \mathcal{L}_{\mathrm{MLE}}(\theta, \beta, \sigma_\phi^2) \\ &= \underset{\theta\in\Theta, \beta\in\mathcal{B}, \sigma_\phi^2\in\mathbb{R}^+}{\operatorname{argmin}} \quad \log\left(\left|\boldsymbol{C}_\phi\right|^q \left|\boldsymbol{C}_y\right|^N\right) + (\boldsymbol{v} - \boldsymbol{m})^T \mathrm{vec}\left(\boldsymbol{C}_y^{-1}(\boldsymbol{V} - \boldsymbol{M})\boldsymbol{C}_\phi^{-1}\right),\end{aligned} \tag{13}$$

which can be used to simultaneously estimate the parameters of the mean and covariance functions of the GP.

We highlight that the optimization problems in Eqs. (9) and (13) both correspond to our generic GP-based operator learning framework except that the latter one exploits the data structure and the kernel separability assumption in Eq. (10) to decrease the computational costs and memory demands. If the locations where the output function is observed or the discretization of the input function vary across the samples, Eq. (13) can still be used by either pre-processing the data (which may introduce some additional errors) or sparse GPs.

For a zero-mean GP, Eq. (13) simplifies to:

$$\begin{aligned}\left[\hat{\beta}, \hat{\sigma}_\phi^2\right] &= \underset{\beta\in\mathcal{B}, \sigma_\phi^2\in\mathbb{R}^+}{\operatorname{argmin}} \quad \mathcal{L}_0(\beta, \sigma_\phi^2) \\ &= \underset{\beta\in\mathcal{B}, \sigma_\phi^2\in\mathbb{R}^+}{\operatorname{argmin}} \quad q\log\left(\left|\boldsymbol{C}_\phi\right|\right) + N\log\left(\left|\boldsymbol{C}_y\right|\right) + \boldsymbol{v}^T \mathrm{vec}\left(\boldsymbol{C}_y^{-1}\boldsymbol{V}\boldsymbol{C}_\phi^{-1}\right),\end{aligned} \tag{14}$$

which can be solved via a first-order gradient-based optimizer (e.g., Adam) that leverages automatic differentiation to obtain the gradients of $\mathcal{L}_0(\beta, \sigma_\phi^2)$ with respect to β and σ_ϕ^2. In typical GP regression problems, it is common practice to initialize gradient-based optimizers multiple times to avoid local optimality. This process substantially increases the cost of minimizing $\mathcal{L}_0(\beta, \sigma_\phi^2)$ due to the high costs of repeatedly constructing and inverting $\boldsymbol{C}_\phi$ and $\boldsymbol{C}_y$ to calculate $\mathcal{L}_0(\beta, \sigma_\phi^2)$ and its gradients. We address this computational issue in Section 2.3 where we show that β_y and β_ϕ can be, respectively, fixed and initialized to some judiciously chosen values to, in turn, optimize $\mathcal{L}_0(\beta_\phi, \sigma_\phi^2)$ only once.

For a GP whose mean function is parameterized with a deep NN (e.g., FNO or DeepONet), minimizing $\mathcal{L}(\theta, \beta, \sigma_\phi^2)$ can be prohibitively costly and memory intensive (even if β are initialized well and the optimization problem in Eq. (13) is solved only once) since differentiating $\mathcal{L}(\theta, \beta, \sigma_\phi^2)$ with respect to θ involves large matrices. Moreover, we show in Section 2.3 that the simultaneous estimation of θ and β renders the covariance matrices ill-conditioned. As detailed in Section 2.3, we address these computational issues by fixing β and σ_ϕ^2 to some values that ensure (1) the covariance matrices are numerically stable, and (2) the posterior

[1] Alternatively, we could have also assumed $\sigma_\phi^2 = 1$ and estimate σ_y^2 instead, leading to the same result.

distributions in Section 2.1.3 regress the training data. Fixing β and σ_ϕ^2 makes the first term in Eq. (13) a constant and hence we can estimate θ via:

$$\widehat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}} \;\; \mathcal{L}_{\text{nn}}(\theta) = \underset{\theta\in\Theta}{\operatorname{argmin}} \;\; (\boldsymbol{v}-\boldsymbol{m})^T \operatorname{vec}\Big(\boldsymbol{C}_y^{-1}(\boldsymbol{V}-\boldsymbol{M})\boldsymbol{C}_\phi^{-1}\Big). \tag{15}$$

where $\boldsymbol{m}$ and $\boldsymbol{M}$ are the only terms that depend on θ, i.e., we can construct $\boldsymbol{C}_y^{-1}$ and $\boldsymbol{C}_\phi^{-1}$ once and store them so that evaluating $\mathcal{L}_{\text{nn}}(\theta)$ and its gradients only relies on matrix multiplications.

We notice in Eq. (15) that if $\boldsymbol{C}_y$ and $\boldsymbol{C}_\phi$ are identity matrices (i.e., if the correlations among different u^p and different query points y are eliminated), $\mathcal{L}_{\text{nn}}(\theta)$ becomes the mean squared error (MSE) loss function that is typically used in operator learning. We demonstrate the benefits of using $\mathcal{L}_{\text{nn}}(\theta)$ instead of MSE in Section 3.

2.1.2. Multi-output operators

Consider now the following multi-output operator

$$\mathcal{G}^\dagger : \mathcal{U} \to \prod_{s=1}^{S} \mathcal{V}^s \tag{16}$$

where S is the total number of (possibly related) output functions. In this case, our goal is to approximate $\mathcal{G}^\dagger$ given the training dataset $\mathcal{D}^S := \{s, \phi(u_i), \psi(v_i^s)\}_{i=1}^N$ where $\{u_i, v_i^s\}_{i=1}^N$ are elements from $\mathcal{U}\times\mathcal{V}^s$ with $s = \{1,\dots,S\}$. Analogously to Section 1.2, we cast the operator learning problem in Eq. (16) as the following regression problem:

$$\widetilde{\mathcal{G}}_p^\dagger : \mathbb{R}^p \times \mathcal{V}^* \to \mathbb{R}^S. \tag{17}$$

where now $\widetilde{\mathcal{G}}_p^\dagger$ is a *multi-output* function and $\mathcal{V}^*$ denotes the dual of the product space $\prod_{s=1}^S \mathcal{V}^s$. To approximate $\widetilde{\mathcal{G}}_p^\dagger$, we start by placing a multi-task or $S-$dimensional GP prior [65,66] on it:

$$\widetilde{\mathcal{G}}_p^\dagger \sim GP\left(m(\phi(u), y; \theta)(s), c\left([s, \phi(u), y], [s, \phi(u), y]'; \beta, \sigma^2\right)\right) \tag{18}$$

where $m(\phi(u), y; \theta)$ is the $S-$dimensional mean function of the GP.

The covariance matrix of the training data cannot be used in training unless $c\left([s, \phi(u), y], [s, \phi(u), y]'; \beta, \sigma^2\right)$ possesses certain features that eliminate the need to build and invert the $NSq \times NSq$ covariance matrix. Hence, we invoke the assumptions made on the data structure in Section 2.1.1 and also presume that all the outputs are observed at the same collocation points (CPs). Following these assumptions, we formulate the kernel in Eq. (18) as:

$$c\left([s, \phi(u), y], [s, \phi(u), y]'; \beta, \sigma^2\right) = c_s(s, s'; \beta_s, \sigma_s^2) c_y(y, y'; \beta_y, \sigma_y^2) c_\phi(\phi(u), \phi(u'); \beta_\phi, \sigma_\phi^2), \tag{19}$$

where $c_y(y, y'; \beta_y, \sigma_y^2)$ and $c_\phi(\phi(u), \phi(u'); \beta_\phi, \sigma_\phi^2)$ are standard kernels such as the Gaussian or Matérn defined in Eq. (7), $\beta = [\beta_s, \beta_y, \beta_\phi]$, and $c_s(s, s'; \beta_s, \sigma_s^2)$ captures the average correlations among the responses. Once again, and without loss of generality, we assume $\sigma_y^2 = 1$ and hence only consider σ_ϕ^2 and σ_s^2. The kernel in Eq. (19) has a separable structure and is stationary since all the kernels on the right-hand side are stationary.

To further simplify the computations we set $c_s(s, s'; \beta_s)$ to a diagonal matrix, that is:

$$c_s(s, s') = \mathbb{1}\left\{s == s'\right\} \sigma_s^2 \tag{20}$$

where $\mathbb{1}\{\cdot\}$ returns 1/0 if the enclosed statement is true/false and σ_s^2 is the kernel variance for the $s-$th response. With this choice, the correlations between the different responses are either neglected or captured by the mean function in zero-mean and NN-mean GPs, respectively.

To estimate θ, β, and σ^2 we use the training dataset $\mathcal{D}^S$ to maximize the likelihood function. We rearrange $\mathcal{D}^S$ as $\{(s, \phi(u_i), y_j), [\psi(v_i^s)]_j\}_{i,j,s=1}^{N,q,S}$ and denote its inputs and outputs via $\boldsymbol{X} = \left\{\phi(u_i), y_j\right\}_{i,j=1}^{N,q}$ and $\boldsymbol{v} = \left\{[\psi(v_i^s)]_j\right\}_{i,j,s=1}^{N,q,S}$, respectively, where we have used the assumptions that $\boldsymbol{X}$ do not depend on s. The MLE process for our multi-output regression task involves the following minimization problem:

$$\begin{aligned}\left[\widehat{\theta}, \widehat{\beta}, \widehat{\sigma}^2\right] &= \underset{\theta\in\Theta, \beta\in\mathcal{B}, \sigma^2\in\mathbb{R}^+}{\operatorname{argmin}} \;\; \frac{1}{2}\log|\Sigma\otimes\boldsymbol{C}| + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{m})^T(\Sigma\otimes\boldsymbol{C})^{-1}(\boldsymbol{v}-\boldsymbol{m}) \\ &= \underset{\theta\in\Theta, \beta\in\mathcal{B}, \sigma^2\in\mathbb{R}^+}{\operatorname{argmin}} \;\; \frac{1}{2}\log|\Sigma|^{Nq} + \frac{1}{2}\log|\boldsymbol{C}|^S + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{m})^T\left(\Sigma^{-1}\otimes\boldsymbol{C}^{-1}\right)(\boldsymbol{v}-\boldsymbol{m}),\end{aligned} \tag{21}$$

where $\sigma^2 = \left[\sigma_\phi^2, \sigma_1^2, \dots, \sigma_S^2\right]$, $\Sigma = \operatorname{diag}\left\{\sigma_1^2, \dots, \sigma_S^2\right\}$, $\boldsymbol{C} = \boldsymbol{C}_\phi \otimes \boldsymbol{C}_y$ is the $Nq \times Nq$ symmetric positive-definite covariance matrix, and $\boldsymbol{m}$ is a column vector of length NqS that corresponds to the evaluations of the mean function for all the outputs, samples, and query points.

Similar to the previous section, we leverage the properties of the Kronecker product in Eq. (12) to simplify Eq. (21) to:

$$\begin{aligned}\left[\widehat{\theta}, \widehat{\beta}, \widehat{\sigma}^2\right] = \underset{\theta\in\Theta, \beta\in\mathcal{B}, \sigma^2\in\mathbb{R}^+}{\operatorname{argmin}} \;\; &\frac{1}{2}Nq\sum_{s=1}^{S}\log\sigma_s^2 + \frac{1}{2}\log\left(\left|\boldsymbol{C}_\phi\right|^{qS}\left|\boldsymbol{C}_y\right|^{NS}\right) \\ &+ \frac{1}{2}\sum_{s=1}^{S}\sigma_s^{-2}(\boldsymbol{v}^s-\boldsymbol{m}^s)^T\operatorname{vec}\Big(\boldsymbol{C}_y^{-1}(\boldsymbol{V}^s-\boldsymbol{M}^s)\boldsymbol{C}_\phi^{-1}\Big),\end{aligned} \tag{22}$$

where $\boldsymbol{V}^s$ and $\boldsymbol{M}^s$ are formed by reshaping $\boldsymbol{v}^s$ and $\boldsymbol{m}^s$ to $q\times N$ matrices, respectively, $\boldsymbol{v}^s = \left\{[\psi(v_i^s)]_j\right\}_{i,j}^{N,q}$, and $\boldsymbol{m}^s = m(s,\boldsymbol{X};\theta)$ denotes the evaluations of the mean function for response s.

Parameter estimation now follows as in Section 2.1.1. Specifically, for a zero-mean GP, Eq. (22) simplifies to:

$$\begin{aligned}\left[\widehat{\beta},\widehat{\sigma}^2\right] &= \underset{\beta\in\mathcal{B},\sigma^2\in\mathbb{R}^+}{\operatorname{argmin}} \quad \mathcal{L}_{0v}(\beta,\sigma^2)\\ &= \underset{\beta\in\mathcal{B},\sigma^2\in\mathbb{R}^+}{\operatorname{argmin}} \quad \frac{1}{2}Nq\sum_{s=1}^{S}\log\left(\sigma_s^2\right)+\frac{1}{2}\log\left(\left|\boldsymbol{C}_\phi\right|^{qS}\left|\boldsymbol{C}_y\right|^{NS}\right)+\frac{1}{2}\sum_{s=1}^{S}\sigma_s^{-2}\left(\boldsymbol{v}^s\right)^T\operatorname{vec}\left(\boldsymbol{C}_y^{-1}\boldsymbol{V}^s\boldsymbol{C}_\phi^{-1}\right).\end{aligned} \tag{23}$$

where we can fix β_ϕ similar to the scalar case in Section 2.1.1. For an NN-mean GP, we fix the kernel hyperparameters which makes the first two terms on the right-hand side of Eq. (22) to become constants. Hence, we can estimate θ via:

$$\widehat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}} \quad \mathcal{L}_{\text{nnv}}(\theta) = \underset{\theta\in\Theta}{\operatorname{argmin}} \quad \frac{1}{2}\sum_{s=1}^{S}\left(\boldsymbol{v}^s-\boldsymbol{m}^s\right)^T\operatorname{vec}\left(\boldsymbol{C}_y^{-1}\left(\boldsymbol{V}^s-\boldsymbol{M}^s\right)\boldsymbol{C}_\phi^{-1}\right). \tag{24}$$

2.1.3. Inference

Once the parameters of the mean and covariance functions are estimated, we obtain the predictive response for a test input function u^* at the query point y^* by computing the expected value of the posterior distribution of the GP conditioned on the training data. That is:

$$\begin{aligned}\eta\left(\phi(u^*),y^*;\widehat{\theta},\widehat{\beta},\widehat{\sigma}_\phi^2\right) &:= \mathbb{E}[\widetilde{\mathcal{G}}_p^\dagger(\phi(u^*),\delta_{y^*})|D]\\ &= m(\phi(u^*),y^*;\widehat{\theta})+c\left(\left[\phi(u^*),y^*\right],\boldsymbol{X};\widehat{\beta},\widehat{\sigma}_\phi^2\right)\boldsymbol{C}^{-1}(\boldsymbol{v}-\boldsymbol{m})\\ &= m(\phi(u^*),y^*;\widehat{\theta})+\\ &\quad\left[c_\phi\left(\phi(u^*),\boldsymbol{U};\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2\right)\otimes c_y\left(y^*,\boldsymbol{Y};\widehat{\beta}_y\right)\right]\operatorname{vec}\left(\boldsymbol{C}_y^{-1}\left(\boldsymbol{V}-\boldsymbol{M}\right)\boldsymbol{C}_\phi^{-1}\right)\\ &= m(\phi(u^*),y^*;\widehat{\theta})+\\ &\quad\operatorname{vec}\left(c_y\left(y^*,\boldsymbol{Y};\widehat{\beta}_y\right)\boldsymbol{C}_y^{-1}\left(\boldsymbol{V}-\boldsymbol{M}\right)\boldsymbol{C}_\phi^{-1}c_\phi\left(\boldsymbol{U},\phi(u^*);\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2\right)\right),\end{aligned} \tag{25}$$

where we have leveraged the Kronecker product properties in Eq. (12) to simplify the matrix multiplications. Eq. (25) accommodates batch computations for accelerated prediction of the output field at the m points $\boldsymbol{Y}^* = \left\{y_1^*,\dots,y_m^*\right\}$:

$$\begin{aligned}\eta\left(\phi(u^*),\boldsymbol{Y}^*;\widehat{\theta},\widehat{\beta},\widehat{\sigma}_\phi^2\right) &= m(\phi(u^*),\boldsymbol{Y}^*;\widehat{\theta})+\\ &\quad\operatorname{vec}\left(c_y\left(\boldsymbol{Y}^*,\boldsymbol{Y};\widehat{\beta}_y\right)\boldsymbol{C}_y^{-1}\left(\boldsymbol{V}-\boldsymbol{M}\right)\boldsymbol{C}_\phi^{-1}c_\phi\left(\boldsymbol{U},\phi(u^*);\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2\right)\right)\end{aligned} \tag{26}$$

where $\boldsymbol{U} = \left\{\phi(u_1),\dots,\phi(u_N)\right\}$ and $\boldsymbol{Y} = \left\{y_1,\dots,y_q\right\}$ as defined previously. In the case of multi-output operators, our predictor for output s is:

$$\begin{aligned}\eta\left(s,\phi(u^*),\boldsymbol{Y}^*;\widehat{\theta},\widehat{\beta},\widehat{\sigma}_\phi^2\right) &= m(\phi(u^*),\boldsymbol{Y}^*;\widehat{\theta})(s)\\ &\quad+\operatorname{vec}\left(c_y(\boldsymbol{Y}^*,\boldsymbol{Y};\widehat{\beta}_y)\boldsymbol{C}_y^{-1}\left(\boldsymbol{V}^s-\boldsymbol{M}^s\right)\boldsymbol{C}_\phi^{-1}c_\phi(\boldsymbol{U},\phi(u^*);\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2)\right).\end{aligned} \tag{27}$$

For a zero-mean GP, Eqs. (25)–(27) can be simplified by removing the mean function and the associated matrices $\boldsymbol{M}$ and $\boldsymbol{M}^s$.

The predictor in Eq. (25) is a differentiable function with respect to y^*. We use this property in Section 2.2 for physics-informed operator learning where the training process leverages the known differential operators that govern the problem. This predictor also interpolates the training data which is a feature that most neural operators do not provide. To see this, we use Eq. (26) to predict the output function at $\boldsymbol{Y}$ for the N input functions $\boldsymbol{U}$ that are observed during training:

$$\begin{aligned}\eta\left(\boldsymbol{U},\boldsymbol{Y};\widehat{\theta},\widehat{\beta},\widehat{\sigma}_\phi^2\right) &= m(\boldsymbol{U},\boldsymbol{Y};\widehat{\theta})+\operatorname{vec}\left(c_y(\boldsymbol{Y},\boldsymbol{Y};\widehat{\beta}_y)\boldsymbol{C}_y^{-1}\left(\boldsymbol{V}-\boldsymbol{M}\right)\boldsymbol{C}_\phi^{-1}c_\phi(\boldsymbol{U},\boldsymbol{U};\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2)\right)\\ &= \boldsymbol{m}+\operatorname{vec}\left(I_y\left(\boldsymbol{V}-\boldsymbol{M}\right)I_\phi\right)=\boldsymbol{m}+\boldsymbol{v}-\boldsymbol{m}=\boldsymbol{v}.\end{aligned} \tag{28}$$

Lastly, we highlight that both terms on the right-hand side of Eq. (25) capture the correlations not only in $\mathcal{U}$, but also in the support of $\mathcal{V}$, i.e., Ω_v. That is, our predictor leverages the strengths of both neural operators and kernel methods for operator learning.

2.2. Physics-informed operator learning

Consider the general differential operator $\mathcal{T}$ acting upon $u\in\mathcal{U}$ and $v\in\mathcal{V}$ such that $\mathcal{T}:\mathcal{U}\times\mathcal{V}\to\mathcal{O}$, where $\mathcal{O}$ is the null space, u is the input function, and v denotes the unknown solution. Then, a general family of (possibly nonlinear) time-dependent PDEs can be expressed as:

$$\mathcal{T}(u,v)=0 \quad \text{in } \Omega_v\times(0,\infty), \tag{29a}$$

$$v=f \quad \text{on } \partial\Omega_v\times(0,\infty), \tag{29b}$$

$$v=g \quad \text{in } \bar{\Omega}_v\times\{0\}. \tag{29c}$$

Given Eq. (29) and N pairs of $\{\phi(u_i), \psi(v_i)\}$ that satisfy it, our goal is to approximate the operator $\mathcal{G}^\dagger : \mathcal{U} \to \mathcal{V}$. In this context, $\mathcal{U}$ may represent a family of PDE coefficients $a(y,t) : \Omega_v \times (0,\infty) \to \mathbb{R}$ where t denotes time, boundary conditions $v(y,t) : \partial\Omega_v \times (0,\infty) \to \mathbb{R}$, or initial conditions $v(y,t) : \bar{\Omega}_v \times \{0\} \to \mathbb{R}$.

To improve the approximation accuracy of the predictor in Eq. (25), we regularize the parameter estimation process via Eq. (29). To this end, we use a batch of N_{pi} samples with inputs $\left\{\phi(u_i)\right\}_{i=1}^{N_{pi}}$ which may or may not be unique from the N training samples used throughout Section 2.1. For each of the N_{pi} samples, we consider n_{PDE}, n_{BC}, and n_{IC} collocation points in the domain, on the boundaries, and at $t = 0$, respectively, to obtain the following losses:

$$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_{pi} n_{\text{PDE}}} \sum_{j=1}^{N_{pi}} \sum_{i=1}^{n_{\text{PDE}}} \left(\mathcal{T}(u_j, \eta(\phi(u_j), \delta_{y_i})) \right)^2, \tag{30a}$$

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N_{pi} n_{\text{BC}}} \sum_{j=1}^{N_{pi}} \sum_{i=1}^{n_{\text{BC}}} \left(\eta(\phi(u_j), \delta_{y_i}) - f(y_i) \right)^2, \tag{30b}$$

$$\mathcal{L}_{\text{IC}}(\theta) = \frac{1}{N_{pi} n_{\text{IC}}} \sum_{j=1}^{N_{pi}} \sum_{i=1}^{n_{\text{IC}}} \left(\eta(\phi(u_j), \delta_{y_i}) - g(y_i) \right)^2, \tag{30c}$$

which quantify the errors of Eq. (25) in satisfying Eq. (29). Note that we calculate $\mathcal{L}_{\text{BC}}$ and $\mathcal{L}_{\text{IC}}(\theta)$ since the N_{pi} samples may be different from the N labeled training samples. But if these two sets of samples have identical ICs/BCs, $\mathcal{L}_{\text{BC}}$ and $\mathcal{L}_{\text{IC}}(\theta)$ in Eq. (30) will be zero due to the reproducing property of the predictor, see Eq. (28).

To estimate θ, we combine the loss terms in Eq. (30) with Eq. (15):

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PDE}}(\theta) + \alpha_{\text{BC}}\mathcal{L}_{\text{BC}}(\theta) + \alpha_{\text{IC}}\mathcal{L}_{\text{IC}}(\theta) + \alpha_{\text{MLE}}\mathcal{L}_{\text{nn}}(\theta), \tag{31}$$

where $\alpha_{\text{BC}}, \alpha_{\text{IC}}$, and α_{MLE} are weights that ensure none of the loss terms in Eq. (31) dominates the optimization process. Specifically, we choose $\mathcal{L}_{\text{PDE}}$ as a reference and assign the unit weight to it. Then, we determine the weights of the other loss terms via an Adam-inspired approach [67,68] such that their gradients with respect to θ have comparable magnitudes at all optimization iterations. For example, we obtain α_{BC} at epoch $k+1$ via:

$$\alpha_{\text{BC}}^{k+1} = (1-\lambda)\alpha_{\text{BC}}^{k} + \lambda \frac{\max\left|\nabla_\theta \mathcal{L}_{\text{PDE}}(\theta)\right|}{\text{mean}\left|\nabla_\theta \mathcal{L}_{\text{BC}}(\theta)\right|}, \tag{32}$$

where $\mathcal{L}_{\text{PDE}}(\theta)$ is chosen as the reference term, $\nabla_\theta \mathcal{L}_{\text{BC}}(\theta)$ denotes the gradient of $\mathcal{L}_{\text{BC}}(\theta)$ with respect to θ, and λ is typically set to 0.9.

We conclude this section by highlighting three points. First, if the regularized loss function in Eq. (31) involves nonlinear functions of the predictor (i.e., if the PDE system in Eq. (29) is nonlinear), then the posterior in Eq. (25) is not a Gaussian and the predictor is merely a maximum a posteriori (MAP) estimator. Second, for physics-informed operator learning we only use NN-mean GPs with fixed kernel parameters because (1) optimizing the kernel parameters requires the repeated inversion of the covariance matrices which is computationally expensive and prone to ill-conditioning, and (2) our kernels are quite simple and have insufficient parameters to accommodate residual minimization. Third, for multi-output operators we replace $\mathcal{L}_{\text{nn}}(\theta)$ by $\mathcal{L}_{\text{nnv}}(\theta)$ in Eq. (24).

2.3. Parameter initialization and stability

For a zero-mean GP we fix β_y to some *large* values and only optimize β_ϕ and σ_ϕ^2 in Eq. (14) or Eq. (23). We rationalize this decision in Section 2.3.1 and then in Section 2.3.2 we elaborate on our logic for initializing β_ϕ to some *small* values before they are optimized. For an NN-mean GP, we optimize θ while $\beta = \left[\beta_y, \beta_\phi\right]$ and σ_ϕ^2 are all fixed to their initial values which are the same as those for a zero-mean GP. As detailed in Section 2.3.3 this approach dramatically accelerates the computations while reducing the numerical instabilities.

2.3.1. Effect of observation operator

In many operator learning problems $\psi(v)$ provides dense observations, i.e., q in Eq. (2) is a relatively large number. To demonstrate the effect of dense observations on the optimum kernel parameters of a zero-mean GP, we consider the simple task of interpolating $u(x) = \sin(2\pi x), x \in [0,1]$ using a zero-mean GP with the Gaussian kernel $c(x,x';\beta,\sigma^2)$. We set $\sigma^2 = 1$ without loss of generality and sample $N \gg 1$ points from $u(x)$. Instead of optimizing β via MLE, we vary it in the $\left[0, 10^6\right]$ range and for each value we calculate the error of the corresponding conditional predictor in Eq. (28) on a large set of test samples.

The results of this study are summarized in Fig. 3 which clearly shows two pathological regimes for $\beta \to 0$ and $\beta \to \infty$ (i.e., not all values of β result in accurate interpolation of the test samples, even when abundant data is available). This trend is expected since $1/\beta$ is proportional to the correlation length. For this particular example, $\beta < 1$ renders the covariance matrix ill-conditioned while $\beta > 10^4$ prevents the GP from modeling the spatial correlations. For instance, consider points A and B on the curve which mark $\beta = 10^{-1}$ and $\beta = 10^3$, respectively, and the corresponding insets illustrate the predictions of the resulting GPs. We observe in inset A that the prediction curve with $\beta = 10^{-1}$ is quite rough which is due to the ill-conditioning of the GP's covariance matrix. In contrast, for $\beta \in \left[10^1, 3.50 \times 10^3\right]$ the resulting GP interpolates unseen data very accurately and provides a smooth prediction curve. Further increasing β reduces the prediction accuracy as the posterior mean cannot leverage the spatial correlations at all.

Our studies indicate that the trends in Fig. 3 extend to not only higher dimensions, but also the profile of the likelihood function (we further elaborate on the latter point in Section 2.3.2). Following these observations, in all of our studies we set β_y to large

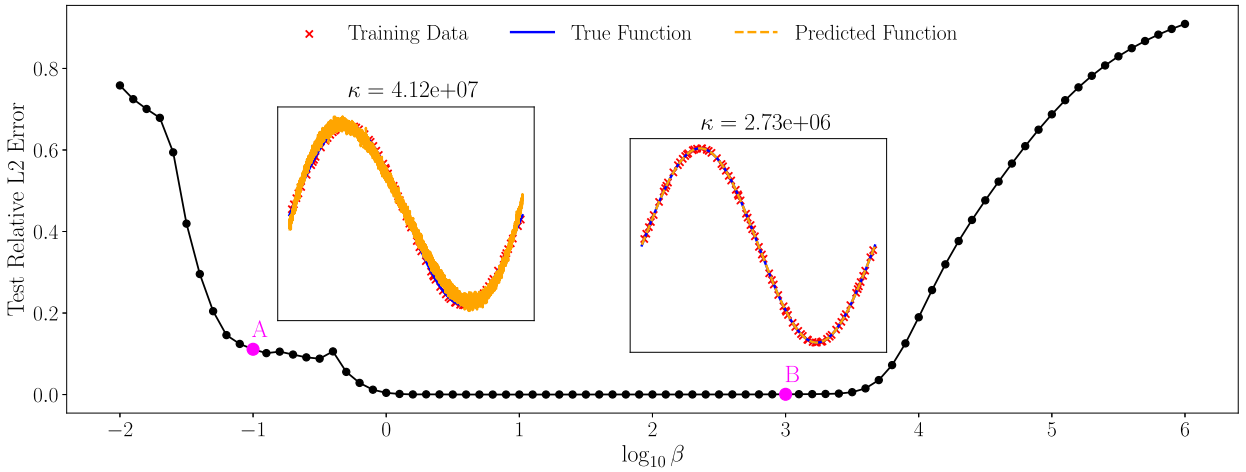


Fig. 3. Effect of β_y on test accuracy with dense observations: Very small and large β values provide poor performance as they cause numerical issues and cannot capture spatial correlations, respectively. However, a relatively large range of values such as 10^3 provide high accuracy and numerical stability.

values, e.g., 3, regardless of d (i.e., the dimensionality of the output space Ω_v) and the choice of the GP's mean function (NN or zero). This decision ensures that C_y is numerically stable and the GP can faithfully interpolate the dense observations in Ω_v.

2.3.2. High-dimensional features in GPs

The performance of GPs deteriorates in very high feature spaces [69] due to the so-called curse of dimensionality: kernels capture the correlations between the samples based on their distance in the feature space. Most kernels such as those in Eq. (7) poorly characterize these correlations in high-dimensional feature spaces as the quantified distances quickly increase and, in turn, render the correlations among samples to become zero. To illustrate this, consider the kernel in Eq. (7a) with $\sigma^2 = 1$. Assuming the same length-scale parameter β is used across all d_x dimensions, we can express $c(x, x'; \beta)$ as a function of the distance between two feature vectors x and x':

$$c(x, x'; \beta) = \exp\left\{-\beta \left|\left|x - x'\right|\right|^2\right\}, \tag{33}$$

which can be solved for β:

$$\beta = -\frac{\log c(x, x')}{||x - x'||^2}. \tag{34}$$

To explore the interaction between d_x and β, we perform the following experiment. Given fixed covariance values $c(x, x') = 0.2$ and $c(x, x') = 0.8$, we generate 10,000 random samples in the d_x−dimensional unit hypercube where the norm of each sample is used as the denominator of Eq. (34). Then, we calculate β corresponding to each sample and repeat this process for various values of d_x. The corresponding histograms are provided in Fig. 4 and demonstrate that as d_x increases, the range of β values that achieve the specified covariances not only shrinks (compare the width of the histograms for $d_x = 2$ and $d_x = 2000$), but also includes smaller values.

Following the above experiment, we initialize β_ϕ at 10^{-2} in all of our studies in this paper since the input function is typically discretized at a large number of points. This value is obviously not an optimal choice so in the case of zero-mean GPs we associate each input feature with its own β and then fine tune them via MLE. In the case of NN-mean GPs, we do not optimize these initial values (due to computational costs and numerical instabilities) and rely on the neural operator part of the GP to correct (if needed) how the correlations between $\phi(u)$ and $\phi(u')$ are quantified.

Our rationale for initializing β_ϕ at a small value, e.g., 10^{-2}, can also be explained as follows. Consider the Gaussian kernel $c(u, u'; \beta, \sigma^2 = 1)$:

$$c(u, u'; \beta, \sigma^2 = 1) = \exp\left\{-\beta \left|\left|u - u'\right|\right|^2\right\}, \tag{35}$$

where $u(x)$ and $u'(x)$ are one-dimensional functions and $||\cdot||$ is the $L2$ norm that can be approximated as:

$$\left|\left|u - u'\right|\right|^2 = \int_\Omega (u(x) - u'(x))^2 dx \approx \sum_{i=1}^{p} (u(x_i) - u'(x_i))^2 \Delta x_i, \tag{36}$$

where Δx_i is the discretization step. Assuming the points are equally spaced, Eq. (35) simplifies to:

$$c(u, u'; \beta, \sigma^2 = 1) = \exp\left\{-\underbrace{\beta \Delta x}_{\beta_u} \sum_{i=1}^{p} (u(x_i) - u'(x_i))^2\right\}. \tag{37}$$

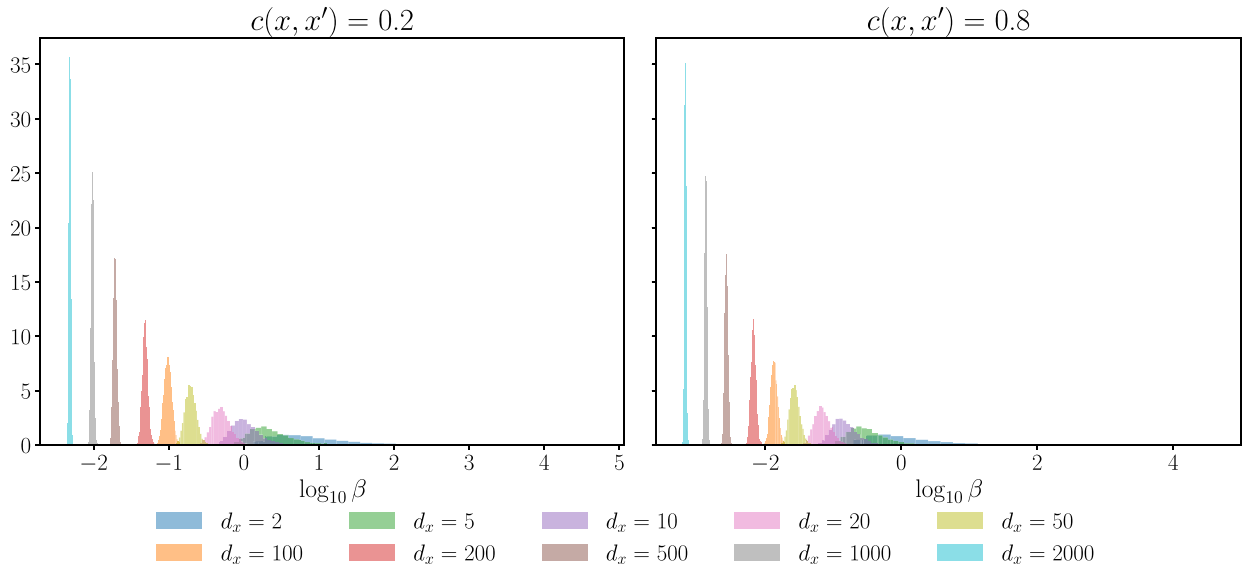


Fig. 4. Interaction between β and feature space dimensionality: Smaller β values are needed to achieve a desired correlation value in high-dimensional feature spaces.

We observe that for function inputs the effective length-scale parameter β_u corresponds to the product of the length-scale β and the discretization step Δx, which is generally small. So, compared to the case where the features are scalars by nature, smaller values should be chosen for the effective length-scale parameter if the features are obtained by finely discretizing a function.

Once again, these findings are also validated in Fig. 5(a.2) and (b.2), where we plot the test relative $L2$ errors as functions of β_y and β_ϕ for the Burgers' and Darcy benchmarks. Our experiments suggest that the range of values for β_ϕ (corresponding to the functional input) that minimize the test error is comparably smaller to β_y (corresponding to scalar inputs). An interesting observation from Fig. 5(a.2) and (b.2) is that the test error is minimized across a broad range of β_y and β_ϕ values. This suggests that a zero-mean GP, when properly initialized, can achieve high accuracy without requiring any training.

To examine the interaction between the initialization of β_y and β_ϕ, we consider the Burgers' and Darcy problems studied in Section 3. Applying zero-mean GPs to these two problems, we visualize the training loss and test error maps as a function of the kernel parameters in Fig. 5. We observe that in all cases our suggested initial values provide minimal values for these metrics in both problems. Interestingly, we notice that many parameter combinations effectively minimize the loss and error maps in both problems, indicating that a high accuracy is obtained by a *zero-shot* zero-mean GP whose kernel parameters are properly initialized but never optimized. We examine the performance of zero-shot and one-shot zero-mean GPs in Section 3 which benefit from zero and one optimization iterations, respectively.

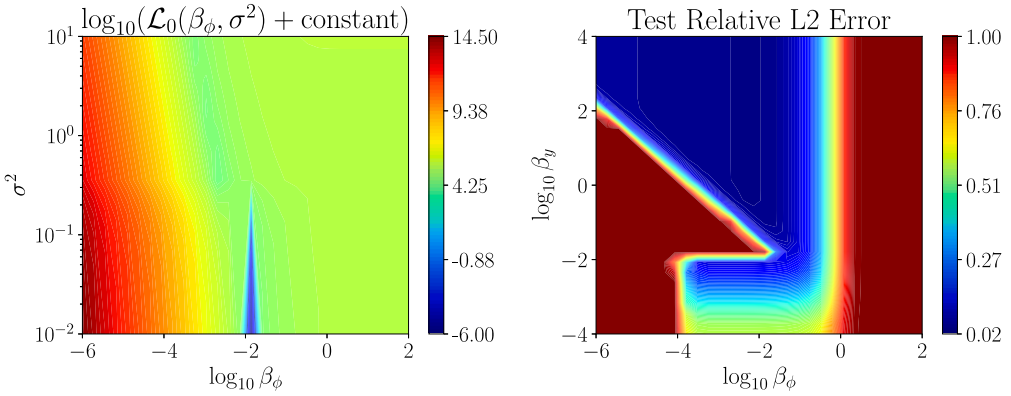
2.3.3. Parameter optimization in NN-mean GPs

Non-zero mean functions have the potential to increase the extrapolation capabilities of GPs [57] and they have been effectively employed for materials modeling [70] or to solve forward and inverse problems in PDEs [62]. Specifically, parameterizing the mean function of a GP by a deep NN provides the means to leverage the strengths of both kernels and NNs as the former effectively learns local trends by interpolating neighboring points in regions with abundant data, while the latter improves generalization by learning complex distributed representations for unseen inputs. However, this added model complexity increases the risk of overfitting and can lead to numerical instability during optimization [57]. We demonstrate these issues with a simple regression example which motivates our decision for fixing the kernel parameters while optimizing θ.

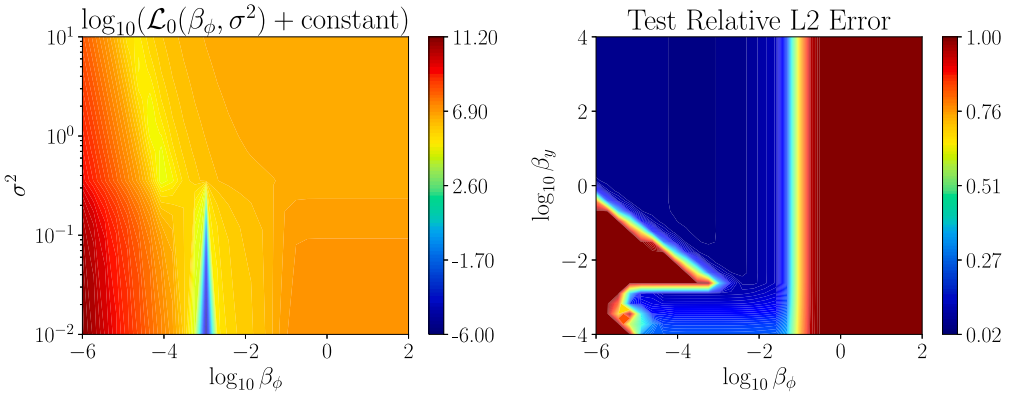
We again consider the function $u(x) = \sin(2\pi x)$, $x \in [0, 1]$ and sample $N = 20$ points from it such that the samples are concentrated close to the lower- and upper-bounds of x, see the red crosses in Fig. 6. We fit three GPs to this dataset:

- Zero-mean GP: We set the kernel to $c(x, x'; \beta, \sigma^2 = 1)$ and use MLE to optimize β.
- NN-mean GP: We set the kernel and the mean function to, respectively, $c(x, x'; \beta, \sigma^2 = 1)$ and a fully-connected feed-forward NN with 4 layers of 20 neurons. We use MLE to jointly optimize the parameters of the kernel and the NN.
- Physics-informed NN-mean GP: We set the kernel to $c(x, x'; \beta = 10^3, \sigma^2 = 1)$ and optimize the NN parameters to minimize a weighted combination of MLE and MSE where the latter encourages the GP-based predictor to satisfy the ordinary differential equation $\frac{du}{dx} = 2\pi \cos(2\pi x)$.

In all cases the Adam optimizer with a learning rate scheduler is used where the initial learning rate and maximum number of epochs are set to 10^{-3} and 20,000, respectively. The results of our studies are summarized in Fig. 6 and demonstrate that the



(a) Loss and error profiles for the $1D$ Burgers' problem in Equation (43).



(b) Loss and error profiles for the Darcy problem in Equation (44).

Fig. 5. Loss and error profiles as functions of kernel parameters in zero-mean GPs: A wide range of parameter combinations provide optimal loss and error values, indicating that proper initialization is relatively easy and results in good performance even without training. $\beta_y = 10^3$ is used for the two loss profiles on the left-hand side of (a) and (b). Note that we do not show the dependency of the test relative $L2$ error with respect to σ_ϕ^2 since this does not affect Eq. (25). For Darcy, β_y has two components and so for plotting the error map we presume that these components are equal (a similar assumption is made for β_ϕ).

zero-mean GP (left plot) expectedly fails to generalize to regions where there are no training samples. This issue is slightly alleviated in the case of the NN-mean GP (middle plot) at the expense of worsening the condition number, κ, of the covariance matrix. The increase in κ is due to the fact that the NN mean has sufficient parameters to interpolate the data which causes the kernel to model very small residuals and, in turn, be numerically ill-conditioned. The magnitude of κ is directly related to the number of parameters of the mean function which is typically a very large number in neural operators.

The results of the third scenario are provided in Fig. 6(c) where we observe that the physics of the problem is effectively used to learn $u(x)$ with high accuracy without increasing the condition number of the covariance matrix. The training mechanism corresponding to this scenario is similar to the one explained in Section 2.2 for operator learning.

2.4. Comparison to optimal recovery-based operator learning

The optimal recovery method of [13] approximates the operator $\mathcal{G}^\dagger$ via the following explicit representer formula:

$$\bar{\mathcal{G}}(u^*)(y) = c_v(y, \boldsymbol{Y}) c_v(\boldsymbol{Y}, \boldsymbol{Y})^{-1} \Lambda(\phi(u^*), \boldsymbol{U}) \Lambda(\boldsymbol{U}, \boldsymbol{U})^{-1} \boldsymbol{V} \tag{38}$$

where $\Lambda(\boldsymbol{U}, \boldsymbol{U}')$ is a matrix-valued kernel in the space of $m \times m$ matrices, $c_v(\cdot, \cdot)$ is the kernel associated with the RKHS of functions in $\mathcal{V}$, and y denotes a location in the output space where observations are made. [13] assumes a diagonal kernel of the form $\Lambda(\boldsymbol{U}, \boldsymbol{U}') = c_\lambda(\boldsymbol{U}, \boldsymbol{U}') I$ where I is the $m \times m$ identity matrix and $c_\lambda : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a real-valued kernel. Following these assumptions and using the Kronecker product, we can rewrite Eq. (38) as:

$$\bar{\mathcal{G}}(u^*)(y) = c_v(y, \boldsymbol{Y}) c_v(\boldsymbol{Y}, \boldsymbol{Y})^{-1} \left[c_\lambda \left(\phi(u^*), \boldsymbol{U} \right) \otimes I \right] \left[c_\lambda(\boldsymbol{U}, \boldsymbol{U}) \otimes I \right]^{-1} \boldsymbol{V}. \tag{39}$$

To directly compare Eq. (39) with our approach, we drop the mean function in our GP predictor to obtain:

$$\mathbb{E}[\widetilde{\mathcal{G}}_p^\dagger(\phi(u^*), \delta_{y^*}) | D] = \left[c_\phi(\phi(u^*), \boldsymbol{U}) \otimes c_y(\boldsymbol{Y}^*, \boldsymbol{Y}) \right] \left[c_\phi(\boldsymbol{U}, \boldsymbol{U}) \otimes c_y(\boldsymbol{Y}, \boldsymbol{Y}) \right]^{-1} \boldsymbol{V}. \tag{40}$$

Comparing Eqs. (39) and (40) highlights the main differences between the two approaches:

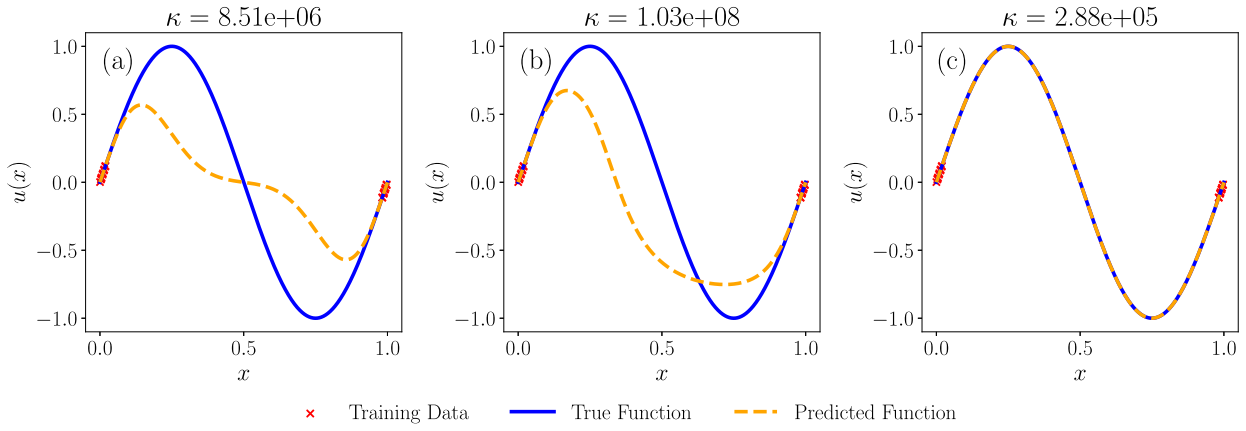


Fig. 6. Parameter optimization in GPs: We consider three scenarios with a zero-mean GP in (a) and NN-mean GPs in (b) and (c). The differences in training settings across these three cases are detailed in the text. The relative $L2$ errors for cases (a), (b) and (c) are 64.77%, 60.38%, and 0.09%, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- Unlike the optimal recovery method of [13], we account for the correlations in the output function space through $c_y(\cdot,\cdot)$. Hence, we solve a single regression problem as opposed to solving m independent optimal recovery ones at the expense of (1) using all the data at once, and (2) having far fewer hyperparameters (and hence flexibility).
- Eq. (39) directly approximates $\mathcal{G}^\dagger$ which explains the presence of the operator-valued kernels. Instead, our method approximates $\mathcal{G}^\dagger(u^*)(y^*)$, i.e., the *evaluation* of the output function at the *arbitrary* query point y^* for the given input function u^*, see Fig. 1. This feature enables us to incorporate additional physical constraints into the approximation process as detailed in Section 2.2.

We also note that our approach can also use deep NNs as the mean function which has the advantage of reducing the sole reliance of the predictor on the kernel whose misspecification can substantially reduce the performance of zero-mean GPs.

2.5. Inference complexity

We quantify the evaluation costs of our predictor by computing the floating point operators (FLOPs) per a test sample [71]. We assume that the product between two matrices $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times q}$ has a cost of $nq(2m-1)$ flops, vectorization has no costs, and that a nonlinear function acting upon an input vector of size n (e.g., activation function in NNs or kernel evaluation in GPs) costs n flops. Additionally, we assume the cost of constructing the $m\times n$ covariance matrix between two matrices $\boldsymbol{A} \in \mathbb{R}^{m\times q}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times q}$ using the Gaussian kernel in Eq. (7a) is $mn(4q+2)$.

We observe in Eq. (25) that the $m\times N$ matrix $\boldsymbol{C}_y^{-1}(\boldsymbol{V}-\boldsymbol{M})\boldsymbol{C}_\phi^{-1}$ can be stored from training and therefore we do not attribute any associated cost to that operation at the inference stage. Hence, the cost of evaluating the learned operator in our framework consists of (1) evaluation cost of the mean function which is either zero for a zero-mean GP, or C_m for a non-zero mean GP, (2) building $c_y(\boldsymbol{Y}^*,\boldsymbol{Y};\widehat{\beta}_y) \in \mathbb{R}^{m\times m}$ and $c_\phi(\phi(u^*),\phi(\boldsymbol{U});\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2) \in \mathbb{R}^{N\times 1}$, which have costs of $m^2(4d+2)$ and $N(4n+2)$, respectively, and (3) matrix multiplication costs in $\mathrm{vec}\Big(c_y(\boldsymbol{Y}^*,\boldsymbol{Y};\widehat{\beta}_y)\boldsymbol{C}_y^{-1}(\boldsymbol{V}^s-\boldsymbol{M}^s)\boldsymbol{C}_\phi^{-1}c_\phi(\phi(u^*),\boldsymbol{U};\widehat{\beta}_\phi,\widehat{\sigma}_\phi^2)\Big)$ which sum up to $m(2N-1)+m(2\ m-1)$. Given these cost components, the total inference cost of predicting the output function at m points for a given test input function discretized at n points is:

$$\begin{aligned} C &= C_m + m^2(4d+2) + N(4n+2) + m(2N-1) + m(2\ m-1) \\ &= C_m + 4m^2(d+1) - 2\ m + 2N(2n+2\ m+1), \end{aligned} \tag{41}$$

where N is the number of training samples and C_m can be calculated as detailed in [71] if neural operators such as DeepONet or FNO are used as the mean function in our approach.

Eq. (41) indicates the price that one has to pay for embedding a neural operator in our framework. This additional cost scales linearly in N but incurring it, as argued in Section 3 based on our comparative studies, is justified since GPs with neural operators as their mean function provide higher accuracy than the base neural operators especially in extrapolation.

3. Numerical studies and discussions

We evaluate the performance of our approach in data-driven and physics-informed operator learning problems in Sections 3.1 to 3.3 where we consider both single- and multi-output operators. While we can leverage any neural operator as the mean function

in our approach, herein we only consider FNO [11] and DeepONet [10] which are two of the most popular neural operators. Throughout, we use the relative $L2$ metric defined below to quantify the accuracy on test data:

$$L2 = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{\left\| \mathcal{G}^\dagger(u^i) - \widetilde{\mathcal{G}}_p^\dagger(u^i) \right\|_{\mathcal{Y}}}{\left\| \mathcal{G}^\dagger(u^i) \right\|_{\mathcal{Y}}} \tag{42}$$

where N_{test} is the number of test samples, $\mathcal{G}^\dagger$ is the target operator and $\widetilde{\mathcal{G}}_p^\dagger$ is the predicted operator. We conclude our studies in Section 3.4 by commenting on the training cost of different approaches.

3.1. Data-driven operator learning: Single-output operators

We consider four canonical examples that are routinely studied in the relevant literature [11,13,71,72]. These examples are defined below and used to compare our approach to DeepONet [10], FNO [11], and GP-OR which is the GP-based optimal recovery approach of [13]. We take the training data from [72] and use the reported accuracy metrics in [13] for these three baselines. In our approach, we consider three cases: a zero-mean GP, DeepONet-mean GP, and FNO-mean GP. In the latter two cases, we set up the architecture of the neural operators to match those of the baselines.[2] We also train our models with the same data used for training the baselines. We train our GP-based models via Adam for 3000 training epochs and set the learning rate to 10^{-3} and 10^{-2} for NN-mean and zero-mean cases, respectively.

Burgers' Equation: We consider the $1D$ Burgers' equation with periodic BCs:

$$\begin{aligned} &u_t + uu_x = \zeta u_{xx}, \quad && \forall (x,t) \in (0,1) \times (0,1], \\ &u(x,0) = u_0(x), && \forall x \in (0,1) \end{aligned} \tag{43}$$

with $\zeta = 0.1$. The operator that we aim to learn is $\mathcal{G}^\dagger : u(x,0) \to u(x,1)$, i.e., the mapping from the IC to the solution at time $t = 1$. The dataset is originally generated in [11] where the IC is sampled from a Gaussian random field with a Riesz kernel. As in [72], we use 1000 samples for training and 200 for testing with a spatial resolution of 128 points, i.e., $p = q = 128$.

Darcy Flow: The $2D$ Darcy flow problem with zero BCs is formulated as:

$$\begin{aligned} &-\nabla \cdot (a(x,y) \nabla u(x,y)) = f(x,y), \quad && \forall (x,y) \times (0,1)^2 \\ &u(x,y) = 0, && \forall (x,y) \in \partial (0,1)^2 \end{aligned} \tag{44}$$

where $a(x,y)$ is the permeability field and $f(x,y)$ is a source term. For a fixed $f(x,y)$, we aim to learn $\mathcal{G}^\dagger : a(x,y) \to u(x,y)$, i.e., the mapping from the permeability field to the solution in the entire domain. The dataset is extracted from [72] and we use 1000 samples for training and 200 for testing where the input and output functions are available on a 29×29 grid. The coefficient field $a(x,y)$ is obtained via $a = \xi(\mu)$ where $\mu \sim \mathcal{GP}(0, (-\Delta + 9I)^{-2})$ is a Gaussian random field with zero Neumann BCs on the Laplacian and ξ is a function that maps positive values to 12 and negative values to 3 [72].

Advection Equation: We consider the $1D$ advection equation with periodic BCs:

$$\begin{aligned} &u_t + u_x = 0, \quad && \forall (x,t) \in (0,1) \times (0,1], \\ &u(x,0) = u_0(x), && \forall x \in (0,1) \end{aligned} \tag{45}$$

where $\zeta = 0.1$ and our goal is to approximate $\mathcal{G}^\dagger : u(x,0) \to u(x,0.5)$, i.e., the mapping from the IC to the solution at time $t = 0.5$. The dataset is originally generated by [72] where the IC is designed as a square wave centered at $x = c$ with width w and height h, i.e., $u_0(x) = h\mathbf{1}_{\left\{c - \frac{w}{2}\right\}, \left\{c + \frac{w}{2}\right\}}$. The parameters c, w and h are randomly drawn from the uniform distribution $U([0.3, 0.7] \times [0.3, 0.6] \times [1, 2])$. We use 1000 samples for training and 200 for testing whose input and output functions are discretized via 40 equally-spaced grid points.

Structural Mechanics: We consider the equation governing the displacement field u of an elastic solid under small deformations:

$$\begin{aligned} &\nabla \cdot \sigma = 0, \quad && \text{in } D, \\ &u = \bar{u}, && \text{on } \Gamma_u, \\ &\nabla \cdot n = \bar{t}, && \text{on } \Gamma_t, \end{aligned} \tag{46}$$

where $D = (0,1)^2$ and the boundary ∂D is divided into $\Gamma_t = [0,1] \times 1$ and $\Gamma_u = \partial D \setminus \Gamma_t$, i.e., Γ_u is the compliment of Γ_t. We aim to learn the operator $\mathcal{G}^\dagger : \bar{t} \to \sigma$, i.e., the mapping between the $1D$ load $\bar{t}$ applied on Γ_t to the von Misses stress field σ. The dataset is extracted from [71] where $\bar{t}$ is drawn from a Gaussian random field $\mathcal{GP}(100, 400^2(-\nabla + 3^2 I)^{-1})$ where ∇ is the Laplacian with Neumann BCs applied to the space of functions with a spatial mean of zero. The output function σ is obtained via the FEM. We use 1250 samples for training and 20,000 for testing. The input function is sampled on 41 discretized points and the output function is interpolated on a regular 41×41 grid.

[2] In some cases, insufficient information about the architecture is available so we highlight the potential differences when applicable.

Table 1
Summary results on four benchmark problems: We report the $L2$ relative test error for four different operator learning methods. The errors in the first three rows are directly taken from the literature [13,71,72]. In our approach, we set the mean function to either (1) zero in which case we also provide the error metrics without training and after one optimization iteration, or (2) DeepONet and FNO whose architectures are exactly the same as those in rows 1 and 2. In the case of Darcy, we follow the references and apply PCA to the discretized input function and keep the first 200 PCs as the features for the kernel part of our GP-based methods.

		Burgers' $N = 1000, p = 128, q = 128$		Darcy $N = 1000, p = 29^2, q = 29^2$		Advection $N = 1000, p = 40, q = 40$		Structural mechanics $N = 1250, p = 41, q = 41^2$	
		Error	# of Parameters	Error	# of Parameters	Error	# of Parameters	Error	# of Parameters
DeepONet		2.15%	148,864	2.91%	715,776	0.22%	471,552	8.70%	137,856
FNO		1.93%	320,705	2.41%	6,304,257	0.66%	320,705	6.62%	888,001
GP-OR		2.15%	1280	2.75%	169,882	**2.75e−3%**	1600	6.95%	68,921
Our GP	0-shot Zero-mean	2.90%	**129**	5.14%	**201**	0.17%	**41**	7.13%	**42**
	1-shot Zero-mean	2.90%	**129**	5.01%	**201**	0.17%	**41**	7.13%	**42**
	Zero-mean	2.99%	**129**	2.89%	**201**	4.11e−3%	**41**	6.74%	**42**
	DeepONet-mean	1.84%	148,864	3.44%	715,776	0.18%	471,552	7.12%	137,856
	FNO-mean	**0.08%**	320,705	**2.19%**	6,304,257	0.23%	320,705	**6.49%**	888,001

The results of our comparative studies are summarized in Table 1 and indicate that our approach provides very competitive results. We observe that except for the Darcy example with DeepONet,[3] the performance of FNO and DeepONet consistently improves if they are used as the mean function of a GP. We attribute this desirable property to two features. First, due to the second term on the right-hand side of Eq. (28) our predictor naturally interpolates the data and hence (a) its mean function (i.e., the neural operator) can focus more on learning the sample-to-sample correlations that are *not* captured by the kernel, see the loss function in Eq. (15), and (b) the irregular part of the target operator (that is hard to approximate with a smooth kernel) can be captured by the neural operator. Second, our use of kernels provides a natural regularization scheme that improves generalizability. To see this, we analyze Eq. (9) within a regularization framework which is also used in contexts such as kernel ridge regression [73]. The second term in Eq. (9) measures the data fit while the first term is a regularizer whose magnitude can be calculated based on the eigenvalues of $\boldsymbol{C}$. Since we freeze the kernel parameters in our NN-mean GPs, these eigenvalues are essentially fixed which imposes a smoothness condition on the underlying function space which is controlled by the decay rate of the sequence of eigenvalues.

We also observe in Table 1 that our zero-mean GP is highly efficient in that it achieves small errors with orders of magnitude fewer hyperparameters. Due to the proper initialization of the kernel parameters, these GPs provide quite competitive results even prior to parameter optimization or just after one training epoch (referred to as $0-$shot and $1-$shot, respectively, in the table).

In Fig. 7 we visualize example test inputs, outputs, and predictions with associated error maps. In these plots we use the best version of our approach, i.e., a zero-mean GP in the case of Advection problem and an FNO-mean GP for the other three cases. We observe that the majority of the errors are concentrated at regions with sharp gradients which correspond to the jumps in the Advection example and the large traction regions at the top left corner in the Structural Mechanics problem. To further examine the performance of our framework in learning solution maps that exhibit large gradients, in Appendix A we study the 1D Burgers' equation with $\zeta = 0.01$ and show that our approach provides accurate results in such settings as well.

3.2. Data-driven operator learning: Multi-output operators

To benchmark our approach for learning multi-output operators we use the $2D$ lid-driven cavity (LDC) problem which is governed by the incompressible Navier–Stokes equations:

$$\begin{aligned}
&u_x + v_y = 0, && \forall (x,y) \in (0,1)^2 \\
&uu_x + vu_y = -\frac{1}{\rho} p_x + \zeta \left(u_{xx} + u_{yy}\right), && \forall (x,y) \in (0,1)^2 \\
&uv_x + vv_y = -\frac{1}{\rho} p_y + \zeta \left(v_{xx} + v_{yy}\right), && \forall (x,y) \in (0,1)^2 \\
&v(x,0) = v(x,1) = v(0,y) = v(1,y) = 0, && \forall x, y \in [0,1] \\
&u(x,0) = u(0,y) = u(1,y) = 0, && \forall x, y \in [0,1] \\
&u(x,1) = f(x), && \forall x \in [0,1] \\
&p(0,0) = 0
\end{aligned} \tag{47}$$

where $\zeta = 0.002$ is the kinematic viscosity, $\rho = 1.0$ denotes the density, and $f(x)$ is the lid velocity profile. To generate the data, we follow [74] and first solve Eq. (47) via the FEM for a variety of lid velocity profiles[4] and then sweep the flow field via a window of size 0.25×0.25 to collect samples, see Fig. 8. We divide the resulting data into two parts: if the window is above the horizontal dashed

[3] This may be due to the fact that we could not exactly replicate the training settings and model architecture since the corresponding Ref. [72] lacks sufficient details.

[4] The velocity profiles are taken from [74] who use GPs to generate them and solve Eq. (47) via OpenFOAM [75].

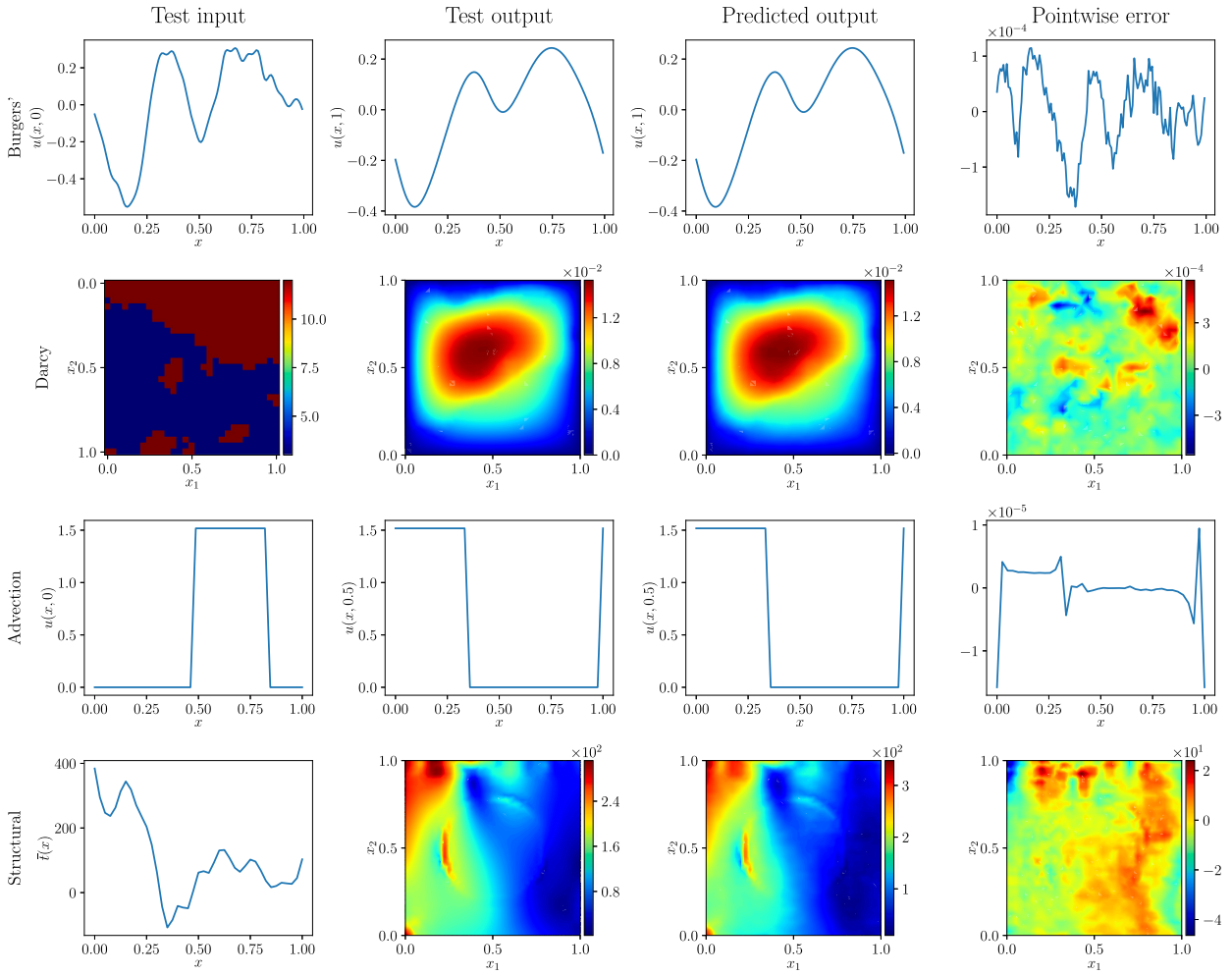
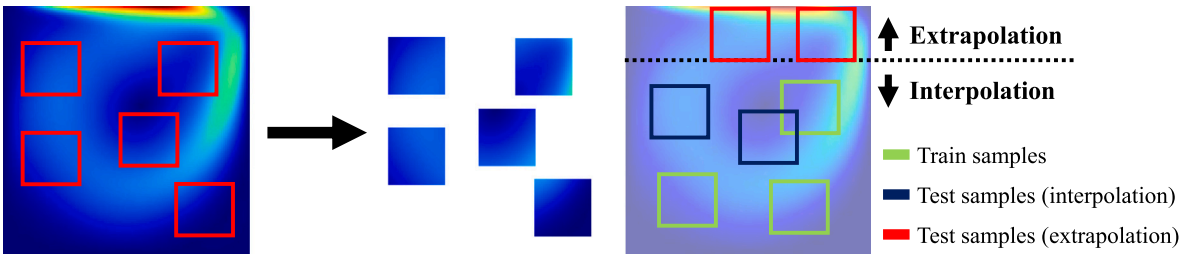


Fig. 7. Example test inputs, outputs, and predictions: We provide one example from each problem where the predicted outputs correspond to the best version of our approach in Table 1, i.e., a zero-mean GP in the case of Advection problem and an FNO-mean GP for the other three problems.



(a) Data collection. **(b)** Train vs test samples.

Fig. 8. Train and test data generation in the LDC problem: (a) The $[0,1]^2$ domain is swept with a smaller window to collect data. (b) If the center of the window is above the dashed line, the sample is used to assess the extrapolation error. Otherwise, the sample is used either for training or obtaining the interpolation error.

line in Fig. 8(b) the corresponding sample is used to assess the extrapolation test error. Otherwise, the sample is used for either training or obtaining the interpolation test error. The samples positioned on the top part of the domain provide good candidates for evaluating the extrapolation capabilities of the models since the solution fields inside them contain much larger velocity magnitudes and gradients than those in the bottom part of the domain.

Table 2
Summary of results for Section 3.2: We use DeepONet and FNO with two architectures that differ significantly in their parameter counts, referred to as *small* and *large*, for comparison in both interpolation and extrapolation tests. In our approach, we either employ these neural operators as the mean function or utilize a zero-mean GP. For the former case, we train the GPs using maximum likelihood estimation (MLE) and mean squared error (MSE) to assess the impact of the loss function on model performance. For the zero-mean GP, we also report errors before parameter optimization (denoted as 0-shot) and after one training epoch (denoted as 1-shot). All errors are reported for the velocity magnitude.

		Number of parameters	Interpolation error		Extrapolation error	
			Training via MLE	Training via MSE	Training via MLE	Training via MSE
DeepONet (small)		13,536	N/A	4.46%	N/A	42.60%
DeepONet (large)		235,800	N/A	3.49%	N/A	35.14%
FNO (small)		6235	N/A	7.80%	N/A	62.66%
FNO (large)		223,155	N/A	1.76%	N/A	44.20%
Ours	0-shot Zero-mean	**361**	7.10e−3%	N/A	24.48%	N/A
	1-shot Zero-mean	**361**	7.10e−3%	N/A	24.48%	N/A
	Zero-mean	**361**	**4.37e−4**%	N/A	41.00%	N/A
	DeepONet-mean (small)	13,536	5.02e−3%	1.43e−2%	31.13%	34.25%
	DeepONet-mean (large)	235,800	2.73e−3%	**7.12e−3**%	**22.77%**	**23.00%**
	FNO-mean (small)	6235	3.76e−3%	9.53e−3%	26.86%	34.41%
	FNO-mean (large)	223,155	1.48e−3%	7.31e−3%	39.15%	26.70%

The above procedure generates a total of 961 samples with observations on a 30×30 regular grid in the 0.25×0.25 subdomain. The operator learning problems involves mapping the arbitrary set of subdomain BCs (pressure and velocity components) to the solution inside the 0.25×0.25 subdomain, i.e., $\mathcal{G}^\dagger : [u_{\text{BC}}, v_{\text{BC}}, p_{\text{BC}}] \rightarrow [u, v, p]$. We use velocity magnitude to calculate the interpolation/extrapolation errors and train all models via Adam with 2000 epochs and learning rates of 10^{-2} and 10^{-3} for the 0-mean and NN-mean GPs, respectively. We use two different architectures for all models with DeepONet and FNO that differ significantly in their parameter counts (denoted as *small* or *large*). Specifically, DeepONet-based models either have a 4− layer branch and trunk nets (each layer with 150 neurons) or a 4− layer branch and trunk nets (each layer with 24 neurons). FNO-based models either have 4 layers with a width of 4 and 8 modes, or 4 layers with a width of 24 and 8 modes. We note that we intentionally did not conduct an exhaustive search to optimize the architecture of these models because we aim to determine whether even sub-optimal architectures can benefit from being integrated within our GP framework. We use 500 samples for training while 151 and 31 samples are used to obtain the interpolation and extrapolation errors, respectively.

The results of our studies are summarized in Table 2 and indicate that our GP-based approach consistently outperforms FNO and DeepONet in this example. The best results in interpolation are achieved with the zero-mean GP, whereas the DeepONet-mean GP trained using MLE yields the lowest errors in extrapolation. Switching to MSE from MLE reduces the accuracy in both extrapolation and interpolation regardless of whether FNO or DeepONet is used as the mean function in our approach since MSE does not take the sample-to-sample correlations into account while MLE does, see Eq. (24). Once again, we observe that the performance of the neural operator significantly improves when it is incorporated as the mean function of our GP framework, regardless of its architecture.

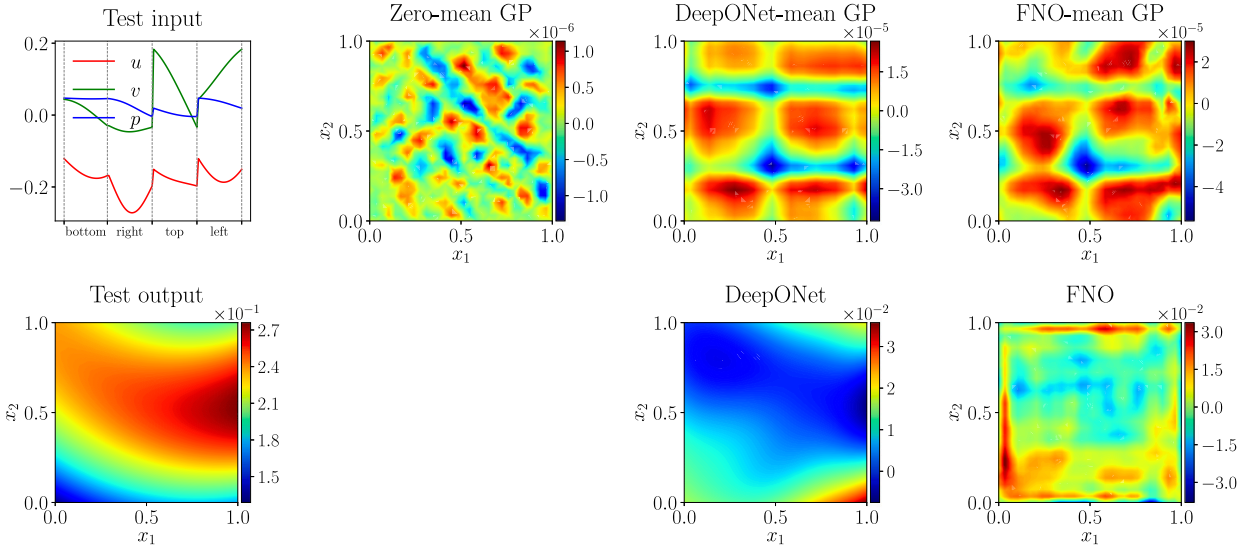
To visually compare the accuracy of different methods in extrapolation and interpolation, in Fig. 9 we provide the error maps corresponding to the velocity magnitude for two test cases. We observe in Fig. 9(a) that all methods performs quite well but our GP-based approach achieves two orders of magnitude smaller errors compared to FNO and DeepONet. However, the performance of all methods expectedly decreases in Fig. 9(b) as the imposed BCs are substantially different than the ones that the models are trained on.

3.3. Physics-informed operator learning

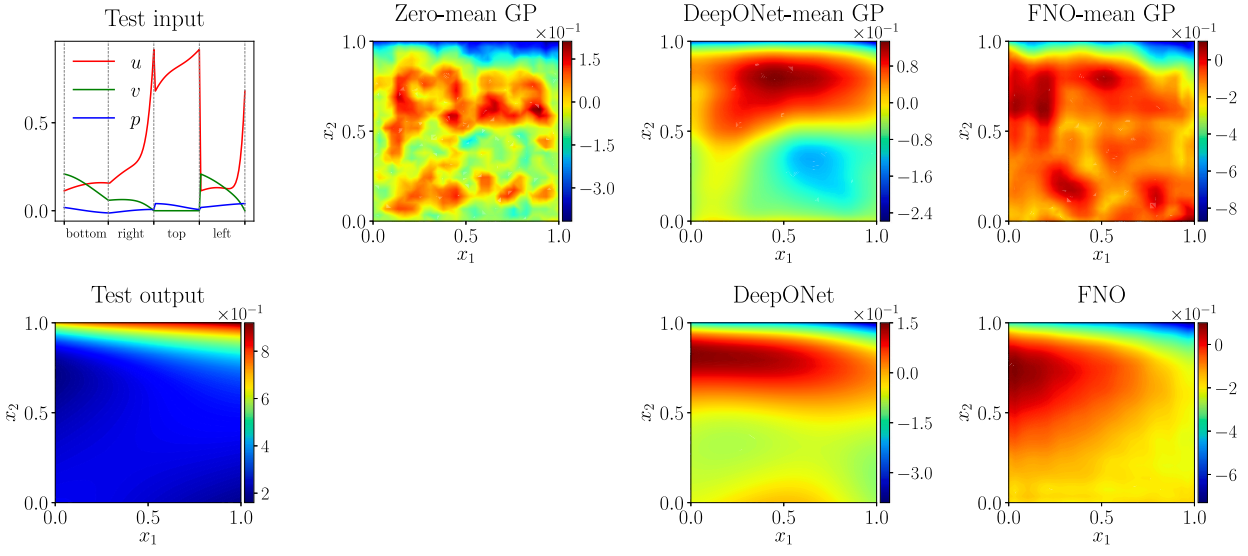
We now consider the effect of augmenting data with physics in two examples, the Burger's equation introduced in Section 1.2 and the LDC problem in Section 2.1.2. In the former example, we discretize the input function via 100 points and record the output at a very coarse grid of size 12×12 and use $N_{pi} = 50, n_{PDE} = 100^2, n_{BC} = 100$, and, $n_{IC} = 100$. In the LDC problem, each variable is discretized via 120 points on the boundaries (30 points on each edge of the square subdomain in Fig. 8(a)) and the PDE solution is recorded at a fine mesh of size 30×30, and use $N_{pi} = 100, n_{PDE} = 30^2, n_{BC} = 30$.

We solve these two problems via FNO, DeepONet, and their physics-informed versions denoted by PI-DeepONet [30] and PINO [76]. For our approach, we consider both zero-mean and NN-mean GPs where the latter is trained either with data or with both data and physics. In our NN-mean GP we only use DeepONet as we were unable to successfully integrate our code with PINO. All the models are trained via Adam with a learning rate of 10^{-3} which, similar to the previous sections, is increased to 10^{-2} for a zero-mean GP. We use 1000 training epochs in all cases except for the physics-informed models of the LDC problem where the number of epochs is reduced to 500 to decrease the computational costs. Since the density of the observations is very low in the Burgers' equation, we optimize β_y along with β_ϕ and σ^2 via MLE.

The results of our studies are summarized in Fig. 10 where the errors on unseen data are reported for each model as N increases. For the Burgers' equation where the observations are sparse, augmenting the training data with the physics substantially helps in the case of GP and DeepONet but not FNO (compare the dashed and solid lines with the same color). This difference is much smaller in the case of LDC where adding physics slightly improves GP and DeepONet when N is very small. We also notice that even in the



(a) Interpolation.



(b) Extrapolation.

Fig. 9. Example input, output, and error maps on the velocity magnitudes in the LDC problem: The test samples in extrapolation look substantially different than those in interpolation (compare the scales of the velocity magnitude profiles in the two test outputs). Our GP-based approach provides smaller errors in both scenarios especially in the interpolation case.

Burgers' problem where the output function is recorded at a coarse resolution, our purely data-driven models, i.e., without physics, achieve lower errors than the vanilla implementations of DeepONet and FNO. In the case of the zero-mean GP for the Burgers' problem, we also optimized β_y to relax the assumption on having dense observations, as opposed to other problems in Section 3.1. More details on this choice can be found in Appendix B. In the DeepONet-mean case, we still fix β_y as suggested in Section 2.3 and leverage the neural operator to learn the irregular part of the target operator.

We notice as N increases that the errors associated with the data-only models in the left panel of Fig. 10 slightly fluctuate while those in the right panel either decrease (in our approach) or negligibly change (FNO and DeepONet). The former trend is due to the fact that we observe the PDE solution at a very coarse grid and that the architecture and training mechanisms are not altered as N increases. We attribute the latter trend to FNO and DeepONet's inability to effectively use data. However, once the exact same DeepONet is used as the mean function of a GP, the prediction accuracy dramatically improves (compare the solid green and blue lines in the right panel). See Section 3.1 for our reasoning behind this trend.

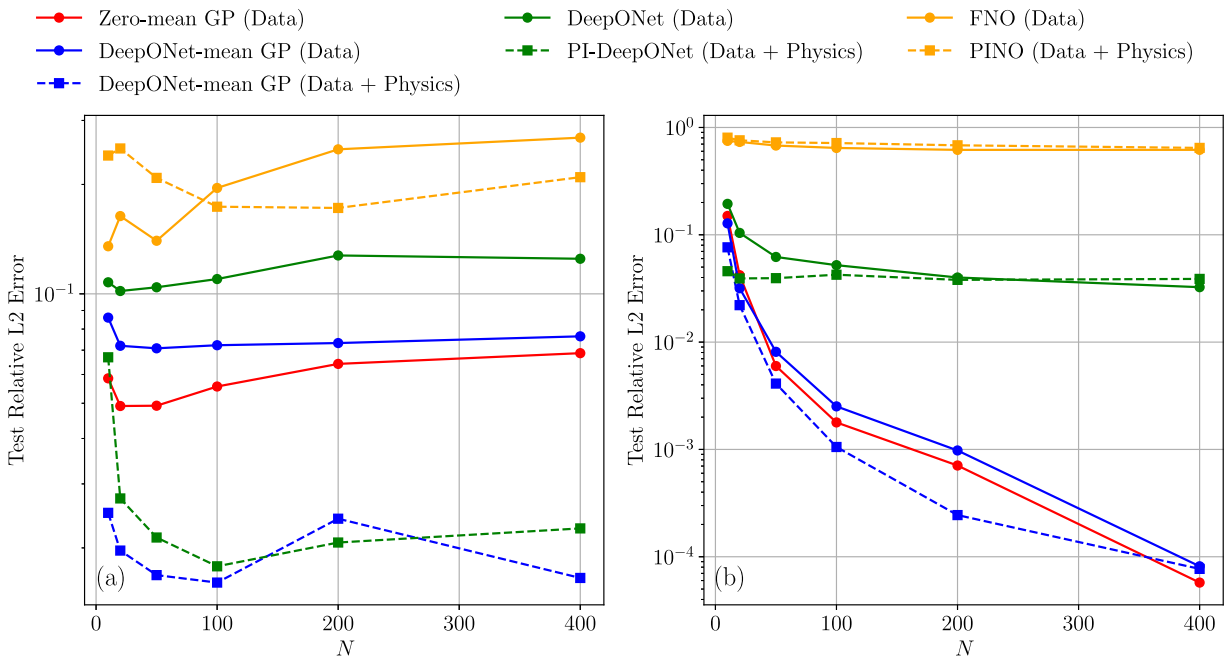


Fig. 10. Convergence studies: We study the effect of data set size N and augmenting the data with physics on the errors for various models in the case of Burgers' (left) and LDC (right) problems. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Training time in seconds per epoch: We report the costs for DeepONet, FNO, and our method with three different mean functions. The training time of each approach varies across the different problems based on the dimensionality of the input/output functions and their respective resolutions. For example, the zero-mean GP appears to be the slowest in problems where the input and output functions are one-dimensional, such as in the Burgers' or Advection equations, but becomes the fastest in the Structural Mechanics problem, where the input and output functions are one- and two-dimensional, respectively. Incorporating neural operators within our GP framework increases the training time per epoch with respect to their vanilla implementations due to the additional computations involved in the loss function (see Eq. (15) versus MSE). However, this results in higher overall accuracy, as shown in Tables 1 and 2.

		Burgers'	Darcy	Advection	Structural mechanics	Lid-driven cavity
DeepONet		0.04	0.17	0.03	12.7	0.22
FNO		0.06	0.53	0.03	3.53	0.03
Ours	Zero-mean	0.12	0.24	0.12	1.74	0.20
	DeepONet-mean	0.06	0.25	0.05	13.4	0.27
	FNO-mean	0.07	1.17	0.05	3.66	0.04

3.4. Training cost

We compare the training costs of our approach to those of FNO and DeepONet in Table 3 where the costs are reported for one epoch of parameter optimization. We observe that the training cost of each approach varies depending on the dimensionality and resolution of the input/output functions in each problem. For instance, the zero-mean GP appears to have the highest training time per epoch for the Burgers' and Advection problems where the input and output functions are one-dimensional. However, it has the lowest training cost for the Structural Mechanics problem where the input and output functions are one- and two-dimensional, respectively. Incorporating neural operators within our GP framework increases the training time per epoch with respect to their baseline implementations, due to the additional computations involved in the loss function (compare Eq. (15) to MSE). We believe this additional cost is justified since the non-zero mean GPs achieve higher accuracy compared to the standalone neural operators, as shown in Tables 1 and 2.

4. Conclusions

We introduce a framework based on GPs for approximating single- or multi-output operators in either a purely data-driven context, or by using both data and physics. In the former case, we use MLE for parameter optimization and set the mean function

of the GP to be either zero or a neural operator. In the physics-informed case, we employ GPs with neural operator mean functions whose parameters are estimated via a weighted combination of MLE and PDE residuals.

We emphasize that our framework is *general*, but we have devised several strategies to (1) ensure scalability to large data and high-dimensional feature spaces, and (2) minimize computational costs and numerical issues associated with inverting the GP's covariance matrix. While these strategies have proven effective for the problems tested in this paper, we recommend that practitioners explore alternative initialization settings, using our insights and guidelines as a basis for developing strong baseline models. That is, they may choose to use the framework in its generic form, i.e., Eqs. (9) and (13), or develop tailored mechanisms to train the model for their specific problem.

Our suggested training strategies rely on the assumption that both input and output functions are richly sampled at the same locations across samples. This allows us to (1) leverage the data structure together with the separability assumption on the kernel to formulate the GP's covariance matrix using the Kronecker product, and (2) initialize the kernel parameters such that they require little to no tuning. In data-driven applications, these strategies enable us to build zero-mean GPs that provide high approximation accuracy in both single- and multi-output cases while having orders of magnitude fewer parameters than competing methods such as neural operators. In fact, we demonstrate that our zero-mean GPs provide competitive results even without training, i.e., we build the first zero-shot mechanism for operator learning. With NN-mean GPs, either in data-driven or physics-informed applications, our strategies eliminate the need for optimizing the kernel hyperparameters and result in a model that augments neural operators with the strengths of kernels. Using FNO and DeepONet as examples, we show that such an integrated model improves the performance of neural operators.

Dense observations may not always be available, especially when data is collected from physical experiments rather than simulations. Our GP framework is flexible enough to accommodate these situations and relax the assumptions we have made on having dense data. For instance, if a specific problem does not allow for a dense set of observations of the output function, one can choose to optimize β_y with a zero-mean GP to leverage correlations in the support of the output function. Alternatively, in the case of a neural-operator-mean GP, one can fix β_y and leverage the mean function to learn such correlations. These strategies have been successfully applied in Section 3.3 to problems where the output function is observed at a coarse resolution.

In all the examples in Section 3 we use the simple kernels in Eq. (7) that have at most a few hundred parameters and, hence, limited learning capacity. We studied various kernels including deep ones [77] but did not observe consistent and significant performance improvements. We believe a promising future direction is designing special kernels whose parameters can be efficiently estimated via cross-validation which is more robust to model misspecification compared to MLE. Another interesting direction is to extend the proposed GP framework in order to relax the assumption that input and output functions are sampled at the same locations across samples. We believe this could potentially be achieved by leveraging sparse GPs [78–83] which were originally developed to increase the scalability of GPs for large datasets.

CRediT authorship contribution statement

Carlos Mora: Writing – review & editing, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Amin Yousefpour:** Investigation. **Shirin Hosseinmardi:** Investigation. **Houman Owhadi:** Writing – review & editing, Funding acquisition, Formal analysis. **Ramin Bostanabad:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

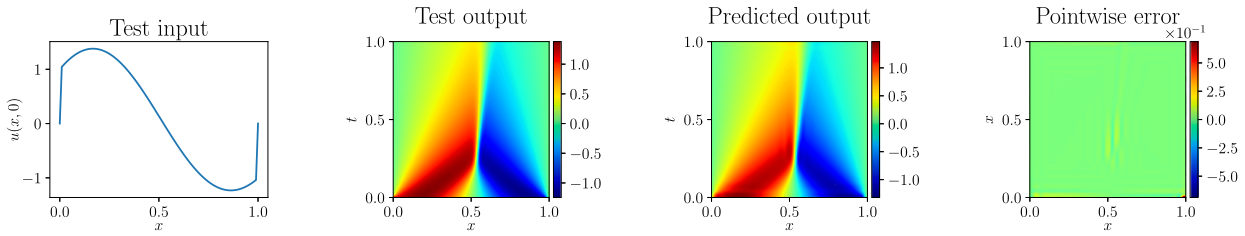
The authors appreciate the support from the Office of Naval Research, United States (grant number $N000142312485$) and the National Science Foundation, United States (grant number 2238038). HO acknowledges support from a Department of Defense Vannevar Bush Faculty Fellowship.

Appendix A. 1D Burgers' equation with high-gradients

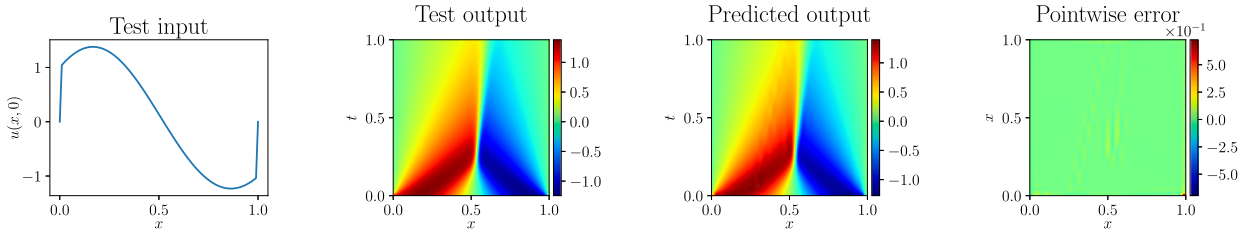
To evaluate the performance of the proposed method when the solution maps exhibit large gradients, we consider the 1D Burgers' problem with zero Dirichlet BCs and $\zeta = 0.01$:

$$\begin{aligned} &v_t + vv_x - \zeta v_{xx} = 0, && \forall x \in (0,1), && t \in (0,1] \\ &v = 0, && \forall x = \{0,1\}, && t \in [0,1] \\ &v = u, && \forall x \in [0,1], && t = 0. \end{aligned} \tag{A.1}$$

We use $N = 800$ samples and set $p = 100$ and $q = 34 \times 34$. We employ both zero- and DeepONet-mean GPs in our framework which achieve $L2$ relative test errors of 2.84% and 3.03%, respectively. Fig. 11 shows the results obtained for a random test sample. As illustrated, both the zero- and DeepONet-mean GP are able to accurately learn the sharp gradients in the solution map.



(a) Our GP framework with zero-mean is able to accurately capture the large gradients in the solution.



(b) Our GP framework with DeepONet-mean is able to accurately capture the large gradients in the solution.

Fig. 11. Example test input, output, predictions and pointwise error: We provide the results obtained with our zero-mean and DeepONet-mean GP for one test example for the 1D Burgers' equation with zero Dirichlet BCs and $\zeta = 0.01$. The $L2$ test relative error across 200 test samples for each model is 2.84% and 3.03% respectively.

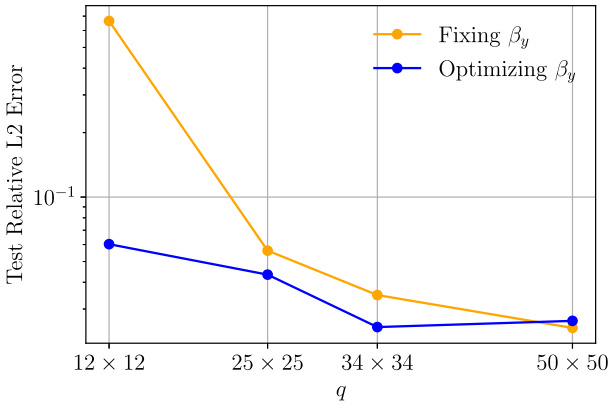


Fig. 12. Optimizing versus fixing β_y in training with our zero-mean GP for the Burgers' equation with Dirichlet BCs: Fixing β_y achieves comparable or slightly better performance as the output function is observed at finer resolutions. Optimizing β_y is the better choice when the output function is not observed at a fine enough resolution.

Appendix B. Fixing versus optimizing β_y

In Section 2.3.1, we propose a training strategy that involves fixing β_y to a large value, e.g., 10^3, based on the assumption that output functions are typically richly sampled for operator learning purposes. This approach enables the GP to effectively capture correlations in Ω_v without any training, while ensuring the numerical stability of $\boldsymbol{C}_y$ and eliminating the need to invert it in every optimization iteration. However, dense observations may not always be available, especially when data is collected from physical experiments rather than simulations. Fortunately, our GP framework is flexible enough to accommodate these situations by optimizing β_y instead of fixing it.

To demonstrate the effect of fixing β_y versus optimizing it, we conduct an experiment on the Burgers' equation with Dirichlet BCs, i.e., Eq. (5). We report the test relative L2 error of our zero-mean GP as a function of the output function resolution q under two scenarios: (1) fixing β_y at 10^3 and following the proposed strategy, and (2) optimizing β_y alongside the other parameters. For this study, we set $N = 800$ and $p = 100$. The results of these studies are summarized in Fig. 12. We observe that fixing β_y achieves comparable or even better performance as the output function resolution increases. In contrast, when the output function is sampled at coarser resolutions, optimizing β_y yields better results since it allows the GP to more accurately capture correlations within Ω_v.

Therefore, practitioners always have the option of optimizing β_y if they suspect the observation density is low. Additionally, given the low computational cost of our approach, one can experiment with both strategies and choose the one that performs best.

Data availability

All data and code are available at https://github.com/Bostanabad-Research-Group/GP-for-Operator-Learning.

References

[1] Roger G. Ghanem, Pol D. Spanos, Stochastic Finite Elements: A Spectral Approach, Courier Corporation, 2003.
[2] James O. Ramsay, Bernard W. Silverman, Fitting differential equations to functional data: Principal differential analysis, Funct. Data Anal. (2005) 327–348.
[3] James O. Ramsay, Bernard W. Silverman, Applied Functional Data Analysis: Methods and Case Studies, Springer, 2002.
[4] David J. Lucia, Philip S. Beran, Walter A. Silva, Reduced-order modeling: new approaches for computational physics, Prog. Aerosp. Sci. 40 (1–2) (2004) 51–117.
[5] Vikramjit Mitra, Yucel Ozbek, Hosung Nam, Xinhui Zhou, Carol Y Espy-Wilson, From acoustics to vocal tract time functions, in: 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2009, pp. 4497–4500.
[6] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, Juan J Alonso, SU2: An open-source suite for multiphysics simulation and design, Aiaa J. 54 (3) (2016) 828–846.
[7] Nikola Kovachki, Burigede Liu, Xingsheng Sun, Hao Zhou, Kaushik Bhattacharya, Michael Ortiz, Andrew Stuart, Multiscale modeling of materials: Computing, data science, uncertainty and goal-oriented optimization, Mech. Mater. 165 (2022) 104156.
[8] Gabriele Boncoraglio, Charbel Farhat, Active manifold and model-order reduction to accelerate multidisciplinary analysis and optimization, AIAA J. 59 (11) (2021) 4739–4753.
[9] Bogdan Raonic, Roberto Molinaro, Tobias Rohner, Siddhartha Mishra, Emmanuel de Bezenac, Convolutional neural operators, in: ICLR 2023 Workshop on Physics for Machine Learning, 2023.
[10] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, George Em Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[11] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Fourier neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2010.08895.
[12] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, Jun Zhu, Gnot: A general neural operator transformer for operator learning, in: International Conference on Machine Learning, PMLR, 2023, pp. 12556–12569.
[13] Pau Batlle, Matthieu Darcy, Bamdad Hosseini, Houman Owhadi, Kernel methods are competitive for operator learning, J. Comput. Phys. 496 (2024) 112549.
[14] Nicholas H. Nelsen, Andrew M. Stuart, Operator learning using random features: A tool for scientific computing, SIAM Rev. 66 (3) (2024) 535–571.
[15] Dongbin Xiu, Numerical Methods for Stochastic Computations: A Spectral Method Approach, Princeton University Press, 2010.
[16] Dongbin Xiu, George Em Karniadakis, The Wiener–Askey polynomial chaos for stochastic differential equations, SIAM J. Sci. Comput. 24 (2) (2002) 619–644.
[17] Olivier Le Maître, Omar M. Knio, Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics, Springer Science & Business Media, 2010.
[18] Dongbin Xiu, Jie Shen, Efficient stochastic Galerkin methods for random diffusion equations, J. Comput. Phys. 228 (2) (2009) 266–281.
[19] 2003.
[20] P.D. Spanos, Roger Ghanem, Stochastic finite element expansion for random media, J. Eng. Mech. 115 (5) (1989) 1035–1053.
[21] Houman Owhadi, Clint Scovel, Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design, vol. 35, Cambridge University Press, 2019.
[22] Houman Owhadi, Bayesian numerical homogenization, Multiscale Model. Simul. 13 (3) (2015) 812–828.
[23] Florian Schäfer, Houman Owhadi, Sparse recovery of elliptic solvers from matrix-vector products, SIAM J. Sci. Comput. 46 (2) (2024) A998–A1025.
[24] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, arXiv preprint arXiv:2003.03485.
[25] Nikola B. Kovachki, Samuel Lanthaler, Andrew M. Stuart, Operator learning: Algorithms and analysis, 2024, arXiv preprint arXiv:2402.15715.
[26] Nicolas Boullé, Alex Townsend, A mathematical guide to operator learning, 2023, arXiv preprint arXiv:2312.14688.
[27] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Neural operator: Learning maps between function spaces with applications to pdes, J. Mach. Learn. Res. 24 (89) (2023) 1–97.
[28] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917.
[29] Lu Lu, Pengzhan Jin, George Em Karniadakis, Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, 2019, arXiv preprint arXiv:1910.03193.
[30] Sifan Wang, Hanwen Wang, Paris Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. 7 (40) (2021) eabi8605.
[31] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Multipole graph neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2006.09535.
[32] Tapas Tripura, Souvik Chakraborty, Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems, Comput. Methods Appl. Mech. Engrg. 404 (2023) 115783.
[33] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, Anima Anandkumar, Physics-informed neural operator for learning partial differential equations, ACM/JMS J. Data Sci. (2021).
[34] Louis Serrano, Lise Le Boudec, Armand Kassaï Koupaï, Thomas X Wang, Yuan Yin, Jean-Noël Vittaut, Patrick Gallinari, Operator learning with neural fields: Tackling PDEs on general geometries, Adv. Neural Inf. Process. Syst. 36 (2024).
[35] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, Jun Zhu, Gnot: A general neural operator transformer for operator learning, in: International Conference on Machine Learning, PMLR, pp. 12556–12569.
[36] Wolfgang Hackbusch, Multi-Grid Methods and Applications, vol. 4, Springer Science & Business Media, 2013.
[37] Juncai He, Jinchao Xu, Mgnet: A unified framework of multigrid and convolutional neural network, Sci. China Math. 62 (2019) 1331–1354.
[38] Juncai He, Xinliang Liu, Jinchao Xu, Mgno: Efficient parameterization of linear operators via multigrid, 2023, arXiv preprint arXiv:2310.19809.
[39] James O. Ramsay, CJ1125714 Dalzell, Some tools for functional data analysis, J. R. Stat. Soc. Ser. B Stat. Methodol. 53 (3) (1991) 539–561.
[40] Charles A. Micchelli, Massimiliano Pontil, On learning vector-valued functions, Neural Comput. 17 (1) (2005) 177–204.

[41] Leo Breiman, Jerome H. Friedman, Predicting multivariate responses in multiple linear regression, J. R. Stat. Soc. Ser. B Stat. Methodol. 59 (1) (1997) 3–54.
[42] Theodoros Evgeniou, Massimiliano Pontil, Regularized multi–task learning, in: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 109–117.
[43] Andreas Argyriou, Theodoros Evgeniou, Massimiliano Pontil, Convex multi-task feature learning, Mach. Learn. 73 (2008) 243–272.
[44] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al., Kernels for vector-valued functions: A review, Found. Trends® Mach. Learn. 4 (3) (2012) 195–266.
[45] Houman Owhadi, Do ideas have shape? Idea registration as the continuous limit of artificial neural networks, Physica D 444 (2023) 133592.
[46] Yifan Chen, Bamdad Hosseini, Houman Owhadi, Andrew M Stuart, Solving and learning nonlinear PDEs with Gaussian processes, J. Comput. Phys. 447 (2021) 110668.
[47] Houman Owhadi, Computational graph completion, Res. Math. Sci. 9 (2) (2022) 27.
[48] Matthew Lowery, John Turnage, Zachary Morrow, John D Jakeman, Akil Narayan, Shandian Zhe, Varun Shankar, Kernel Neural Operators (KNOs) for scalable, memory-efficient, geometrically-flexible operator learning, 2024, arXiv preprint arXiv:2407.00809.
[49] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind, Automatic differentiation in machine learning: a survey, J. Marchine Learn. Res. 18 (2018) 1–43.
[50] Kurt Cutajar, Mark Pullin, Andreas Damianou, Neil Lawrence, Javier González, Deep gaussian processes for multi-fidelity modeling, 2019, arXiv preprint arXiv:1903.07320.
[51] José Betancourt, François Bachoc, Thierry Klein, Déborah Idier, Rodrigo Pedreros, Jérémy Rohmer, Gaussian process metamodeling of functional-input code for coastal flood hazard assessment, Reliab. Eng. Syst. Saf. 198 (2020) 106870.
[52] Samuel Stanton, Wesley Maddox, Ian Delbridge, Andrew Gordon Wilson, Kernel interpolation for scalable online Gaussian processes, 2021.
[53] Jonathan Tammer Eweis-Labolle, Nicholas Oune, Ramin Bostanabad, Data fusion with latent map Gaussian processes, J. Mech. Des. 144 (9) (2022).
[54] Zahra Zanjani Foumani, Mehdi Shishehbor, Amin Yousefpour, Ramin Bostanabad, Multi-fidelity cost-aware Bayesian optimization, Comput. Methods Appl. Mech. Engrg. 407 (2023) 115937.
[55] Amin Yousefpour, Zahra Zanjani Foumani, Mehdi Shishehbor, Carlos Mora, Ramin Bostanabad, Gp+: A python library for kernel-based learning via Gaussian processes, 2023, arXiv preprint arXiv:2312.07694.
[56] Jiahao Zhang, Shiqi Zhang, Guang Lin, Pagp: A physics-assisted Gaussian process framework with active learning for forward and inverse problems of partial differential equations, 2022, arXiv preprint arXiv:2204.02583.
[57] Tomoharu Iwata, Zoubin Ghahramani, Improving output uncertainty estimation and generalization in deep learning via neural network Gaussian processes, 2017, arXiv preprint arXiv:1707.05922.
[58] Rui Meng, Xianjin Yang, Sparse Gaussian processes for solving nonlinear PDEs, J. Comput. Phys. 490 (2023) 112340.
[59] Xianjin Yang, Houman Owhadi, A mini-batch method for solving nonlinear PDEs with Gaussian processes, 2023, arXiv preprint arXiv:2306.00307.
[60] Yifan Chen, Houman Owhadi, Florian Schäfer, Sparse cholesky factorization for solving nonlinear PDEs via Gaussian processes, Math. Comp. (2024).
[61] Kang Wang, Lei Zhang, Shaoqiang Tang, Discovery of PDEs driven by data with sharp gradient or discontinuity, Comput. Math. Appl. 140 (2023) 33–43.
[62] Carlos Mora, Amin Yousefpour, Shirin Hosseinmardi, Ramin Bostanabad, Neural networks with kernel-weighted corrective residuals for solving partial differential equations, 2024, arXiv preprint arXiv:2401.03492.
[63] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, Andrew G Wilson, Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, Adv. Neural Inf. Process. Syst. 31 (2018).
[64] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[65] Edwin V. Bonilla, Kian Chai, Christopher Williams, Multi-task Gaussian process prediction, Adv. Neural Inf. Process. Syst. 20 (2007).
[66] Stefano Conti, Anthony O'Hagan, Bayesian emulation of complex multi-output and dynamic computer models, J. Statist. Plann. Inference 140 (3) (2010) 640–651, 531at Times Cited:195 Cited References Count:30.
[67] Mehdi Shishehbor, Shirin Hosseinmardi, Ramin Bostanabad, Parametric encoding with attention and convolution mitigate spectral bias of neural partial differential equation solvers, Struct. Multidiscip. Optim. 67 (7) (2024) 128.
[68] Sifan Wang, Yujun Teng, Paris Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.
[69] Peter I. Frazier, A tutorial on Bayesian optimization, 2018, arXiv preprint arXiv:1807.02811.
[70] R. Planas, N. Oune, R. Bostanabad, Evolutionary Gaussian processes, J. Mech. Des. 143 (11) (2021) 111703.
[71] Maarten V de Hoop, Daniel Zhengyu Huang, Elizabeth Qian, Andrew M Stuart, The cost-accuracy trade-off in operator learning with neural networks, 2022, arXiv preprint arXiv:2203.13181.
[72] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, George Em Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114778.
[73] Carl Edward Rasmussen, Gaussian processes for machine learning, 2006.
[74] Hengjie Wang, Robert Planas, Aparna Chandramowlishwaran, Ramin Bostanabad, Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains, Comput. Methods Appl. Mech. Engrg. 389 (2022) 114424.
[75] H. Jasak, Z. Tukovic, Automatic mesh motion for the unstructured finite volume method, Trans. FAMENA 30 (2) (2007) 1–18, Cited by: 2.
[76] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, Anima Anandkumar, Physics-informed neural operator for learning partial differential equations, ACM/JMS J. Data Sci. 1 (3) (2024) 1–27.
[77] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, Eric P Xing, Deep kernel learning, in: Artificial Intelligence and Statistics, PMLR, pp. 370–378.
[78] Christopher KI Williams, Carl Edward Rasmussen, A Scwaighofer, Volker Tresp, Observations on the nyström method for Gaussian process prediction, 2002.
[79] Matthias Seeger, Christopher Williams, Neil Lawrence, Fast forward selection to speed up sparse Gaussian process regression, in: Artificial Intelligence and Statistics 9.
[80] Edward Snelson, Zoubin Ghahramani, Sparse Gaussian processes using pseudo-inputs, in: Advances in Neural Information Processing Systems, pp. 1257–1264.
[81] 2009.
[82] Edward Snelson, Zoubin Ghahramani, Variable noise and dimensionality reduction for sparse Gaussian processes, 2012, arXiv preprint arXiv:1206.6873.
[83] James Hensman, Nicolo Fusi, Neil D. Lawrence, Gaussian processes for big data, 2013, arXiv preprint arXiv:1309.6835.