Computational Mechanics (2023) 72:451–470
https://doi.org/10.1007/s00466-023-02298-8

ORIGINAL PAPER

Semi-supervised invertible neural operators for Bayesian inverse problems

Sebastian Kaltenbach[1] · Paris Perdikaris[2] · Phaedon-Stelios Koutsourelakis[1,3]

Received: 18 September 2022 / Accepted: 19 February 2023 / Published online: 30 March 2023

Abstract
Neural Operators offer a powerful, data-driven tool for solving parametric PDEs as they can represent maps between infinite-dimensional function spaces. In this work, we employ physics-informed Neural Operators in the context of high-dimensional, Bayesian inverse problems. Traditional solution strategies necessitate an enormous, and frequently infeasible, number of forward model solves, as well as the computation of parametric derivatives. In order to enable efficient solutions, we extend Deep Operator Networks (DeepONets) by employing a RealNVP architecture which yields an invertible and differentiable map between the parametric input and the branch-net output. This allows us to construct accurate approximations of the full posterior, irrespective of the number of observations and the magnitude of the observation noise, without any need for additional forward solves nor for cumbersome, iterative sampling procedures. We demonstrate the efficacy and accuracy of the proposed methodology in the context of inverse problems for three benchmarks: an anti-derivative equation, reaction-diffusion dynamics and flow through porous media.

Keywords Data-driven surrogates · Invertible neural networks · Bayesian inverse problems · Semi-supervised learning

1 Introduction

Nonlinear Partial Differential Equations (PDEs) depending on high- or even infinite-dimensional parametric inputs are ubiquitous in applied physics and engineering and appear in the context of several problems such as model calibration and validation or model-based design/optimization/control. In all these cases, they must be solved repeatedly for different values of the input parameters which poses an often insurmountable obstacle as each of these simulations can imply a significant computational cost. An obvious way to overcome these difficulties is to develop less-expensive but accurate surrogates which can be used on their own or in combination with a reduced number of runs of the high-fidelity, expensive, reference solver. The construction of such surrogates has been based on physical/mathematical considerations or data i.e. input-output pairs (and sometimes derivatives). Our contribution belongs to the latter category of data-driven surrogates which has attracted a lot of attention in recent years due to the significant progress in the fields of statistical or machine learning [18,24]. We emphasize however that unlike typical supervised learning problems in data sciences, in the context of computational physics there are several distinguishing features. Firstly, surrogate construction is by definition a Small (or smallest possible) Data problem. The reason we want to have a surrogate in the first place is to avoid using the reference solver which is the one that generates the training data. Secondly, pertinent problems are rich in domain knowledge which should be incorporated as much as possible, not only in order to reduce the requisite training data but also to achieve higher predictive accuracy particularly in out-of-distribution settings. In the context of Bayesian inverse problems which we investigate in this paper, one does

✉ Phaedon-Stelios Koutsourelakis
p.s.koutsourelakis@tum.de
https://www.mdsi.tum.de

Sebastian Kaltenbach
sebastian.kaltenbach@tum.de

Paris Perdikaris
pgp@seas.upenn.edu

[1] Professorship of Data-driven Materials Modeling, School of Engineering and Design, Technical University of Munich , Boltzmannstr. 15, 85748 Garching, Germany

[2] Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia 19104, USA

[3] Munich Data Science Institute (MDSI - Core member), Garching, Germany

not know a priori where the posterior might be concentrated in the parametric space and cannot guarantee that all such regions will be sufficiently represented in the training dataset. Nevertheless the surrogate learned must be accurate enough in these regions in order to resolve the sought posterior.

Data-driven surrogates which are trained in an offline phase and are subsequently used for various downstream tasks have attracted a lot of attention in recent years [5]. Most of these surrogates are constructed by learning a non-linear operator, e.g. a mapping between function spaces and thus between the inputs and the outputs of the PDE, which may depend on additional input parameters. A notable such strategy based on Deep Learning are the Physics-informed Neural Networks (PINNs) [25,33]. An alternative is offered by Deep Operator Networks (DeepONets, [31,36]), which in contrast to PINNs, not only take the spatial and temporal location as an input but can also account for the dependence of the PDE solution on input parameters such as the viscosity in the Navier-Stokes equation. Furthermore, Fourier Neural Networks [28] have shown promising results by parametrizing the integral kernel directly in Fourier Space and thus restricting the operator to a convolution. Finally, the Learning Operators with Coupled Attention (LOCA) framework [20] builds upon the well-known attention mechanism that has already shown promising results in natural language processing.

We note that all of the Deep Learning frameworks mentioned fulfill the universal approximation theorem and, under certain conditions, can approximate the non-linear operator to arbitrary accuracy. Another option, is offered by the Optimizing a Discrete Loss (ODIL, [17]) framework. It does not rely on Deep Learning and was shown to be faster than PINNs due to the reduced number of tunable parameters but can only approximate the solution on a discrete grid.

Apart from the aforementioned techniques and for time-dependent PDEs in particular, the solution can be approximated by methods based on Koopman-operator theory [22] which identifies a transformation of the original system that gives rise to linear dynamics [21]. Nevertheless, these methods [8,12,27] usually require a large set of reduced-order coordinates or an effective encoder/decoder structure. Especially for physical systems, the restricted dynamics can be endowed with stability and physical, inductive bias [14–16].

A common limitation of the aforementioned architectures is that they usually learn only the forward operator whereas for the solution of an inverse problem, its inverse would be more useful. In this work, we extend the DeepONet framework by replacing parts of the previously proposed neural-network architecture with an invertible one. To the authors' best knowledge, we are thus presenting the first invertible Neural Operator framework.This allows one to perform both forward and inverse passes with the same neural network and the forward and inverse operators can be learned simultaneously. In particular, we make use of the RealNVP architecture [10] which has an analytical inverse.

Furthermore we make use of both labeled and unlabeled (i.e. only inputs and residuals) training data in a physics-aware, semi-supervised approach. While the use of labeled training data is straight-forward, unlabeled training data are incorporated by using the governing equations and minimizing the associated residuals, similarly to the physics-informed DeepONet [36]. Since it is easier and less-expensive to procure unlabeled data in comparison to labeled ones, this leads to significant efficiency gains. Even though our algorithm can produce accurate predictions without any labeled training data and by using only a physics-informed loss, we observe empirically that the addition of labeled training data generally improves the results.

Finally, we show that the proposed invertible DeepONet can be used to very efficiently solve Bayesian inverse problems, i.e. to approximate the whole posterior distribution, without any need for multiple likelihood evaluations and cumbersome iterations as required by alternative inference schemes such as Markov Chain Monte Carlo (MCMC, [4]) or Sequential Monte Carlo (SMC, [23]) or Stochastic Variational Inference (SVI, [9]). In particular, we propose a novel approximation that employs a mixture of Gaussians, the parameters of which are computed semi-analytically. When the proposed Neural Operator framework is trained solely on unlabeled data, this means that we can obtain the solution to the (forward and) inverse problem without ever solving the underlying PDE. While Deep Learning has been successfully applied to inverse problems before [1,2,32], our work differs by making use of a fully invertible, operator-learning architecture which leads to a highly efficient approximation of the whole posterior.

The rest of the paper is structured as follows. In Sect. 2 we review the basic elements of invertible neural networks (NNs) and DeepONets and subsequently illustrate how these can be combined and trained with labeled and unlabeled data. Furthermore we present how the resulting invertible DeepONet can be employed in order to approximate the posterior of a model-based, Bayesian inverse problem at minimal additional cost. We illustrate several features of the proposed methodology and assess its performance in Sect. 3 where it is applied to a reaction-diffusion PDE and a Darcy-diffusion problem. The cost and accuracy of the posterior approximation in the context of pertinent Bayesian inverse problems are demonstrated in Sect. 3.4. Finally, we conclude in Sect. 4 with a summary of the main findings and a discussion on the (dis)advantages of the proposed architecture and potential avenues for improvements.

2 Methodology

We first review some basic concepts of invertible neural networks and DeepONets. We subsequently present our novel contributions which consist of an invertible DeepONet architecture and its use for solving efficiently Bayesian inverse problems.

2.1 Invertible neural networks

Neural Networks are in general not invertible which restricts their application in problems requiring inverse operations. Invertibility can be achieved by adding a momentum term [34], restricting the Lipschitz-constant of each layer to be smaller than one [3] or using special building blocks [10]. These formulations have primarily been developed for flow-based architectures but we will apply them to operator learning in this work. In particular, we make use of the RealNVP [10] as this architecture enables an analytical inverse which ensures efficient computations. Each RealNVP building block consists of the transformation below which includes two neural networks denoted by $k(.)$ and $r(.)$. Given a D dimensional input $\boldsymbol{x} = \{x_i\}_{i=1}^{D}$ of an invertible layer, the output $\boldsymbol{y} = \{y_i\}_{i=1}^{D}$ is obtained as follows:

$$y_{1:d} = x_{1:d} \tag{1}$$

$$y_{d+1:D} = x_{d+1:D} \circ exp(k(x_{1:d})) + r(x_{1:d}), \tag{2}$$

where $d < D$. Here, $\circ$ is the Hadamard or element-wise product and d is usually chosen to be half of the dimension of the input vector i.e. $d = D/2$.
As only d of the components are updated, the input entries after each building block are permuted, e.g. by reversing the vector, to ensure that after a second building block all of them are modified. Therefore, for $d = D/2$, at least two building blocks are needed in order to modify all entries. We note, that the dimension of the input cannot change and it needs to be identical to the dimension of the output. The two neural networks involved can consist of arbitrary layers as long as their output and input dimensions are consistent with Eq. (2). The maps defined can be easily inverted which leads to the following equations:

$$x_{1:d} = y_{1:d} \tag{3}$$

$$x_{d+1:D} = (y_{d+1:D} - r(x_{1:d})) \circ exp(-k(x_{1:d})) \tag{4}$$

We note that due to this structure, the Jacobian is lower-triangular and its determinant can be obtained by multiplying the diagonal entries only.

2.2 DeepONets

Before presenting our novel architecture for invertible DeepONets, we briefly review the original DeepONet formulation by [31]. DeepONets have been developed to solve parametric PDEs and significantly extend the Physics-Informed Neural Network (PINNs, [33]) framework as no additional training phase is required if the input parameters of the PDE are changed. We consider a, potentially nonlinear and time-dependent, PDE with an input function $u \in \mathcal{U}$ and solution function $s \in \mathcal{S}$ where $\mathcal{U}, \mathcal{S}$ are appropriate Banach spaces. The former can represent e.g. source terms, boundary or initial conditions, material properties. Let:

$$\mathcal{N}(u,\ s)(\boldsymbol{\xi}) = 0 \tag{5}$$

denote the governing PDE where $\mathcal{N} : \mathcal{U} \times \mathcal{S} \rightarrow \mathcal{V}$ is an appropriate differential operator and $\boldsymbol{\xi}$ the spatio-temporal coordinates. Furthermore, let:

$$\mathcal{B}(u,\ s)(\boldsymbol{\xi}) = 0 \tag{6}$$

denote the operator $\mathcal{B} : \mathcal{U} \times \mathcal{S} \rightarrow \mathcal{V}$ associated with the boundary or initial conditions. Assuming that the solution s for each $u \in \mathcal{U}$ is unique, we denote with $\mathcal{G} : \mathcal{U} \rightarrow \mathcal{S}$ the solution operator that maps from any input u to the corresponding solution s. The goal of DeepONets is to approximate it with an operator $G_{\boldsymbol{\theta}}$ that depends on tunable parameters $\boldsymbol{\theta}$. The latter can yield an approximation to the actual solution at any spatio-temporal point $\boldsymbol{\xi}$ which we denote by $G_{\boldsymbol{\theta}}(\boldsymbol{\xi})$. It is based on a separated representation [31][1]:

$$G_{\boldsymbol{\theta}}(u)(\boldsymbol{\xi}) = \sum_{j=1}^{Q} b_j \left(\underbrace{u(\boldsymbol{\eta}_1), ..., u(\boldsymbol{\eta}_F)}_{\boldsymbol{u}} \right) t_j(\boldsymbol{\xi}) \tag{7}$$

and consists of the so-called *branch network* whose terms b_j depend on the values of the input function u at F fixed spatio-temporal locations[2] $\{\boldsymbol{\eta}_l\}_{l=1}^{F}$ which we summarily denote with the vector $\boldsymbol{u} \in \mathbb{R}^F$, and the so-called *trunk network* whose terms t_j depend on the spatio-temporal coordinates $\boldsymbol{\xi}$ (see Fig. 1b). Both networks have trainable weight and bias parameters which we denote collectively by $\boldsymbol{\theta}$. We emphasize that, once trained, the DeepONet can provide predictions of the solution at *any spatio-temporal location* $\boldsymbol{\xi}$, a feature that is very convenient in the context of inverse problems as the same DeepONet can be used for solving problems with different sets of observations.

[1] We omit the NN parameters $\boldsymbol{\theta}$ on the right-hand side in order to simplify the notation.

[2] These points are usually chosen to be uniformly distributed over the entire domain, but it is also possible to increase their density in certain areas, e.g. with high variability.

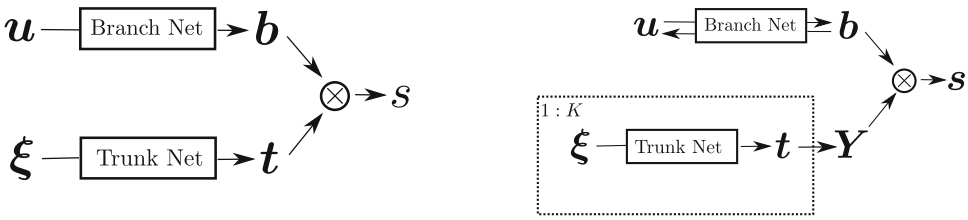

Fig. 1 (Left) Classical DeepONet [30] and (Right) proposed Invertible DeepONet architecture

We note that in the next section, we will use a vectorized formulation of Eq. (7) and process various spatio-temporal coordinate datapoints together as this is needed to ensure invertibility of the DeepONet.

Labeled data can be used for training which consist of pairs of u and corresponding solutions $s = \mathcal{G}(u)$ evaluated at certain spatio-temporal locations. Unlabeled training data (i.e. only inputs) can also be employed in a physics-informed approach as introduced in [36], by including the governing PDE in Eq. (5) in an additional loss term as discussed Sect. 2.4.

2.3 Invertible DeepONets

The invertible RealNVP introduced in Sect. 2.1 is employed exclusively on the branch network i.e. we assume that:

$$D = F = Q \tag{8}$$

and the input $\boldsymbol{x}$ of Sect. 2.1 is the vector $\boldsymbol{u} \in \mathbb{R}^F$ containing the values of the PDE-input at $D = F$ spatio-temporal locations whereas the output $\boldsymbol{y}$ of Sect. 2.1 is now the $D = Q$ values of the branch net $\boldsymbol{b} = [b_1, \ldots, b_Q]^T \in \mathbb{R}^D$. We note that this restriction regarding the equality of the dimension of the input $\boldsymbol{u}$ and the output of the branch network $\boldsymbol{b}$ is due to the use of an invertible architecture. As a consequence, the dimension of the trunk-network output i.e. $\{t_j(\boldsymbol{\xi})\}_{j=1}^{Q}$ is also the same as the dimension of $\boldsymbol{u}$. This requirement does not reduce the generality of the methodology advocated as Q is a free parameter in the definition of the operator $G_{\boldsymbol{\theta}}$ in Eq. (7).

In view of the inverse problems we would like to address, we consider K spatio-temporal locations, $\{\boldsymbol{\xi}_k\}_{k=1}^{K}$ and we denote with $\boldsymbol{s} \in \mathbb{R}^K$ the vector containing the PDE-solution's values at these locations i.e. $\boldsymbol{s} = \left[s(\boldsymbol{\xi}_1), \ldots, s(\boldsymbol{\xi}_K)\right]^T$. Finally we denote with $\boldsymbol{Y}$ the $K \times D$ matrix constructed by the values of the trunk network outputs at the aforementioned locations, i.e.:

$$\boldsymbol{Y} = \begin{bmatrix} t_1(\boldsymbol{\xi}_1) & \ldots & t_D(\boldsymbol{\xi}_1) \\ \ldots & & \ldots \\ t_1(\boldsymbol{\xi}_K) & \ldots & t_D(\boldsymbol{\xi}_K) \end{bmatrix}. \tag{9}$$

As a result of Eq. (7), we can write that:

$$\boldsymbol{s} = \boldsymbol{Y}\boldsymbol{b} \tag{10}$$

As the matrix $\boldsymbol{Y}$ is in general non-invertible, one can determine $\boldsymbol{b}$ given $\boldsymbol{s}$ by solving a least-squares problem, i.e.:

$$\min_{\boldsymbol{b}} \ \|\boldsymbol{s} - \boldsymbol{Y}\boldsymbol{b}\|_2^2 \tag{11}$$

or a better-behaved, regularized version thereof:

$$\min_{\boldsymbol{b}} \ \|\boldsymbol{s} - \boldsymbol{Y}\boldsymbol{b}\|_2^2 + \epsilon\|\boldsymbol{b}\|_2^2 \tag{12}$$

where a small value is generally sufficient for the regularization parameter $\epsilon << 1$. We note that given $\boldsymbol{s}$ and once $\boldsymbol{b}$ has been determined by solving Eqs. (11) or eqrefeq:min2, we can make use of the invertibility of the branch net in order to obtain the input vector $\boldsymbol{u}$. While other approaches are possible in order to determine $\boldsymbol{b}$, we recommend using the regularized, least-squares formulation, as this led to robust results in our experiments. It is nevertheless important to use the same method during training and when deterministic predictions are sought, since different methods can lead to different $\boldsymbol{b}$'s for the same $\boldsymbol{s}$. We note that in the proposed method for the solution of Bayesian inverse problems (see Sect. 2.5), no use of Eq. (12) is made except for the training of the DeepONet (see Sect. 2.4).

For the ensuing equations we denote the forward map implied by equation (10) as:

$$\boldsymbol{s} = \mathcal{F}_{\boldsymbol{\theta}}(\boldsymbol{u}, \boldsymbol{Y}) \tag{13}$$

and the inverse obtained by the two steps described above as:

$$\boldsymbol{u} = \mathcal{I}_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{Y}) \tag{14}$$

where we explicitly account for the NN parameters $\boldsymbol{\theta}$.

2.4 A semi-supervised approach for invertible DeepONets

As mentioned earlier and in order to train the invertible DeepONet proposed, i.e. to find the optimal values for the parameters $\boldsymbol{\theta}$, we employ both labeled (i.e pairs of PDE-inputs u and PDE-outputs s) and unlabled data (i.e. only

PDE-inputs u) in combination with the governing equations. The loss function L employed is therefore decomposed into two parts as[3]:

$$L = L_{labeled} + L_{unlabeled} \tag{15}$$

The first term $L_{labeled}$ pertains to the labeled data and is further decomposed as:

$$L_{labeled} = L_{l,forward} + L_{l,inverse} \tag{16}$$

Without loss of generality and in order to keep the notation as simple as possible we assume that N_l pairs of labeled data are available, each of which consists of the values of the PDE-input u at D locations which we denote with $\boldsymbol{u}^{(i)} \in \mathbb{R}^D$, $i = 1, \dots N_l$ and the values of the PDE-output at K spatio-temporal locations which we denote with $\boldsymbol{s}^{(i)} \in \mathbb{R}^K$, $i = 1, \dots N_l$. If the $K \times D$ matrix $\boldsymbol{Y}$ is defined as in Eq. (9) and in view of the forward (Eq. (13)) and inverse (Eq. (14)) maps defined earlier, we write:

$$L_{l,forward} = \frac{1}{N_l} \sum_{i=1}^{N_l} \|\boldsymbol{s}^{(i)} - \mathcal{F}_{\boldsymbol{\theta}}(\boldsymbol{u}^{(i)}, \boldsymbol{Y})\|_2^2 \tag{17}$$

and:

$$L_{l,inverse} = \frac{1}{N_l} \sum_{i=1}^{N_l} \|\boldsymbol{u}^{(i)} - \mathcal{I}_{\boldsymbol{\theta}}(\boldsymbol{s}^{(i)}, \boldsymbol{Y})\|_2^2. \tag{18}$$

By employing both loss terms, the NN parameters $\boldsymbol{\theta}$ can balance the accuracy of the approximation in both maps.

Furthermore and assuming N_u PDE-inputs are available each of which is evaluated at D spatio-temporal points $\{\boldsymbol{\xi}^{(l)}\}_{l=1}^D$ with $\boldsymbol{u}^{(i)} \in \mathbb{R}^D$ denoting these values, we express the $L_{unlabeled}$ loss term as:

$$L_{unlabeled} = L_{BC} + L_{res} + L_{u,inverse}. \tag{19}$$

The first L_{BC} and second L_{res} terms are physics-informed [36] and account for the residuals in the boundary (and/or initial) conditions and the governing PDE respectively. In the case of L_{BC} we select N_B (uniformly distributed) points along the boundary, say $\boldsymbol{\xi}_B^{(j)}, l = 1, \dots, N_B$. Then, in view of Eq. (6), we employ:

$$L_{BC} = \frac{1}{N_u N_B} \sum_{i=1}^{N_u} \sum_{l=1}^{N_B} \|\mathcal{B}(u^{(i)}, G_{\boldsymbol{\theta}}(u^{(i)}))(\boldsymbol{\xi}_B^{(l)})\|_2^2 \tag{20}$$

[3] All loss functions depend on $\boldsymbol{\theta}$ which we omit in order to simplify the notation.

In the interior of the problem domain and in view of Eq. (5), we employ a loss:

$$L_{res} = \frac{1}{N_u\, N_{res}} \sum_{i=1}^{N_u} \sum_{l=1}^{N_{res}} \|\mathcal{N}(u^{(i)}, G_{\boldsymbol{\theta}}(u^{(i)})(\boldsymbol{\xi}^{(l)})\|_2^2 \tag{21}$$

which involves N_{res} collocation points.

The third term $L_{u,inverse}$ pertains to the forward and inverse maps in Eqs. (13), (14) and can be expressed as:

$$L_{u,inverse} = \frac{1}{N_u} \sum_{i=1}^{N_u} \|\boldsymbol{u}^i - \mathcal{I}_{\boldsymbol{\theta}}(\mathcal{F}_{\boldsymbol{\theta}}(\boldsymbol{u}^{(i)}, \boldsymbol{Y}), \boldsymbol{Y}))\|_2^2 \tag{22}$$

where the matrix $\boldsymbol{Y}$ is defined as in Eq. (9).

The minimization of the combined loss L, with respect to the NN parameters $\boldsymbol{\theta}$ of the branch and trunk network, is performed with stochastic gradient descent and the ADAM [19] scheme in particular. Gradients of the loss were computed using the automatic differentiation tools of the JAX library [7]. We finally note that the D spatioemporal locations need not be the same nor do they need to be equal in number in all data instances as assumed in the equations above. In such cases the vector of the observables and the matrices $\boldsymbol{Y}$ involved would differ which would further complicate the notation but the same DeepONet parameters $\boldsymbol{\theta}$ would appear in all terms.

2.5 Invertible DeepONets for Bayesian inverse problems

In this section we present how the invertible DeepONets proposed and trained as previously discussed, can be used to efficiently approximate the solution of a Bayesian inverse problem in the presence of, potentially noisy, observations as well as prior uncertainty about the unknowns. A central role is played by the readily available invertible map which the RealNVP architecture affords. In particular, let $\hat{\boldsymbol{s}} \in \mathbb{R}^K$ denote a vector of noisy observations of the PDE-solution at certain K spatio-temporal locations. These are assumed to be related to the PDE-solution's values at these locations, denoted summarily by $\boldsymbol{s} \in \mathbb{R}^K$, as follows:

$$\hat{\boldsymbol{s}} = \boldsymbol{s} + \sigma\, \boldsymbol{\eta}, \qquad \boldsymbol{\eta} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{23}$$

where σ^2 is the variance of the observational noise. This in turn defines a conditional density (likelihood) $p(\hat{\boldsymbol{s}} \mid \boldsymbol{s})$:

$$p(\hat{\boldsymbol{s}} \mid \boldsymbol{s}) = \mathcal{N}(\hat{\boldsymbol{s}} \mid \boldsymbol{s}, \sigma^2 \boldsymbol{I}). \tag{24}$$

In the context of a Bayesian formulation and given the implicit dependence of the PDE-output $\boldsymbol{s}$ on $\boldsymbol{u}$, the likelihood would be combined with the a prior density $p_u(\boldsymbol{u})$ on

the PDE-inputs in order to define the sought posterior:

$$p(\boldsymbol{u} \mid \hat{\boldsymbol{s}}) \propto p(\hat{\boldsymbol{s}} \mid \boldsymbol{s})\ p_u(\boldsymbol{u}).$$

Even if the trained DeepONet were used to infer $p(\boldsymbol{u} \mid \hat{\boldsymbol{s}})$ (e.g. using MCMC) several evaluations would be needed especially if the dimension of $\boldsymbol{u}$ was high. In the sequel we demonstrate how one can take advantage of the invertible NN architecture in order to obtain a semi-analytic approximation of the posterior in the form of a mixture of Gaussians and by avoiding iterative algorithms like MCMC altogether.

We note first that by combining the likelihood with Eq. (10), we can write it in terms of the $D-$dimensional, branch-network output vector $\boldsymbol{b}$ as:

$$p(\hat{\boldsymbol{s}} \mid \boldsymbol{b}) = \mathcal{N}(\hat{\boldsymbol{s}} \mid \boldsymbol{Y}\ \boldsymbol{b}, \sigma^2 \boldsymbol{I}). \tag{25}$$

Since $\boldsymbol{u} \in \mathbb{R}^D$ is related to $\boldsymbol{b}$ through the invertible RealNVP $\boldsymbol{b}_{NN} : \mathbb{R}^D \to \mathbb{R}^D$, we can also obtain a prior density $p_b(\boldsymbol{b})$ on $\boldsymbol{b}$ as:

$$p_b(\boldsymbol{b}) = p_u(\boldsymbol{b}_{NN}^{-1}(\boldsymbol{b}))\ J(\boldsymbol{b}) \tag{26}$$

where $\boldsymbol{b}_{NN}^{-1}$ denotes the inverse and $J(\boldsymbol{b}) = |\frac{\partial \boldsymbol{b}_{NN}^{-1}}{\partial \boldsymbol{b}}|$ is the determinant of its Jacobian. The latter, as mentioned in Sect. 2.1, is a triangular matrix and its determinant can be readily computed at a cost $\mathcal{O}(D)$.

We choose not to directly operate with the prior $p_b(\boldsymbol{b})$, but construct an approximation $p_{b,G}(\boldsymbol{b})$ to this in the form of a mixture of $D-$dimensional Gaussians as this allows as to facilitate subsequent steps in finding the posterior. In particular:

$$p_{b,G}(\boldsymbol{b}) = \sum_{m=1}^{M} w_j\ \mathcal{N}(\boldsymbol{b} \mid \boldsymbol{m}_{b,m}, \boldsymbol{S}_{b,m}) \tag{27}$$

where M denotes the number of mixture components and $\boldsymbol{m}_{b,m}$, $\boldsymbol{S}_{b,m}$ the mean vector and covariance matrix of the m^{th} component respectively. Such an approximation can be readily computed, e.g. using Variational Inference [35] and *without* any forward or inverse model evaluations by exploiting the fact that samples from p_b can be readily drawn using ancestral sampling i.e. by drawing samples of $\boldsymbol{u}$ from p_u and propagating those with $\boldsymbol{b}_{NN}$. We note that finding this representation can become more diffucult in case M is large but the complexity of the algorithms involved in general scales linearly with M [6].

By combining the (approximate prior) $p_{b,G}(\boldsymbol{b})$ above with the Gaussian likelihood $p(\hat{\boldsymbol{s}} \mid \boldsymbol{b})$ of Eq. (25) we obtain an expression for the posterior $\tilde{p}(\boldsymbol{b} \mid \hat{\boldsymbol{s}})$ using Bayes' theorem:

$$\tilde{p}(\boldsymbol{b} \mid \hat{\boldsymbol{s}}) \propto p(\hat{\boldsymbol{s}} \mid \boldsymbol{b}) p_{b,G}(\boldsymbol{b}) \tag{28}$$

Due to the conjugacy of prior and likelihood, we can directly conclude that the (approximate) posterior is also a mixture of Gaussians [6]. Therefore, using expressions for the aforementioned likelihood/prior pair, we obtain a closed-form posterior $\tilde{p}(\boldsymbol{b} \mid \hat{\boldsymbol{s}})$ on $\boldsymbol{b}$ of the form:

$$\tilde{p}(\boldsymbol{b} \mid \hat{\boldsymbol{s}}) = \sum_{m=1}^{M} \tilde{w}_j\ \mathcal{N}(\boldsymbol{b} \mid \boldsymbol{\mu}_{b,m}, \boldsymbol{C}_{b,m}) \tag{29}$$

where the mean $\boldsymbol{\mu}_{b,m}$ and covariance $\boldsymbol{C}_{b,m}$ of each mixture component can be computed as:

$$\begin{aligned} \boldsymbol{C}_{b,m}^{-1} &= \sigma^{-2} \boldsymbol{Y}^T \boldsymbol{Y} + \boldsymbol{S}_{b,m}^{-1} \\ \boldsymbol{C}_{b,m}^{-1} \boldsymbol{\mu}_{b,m} &= \sigma^{-2} \boldsymbol{Y}^T \hat{\boldsymbol{s}} + \boldsymbol{S}_b^{-1} \boldsymbol{m}_{b,m} \end{aligned} \tag{30}$$

The weights $\tilde{w}_m$ ($\sum_{m=1}^{M} \tilde{w}_m = 1$) would be proportional to:

$$\begin{aligned} \tilde{w}_m \propto w_m \mid \boldsymbol{D}_m \mid^{-1/2} \exp(-\tfrac{1}{2} (\hat{\boldsymbol{s}} - \boldsymbol{Y} \boldsymbol{m}_{b,m})^T \\ \boldsymbol{D}_m^{-1} (\hat{\boldsymbol{s}} - \boldsymbol{Y} \boldsymbol{m}_{b,m})) \end{aligned} \tag{31}$$

where:

$$\boldsymbol{D}_m = \sigma^2 \boldsymbol{I} + \boldsymbol{Y} \boldsymbol{S}_{b,m} \boldsymbol{Y}^T \tag{32}$$

Therefore inference tasks on the sought $\boldsymbol{u}$ can be readily carried out by sampling $\boldsymbol{b}$ from the mixture-of-Gaussians posterior above and propagating those samples through the inverse map $\boldsymbol{b}_{NN}^{-1}$ to obtain $\boldsymbol{u}$-samples. We note that by employing a mixture of Gaussians with sufficient components M, one can approximate with arbitrary accuracy any non-Gaussian density as well as capture multimodal posteriors, a task that is extremely cumbersome with standard, Bayesian inference schemes [11].

3 Numerical illustrations

We applied the proposed framework to three examples, i.e. the antiderivate operator , a reaction-diffusion PDE as well as a Darcy-type elliptic PDE. In each of these cases, we report the relative errors of forward and inverse maps (on test data) when trained with varying amounts of labeled and unlabeled training data. For the reaction-diffusion PDE and the Darcy-type elliptic PDE, we also use the proposed invertible-DeepONet-surrogate to solve pertinent Bayesian inverse problems. The code for the aforementioned numerical illustrations is available here[4] In Table 1, we summarize the most important dimensions for each of the following examples, namely D: the dimension of the PDE-input, K:

[4] https://github.com/pkmtum/Semi-supervised_Invertible_Neural_Operators.

Table 1 Main dimensions for each numerical illustration

	Sects. 3.1	3.2	3.3	3.4.1	3.4.2
D	100	100	64	100	64
K	200	200	3844	25, 100	1922, 3844
N_l	10^2, 10^3, 10^4	0, 500, 5000	10^3	500	5000
N_u	10^4	5000	10^3	10^4	5000
N_{res}	200	200	3844	200	200
N_{BC}	–	300	–	–	–

Table 2 Relative test errors and their standard deviations depending on the amount of labeled training data for the anti-derivative operator. The percentage of labeled data is the amount of data used in comparison to unlabeled training data, e.g. in the 10% case we used ten times more unlabeled training data whereas in the 100% case the amount of labeled and unlabeled training data was the same

labeled data [%]	1	10	100
Relative error s (forward map)	0.0152 ± 0.0151	0.00791 ± 0.00799	0.00728 ± 0.00797
Relative error u (inverse map)	0.0371 ± 0.0241	0.034 ± 0.024	0.0215 ± 0.0153

the dimension of the observed PDE-output, N_l: number of labeled data (Eqs. (17), (18)), N_u: the number of unlabeled data (e.g. Eq. (22)), N_{res}: the number of interior collocation points (Eq. (21)) and N_{BC} the number of boundary collocation points (Eq. (20)).

3.1 Anti-derivative operator

As a first test case we considered the antiderivative operator on the interval $\xi \in [0, 1]$ with:

$$\frac{ds(\xi)}{d\xi} = u(\xi) \text{ with } s(0) = 0 \tag{33}$$

i.e. when the input u corresponds to the right-hand-side of this ODE and the operator $\mathcal{G}(u)$ that we attempt to approximate is simply the integral operator $\mathcal{G}(u)(\xi) = \int_0^\xi u(t)\, dt$. We generated $N_u = 10000$ unlabeled training data by sampling inputs u from a Gaussian process with zero mean and exponential quadratic covariance kernel with a length scale $\ell = 0.2$. Their values at the same $D = 100$ uniformly-distributed locations in $[0, 1]$ were recorded. We subsequently randomly choose $N_{res} = 200$ collocation points to evaluate the residuals (see Eq. (21)).

Moreover, we used up to $N_l = 10000$ labeled training data, for which the inputs were generated as for the unlabeled training data, and the outputs were obtained by solving the ODE above and evaluating it at $K = 200$ randomly chosen points. We trained the invertible DeepONet on $N_u = 10000$ unlabeled training data with a batch size of 100. In each batch we added 1, 10 or 100 labeled training data points per batch (i.e. $N_l = 100, 1000, 10000$ respectively in Eqs. (17), (18)). A minimum of one labeled datapoint is required in order to set the initial condition correctly as we did not enforce this separately in the unlabeled loss part. With regards to the architecture of the networks used, we employed a MLP with four layers and 100 neurons each for the trunk network and 6 RealNVP building blocks for the branch network which were parametrized by a two-layered MLP. Variations around these values in the number of neurons, layers were also explored (in the subsequent examples as well) and did not impact significantly the performance.

Using the ADAM optimizer and an initial learning rate of 10^{-3}, we run the model training for 4×10^4 iterations with an exponential learning rate decay with rate 0.9 every 1000 iterations. As test data, we used 1000 new (i.e. not included in the training data) input-output pairs and compared the predicted forward and inverse solutions with the actual ones. The results obtained in terms of the relative errors are summarized in Table 2.

The error values indicate that both the forward as well as the inverse maps are very well approximated by the proposed invertible DeepONet. The addition of more labeled training data results in even lower errors especially for the inverse map for which the relative error is decreased from almost $\sim 4\%$ to $\sim 2\%$.

In order to visualize the results we plot for four randomly-chosen test cases the predictions (when trained with 10% labeled data) of both the forward (Fig. 2) and inverse (Fig. 3) operator. In all cases, the predictions are indistinguishable from the reference functions.

In Appendix A we include additional results for this problem with varying amounts of unlabeled and labeled training data in order to further show their influence.

3.2 Reaction-diffusion dynamics

The second illustrative example involves the reaction-diffusion equation on the space-time domain $\boldsymbol{\xi} = (x, t) \in [0, 1] \times$

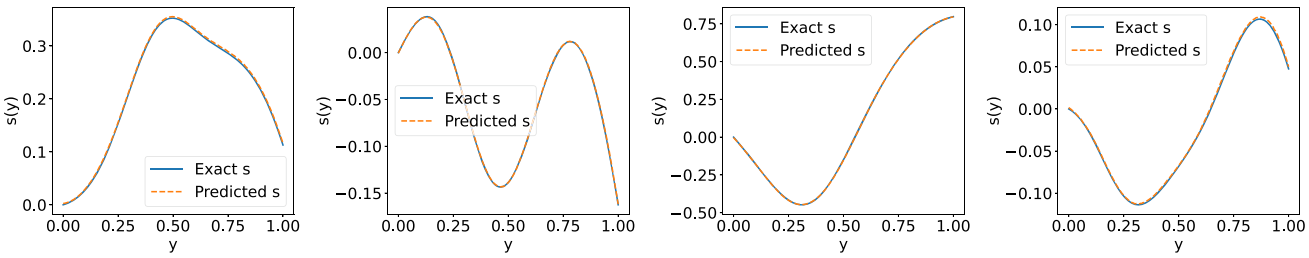

Fig. 2 Forward map - Comparison of the true PDE-output/solution s (given a PDE-input u) with the one predicted by the proposed invertible DeepONet and for the anti-derivative operator

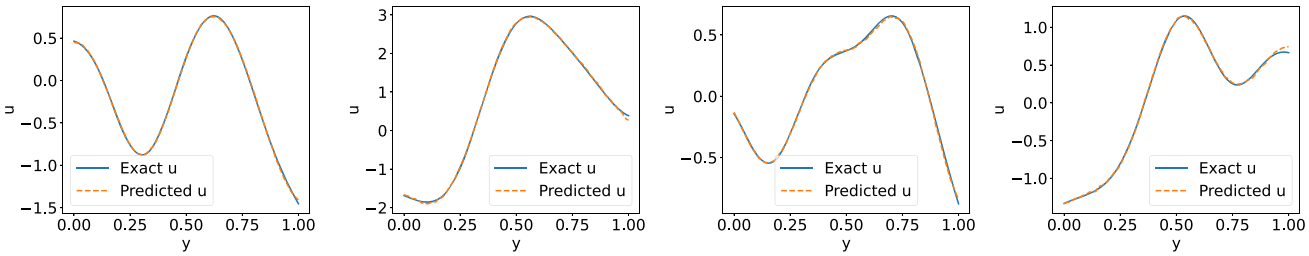

Fig. 3 Inverse map - Comparison of the true PDE-input u (given the PDE-output/solution s) with the one predicted by the proposed invertible DeepONet and for the anti-derivative operator

Table 3 Relative errors on test data depending on the amount of labeled training data for the reaction-diffusion case

labeled data [%]	0	10	100
Relative error for s	0.00925 ± 0.00492	0.0105 ± 0.00519	0.00813 ± 0.00445
Relative error for u	0.024 ± 0.01021	0.0184 ± 0.00578	0.0162 ± 0.00592

$[0, 1]$:

$$\frac{\partial s}{\partial t} = D_s \frac{\partial^2 s}{\partial x^2} + ks^2 + u(x) \tag{34}$$

Here, $D_s = 0.01$ is the diffusion constant, $k = 0.01$ the reaction rate and the source-term $u(x)$ is chosen to be the PDE-input. We used zero values as initial conditions and boundary conditions. We generated random source terms by sampling from a Gaussian process with zero mean and and exponential quadratic covariance kernel with a length scale $\ell = 0.2$ which were then evaluated at $D = 100$ uniformly distributed points over $[0, 1]$. The PDE was subsequently solved using an implicit Finite-Difference scheme and evaluated at 200 randomly chosen points to generate the labeled training data.

We trained our model with $N_u = 5000$ unlabeled data which were processed in batches of 100 samples and to which varying amounts of labeled data were added. Since for this problem the boundary conditions were enforced separately, the amount of labeled training data used could also be zero. All unlabeled training data points were evaluated at $N_{res} = 200$ randomly selected collocation points. With regards to the network architecture, we employed a MLP with five layers and 100 neurons each for the trunk network and 3 RealNVP building blocks for the branch network which were parametrized by a three-layered MLP. Using the ADAM optimizer and an initial learning rate of 10^{-3}, we run the model training for 12×10^4 iterations with an exponential learning rate decay with rate 0.9 every 2000 iterations. For our test dataset, we generated 1000 new (unseen) source terms u and corresponding solutions s. A summary of the relative errors obtained is contained in Table 3.

We note that again for all three settings we achieve very low error rates, which decrease as the amount of labeled training data increases. In Figs. 4 and 5 we show the predictions (trained with 500 i.e. 10% labeled data) of both forward and inverse map for three randomly chosen test cases.

3.3 Flow through porous media

In the final example we considered the Darcfy-flow elliptic PDE in the two-dimensional domain $\boldsymbol{\xi} = (x_1, x_2) \in [0, 1]^2$

$$\nabla \cdot (u(\boldsymbol{\xi}) \nabla s(\boldsymbol{\xi})) = 10 \tag{35}$$

where the PDE-input u corresponds to the permeability field. We assumed zero values for the solution s along all boundaries which we a-priori incorporated in our operator

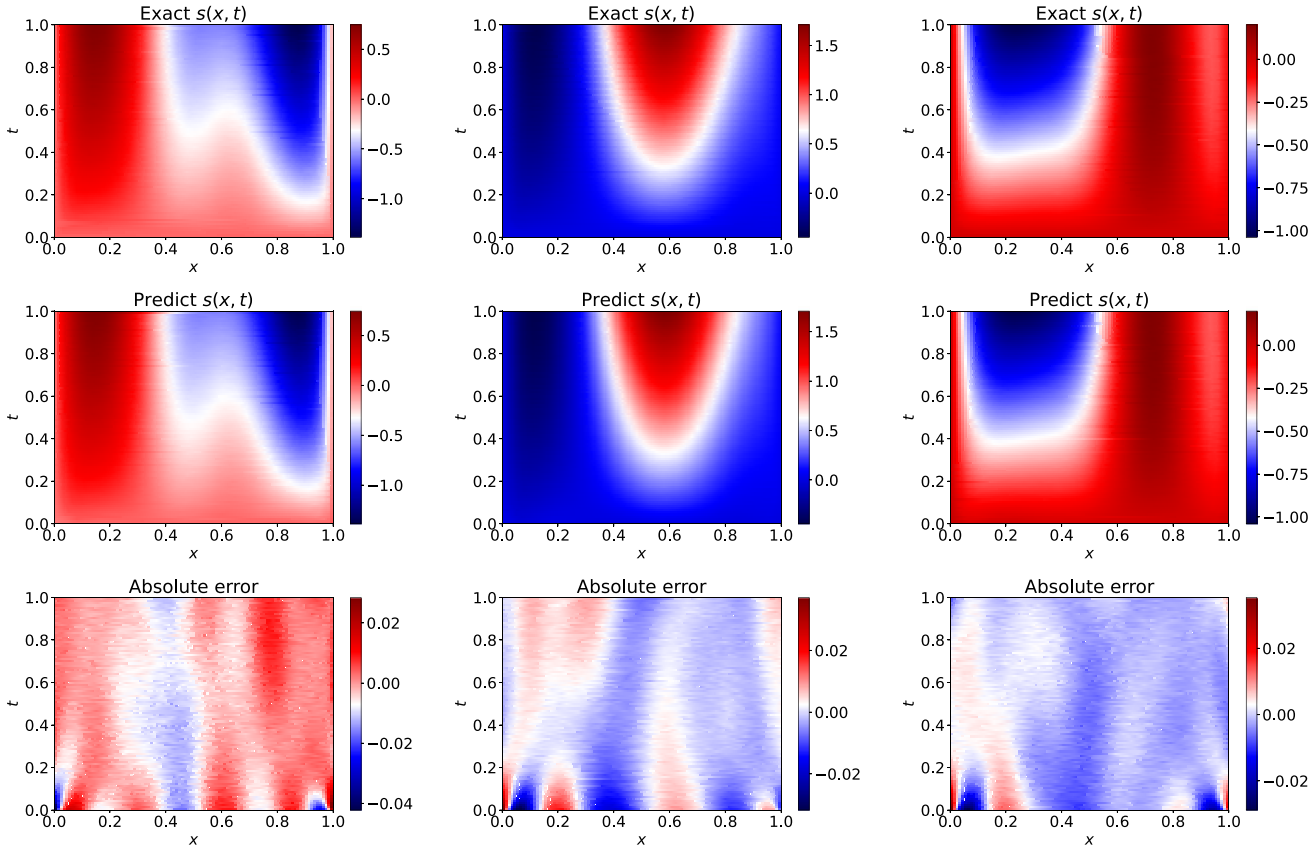

Fig. 4 Forward map - Comparison of the true PDE-output/solution s (given the PDE-input u) with the one predicted by the proposed invertible DeepONet and for Reaction-Diffusion PDE

approximation by multiplying the DeepONet expansion in Eq. (7) with the polynomial $x_1(1-x_1)\,x_2(1-x_2)$. We used $N_u = 1000$ unlabeled training data points with $N_{res} = 3844$ collocation points (Eq. (21)) during training and added either no labeled training data at all (i.e. $N_l = 0$) or $N_l = 1000$. In order to obtain the latter we solved Eq. (35) with the Finite Element library FEniCS [29] on a 128×128 mesh with linear elements and evaluated the solution at 3844 regularly distributed points. We represent the PDE-input u as follows[5]:

$$\begin{aligned}\ln(u) = \sum_{f_1=1}^{4}\sum_{f_2=1}^{4} & c_{f_1,f_2,1}\sin(f_1x_1)\cos(f_2x_2) \\ & + c_{f_1,f_2,2}\sin(f_1x_1)\sin(f_2x_2) \\ & + c_{f_1,f_2,3}\cos(f_1x_1)\sin(f_2x_2) \\ & + c_{f_1,f_2,4}\cos(f_1x_1)\cos(f_2x_2) \end{aligned} \tag{36}$$

using 64 feature functions and corresponding coefficients c. In order to generate the training data, we sampled each of the aforementioned 64 coefficients from a uniform distribution in $[0, 1]$. In this example the 64-dimensional vector of the c's serves as the input in the branch network (i.e. $D = 64$). With the help of the c's and of Eq. (36), one can reconstruct the full permeability field.

With regards to the network architecture, we employed a MLP with five layers and 64 Neurons each for the trunk network and 3 RealNVP building blocks for the branch network which were parametrized by a three-layered MLP. Using the ADAM optimizer and an initial learning rate of 10^{-3}, we run the model training for 10^5 iterations with an exponential learning rate decay with rate 0.9 every 2000 iterations. We tested the trained model on 2500 unseen test data and obtained the results in Table 4. As in the previous examples, the inclusion of labeled data significantly improves the predictive accuracy of the trained model. For the case without data the predictive accuracy of the forward map is slightly lower but the accuracy in the inverse map is comparably low. The addition of labeled data improves the predictive accuracy for both maps.

In Fig. 6 we compare the reference solution for two illustrative test cases with the the forward map learned with labeled training data. As suggested by the cumulative results in Table 4 the two predictions are very close to the reference

[5] We employ this expansion for the logarithm of u in order to ensure that the resulting permeability field is positive

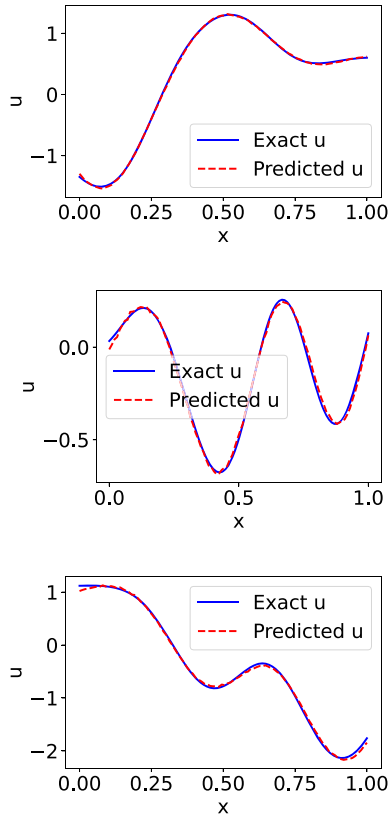

Fig. 5 Inverse map - Comparison of the true PDE-input u (given the PDE-output/solution s) with the one predicted by the proposed invertible DeepONet and for Reaction-Diffusion PDE

Table 4 Relative errors on test data depending on the amount of labeled training data for the Darcy example with feature coefficients as inputs

labeled data [%]	0	100
Relative error for s	0.0134 ± 0.00509	0.0245 ± 0.0108
Relative error for u	0.235 ± 0.137	0.0566 ± 0.0198

and the accuracy is very high. In Figs. 7 (without labeled training) and 8 (with labeled training) the results for two illustrative inverse test cases are shown. While locally the error can be significant, the main characteristics of the PDE-input field u can be captured.

We discuss in the next section the case where the input permeability field u is not represented with respect to some feature functions but rather as a discretized continuous field [13].

3.3.1 Coarse-grained (CG) input parameters

In this sub-case, we modeled the permeability field u with an exponentiated (to ensure positivity) Gaussian Process with mean zero and exponential quadratic covariance with length scale $\ell = 0.1$. The PDE was then again solved on a 128×128 FE mesh and the values of the solution s were assumed to be observed at 3844 regularly distributed points. We moreover sub-sampled the generated PDE input on a regular 8×8 grid and its $D = 64$ values represented the branch network input $\boldsymbol{u}$. We generated $N_u = 1000$ unlabeled fields u in total and used $N_{res} = 3844$ collocation points (Eq. (21)) during training. We also trained the model with $N_l = 1000$ labeled training data.

The results obtained can be found in Table 5. The test data in this table consists of 2500 unseen, discretized, permeability fields and their respective solutions. The error rates are computed with respect to the coarse-grained reference input. As in the previous setting, we observe a significant improvement in the accuracy of the inverse map when labeled data are used in training.

In Fig. 9 we compare the reference solution for two illustrative test cases with the the forward map learned with labeled training data. As suggested by the cumulative results in Table 5 the two predictions are very close to the reference and the accuracy is very high.

In Figs. 10 (without labeled training) and 11 (with labeled training) the results for two illustrative inverse test cases are shown. We note again that the main features of the PDE-input's spatial variability are captured, despite the presence of localized errors.

3.4 Bayesian inverse problems

In this section we demonstrate the utility of the invertible DeepONet proposed in the solution of Bayesian inverse problems and in obtaining accurate approximations of the posterior without any need for additional reference model runs nor for any costly and asymptotically-exact sampling. For each of the examples considered, only one observed output $\hat{\boldsymbol{s}}$ was assumed to be given. The variance of the observational noise σ^2 was assumed to be given although this could readily be inferred, especially if a conjugate inverse-Gamma prior was used for it. In this manner, any deviations from the actual posterior could be attributed to inaccuracies of the DeepONet-based surrogate. Errors due to the approximation of the prior with a mixture of Gaussians as in Eq. (27) can be made arbitrarily small by increasing the number of mixture components M.

3.4.1 Reaction-diffusion dynamics

We employed the trained model of the reaction-diffusion system (with 10% labeled training data), in combination with the formulation detailed in Sect. 2.5 for approximating the posterior. We use a prior $p_u(\boldsymbol{u})$ arising from the discretization of Gaussian Process with zero mean and exponential quadratic covariance kernel with a length scale $\ell = 0.2$. For the Gaussian mixture models involved for the prior and subsequently

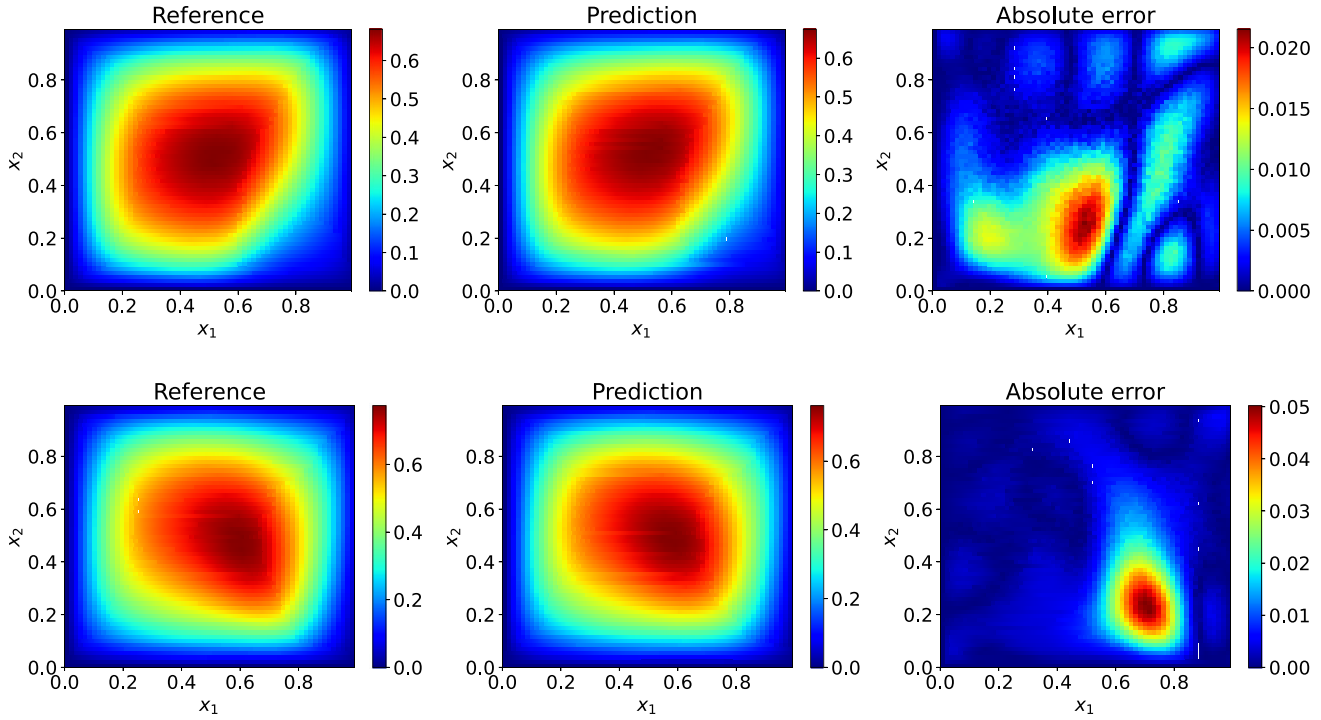

Fig. 6 Forward map - Comparison of the true PDE-output/solution s (given feature coefficients as the PDE-input u) with the one predicted by the proposed invertible DeepONet and for Darcy-type PDE

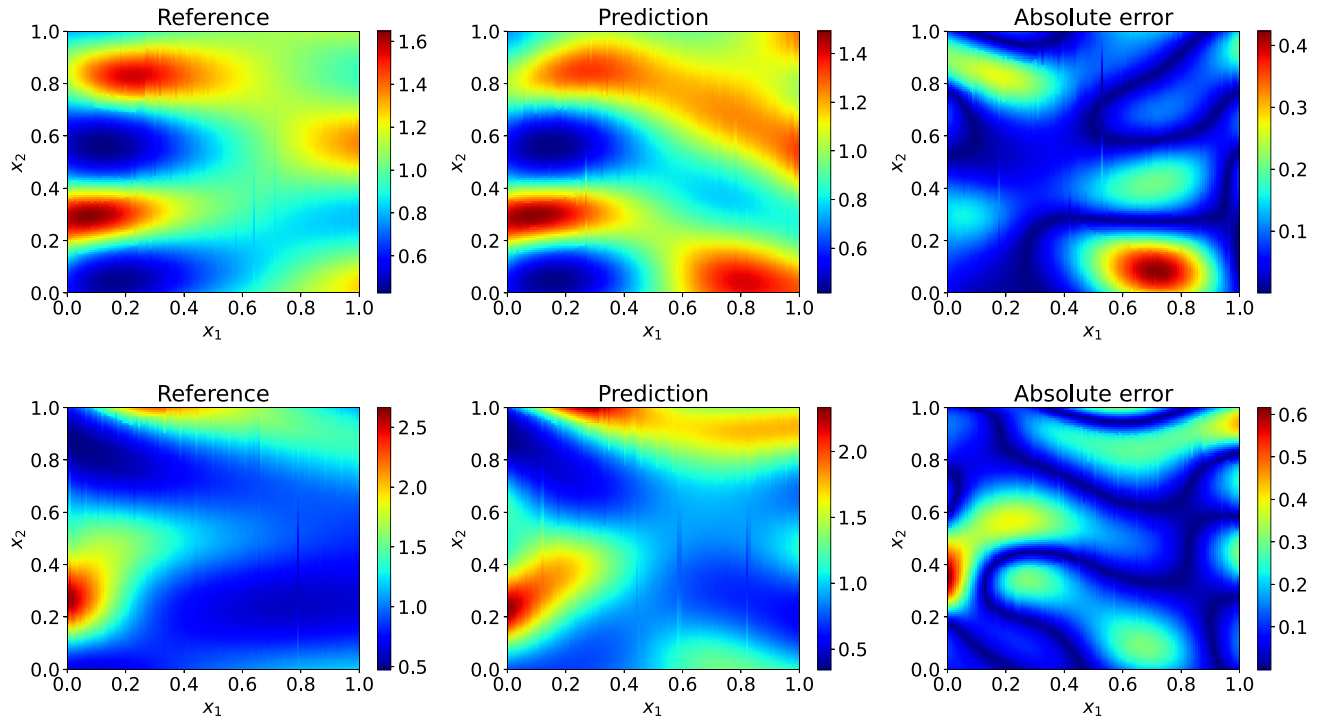

Fig. 7 Inverse map - Comparison of the reconstructed PDE-input (given the PDE-output/solution s) with the one predicted by the proposed invertible DeepONet and for Darcy-type PDE with zero labeled training data

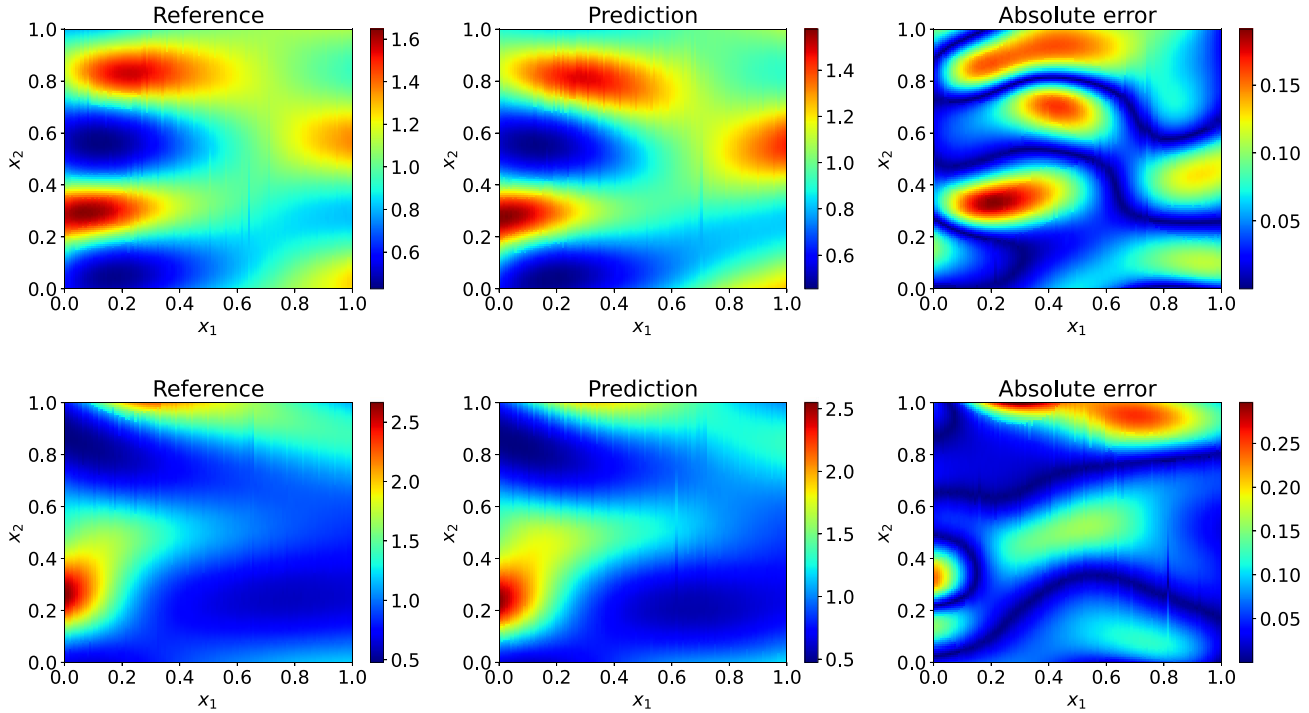

Fig. 8 Inverse map - Comparison of the reconstructed PDE-input (given the PDE-output/solution s) with the one predicted by the proposed invertible DeepONet and for Darcy-type PDE

Table 5 Relative errors on test data depending on the amount of labeled training data for the Darcy example with coarse-grained input parameters

labeled data [%]	0	100
Relative error for s	0.0164 ± 0.00712	0.0164 ± 0.00748
Relative error for u	0.121 ± 0.041	0.0656 ± 0.0168

the posterior on $\boldsymbol{b}$ we used two components i.e. $M = 2$ in Eqs. (27), (29). The results can be seen in the following Figures. The obtained posterior encapsulates the true parameter input for all three cases.

In Fig. 12 we used test cases with 100 observed solution data points for each parameter input and a noise level of $\sigma^2 = 0.001$ (see Eq. (23)). In Fig. 13 we increased the noise level ten-fold, to $\sigma^2 = 0.01$ and, as expected, so did the posterior uncertainty. In Fig. 14 we used $\sigma^2 = 0.001$ but decreased the number of observations of the PDE-solution to 25 points (instead of 100). As expected, this led to an increase in posterior uncertainty.

Our method can therefore be used as a fast approach without any need for optimization and MCMC sampling to generate an approximate posterior. We note that the posterior uncertainty increases if number of observations decreases or if the observation noise σ^2 increases. In Appendix B, we show the excellent agreement of the approximate posterior computed with the actual one as obtained by costly and time-consuming MCMC simulations.

3.4.2 Flow through porous media

We also solved a Bayesian inverse problem in the context of the Darcy-type PDE by using our trained model of Sect. 3.3 with added labeled training data. We computed an approximate posterior based on the algorithm presented in Sect. 2.5 and compared it with the true PDE-input. For the Gaussian mixture models involved for the prior and subsequently the posterior on $\boldsymbol{b}$ we used two mixture components i.e. $M = 2$ in Eqs. (27), (29).

Firstly, we considered permeability fields represented with respect to 64 known feature functions as described in Sect. 3.3. The 64 coefficients c (Eq. (36)) represented the sought PDE-inputs and a uniform prior in $[0, 1]^{64}$ was employed. The results in terms of the permeability field u can be seen in the following Figures. The obtained posterior is in good agreement with the ground truth, e.g. the PDE-input field used to generate the data with the PDE-solver.

In particular, in Fig. 15 we assumed that 3844 observations of the PDE-output were available, on a 62×62 regular grid. The data that was synthetically generated was contaminated with Gaussian noise with $\sigma^2 = 0.001$ (see Eq. (23)). In Fig. 16 we increased the noise level and subsequently the posterior uncertainty was slightly higher but the posterior

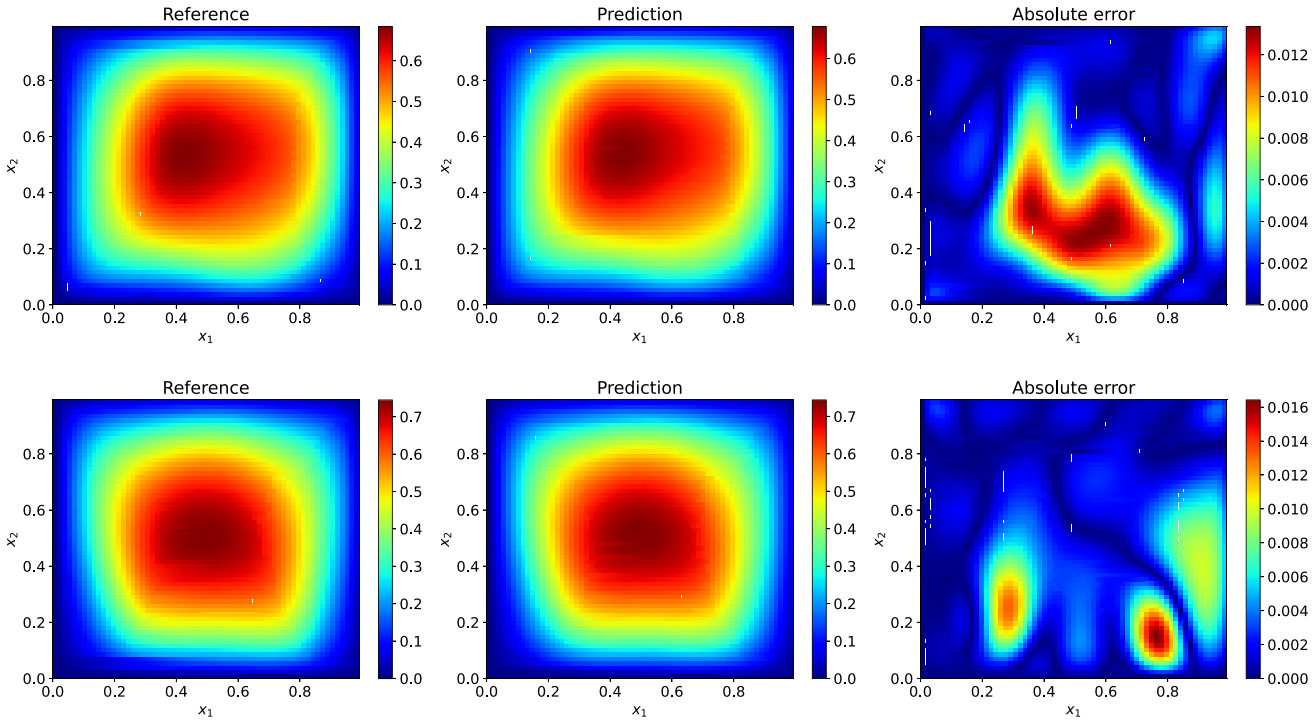

Fig. 9 Forward map - Comparison of the true PDE-output/solution s (given the coarse-grained PDE-input u) with the one predicted by the proposed invertible DeepONet and for Darcy-type PDE

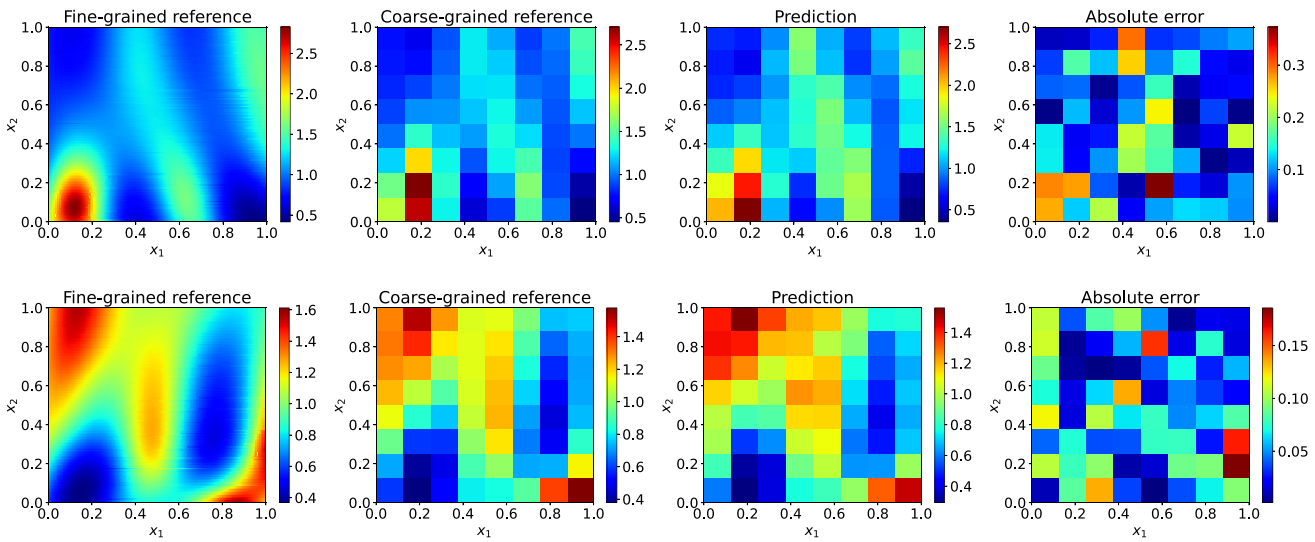

Fig. 10 Inverse map - Comparison of the coarse-grained PDE-input u (given the PDE-output/solution s) with the one predicted by the proposed invertible DeepONet and for Darcy-type PDE with zero labeled training data

mean is still close to the ground truth. In Fig. 17 we used $\sigma^2 = 0.01$ but decreased the number of observations by 50% to 1922. As expected, the posterior uncertainty increased again but still encapsulated the ground truth.

Finally, we considered the case where the PDE-input is represented on a regular 8×8 grid as in Sect. 3.3.1. The discretized GP described therein was used as the prior. In Fig. 18 we compare the ground truth with the posterior mean and standard deviation as obtained from 3844 observations on a 62×62 regular grid and for a noise level of $\sigma^2 = 0.01$ (see Eq. (23)). In Fig. 19 we used lower noise with $\sigma^2 = 0.001$ level and, as expected, the posterior uncertainty was lower and the posterior mean was closer to the ground truth. In Fig. 20 we again choose the previous noise level but decreased the number of observations by half, to 1922. As expected,

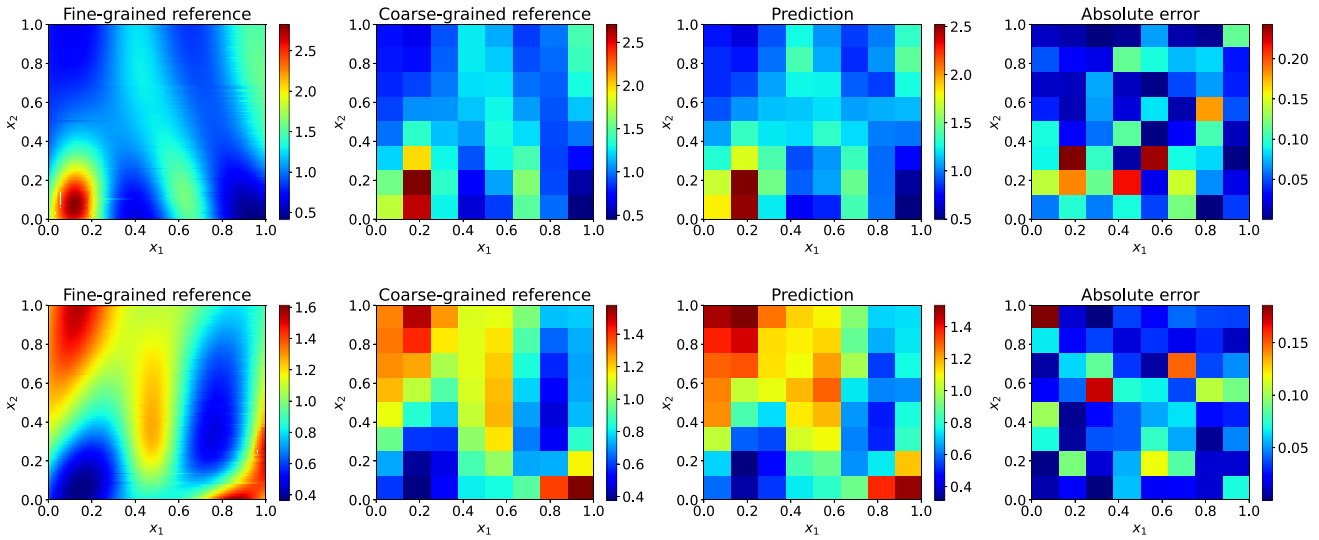

Fig. 11 Inverse map - Comparison of the coarse-grained PDE-input (given the PDE-output/solution s) with the one predicted by the proposed invertible DeepONet and for Darcy-type PDE

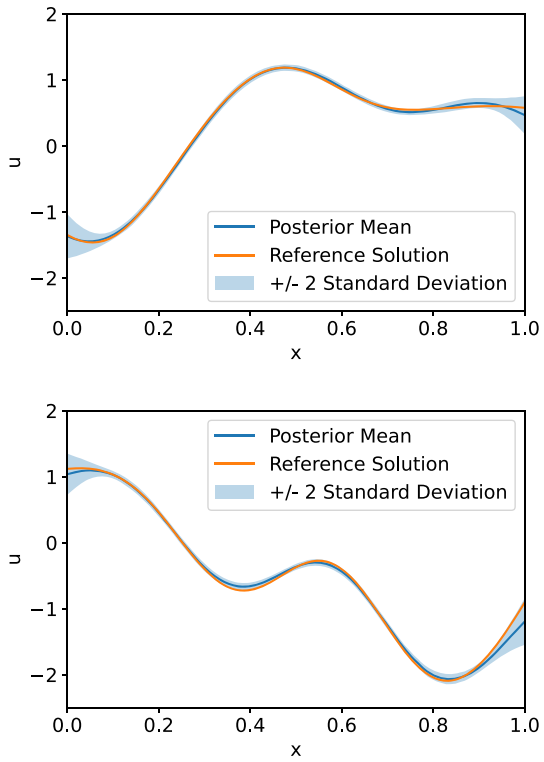

Fig. 12 Bayesian Inverse Problem for Reaction-Diffusion PDE: 100 observed data points with $\sigma^2 = 0.001$ and a 100-dimensional parametric input

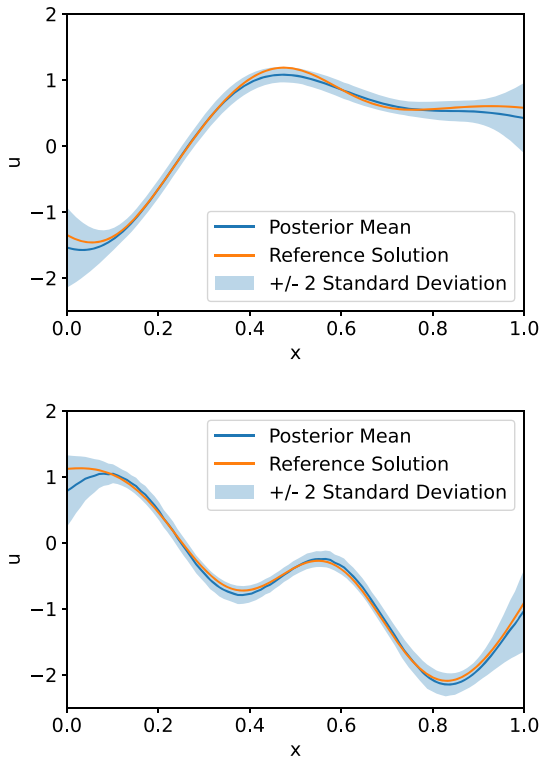

Fig. 13 Bayesian Inverse Problem for Reaction-Diffusion PDE: 100 observed data points with $\sigma^2 = 0.01$ and a 100-dimensional parametric input

the posterior uncertainty increased but still encapsulated the ground truth.

4 Conclusions

We introduced an invertible DeepONet architecture for constructing data-driven surrogates of PDEs with parametric inputs. The use of the RealNVP architecture in the

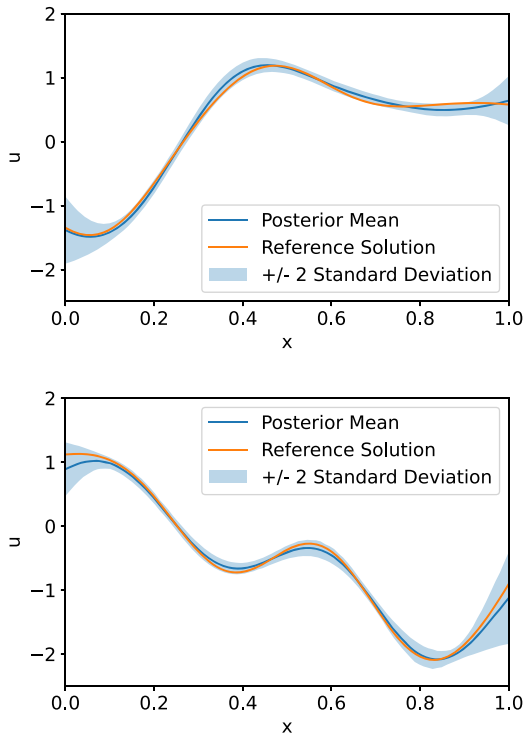

Fig. 14 Bayesian Inverse Problem for Reaction-Diffusion PDE: 25 observed data points with $\sigma^2 = 0.001$ and a 100-dimensional parametric input

branch-network enables one to obtain simultaneously accurate approximations of both the forward and the inverse map (i.e. from PDE-solution to PDE-input). The latter is particularly useful for deterministic and stochastic (Bayesian), PDE-based, inverse problems for which accurate solutions can be readily obtained once the proposed DeepONet has been trained offline. The training framework can make use of expensive, labeled data (i.e. PDE input-output pairs) as well as inexpensive, unlabeled data (i.e. only PDE-inputs) by incorporating residuals of the governing PDE and its boundary/initial conditions into the loss function. The use of labeled data was generally shown to improve predictive accuracy and especially in terms of the inverse map which is something that warrants further investigation.

In the case of Bayesian formulations in particular, we showed that the availability of the inverse map can lead to highly-efficient approximations of the sought posterior without the need of additional PDE solves and without any cumbersome sampling (e.g. due to MCMC, SMC) or iterations (e.g. due to SVI).

The performance of the proposed strategy was demonstrated on several PDEs with modest- to high-dimensional parametric inputs and its efficiency was assessed in terms of the amounts of labeled vs unlabeled data. Furthermore, the approximate posterior obtained was in very good agreement with the exact posterior obtained with the reference solver and MCMC. The accuracy persisted for various levels of noise in the data as well as when changing the number of available observations. We note finally that unbiased estimates with respect to the exact posterior could be readily obtained with Importance Sampling and by using the approximate posterior as the importance sampling density. This

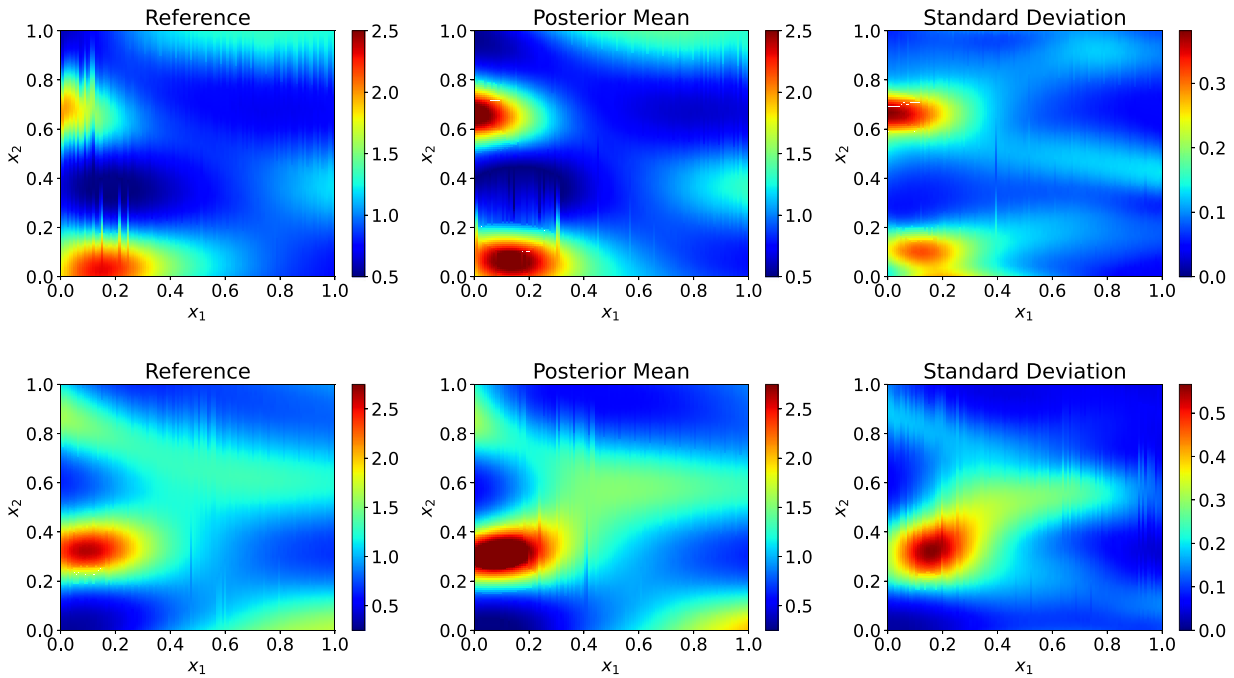

Fig. 15 Bayesian Inverse Problem for Darcy-type PDE: 3844 observed data points with $\sigma^2 = 0.001$ and a 64-dimensional parametric input representing feature coefficients

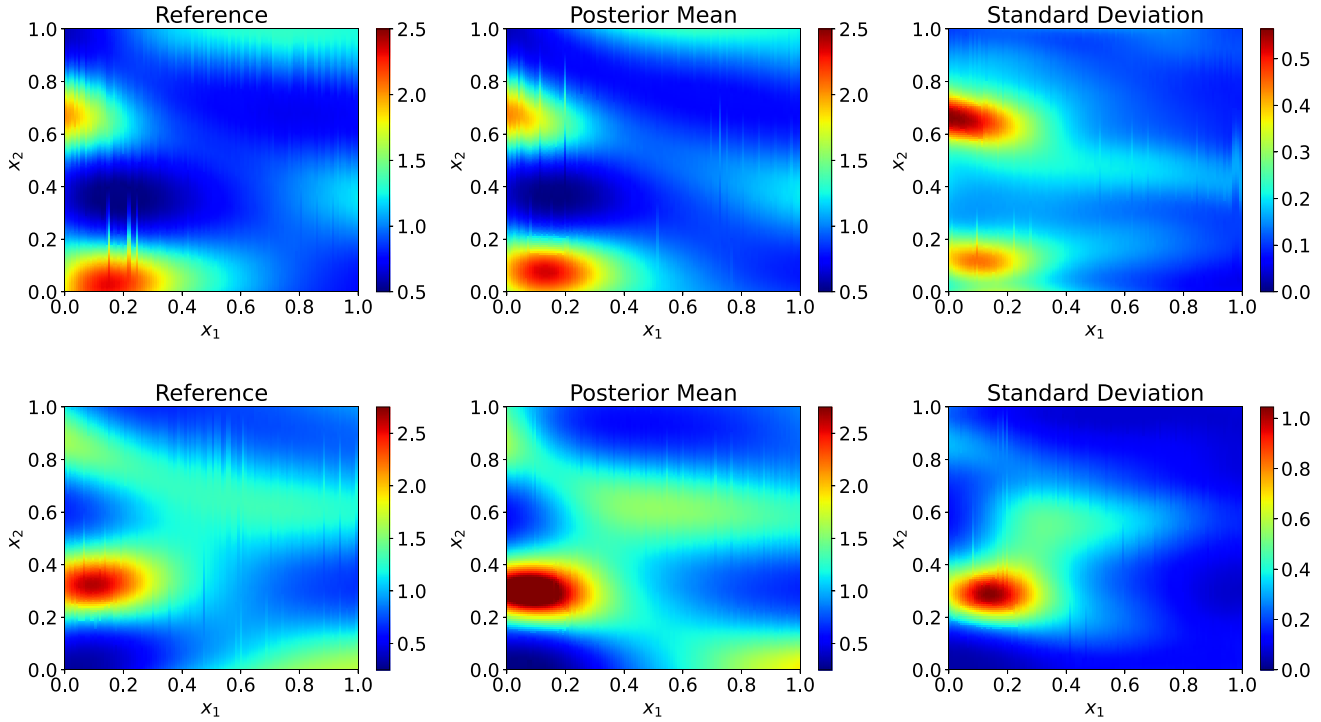

Fig. 16 Bayesian Inverse Problem for Darcy-type PDE: 3844 observed data points with $\sigma^2 = 0.01$ and a 64-dimensional parametric input representing feature coefficients

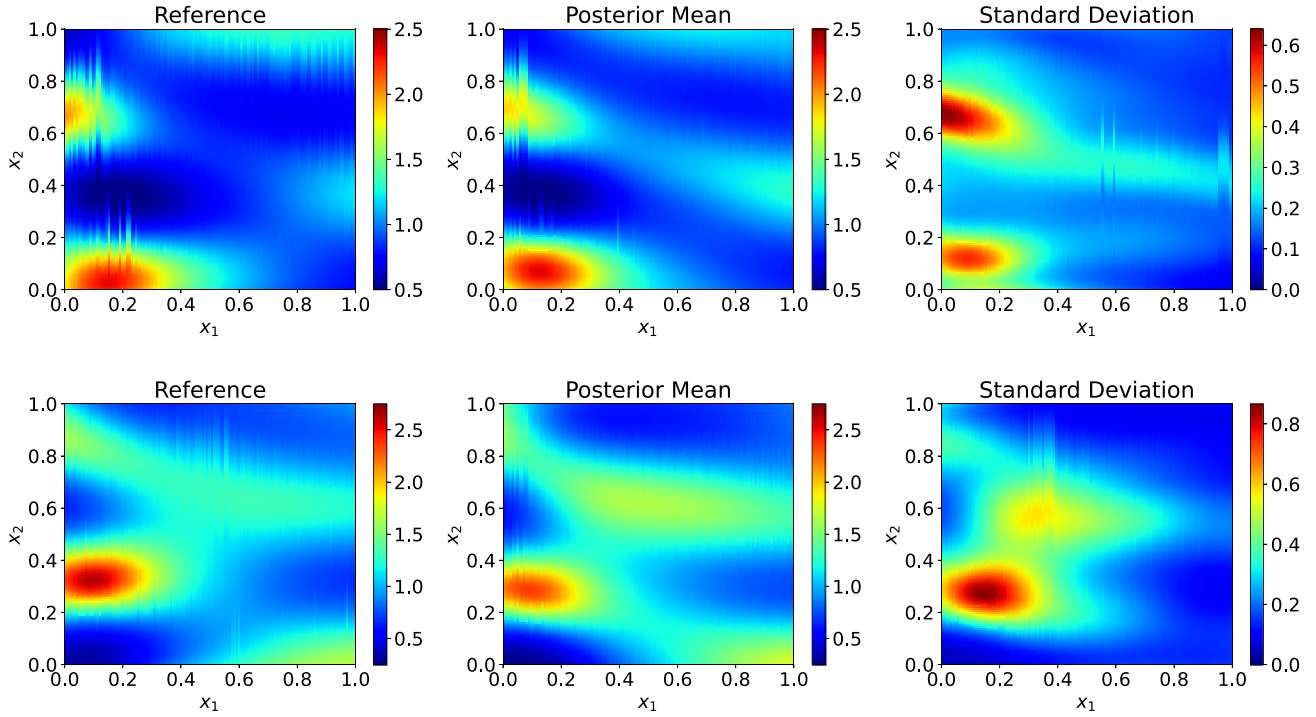

Fig. 17 Bayesian Inverse Problem for Darcy-type PDE: 1922 observed data points with $\sigma^2 = 0.01$ and a 64-dimensional parameter input representing feature coefficients

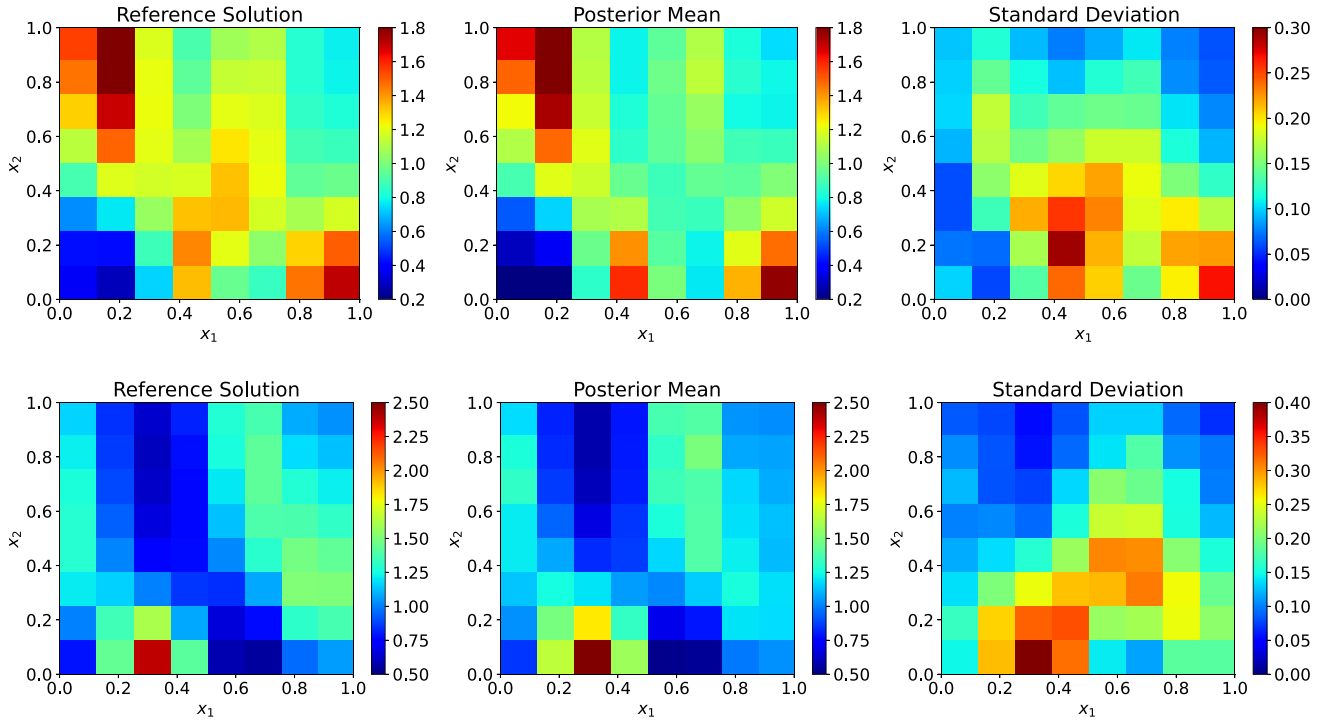

Fig. 18 Bayesian Inverse Problem for Darcy-type PDE: 3844 observations with $\sigma^2 = 0.01$ and a 64-dimensional, discretized permeability field

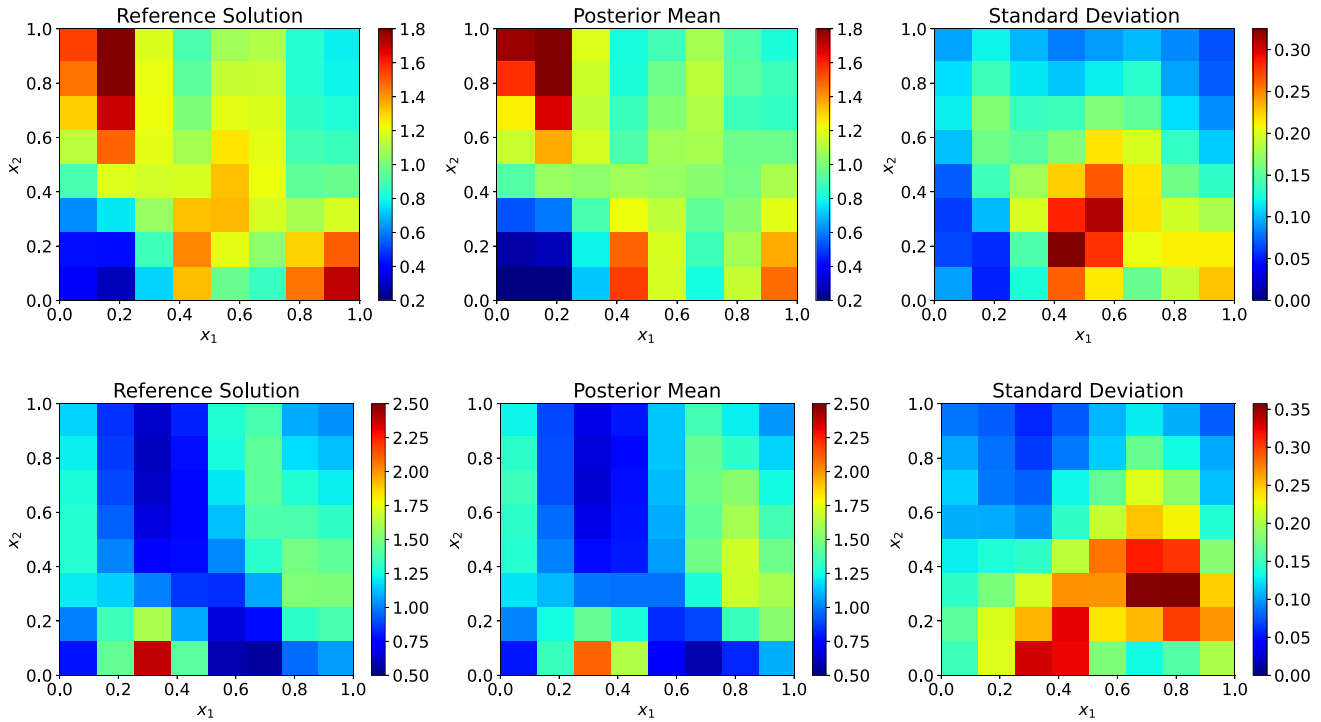

Fig. 19 Bayesian Inverse Problem for Darcy-type PDE: 3844 observations with $\sigma^2 = 0.001$ and a 64-dimensional discretized permeability field

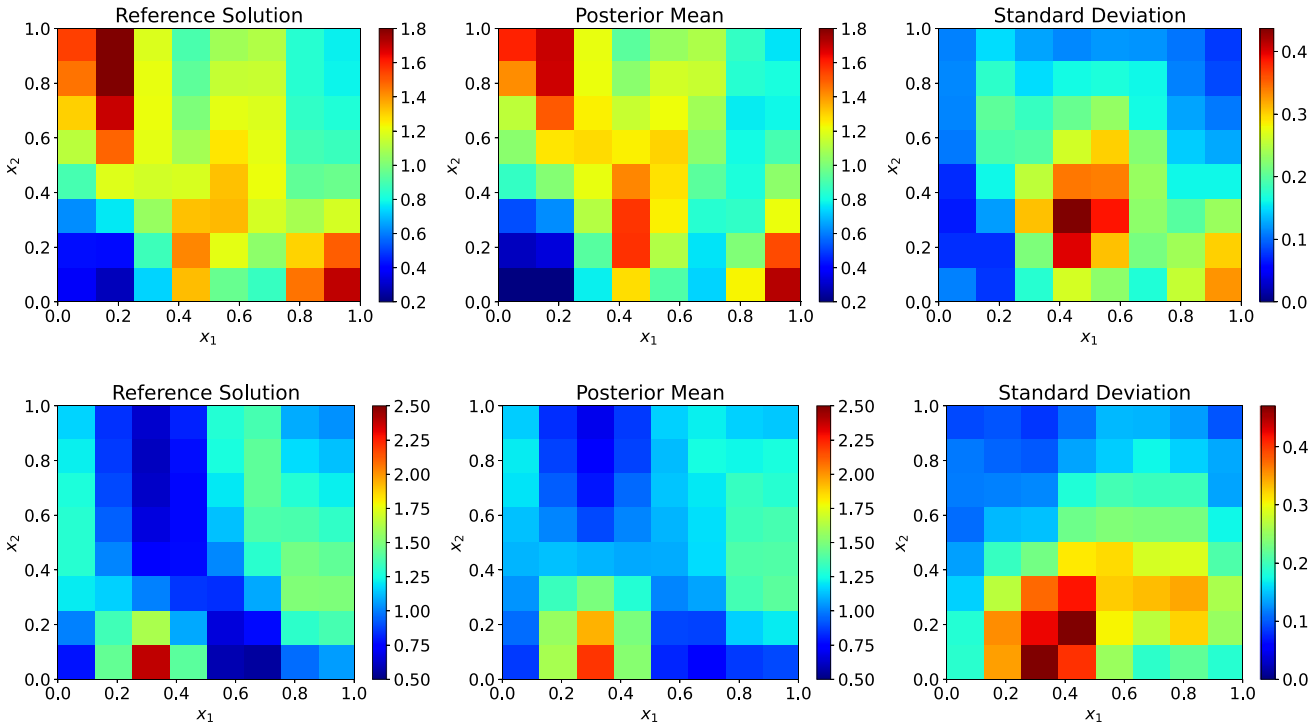

Fig. 20 Bayesian Inverse Problem for Darcy-type PDE: 1922 observations with $\sigma^2 = 0.01$ and a 64-dimensional discretized permeability field

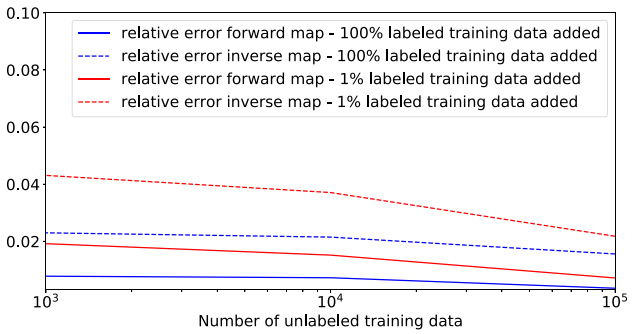

Fig. 21 Relative errors on test data for the forward and inverse map depending on the amount of labeled and unlabeled training data

would nevertheless imply additional PDE solves which we would expect to be modest in number given the accuracy of the approximation i.e. the proximity of the Importance Sampling density with the actual posterior.

Funding Open Access funding enabled and organized by Projekt DEAL.

Appendix: A influence of the amount of data

This section contains additional results as obtained for the antiderivative example and for different amounts of training data. We chose exactly the same settings as described in Sect. 3.1 and varied only the amount of labeled and unlabeled training data. In Fig. 21 we plot the relative error in the foward and inverse map with regards to the amount of unlabeled training data. The color indicates the amount of labeled training data used, i.e. blue curves correspond to 1% labeled training data, whereas red curves correspond to 100% labeled training data.

We observe that although the relative errors decrease with the addition of more data, the benefit is more pronounced with the addition of labeled data.

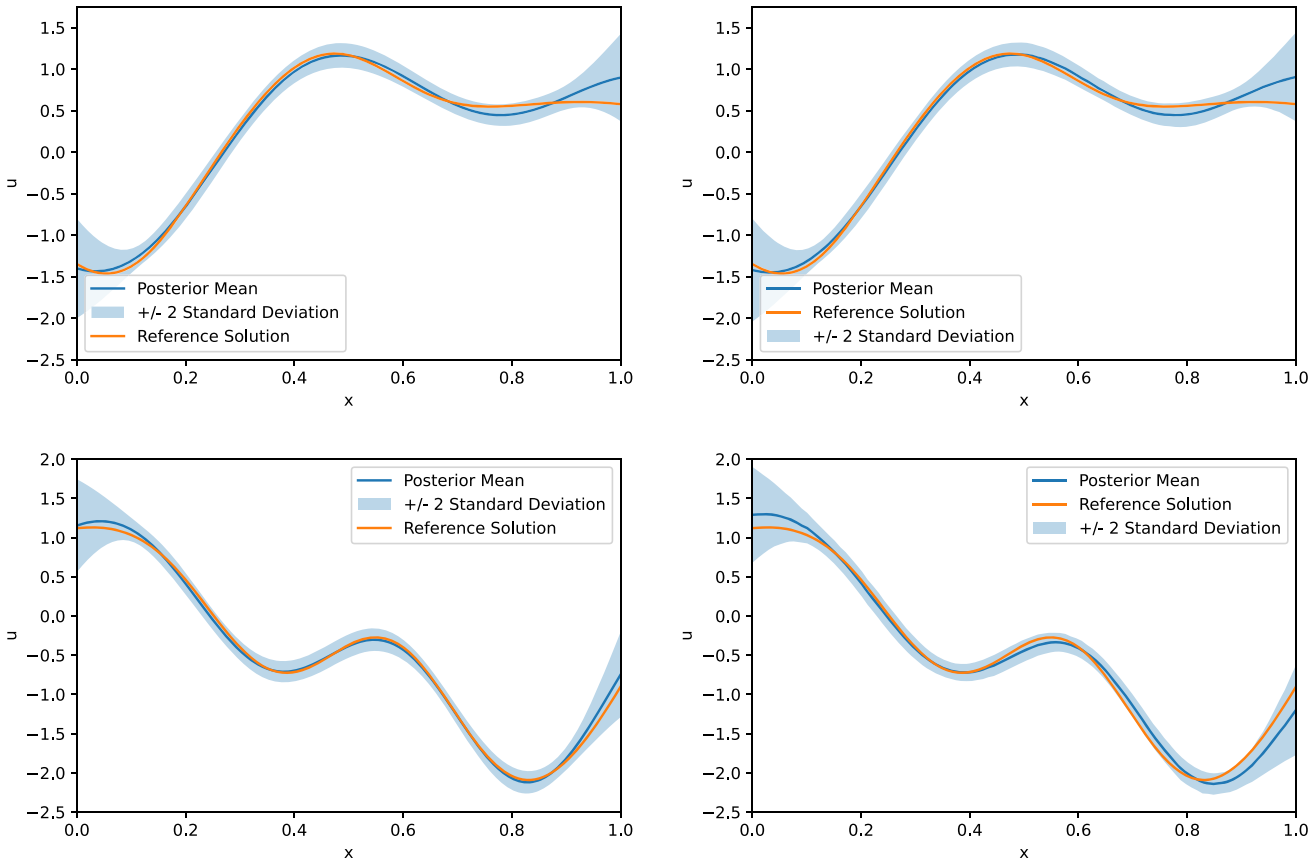

Fig. 22 Bayesian Inverse Problem for Reaction-Diffusion PDE: 100 observed data points with $\sigma^2 = 0.01$ and a 100-dimensional parameter input. Left: Posterior based on MCMC (NUTS), Right: Posterior obtained by our algorithm

B Comparison with MCMC

In the main part of this article we already showed that the true parameter input is encapsulated by the posterior. In this section we compare the approximate posterior computed with the reference posterior obtained by MCMC.

In particular, for two, randomly-chosen cases in the reaction-diffusion example, the true posterior was computed using the NUTS sampler from the Blackjax library [26]. As is the case with all MCMC-based inference schemes, these provide the reference posterior (asymptotically). The results shown in Fig. 22 in terms of the posterior *mean* $\pm$ 2 posterior standard deviations indicate excellent accuracy of the posterior approximation proposed. While our method does not require any new forward model evaluation or model gradients, the MCMC algorithms require a forward model solve and its gradients for each sample. For the MCMC-based results displayed in total 40000 samples were generated.

References

1. Adler J, Öktem O (2017) Solving ill-posed inverse problems using iterative deep neural networks. Inverse Problems 33(12):124,007
2. Ardizzone L, Kruse J, Wirkert S, et al. (2018) Analyzing inverse problems with invertible neural networks. arXiv preprint arXiv:1808.04730
3. Behrmann J, Grathwohl W, Chen RTQ, et al. (2019) Invertible residual networks. ICML
4. Beskos A, Girolami M, Lan S et al (2017) Geometric MCMC for infinite-dimensional inverse problems. J Comput Phys 335:327–351. https://doi.org/10.1016/j.jcp.2016.12.041 (www.sciencedirect.com/science/article/pii/S0021999116307033)
5. Bhattacharya K, Hosseini B, Kovachki NB, et al. (2020) Model reduction and neural networks for parametric pdes. arXiv preprint arXiv:2005.03180
6. Bishop CM, Nasrabadi NM (2006) Pattern recognition and machine learning. Springer, Berlin
7. Bradbury J, Frostig R, Hawkins P, et al. (2018) JAX: composable transformations of Python+NumPy programs. http://github.com/google/jax
8. Champion KP, Brunton SL, Kutz JN (2019) Discovery of nonlinear multiscale systems: sampling strategies and embeddings. SIAM J Appl Dyn Syst 18(1):312–333
9. Detommaso G, Cui T, Marzouk Y, et al. (2018) A Stein variational Newton method. In: Advances in Neural Information Processing Systems, pp 9169–9179

10. Dinh L, Sohl-Dickstein J, Bengio S (2016) Density estimation using real nvp. arXiv preprint arXiv:1605.08803
11. Franck IM, Koutsourelakis PS (2017) Multimodal, high-dimensional, model-based, Bayesian inverse problems with applications in biomechanics. J Comput Phys 329:91–125. https://doi.org/10.1016/j.jcp.2016.10.039 (www.sciencedirect.com/science/article/pii/S002199911630537X)
12. Gin C, Lusch B, Brunton SL, et al. (2019) Deep learning models for global coordinate transformations that linearize pdes. arXiv preprint arXiv:1911.02710
13. Grigo C, Koutsourelakis P-S (2019) Bayesian model and dimension reduction for uncertainty propagation: applications in random media. SIAM/ASA J. Uncertain. Quantif 7(1):292–323
14. Kalia M, Brunton SL, Meijer HG, et al. (2021) Learning normal form autoencoders for data-driven discovery of universal, parameter-dependent governing equations. arXiv preprint arXiv:2106.05102
15. Kaltenbach S, Koutsourelakis PS (2020) Incorporating physical constraints in a deep probabilistic machine learning framework for coarse-graining dynamical systems. J Comput Phys 419(109):673
16. Kaltenbach S, Koutsourelakis PS (2021) Physics-aware, probabilistic model order reduction with guaranteed stability. ICLR
17. Karnakov P, Litvinov S, Koumoutsakos P (2022) Optimizing a discrete loss (odil) to solve forward and inverse problems for partial differential equations using machine learning tools. arXiv preprint arXiv:2205.04611
18. Karniadakis GE, Kevrekidis IG, Lu L et al (2021) Physics-informed machine learning. Nat Rev Phy 3(6):422–440. https://doi.org/10.1038/s42254-021-00314-5 (www.nature.com/articles/s42254-021-00314-5, number: 6 Publisher: Nature Publishing Group)
19. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980
20. Kissas G, Seidman J, Guilhoto LF, et al. (2022) Learning operators with coupled attention. arXiv preprint arXiv:2201.01032
21. Klus S, Nüske F, Koltai P et al (2018) Data-driven model reduction and transfer operator approximation. J Nonlinear Sci 28(3):985–1010. https://doi.org/10.1007/s00332-017-9437-7
22. Koopman BO (1931) Hamiltonian systems and transformations in Hilbert Space. Proceedings of the National Academy of Sciences of the United States of America 17(5):315–318. https://www.jstor.org/stable/86114
23. Koutsourelakis P (2009) A multi-resolution, non-parametric, Bayesian framework for identification of spatially-varying model parameters. J Comput Phys 228(17):6184–6211
24. Koutsourelakis P, Zabaras N, Girolami M (2016) Big data and predictive computational modeling. JCoPh 321:1252-1254
25. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000
26. Lao J, Louf R (2020) Blackjax: A sampling library for JAX. http://github.com/blackjax-devs/blackjax
27. Lee K, Carlberg KT (2020) Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J Comput Phys 404(108):973
28. Li Z, Kovachki N, Azizzadenesheli K, et al. (2020) Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895
29. Logg A, Mardal KA, Wells G (2012) Automated solution of differential equations by the finite element method: The FEniCS book, vol 84. Springer Science & Business Media
30. Lu L, Jin P, Karniadakis GE (2019) Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. arXiv preprint arXiv:1910.03193
31. Lu L, Jin P, Pang G et al (2021) Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature Mach Intell 3(3):218–229
32. Mo S, Zabaras N, Shi X, et al. (2019) Deep Autoregressive Neural Networks for High-Dimensional Inverse Problems in Groundwater Contaminant Source Identification. Water Resources Research 55(5):3856–3881. https://doi.org/10.1029/2018WR024638, https://onlinelibrary.wiley.com/doi/abs/10.1029/2018WR024638
33. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
34. Sander ME, Ablin P, Blondel M, et al (2021) Momentum residual neural networks. arXiv:2102.07870
35. Wainwright M, Jordan M (2008) Graphical models, exponential families, and variational inference. Foundations and Trends in Mach Learn 1:1–305
36. Wang S, Wang H, Perdikaris P (2021) Learning the solution operator of parametric partial differential equations with physics-informed deeponets. arXiv preprint arXiv:2103.10974