Journal of Computational Physics 473 (2023) 111713

Contents lists available at ScienceDirect

Journal of Computational Physics

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

B-DeepONet: An enhanced Bayesian DeepONet for solving noisy parametric PDEs using accelerated replica exchange SGLD

Guang Lin [a,b,*], Christian Moya [b], Zecheng Zhang [b]

[a] Department of Mechanical Engineering, Purdue University, 610 Purdue Mall, West Lafayette, 47907, IN, USA
[b] Department of Mathematics, Purdue University, 610 Purdue Mall, West Lafayette, 47907, IN, USA

ARTICLE INFO

Article history:
Received 13 March 2022
Received in revised form 11 September 2022
Accepted 14 October 2022
Available online 26 October 2022

Keywords:
Parametric PDE
Deep operator network
Noisy data
Bayesian method
Langevin diffusion
Replica exchange

ABSTRACT

The Deep Operator Network (DeepONet) is a neural network architecture used to approximate operators, including the solution operator of parametric PDEs. DeepONets have shown remarkable approximation ability. However, the performance of DeepONets deteriorates when the training data is polluted with noise, a scenario that occurs in practice. To handle noisy data, we propose a Bayesian DeepONet based on replica exchange Langevin diffusion (reLD). Replica exchange uses two particles. The first particle trains a DeepONet to exploit the loss landscape and make predictions. The other particle trains a different DeepONet to explore the loss landscape and escape local minima via swapping. Compared to DeepONets trained with state-of-the-art gradient-based algorithms (e.g., Adam), the proposed Bayesian DeepONet greatly improves the training convergence for noisy scenarios and accurately estimates the uncertainty. To further reduce the high computational cost of the reLD training of DeepONets, we propose (1) an accelerated training framework that exploits the DeepONet's architecture to reduce its computational cost up to 25% without compromising performance and (2) a transfer learning strategy that accelerates training DeepONets for PDEs with different parameter values. Finally, we illustrate the effectiveness of the proposed Bayesian DeepONet using four parametric PDE problems.

1. Introduction

Many problems in Science and Engineering require solving parametric PDEs [1,19,16,28]. That is, they require the repeated evaluation of a PDE model for a given distribution of inputs. For example, in petroleum engineering applications, engineers seek to calculate, using the porous media equations, the pressure field of the oil based on its permeability. In practice, the value of the oil permeability varies frequently; hence, one needs to calculate the oil pressure field for a distribution of fast-varying permeabilities [28,9,11]. The traditional numerical frameworks require intense computations to solve

* Corresponding author.
E-mail addresses: guanglin@purdue.edu (G. Lin), cmoyacal@purdue.edu (C. Moya), zecheng.zhang.math@gmail.com (Z. Zhang).

https://doi.org/10.1016/j.jcp.2022.111713
0021-9991/

these parametric PDEs. Furthermore, the computational cost for these traditional frameworks increases when the problem is time-dependent and multiscale [10,6,15].

To reduce the computational cost of solving parametric PDEs, many works have proposed using deep neural networks, which transfer most of the cost to offline training. However, in practice, the data required to train these networks is often polluted with noise or numerical errors. Deep neural networks often have poor generalization performance in these noisy scenarios because the gradient-based algorithms used to train the networks tend to chase the noise. As a result, the trained model will fail if used in practice.

This work focuses on Deep Operator Networks (DeepONet) [24], a fundamentally different class of neural networks that learns to approximate nonlinear operators, i.e., mappings between infinite-dimensional function spaces. DeepONets are constructed to realize and exploit the power of the universal approximation theorem of nonlinear operators [3]. To that end, a DeepONet uses two sub-networks: the Branch Net, which encodes the input function at a fixed number of locations, and the Trunk Net, which encodes the locations where one evaluates the output function. Such an architecture enables DeepONets to learn nonlinear operators efficiently from relatively small datasets. The remarkable capabilities of DeepONets have been reported in many application areas, including electroconvection [2], chemistry [22], economics [21], and for solving parametric PDEs [24]. These previous works, however, have trained DeepONets using data without noise. In real applications, data is often polluted with noise. So, a framework that can handle noisy data during DeepONets' training must be developed.

Many works have studied the problem of dealing with noisy datasets and avoiding overfitting. Among these works, Bayesian frameworks that use the Langevin diffusion [25,4] have provided promising results. For example, in [25], the authors introduced the Stochastic Gradient Langevin diffusion algorithm to train neural networks. They demonstrated that the samples generated by their proposed algorithm converge to the target posterior distribution. Replica exchange MCMC (and its stochastic gradient variants) [23,5,12] were proposed to accelerate the Langevin diffusion. In replica exchange frameworks, one uses two particles with different temperature parameters to sample from the posterior distribution. The high-temperature particle explores the landscape of the loss function, which enables escaping from local minima, while the low temperature exploits the same landscape, enforcing local convergence [18,27]. The two particles can be exchanged according to a swapping probability. If this swapping probability is designed so that the Markov process is reversible, then the low-temperature particle may attain the global optima faster.

As a result, these replica exchange frameworks provide excellent convergence performance. They, however, double the computational cost of each training iteration. Hence, one needs to balance having fast convergence (i.e., fewer iteration steps) and having high per-iteration computational cost. To tackle such a problem, in our previous work [23], we proposed the accelerated multi-variance stochastic gradient Langevin diffusion (m-reSGLD) framework, which assumes that the estimators for the two particles have different accuracy.

In this work, our objective is to develop a Bayesian DeepONet framework to approximate the solution operator of parametric PDEs using noisy measurements. To this end, we develop a replica exchange stochastic gradient Langevin diffusion (reSGLD) algorithm tailored for training the Bayesian DeepONets. The reSGLD algorithm greatly improves the performance of the DeepONet when compared to classical gradient-based training methods, e.g., Adam. Furthermore, we design an accelerated training regime based on reSGLD for DeepONets. Such a training regime results in a multi-variance replica exchange SGLD (m-reSGLD) algorithm that reduces the cost of reSGLD. Finally, we propose a transfer learning strategy to accelerate the use of reSGLD on PDEs with different parameter values. We summarize the contributions of this work next.

- We design a Bayesian DeepONet to approximate the solution operator of parametric PDEs in scenarios where the training data is polluted with noise. To the best of the authors' knowledge, this is the first paper that addresses such a problem. In the proposed framework, the Bayesian DeepONet represents the prior for the trainable parameters, while replica exchange Stochastic Gradient Langevin Diffusion (reSGLD) enables estimating the posterior, which we use to estimate the uncertainty of DeepONet predictions.
- We demonstrate that the replica exchange algorithm's ability to escape local minima (using one particle to explore and another particle to exploit the DeepONet's loss function landscape) enables the proposed framework to provide improved training convergence in noisy scenarios when compared to vanilla DeepONets trained with gradient-based optimization algorithms (e.g., Adam). Remarkably, the proposed framework also provides a more accurate mean predictive performance than vanilla DeepONets.
- We then propose a novel accelerated training regime for Bayesian DeepONets. This regime can reduce the computational cost of the replica exchange algorithm up to 25% without compromising the DeepONet's predictive performance. More precisely, for the particle that explores the loss function landscape, we randomly choose to train one of the DeepONet's sub-networks (the Branch Net or the Trunk Net) while keeping the other fixed. Empirical results verify our estimate.
- A transfer learning strategy, which exploits the structure of the DeepONet, is then proposed to accelerate the training further. More specifically, we exploit the DeepONet's architecture again to design a transfer learning that disseminates previous reSGLD training results to (1) other datasets with smaller noise and (2) other PDEs with different parameter values. We will show that the proposed transfer learning strategy can reach the same predictive performance as direct training with less training cost.
- Finally, we test the effectiveness of the proposed Bayesian DeepONet using four parametric PDE problems. In particular, we compare the training convergence speed, mean predictive accuracy, and uncertainty quantification performance

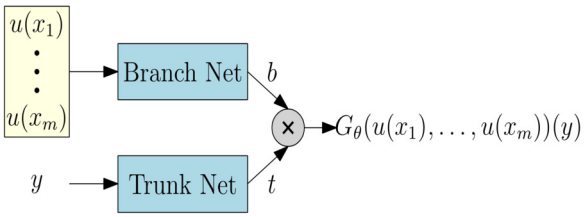

Fig. 1. DeepONet architecture. The crossed node on the right indicates the DeepONet output, which we obtain by taking the inner product between the output features of the Branch (b) and Trunk (t) Nets.

of reSGLD, the proposed accelerated multi-variance reSGLD (denoted as m-reSGLD), and Adam with dropout (a non-Bayesian framework for uncertainty quantification).

The rest of the paper is organized as follows. In Section 2, we provide a brief review of Deep Operator Networks (DeepONet). Section 3 introduces the Bayesian setting of our problem and details the replica exchange Stochastic Gradient Langevin diffusion (reSGLD) algorithm. We present the proposed accelerated replica exchange algorithm and the transfer learning strategy in Section 4. In Section 5, we test the performance of the proposed Bayesian framework using a series of experiments on four parametric PDE problems. Finally, Section 6 concludes this work.

2. Background information

In this work, we propose a Bayesian data-driven deep learning framework to approximate the solution operator of the prototypical parametric PDE

$$(\mathcal{L}_a s)(x) = f(x), \qquad x \in D, \tag{1}$$

where $D \subset \mathbb{R}^d$ is a bounded open set, $a : D \to \mathbb{R}$ is a parameter entering the definition of the nonlinear operator $\mathcal{L}_a$, and $s : D \to \mathbb{R}$ is the solution to the PDE (given appropriate boundary conditions). To approximate the solution operator, we use the deep operator neural network (DeepONet) framework introduced in [24], which we review next.

2.1. Review of DeepONet

Let $G^\dagger$ denote the nonlinear operator arising as the solution operator of the parametric PDE (1). This operator $G^\dagger$ maps an input function u (corresponding, for example, to the parameter a, the forcing f, or initial/boundary conditions) to an output function $G^\dagger(u)$ (corresponding to the solution s of the parametric PDE). Let $y \in Y$ denote a point in the output function domain $Y \subset \mathbb{R}^d$. Then, the goal of the DeepONet G_θ, with trainable parameters $\theta \in \mathbb{R}^p$, is to approximate the operator $G^\dagger(u)(y)$ at $y \in Y$. To this end, the DeepONet G_θ uses the architecture depicted in Fig. 1, consisting of two sub-networks referred to as the Branch Net and Trunk Net.

The *Branch* Net processes the input function information. Let $(x_1, \ldots, x_m)$ denote points in the domain of u (we refer to these points as the *sensors*), such that $(u(x_1), \ldots, u(x_m))$ is a discrete representation of the input u. The Branch Net takes this discretized u as the input and outputs a vector of features $b \in \mathbb{R}^q$. On the other hand, the *Trunk* Net processes the points in the domain Y of the output function. To this end, the Trunk Net takes $y \in Y$ as the input and outputs a vector of features $t \in \mathbb{R}^q$. Note that since the Trunk Net's output t solely depends on the input coordinates y, it is natural to interpret the components of t as a collection of basis functions defined on Y, i.e.,

$$t = (\varphi_1(y), \ldots, \varphi_q(y)).$$

The output of the DeepONet then combines the output features from the Branch Net b and the Trunk Net t using the inner product:

$$G_\theta\left(u(x_1), \ldots, u(x_m)\right)(y) := \langle b, t \rangle = \sum_{i=1}^{q} b_i \cdot \varphi_i(y). \tag{2}$$

From the above, one can interpret the output of the Branch Net b as the trainable coefficients for the basis functions t produced by the Trunk Net. To simplify our notation, in the rest of this work, we omit writing the DeepONet explicit dependency on the discretized input and use the simplified notation $G_\theta(u)(y)$.

Finally, we train the DeepONet G_θ to approximate the nonlinear solution operator $G^\dagger$ by minimizing a mean square loss function on the training dataset $\left\{u_i, y_i, G^\dagger(u_i)(y_i)\right\}_{i=1}^N$, i.e.,

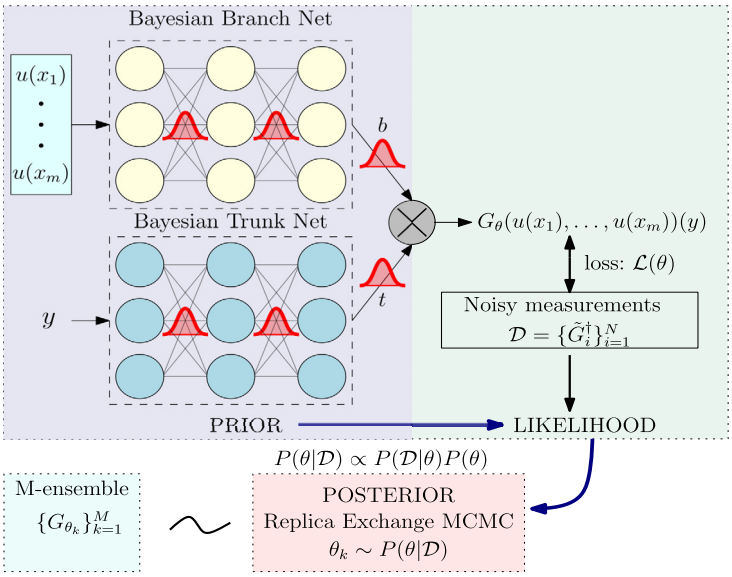

Fig. 2. The Bayesian DeepONet framework. The Bayesian DeepONet represents the prior of the trainable parameters θ. One computes the likelihood using noisy measurements $\mathcal{D} = \{\tilde{G}_i^\dagger\}_{i=1}^N$. Replica Exchange MCMC enables sampling from the posterior $\theta_k \sim P(\theta|\mathcal{D})$. The sampled parameters are used to build the DeepONet M-ensemble $\{G_{\theta_k}\}_{k=1}^M$ (see Fig. 3) for prediction and uncertainty quantification.

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left|G_\theta(u_i)(y_i) - G^\dagger(u_i)(y_i)\right|^2. \tag{3}$$

In this work, we aim to design a Bayesian DeepONet that enables us to (1) better approximate $G^\dagger$ in a noisy scenario and (2) estimate the solution/prediction uncertainty. We refer to this framework as the *Bayesian DeepONet*.

3. The Bayesian DeepONet

To derive the proposed Bayesian DeepONet framework depicted in Fig. 2, we consider the scenario when our available operator training targets correspond to scattered noisy measurements of $G^\dagger(u)(y)$, i.e., $\{\tilde{G}_i^\dagger\}_{i=1}^N$. In particular, we assume these measurements are independently Gaussian distributed centered at the latent true operator target value, i.e.,

$$\tilde{G}_i^\dagger = G^\dagger(u_i)(y_i) + \epsilon_i, \qquad i = 1, 2, \ldots, N.$$

Here, ϵ_i is an independent Gaussian noise with zero mean and *known* standard deviation σ, i.e., $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Let $\mathcal{D} := \{\tilde{G}_i^\dagger\}_{i=1}^N$ denote the noisy dataset of targets; we can then calculate the likelihood as:

$$P(\mathcal{D}|\theta) = \prod_{i=1}^{N} P(\tilde{G}_i^\dagger|\theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(G_\theta(u_i)(y_i) - \tilde{G}_i^\dagger)^2}{2\sigma^2}\right). \tag{4}$$

To compute the posterior distribution, we use Bayes' theorem, i.e.,

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})} \propto P(\mathcal{D}|\theta)P(\theta),$$

where "$\propto$" denotes equality up to a constant. In practice, computing $P(\mathcal{D})$ is usually analytically intractable. Hence, to sample from the posterior $P(\theta|\mathcal{D})$, we usually use the unnormalized expression $P(\mathcal{D}|\theta)P(\theta)$.

To predict and estimate the uncertainty of a solution trajectory, denoted as $\{G^\dagger(u)(y) : y \in Y_m\}$, for a given input u and over the mesh of points $y \in Y_m$, we must sample from the posterior $P(\theta|\mathcal{D})$ and obtain an M-ensemble of DeepONets with parameters $\{\theta_k\}_{k=1}^M$ (see Fig. 3). Then, using this ensemble, we can obtain statistics from simulated trajectories $\{G_{\theta_k}(u)(y) : y \in Y_m\}_{k=1}^M$. In this work, we compute the mean and standard deviation of $\{G_{\theta_k}(u)(y) : y \in Y_m\}_{k=1}^M$. We use the mean to predict the true solution $\{G^\dagger(u)(y) : y \in Y_m\}$ and the standard deviation to quantify the predictive uncertainty.

To sample from the posterior and obtain the M-ensemble of DeepONets with parameters $\{\theta_k\}_{k=1}^M$, in this work, we use replica exchange stochastic Langevin diffusion (re-SGLD) [12], which we review in the next section and detail in Algorithm 1.

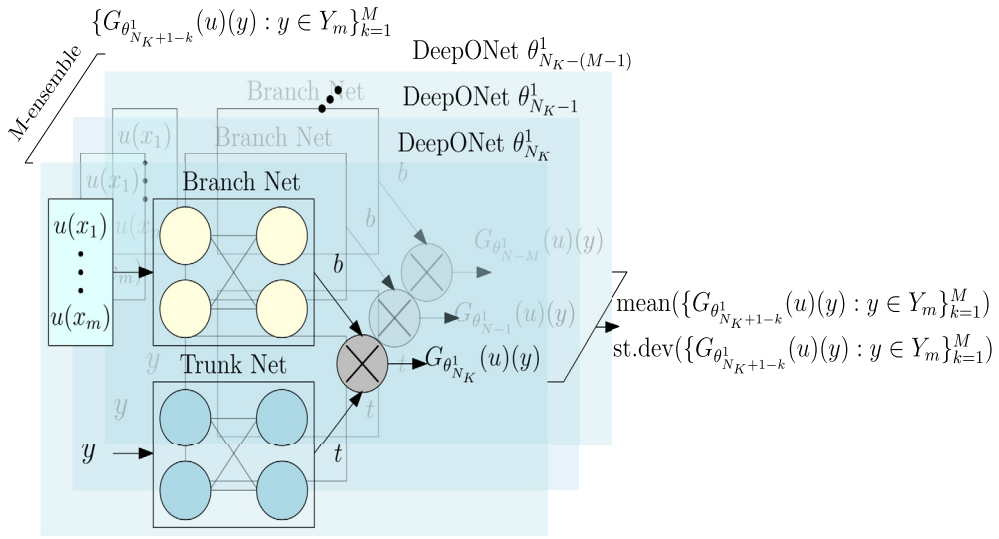

Fig. 3. The DeepONet M-ensemble $\{G_{\theta^1_{N_K+1-k}}\}_{k=1}^M$. Using this ensemble, we simulate trajectories, i.e., $\{G_{\theta^1_{N_K+1-k}}(u)(y) : y \in Y_m\}_{k=1}^M$, and compute statistics, e.g., the mean and standard deviation (st.dev).

Algorithm 1: Replica Exchange Stochastic Gradient Langevin Diffusion.

Require: initial DeepONet parameters θ_0^1, θ_0^2, learning rate η_k, temperatures τ_1, τ_2, stochastic gradient variances σ_1^2, σ_2^2, and two constants $a_1, a_2 > 0$ satisfying $a_1 + a_2 = 1$.
for $k = 0, \ldots, N_K - 1$ **do**
  sampling step:
  sample $B_k^1 \sim \mathcal{N}(0, I)$ and $B_k^2 \sim \mathcal{N}(0, I)$
    $\theta_{k+1}^1 = \theta_k^1 - \eta_k \nabla \hat{U}_1(\theta_k^1) + \sqrt{2\tau_1 \eta_k} B_k^1$
    $\theta_{k+1}^2 = \theta_k^2 - \eta_k \nabla \hat{U}_2(\theta_k^2) + \sqrt{2\tau_2 \eta_k} B_k^2$
  swapping step:
  Generate a uniform random number $\psi \in [0, 1]$
  let $\tau_\delta = \frac{1}{\tau_1} - \frac{1}{\tau_2}$ and compute
    $\hat{r} = e^{\tau_\delta \left((a_1 - a_2)(\hat{U}_1(\theta_{k+1}^1) - \hat{U}_2(\theta_{k+1}^2)) - (a_1\sigma_1 + a_2\sigma_2)^2 \tau_\delta\right)}$
  if $\psi < \hat{r}$ **then**
    swap θ_{k+1}^1 and θ_{k+1}^2
  end
end
Calculate $\{G_{\theta^1_{N_K+1-k}}(u)(y) : y \in Y_m\}_{k=1}^M$ as prediction samples of the true posterior trajectory $\{G^\dagger(u)(y) : y \in Y_m\}$.

3.1. Review of the re-SGLD

To derive re-SGLD (Algorithm 1), we start from the Langevin diffusion, which reads:

$$d\theta_t = -\nabla U(\theta_t) + \sqrt{2\tau} dB_t, \tag{5}$$

where $\theta_t \in \mathbb{R}^p$ is called the particle and corresponds to the target trainable parameters of the DeepONet, B_t is the Brownian motion in $\mathbb{R}^p$, $\tau > 0$ is the temperature parameter, and $U : \mathbb{R}^p \to \mathbb{R}$ is an energy function defined as:

$$U(\theta) \sim -\log P(\theta) - \sum_{i=1}^{N} \log P(\tilde{G}_i^\dagger | \theta), \tag{6}$$

where $P(\theta)$ denotes the prior and $P(\tilde{G}_i^\dagger|\theta)$ the likelihood (4).

To obtain the target parameters, we must solve an optimization problem. More precisely, we must find $\theta^* \in \mathbb{R}^p$ that minimizes the energy (6). To this end, one can show that, under proper assumptions for the energy function U, the sequence of targets $\{\theta_t\}_{t \geq 0}$ converges to the target distribution $\pi_{\mathcal{D}} \sim \exp\left(-\frac{U(\theta)}{\tau}\right)$ as $t \to \infty$ [7]. Furthermore, when the temperature parameter τ is small enough, $\pi_{\mathcal{D}}(\theta)$ distributes around the minimizer θ^* [18]. Hence, one can obtain the minimizer θ^* of the energy (6) by sampling from the target distribution $\pi_{\mathcal{D}}(\theta)$.

In addition, when the temperature τ is low, the particle tends to reach stationary points with fast local convergence [27], by *exploiting* the landscape of the energy function U. On the other hand, when the temperature τ is high, the particle tends to traverse globally, exploring the landscape of the energy function U. This exploitation-exploration dilemma has led researchers to design algorithms (e.g., re-SGLD) that achieve superior performance by balancing exploration and exploitation.

In this work, we use exploration and exploitation by considering two separate particles θ_t^1 and θ_t^2. The low-temperature particle θ_t^1 trains an exploitation DeepONet that exploits the landscape of U. The high-temperature particle θ_t^2 trains an exploration DeepONet that enables escaping local minima via swapping (see Fig. 4). These particles satisfy the Langevin diffusion, i.e.,

$$d\theta_t^1 = -\nabla U(\theta_t^1) + \sqrt{2\tau_1} dB_t^1, \tag{7a}$$

$$d\theta_t^2 = -\nabla U(\theta_t^2) + \sqrt{2\tau_2} dB_t^2, \tag{7b}$$

and use swapping, with controlled rate $r(\theta_t^1, \theta_t^2)$, to help the low-temperature particle θ_t^1 escape local minima and achieve improved and accelerated convergence.

The theoretical support for the previous claims was established in the convergence result of [5]. That is, the *replica exchange Langevin diffusion* (reLD) algorithm, which simulates the dynamics (7) and allows the swapping of particles, i.e., $(\theta_{t+dt}^1, \theta_{t+dt}^2) = (\theta_{t+dt}^2, \theta_{t+dt}^1)$, with rate:

$$r(\theta_t^1, \theta_t^2) = e^{\tau_\delta (U(\theta_t^1) - U(\theta_t^2))}, \tag{8}$$

converges to the invariant distribution with density:

$$\pi(\theta^1, \theta^2) \propto e^{-\frac{U(\theta^1)}{\tau_1} - \frac{U(\theta^2)}{\tau_2}}. \tag{9}$$

3.2. Errors in the energy function

In practice, computing the energy function U may fail due to, for example, errors in the dataset $\mathcal{D}$ or an insufficient number of samples of $\mathcal{D}$ to represent the likelihood and cover the target output space. Moreover, when the size N of the dataset is large, it is expensive to compute U and its gradient. Hence, the authors in [12] proposed to approximate the energy function U using a mini-batch of data $\{\tilde{G}_{s_i}^\dagger\}_{i=1}^n \subset \mathcal{D}$. In any case, an unbiased estimator of the energy function is then

$$\hat{U}(\theta) = -\log P(\theta) - \frac{N}{n} \sum_{i=1}^{n} \log P(\tilde{G}_{s_i}^\dagger | \theta). \tag{10}$$

Moreover, we assume the estimator (10) follows the normal distribution

$$\hat{U}(\theta) \sim \mathcal{N}(U(\theta), \sigma_e^2),$$

where σ_e^2 is the variance of the estimator. To simulate from the dynamics (5), we discretize them and obtain

$$\theta_{k+1}^1 = \theta_k^1 - \eta_k \nabla \hat{U}_1(\theta_k^1) + \sqrt{2\tau_1 \eta_k} B_k^1, \tag{11a}$$

$$\theta_{k+1}^2 = \theta_k^2 - \eta_k \nabla \hat{U}_2(\theta_k^2) + \sqrt{2\tau_2 \eta_k} B_k^2, \tag{11b}$$

where η_k is a positive learning rate and $\hat{U}_1(\theta^1) \sim \mathcal{N}(U(\theta^1), \sigma_1^2)$, $\hat{U}_2(\theta^2) \sim \mathcal{N}(U(\theta^2), \sigma_2^2)$ are the energy function estimators of the two particles. However, using the unbiased estimators for the energy, $\hat{U}_1(\theta^1)$ and $\hat{U}_2(\theta^2)$, in reLD with discretized dynamics (11) leads to a large bias for the estimator of the swapping rate $r(\theta_t^1, \theta_2^2)$ defined in (8).

To remove the bias from the swaps, we allow the particles swapping $(\theta_{k+1}^1, \theta_{k+1}^2) = (\theta_{k+1}^2, \theta_{k+1}^1)$ with the following unbiased rate estimator [12],

$$\hat{r} = e^{\tau_\delta \left(a_1 (\hat{U}_1(\theta^1) - \hat{U}_1(\theta^2)) + a_2 (\hat{U}_2(\theta^1) - \hat{U}_2(\theta^2)) - (a_1\sigma_1 + a_2\sigma_2)^2 \tau_\delta \right)} \tag{12}$$

where $a_1 + a_2 = 1$ are two positive constants. The replica exchange algorithm then converges to the target invariant distribution with this new unbiased rate estimator $\hat{r}$. We refer to this algorithm as the *replica exchange stochastic gradient Langevin diffusion* (reSGLD) if $\sigma_1 = \sigma_2$ (see Algorithm 1 and Fig. 4); otherwise, we refer to the algorithm as the *multi-variance replica exchange stochastic gradient Langevin diffusion* (m-reSGLD).

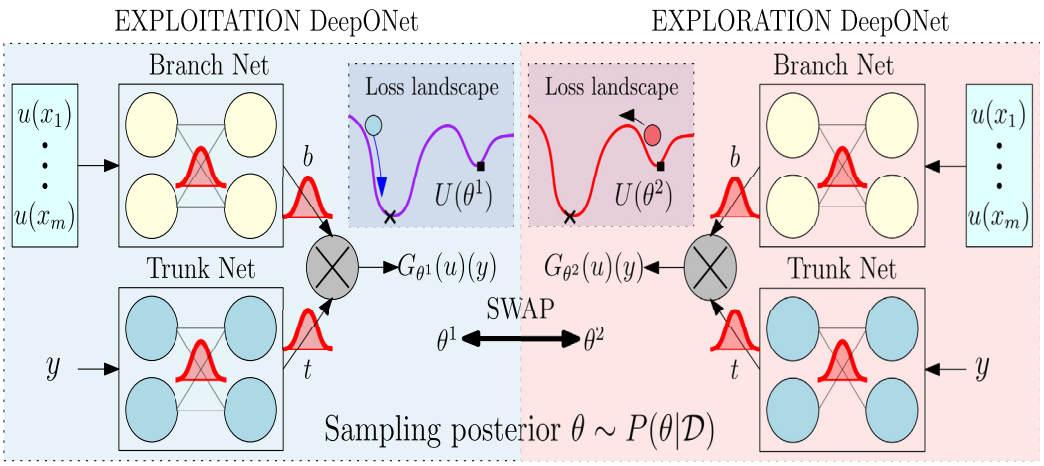

Fig. 4. The replica exchange Stochastic Gradient Langevin Diffusion (reSGLD) algorithm enables swapping between an exploitation DeepONet (with a low temperature particle) and an exploration DeepONet (with a high temperature particle). The exploitation DeepONet exploits the loss landscape to reach stationary points. The exploration DeepONet enables escaping local minima.

Algorithm 2: Accelerated Multi-variance Replica Exchange Stochastic Gradient Langevin Diffusion.

Require: initial DeepONet parameters θ_0^1, θ_0^2, learning rate η_k, temperatures τ_1, τ_2, stochastic gradient variances σ_1^2, σ_2^2, constants $a_1, a_2 > 0$ satisfying $a_1 + a_2 = 1$, and training control parameter $c \in [0, 1]$.
for $k = 0, \ldots, N_K - 1$ **do**
  sampling step:
  sample $B_k^1 \sim \mathcal{N}(0, I)$ and $B_k^2 \sim \mathcal{N}(0, I)$
    $\theta_{k+1}^1 = \theta_k^1 - \eta_k \nabla \hat{U}_1(\theta_k^1) + \sqrt{2\tau_1 \eta_k} B_k^1$
  Generate a uniform random number $\gamma \in [0, 1]$
  if $\gamma < c$ **then**
    $\theta_{k+1}^{2,\mathrm{BN}} = \theta_k^{2,\mathrm{BN}} - \eta_k \nabla \hat{U}_2(\theta_k^{2,\mathrm{BN}}) + \sqrt{2\tau_2 \eta_k} B_k^2$
    $\theta_{k+1}^{2,\mathrm{TN}} = \theta_k^{2,\mathrm{TN}}$
  else
    $\theta_{k+1}^{2,\mathrm{BN}} = \theta_k^{2,\mathrm{BN}}$
    $\theta_{k+1}^{2,\mathrm{TN}} = \theta_k^{2,\mathrm{TN}} - \eta_k \nabla \hat{U}_2(\theta_k^{2,\mathrm{TN}}) + \sqrt{2\tau_2 \eta_k} B_k^2$
  end
  swapping step:
  Generate one uniform random number $\psi \in [0, 1]$
  let $\tau_\delta = \frac{1}{\tau_1} - \frac{1}{\tau_2}$ and compute
    $\hat{r} = e^{\tau_\delta \left((a_1 - a_2)(\hat{U}_1(s_{k+1}^1) - \hat{U}_2(s_{k+1}^2)) - (a_1\sigma_1 + a_2\sigma_2)^2 \tau_\delta \right)}$
  if $\psi < \hat{r}$ **then**
    swap θ_{k+1}^1 and θ_{k+2}^2
  end
end
Calculate $\{G_{\theta_{N_K+1-k}^1}(u)(y) : y \in Y_m\}_{k=1}^M$ as prediction samples of the true posterior trajectory $\{G^\dagger(u)(y) : y \in Y_m\}$.

4. Accelerated training of Bayesian DeepONets

This section proposes exploiting the DeepONet's architecture to develop an accelerated training strategy for Bayesian DeepONets. The proposed training regime for the DeepONets can (1) reduce the time reSGLD (Algorithm 1) uses to train the DeepONet parameters and (2) achieve a performance comparable to reSGLD (and better than the stochastic gradient-based algorithm Adam). Such training strategy results in an accelerated multi-variance replica exchange SGLD (m-reSGLD) algorithm (Algorithm 2), which we describe next.

Many theoretical results have shown that over-parametrized neural networks, trained with gradient descent-based methods, converge to zero-training loss exponentially fast, with the network parameters hardly changing [14,13]. Motivated by this convergence result, other works have studied whether the same holds for under-parametrized neural networks. Their results [8] show that exponential convergence to zero-training loss can happen, with parameters hardly varying, but depends on an implicit scaling. Inspired by these results, in this work, we propose a more efficient Bayesian framework that trains a less parametrized DeepONet, i.e., a DeepONet that keeps some of its parameters hardly varying during training.

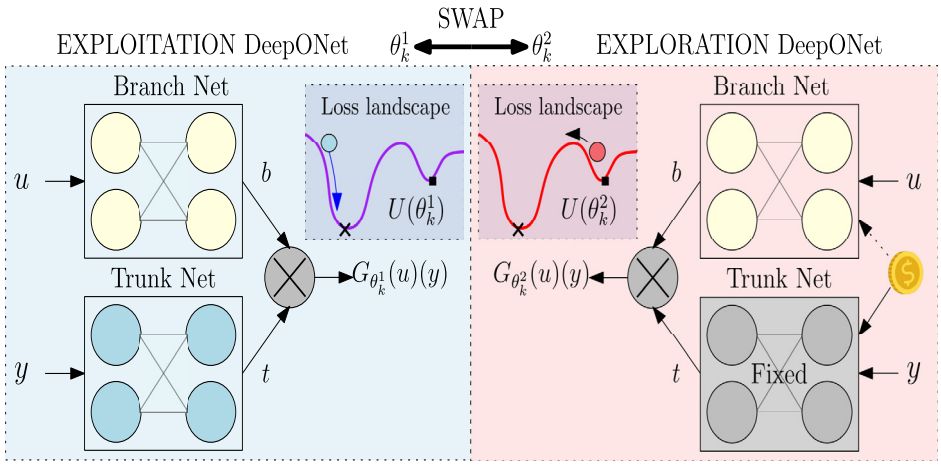

Fig. 5. The accelerated training of Bayesian DeepONets enables saving up to 25% of the computational cost of replica exchange SGLD. To this end, at each iteration, for the exploration DeepONet, we randomly (see the "biased" coin) select to train the Branch Net or the Trunk Net.

Intuitively, our proposed framework follows from the fact that empirically (see [24]), DeepONets converge exponentially fast to zero-training loss. Hence, we expect that a DeepONet that keeps, during training, some of its parameters hardly varying should also converge to zero-training loss provided an appropriate training regime. In this paper, such a regime corresponds to training either the Branch Net or the Trunk Net randomly. We refer to this training regime as the *accelerated Bayesian training of DeepONets*, which we use to reduce the computational cost of reSGLD.

The reSGLD framework has demonstrated excellent convergence capabilities when used for training neural networks. However, compared to vanilla SGLD [25], the computational cost of reSGLD doubles. To reduce the computational cost of reSGLD, we adopt the idea proposed in [23] and develop a multi-variance m-reSGLD (see Algorithm 2 and Fig. 5); that is, an algorithm that uses (1) a full training regime for the low temperature particle θ^1, which *exploits* the landscape of the energy function U, and (2) an accelerated training regime for the high temperature particle θ^2, which *explores* the landscape of U. To this end, we first fully train both particles for a fixed number of burn-in epochs. After burn-in, we let the high-temperature particle enter the accelerated training regime. Let us split the high temperature particle into its Branch Net and Trunk Net components, i.e., $\theta^2 = \{\theta^{2,\mathrm{BN}}, \theta^{2,\mathrm{TN}}\}$. Then, to control which sub-network parameters ($\theta^{2,\mathrm{BN}}$ or $\theta^{2,\mathrm{TN}}$) are trained more, we define the training control parameter $c \in [0,1]$ and generate a random number γ in $[0,1]$. If $\gamma < c$, we train more the Branch Net parameters $\theta^{2,\mathrm{BN}}$. If, on the other hand, $\gamma > c$, we train more the Trunk Net parameters $\theta^{2,\mathrm{TN}}$.

In practice, at each epoch after the burn-in, we use the parameter c to select only one group of parameters (the Branch Net's or the Trunk Net's parameters) for training. Motivated by the DeepONet's neural network architecture, we will train the Branch Net's parameters more, i.e., we set $c \in [0.5, 1.0]$. As explained in Section 2.1, the DeepONet architecture, depicted in Fig. 1 and whose output is given by (2), enables us to interpret the Branch Net output b as trainable coefficients for the trainable basis functions t produced by the Trunk Net. Hence, DeepONets, as proved in [20], fall within the class of linear approximation methods because they approximate the nonlinear space of the operator $G^\dagger$ using the linear space generated by the Trunk Net, i.e., $\mathrm{span}\{\varphi_1(y), \ldots, \varphi_q(y)\}$. As a result, in our numerical experiments (see Section 5), we choose to train the Branch Net's parameters more, i.e., we let $c \in [0.5, 1.0]$ (e.g., $c = 0.75$) because the Branch Net's output represents the coefficients for the basis functions generated by the Trunk. We remark, however, that one can always perform an expensive hyper-parameter optimization routine to find the best value for c by minimizing an estimate of the computational cost of the proposed accelerated regime.

Let us now assess the computational cost of m-reSGLD. Suppose the hidden and output layers of the Branch and Trunk net are identical. In that case, the accelerated regime could reduce roughly half the computational cost for training the Trunk Net's parameters θ^2. Thus, overall, m-reSGLD reduces at most 25% of the reSGLD computational cost.

Finally, one should note that as soon as the accelerated training regime (m-reSGLD) freezes one of the particles, the corresponding gradient of the stochastic energy becomes truncated. And hence the estimate of the stochastic energy changes. In particular, the variances of the estimates are no longer the same [23], i.e.,

$$\hat{U}_1(\theta_k^1) \sim \mathcal{N}(U(\theta_k^1), \sigma_1^2) \text{ and } \hat{U}_2(\theta_k^2) \sim \mathcal{N}(U(\theta_k^2), \sigma_2^2).$$

As a result, we use the name multi-variance replica exchange stochastic gradient Langevin diffusion (m-reSGLD) for denoting Algorithm 2.

4.1. DeepONet transfer learning

This section describes a transfer learning strategy to reduce the recalibration effort between PDE problems with different (1) unknown noise levels of the training dataset and (2) parameter values.

First, during testing (see Section 5), we will assume we know the noise level for all experiments. However, in practice, one may not have access to prior information about the noise. So, in this section, we propose a transfer learning strategy that tackles real problems with unknown and smaller noise levels. We will show with our experiments that we can efficiently train the new problem with unknown and smaller noise levels and predict with great accuracy. In addition, given multiple related PDE problems, we can train (from random initialization) individual Bayesian DeepONets to approximate the operator of each problem with different parameters. However, training all these DeepONets from scratch may be computationally expensive. To alleviate this cost, we propose using *transfer learning*.

In transfer learning, we consider two objects: the source and the target. The *source* corresponds to a fully trained Bayesian DeepONet that approximates the operator of a given PDE problem. The *target* corresponds to a Bayesian DeepONet that we want to train to approximate the operator of a different but related PDE problem. Here we relate the source and a target as follows. (1) The source and the target have the same parametric PDE, but we train the target using unknown and smaller noise levels. (2) The PDE of the target has different parameter values than the source PDE, and we train the target Bayesian DeepONet using a different distribution of inputs.

To transfer learning from the source to the target, we exploit the structure of the DeepONet and proceed as follows. We start by initializing the parameters of the target Bayesian DeepONet using the already trained parameters of the source DeepONet. We then train the target Bayesian DeepONet as follows. As described before, the Trunk Net represents a collection of trainable basis functions, and the Branch Net represents the trainable coefficients of these basis functions. So, once the source DeepONet is well trained, it is reasonable to assume the basis has been well constructed and can be transferred to the related target PDE problem. As a result, for the target Bayesian DeepONet, we can fix the parameters of the Trunk Net and use reSGLD or m-reSGLD with the new dataset to recalibrate the parameters of the Branch Net. In the numerical experiments section, we will demonstrate that this transfer learning strategy reduces the operator learning effort for related PDEs.

5. Numerical experiments

This section tests the efficacy of the proposed Bayesian DeepONet using two examples: an anti-derivative operator and a diffusion-reaction system. In addition, in Appendix A and Appendix B, we have added two more examples: a gravity pendulum system and an advection-diffusion system.

We first test how well the Bayesian DeepONet, trained with replica exchange SGLD (reSGLD) and accelerated replica exchange SGLD (m-reSGLD), approximates the solution operator of the examples. We compare the Bayesian DeepONet's approximation and generalization power against the vanilla DeepONet trained with the state-of-the-art Adam algorithm. To this end, we systematically test the DeepONets using datasets of noisy output targets $\{\tilde{G}_i^\dagger\}_{i=1}^N$ with different noise levels.

Predictive metrics. To test the performance of the Bayesian DeepONets and the vanilla DeepONet, we compute the L^1 and L^2 relative errors of 100 test trajectories selected from outside the training dataset. That is, for a given discretized input $(u(x_1), \ldots, u(x_m))$, we predict using the DeepONets the solution trajectory at a collection of selected mesh points $y \in Y_m \subset Y$. Denote the (mean) predicted solution of the test trajectory as $v_u := \{G_\theta(u)(y) : y \in Y_m\}$ and the true solution as $w_u := \{G^\dagger(u)(y) : y \in Y_m\}$, then we calculate the relative error as follows:

$$e_1 = \frac{\|v_u - w_u\|_1}{\|w_u\|_1} \cdot 100\%, \quad e_2 = \frac{\|v_u - w_u\|_2}{\|w_u\|_2} \cdot 100\%. \tag{13}$$

We then verify how the proposed Bayesian DeepONets (trained with reSGLD and m-reSGLD) handle the noisy targets and estimate the predictive uncertainty. To this end, we construct a 95% (2σ) confidence interval for every test trajectory. In particular, the Bayesian DeepONet, trained with reSGLD (resp. m-reSGLD), constructs the confidence interval using the $M-$ prediction samples $\{G_{\theta_{N_K+1-k}}(u)(y) : y \in Y_m\}_{k=1}^M$ obtained using Algorithm 1 (resp. Algorithm 2). In all of our experiments, we selected an ensemble of $M = 2000$. To enable the vanilla DeepONet to estimate the uncertainty, we adopt the non-Bayesian framework of Dropout [17] and use the strategy described in [26].

Uncertainty metrics. To measure how well the confidence intervals quantify uncertainty, we compute the ratio of the true test trajectory that is within the confidence interval, i.e.,

$$e_3 = \frac{\#\text{ of points of } w_u \text{ in the confidence interval}}{\#\text{ of points of } w_u} \cdot 100\%. \tag{14}$$

5.1. Experiment 1: antiderivative operator

In this experiment, we use the proposed Bayesian DeepONet to approximate the solution operator of the following ordinary differential equation (ODE):

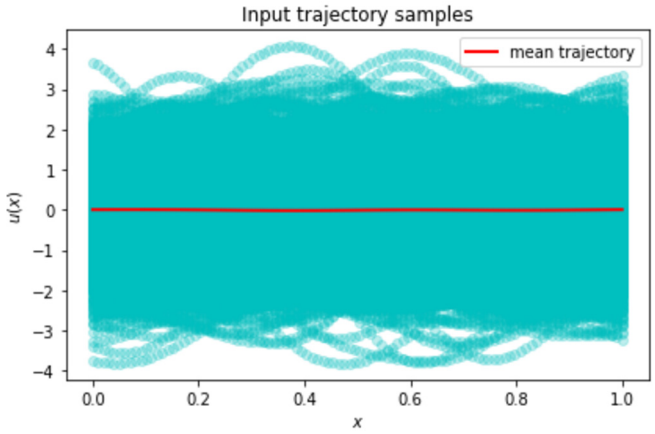

Fig. 6. Distribution of the input samples u simulated from the GRF $\mathcal{G}(0, k_l(x_1, x_2))$ with RBF kernel and length-scale $l = 0.2$.

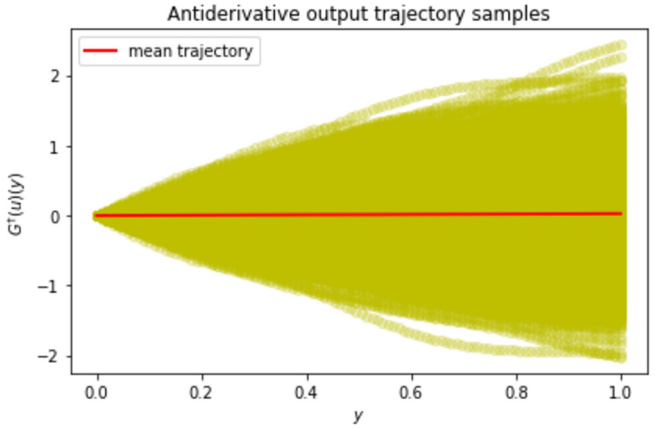

Fig. 7. Distribution of the operator targets $G^\dagger(u)(y)$ obtained from the distribution of inputs depicted in Fig. 6 and used to train the DeepONet for the anti-derivative example.

$$\frac{ds}{dt} = u(t), \quad t \in [0, 1], \tag{15}$$

with initial condition $u(0) = 0$. Note that the solution operator for (15) corresponds to the anti-derivative operator:

$$s(t) = \int_0^t u(\tau) d\tau, \quad t \in [0, 1].$$

Here $u(t)$ denotes an external force that drives the response of the ode system (15). We sample this external force from the following mean-zero Gaussian Random Field (GRF):

$$u \sim \mathcal{G}(0, k_l(x_1, x_2)),$$

where the covariance kernel $k_l(x_1, x_2) = \exp(-||x_1 - x_2||^2/2l^2)$ is the radial-basis function (RBF) kernel with length-scale parameter $l > 0$. In this experiment, we set the length-scale parameter to $l = 0.2$. Fig. 6 illustrates the distribution of input samples simulated from the GRF and used to train the DeepONets. More specifically, the Branch Net takes the discretized version $(u(x_1), \ldots, u(x_m))$ of these input samples u during training. Here, we discretized u using $m = 100$ sensors.

In addition, we present in Fig. 7 the distribution of the true operator targets associated with the input samples u. We can observe that these targets deviate largely from the mean. Thus, any predictive framework must capture the operator response over the whole output target space. Such a requirement makes this problem very challenging.

5.1.1. Small noise

We consider first training the DeepONet using operator targets $\{\tilde{G}_i^\dagger\}_{i=1}^N$ with small noise. That is, we assume the noise ϵ_i follows the normal distribution $\mathcal{N}(0, 0.01^2)$. We train the DeepONet for 8,000 epochs using the three frameworks: reSGLD,

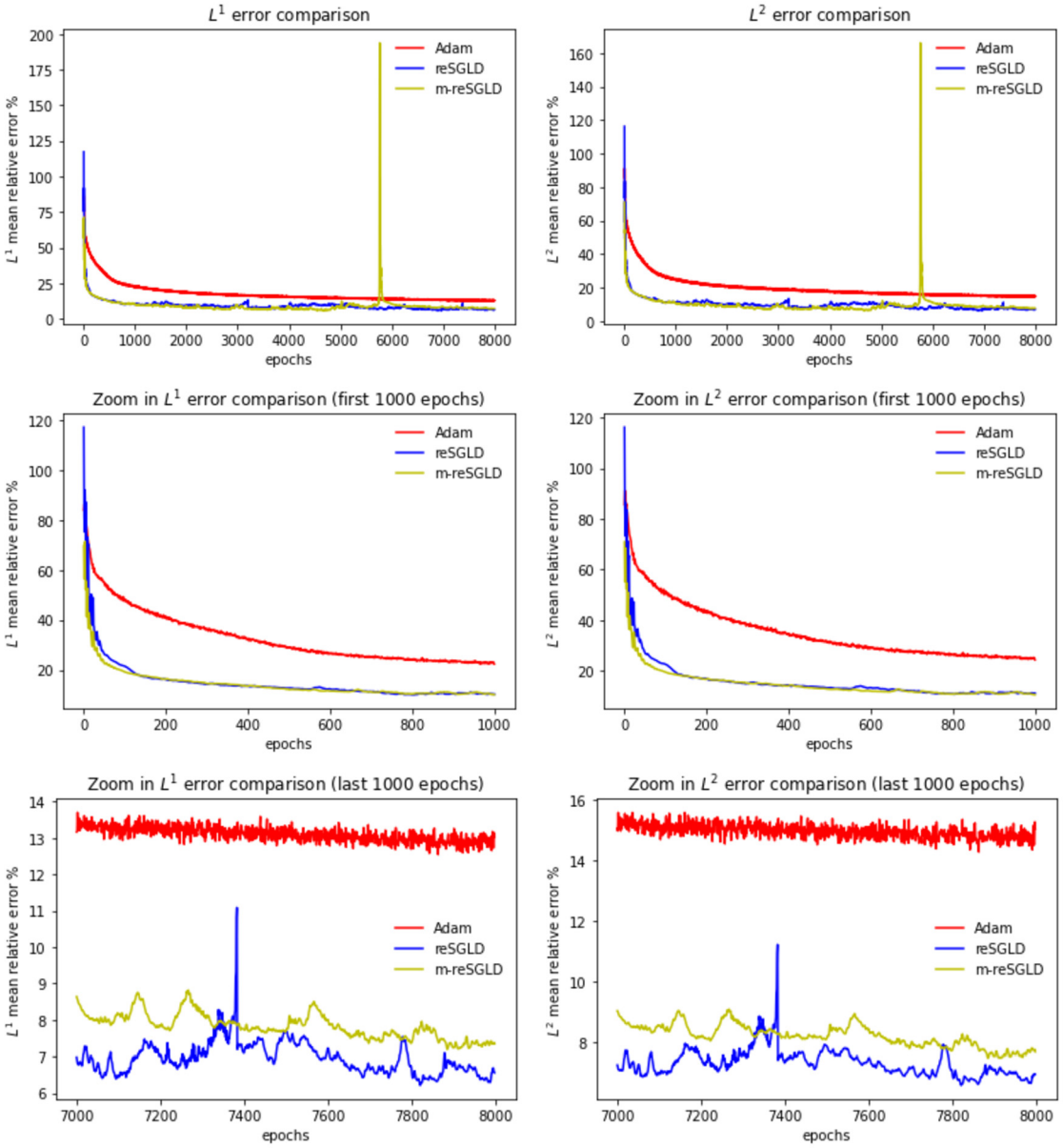

Fig. 8. Convergence results for the anti-derivative example with noise $\mathcal{N}(0, 0.01^2)$. The e_1 errors are presented on the left while e_2 errors are shown on the right. Top row: entire history. Middle row: zoom in the first $1,000$ epochs. Last row: zoom in the last $1,000$ epochs. The average errors after the burn in are presented in the Table 1. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 1
Anti-derivative example with noise $\mathcal{N}(0, 0.01^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	13.099151	14.961829
reSGLD	6.953939	7.35423
m-reSGLD	7.82799	8.234294

m-reSGLD, and Adam. Furthermore, we compute the mean of the relative errors e_1 and e_2 of 100 test trajectories at each epoch. Fig. 8 shows how these errors converge during training, and Table 1 presents the average mean relative errors after a given number of burn-in epochs.

From Fig. 8 and Table 1, we can observe that the proposed Bayesian DeepONet trained with reSGLD and m-reSGLD converge much faster than the one trained with Adam. Moreover, the average relative errors after burn-in show that the

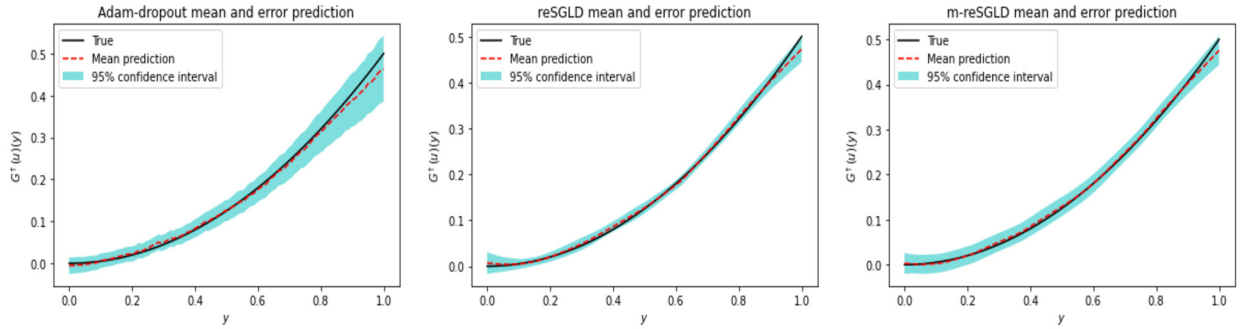

Fig. 9. Confidence intervals for the anti-derivative example with noise $\mathcal{N}(0, 0.01^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, e_3 for all three frameworks is 100%.

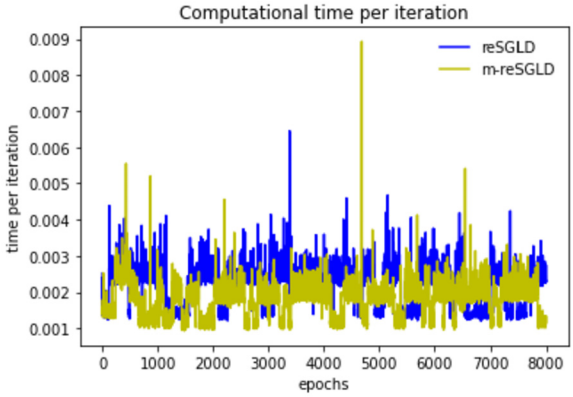

Fig. 10. Anti-derivative example with noise $\mathcal{N}(0, 0.01^2)$. Computer running time per iteration for reSGLD and m-reSGLD. The average computation time of m-reSGLD is around 80.1% of reSGLD framework.

mean predictive performance of the proposed Bayesian framework outperforms the state-of-the-art Adam. Such improved accuracy illustrates that reSGLD and m-reSGLD handle much better noisy training data.

Besides the fast training convergence and improved mean predictive performance, we must check if the proposed frameworks can capture the predictive uncertainty of the problem. To this end, we illustrate in Fig. 9, for a randomly selected test trajectory, the confidence interval and the error e_3 for Adam with dropout and the Bayesian DeepONet trained reSGLD and m-reSGLD. We observe that the three frameworks can capture the true trajectory within their confidence intervals ($e_3 = 100\%$). Adam with dropout, however, seems to overestimate its predictive uncertainty, producing a wider confidence band.

We also verify how much of the computational cost of reSGLD is reduced when using m-reSGLD. To this end, during training, we record the per-iteration computational time for reSGLD and m-reSGLD (see Fig. 10). The results show that the computational time for m-reSGLD corresponds to 80.1% the computational time of reSGLD. This reduction of approximately 20% of the cost of reSGLD is close to the theoretical maximum of 25%. Furthermore, m-reSGLD does not deteriorate the predictive performance of the DeepONet (see Table 1).

5.1.2. Increased noise

We now verify the performance of the proposed frameworks when the noise associated with the operator output targets $\{\tilde{G}_i^\dagger\}_{i=1}^N$ increases. In particular, we assume the increased noise follows the normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.05^2)$. Fig. 11 depicts the training convergence results for the relative errors e_1 and e_2, and Table 2 presents the average errors.

We observe from Fig. 11 and Table 2 that the convergence performance of the three frameworks deteriorates in this increased noise scenario. However, our proposed Bayesian DeepONet (reSGLD and m-reSGLD) provides a faster training convergence and has much better mean predictive performance than Adam.

We also construct the confidence intervals for the three frameworks. Fig. 12 shows that, as expected, the predictive uncertainty (i.e., the width of the confidence band) estimated by the proposed framework has increased in this scenario. However, the confidence band of Adam with dropout is observed to be noisier and uncertain. Also, we note that the three frameworks capture the true solution trajectory, i.e., they have $e_3 = 100\%$. Thus, we conclude that the proposed frameworks are more effective in predicting uncertainty in this increased noise scenario.

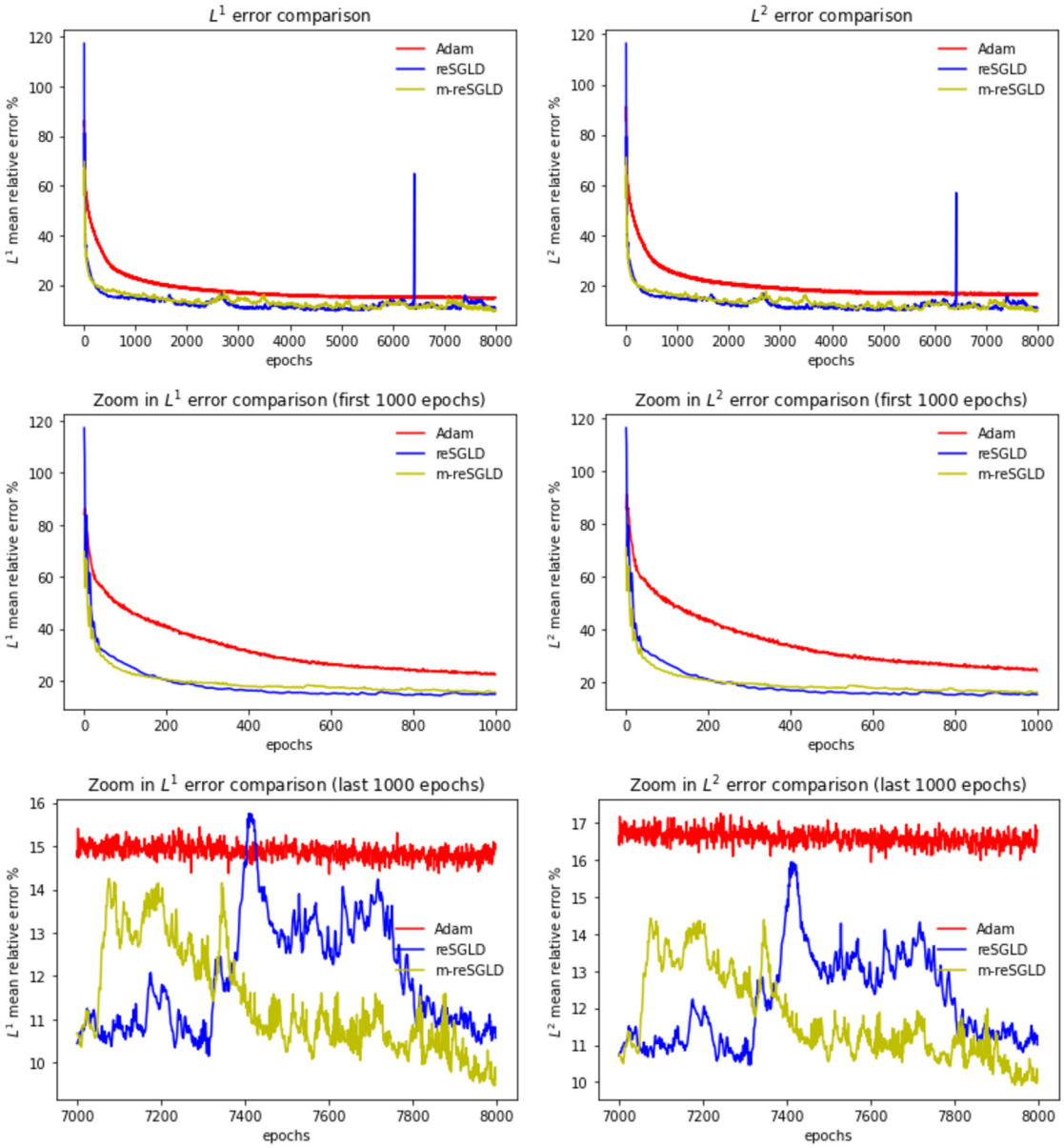

Fig. 11. Convergence results for the anti-derivative with noise $\mathcal{N}(0, 0.05^2)$. The e_1 errors are presented on the left while e_2 errors are shown on the right. Top row: entire history. Middle row: zoom in the first $1,000$ epochs. Last row: zoom in the last $1,000$ epochs. The average errors after the burn in are presented in the Table 2.

Table 2
Anti-derivative example with noise $\mathcal{N}(0, 0.05^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	14.877717	16.611967
reSGLD	11.970458	12.182682
m-reSGLD	11.367707	11.708988

We also illustrate (see Fig. 13) how much computational cost is saved when using m-reSGLD instead of reSGLD. The average per-iteration time (i.e., the computational cost) for m-reSGLD represents 76.9% of the per-iteration time used by reSGLD. Moreover, for this increased noise scenario, m-reSGLD is not only more efficient but also provides a better mean predictive performance (see Table 2).

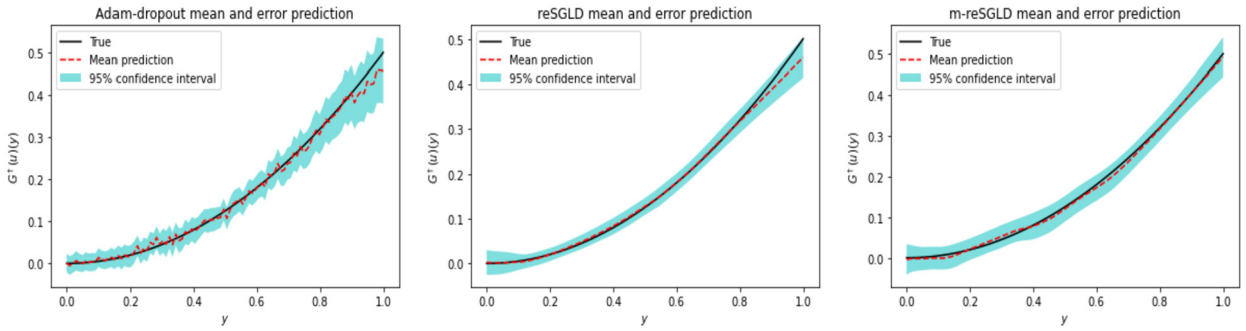

Fig. 12. Confidence intervals for the anti-derivative example with noise $\mathcal{N}(0, 0.05^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, e_3 of Adam, reSGLD and m-reSGLD are all 100%.

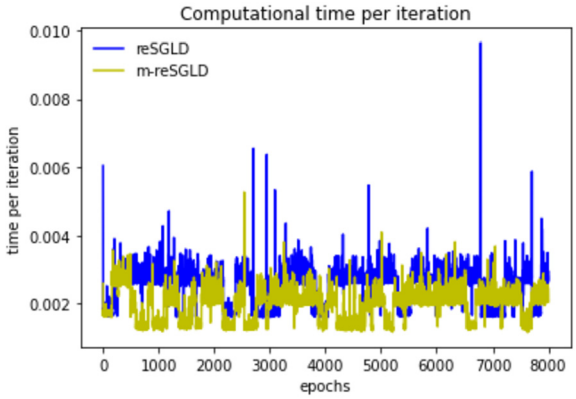

Fig. 13. Anti-derivative example with noise $\mathcal{N}(0, 0.05^2)$. Computer running time per iteration for reSGLD and m-reSGLD. The average computation time of m-reSGLD is around 76.9% of reSGLD framework.

Table 3
Antiderivative operator transfer learning for *small* noise $\mathcal{N}(0, 0.01^2)$. The network is initialized using the model trained with *increased* noise (see section 5.1.2).

	e_1	e_2
Direct training	6.953939	7.35423
Transfer learning	6.07787	6.370388

5.1.3. *DeepONet transfer learning*

We conclude this experiment by testing the efficacy of the proposed transfer learning strategy. The source for transfer learning corresponds to the Bayesian DeepONet trained using reSGLD and operator data with *increased* noise $\mathcal{N}(0, 0.05^2)$. The target is a Bayesian DeepOnet to approximate that we aim to train to approximate the same antiderivative operator but using operator data with smaller noise, i.e., $\epsilon_i \sim \mathcal{N}(0, 0.01^2)$.

The proposed transfer learning strategy starts by initializing the target exploitation and exploration DeepONets for reSGLD using the source DeepONet. We fix the Trunk Net and train the Branch Net of the target DeepONet for 2000 epochs. Table 3 reports the average of the predictive errors for the transfer learning strategy after a given number of burn-in epochs. We compare these results with the predictive errors obtained when training the target DeepONet from random initialization (i.e., via direct training). One can observe that transfer learning obtained improved accuracy as direct training with a fraction of the computational cost.

5.2. *Experiment 2: diffusion reaction system*

In this experiment, we test the performance of the proposed Bayesian DeepONet with the following diffusion-reaction system,

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad x \in \Omega,\ t \in [0, 1], \tag{16a}$$

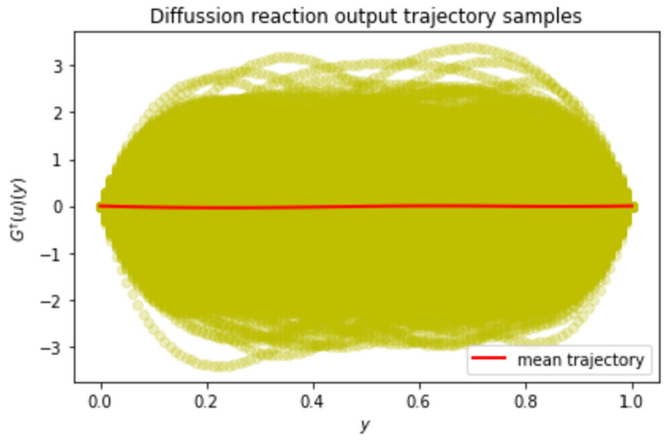

Fig. 14. Distribution of the operator targets $G^\dagger(u)(y)$ obtained from the distribution of inputs depicted in Fig. 6 and used to train the diffusion reaction example.

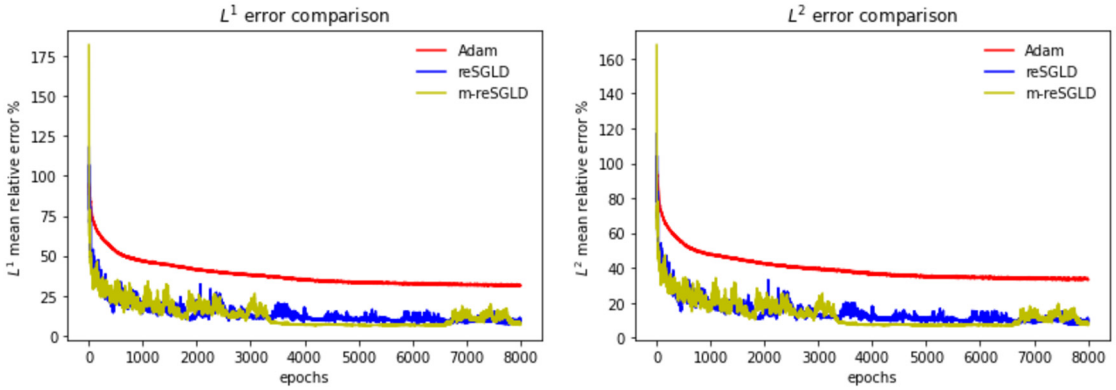

Fig. 15. Convergence results for the diffusion reaction example with noise $\mathcal{N}(0, 0.01^2)$. The e_1 errors are presented on the left while e_2 errors are shown on the right. The average errors after the burn-in are presented in the Table 4.

Table 4
Diffusion reaction example with noise $\mathcal{N}(0, 0.01^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	31.854993	33.759108
reSGLD	9.644058	9.904289
m-reSGLD	11.344359	11.62751

$$s(x,t) = 0, x \in \partial\Omega, \tag{16b}$$

$$s(x,0) = 0, x \in \Omega \tag{16c}$$

where $\Omega = [0,1]$ and $k = -0.01$. Similar to the antiderivative example, the input samples $u(x)$ are simulated using the GRF (see Fig. 6) and discretized using $m = 100$ sensors. Fig. 14 shows the distribution of the true operator targets associated with the input samples u for the diffusion-reaction system.

5.2.1. Small noise

In this small noise scenario, the noise ϵ_i added to the operator targets follows the normal distribution $\mathcal{N}(0, 0.01^2)$. Fig. 15 shows the convergence results of the relative errors e_1 and e_2 during training. Also, Table 4 presents the average of these relative errors after the burn-in. Compared to the state-of-the-art Adam optimizer, the results show that the proposed Bayesian DeepONet (trained with reSGLD and m-reSGLD) improves the training convergence and mean predictive performance.

We also show in Fig. 16 the estimated confidence intervals for the three frameworks on a test trajectory, representing the solution of the diffusion-reaction system at time $t = 1.0$. The results show that Adam with dropout overestimates the

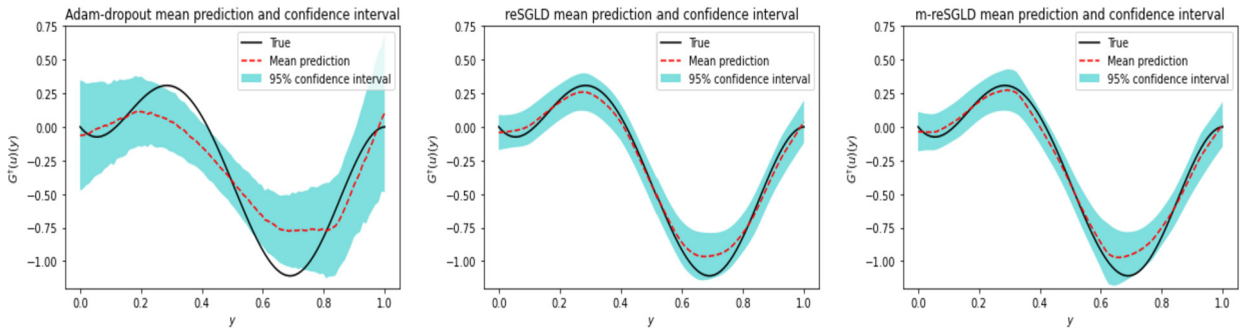

Fig. 16. Confidence intervals for the diffusion reaction example with noise $\mathcal{N}(0, 0.01^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, e_3 of Adam is 71%, for reSGLD is 100%, and for m-reSGLD is 100%.

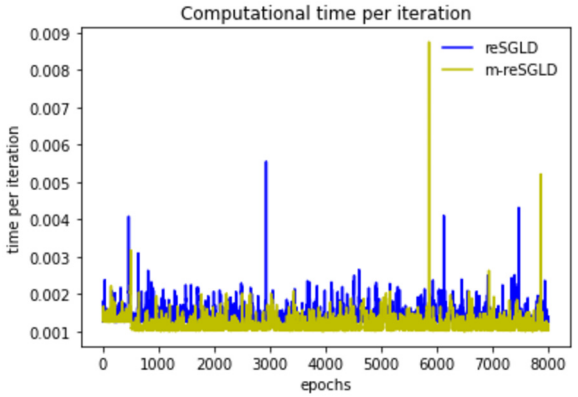

Fig. 17. Diffusion reaction example with noise $\mathcal{N}(0, 0.01^2)$. Computer running time per iteration for reSGLD and m-reSGLD. The average computation time of m-reSGLD is around 83.0% of reSGLD framework.

uncertainty (wider confidence band in some regions) and fails to capture the whole true test trajectory ($e_3 = 71\%$) within the confidence interval. On the other hand, the proposed Bayesian DeepONets, trained with reSGLD and m-reSGLD, provide more balanced confidence intervals that can capture the whole true trajectory ($e_3 = 100\%$).

We conclude the small noise scenario by plotting in Fig. 17 the per-iteration time consumed by the proposed training frameworks of reSGLD and m-reSGLD. Note that by using m-reSGLD, we can save 17% of the computational cost of re-SGLD. Furthermore, we can observe from Table 4 that the state-of-art algorithm Adam predicts with a relative error of more than 30%. This indicates that the noisy scenario is very hard to learn. However, the proposed methods greatly reduce the error to around 10%. In particular, m-reSGLD achieves such a reduction (see Table 4) with less computational cost than reSGLD.

5.2.2. Increased noise

In this section, we increase the noise variance so that the noise follows the normal distribution $\mathcal{N}(0, 0.1^2)$. Fig. 18 shows the convergence results of e_1 and e_2 relative errors during training. Moreover, Table 5 presents the average of these errors after the burn-in. Compared to Adam, we observe that the proposed Bayesian DeepONet improves the mean predictive performance and training convergence.

The estimated confidence intervals are depicted in Fig. 19. The proposed frameworks capture much better the true trajectory in their confidence intervals ($e_3 = 100\%$). On the other hand, Adam with dropout seems affected by the noise and fails to capture the whole trajectory ($e_3 = 68\%$).

We conclude the increased noise scenario by plotting the per-iteration time for reSGLD and m-reSGLD (Fig. 20). The results show that m-reSGLD uses only 81.5% of the computational resources used by reSGLD. Such a reduction is achieved by the proposed m-reSGLD with (remarkably) improved predictive performance (see Table 5).

5.2.3. DeepONet transfer learning

To conclude this example, we test the efficacy of the proposed transfer learning strategy. The *source* for the transfer learning is the Bayesian DeepONet with increased noise (see Section 5.2.2). The target is to learn the solution operator of a diffusion-reaction system with different parameter $k' = 0.005$ using a different input data distribution. We generate this input data using a GRF with an RBF kernel of length $l' = 0.6$. Also, we assume the operator data is contaminated with smaller noise distributed as follows: $\epsilon_i \sim \mathcal{N}(0, 0.075)$.

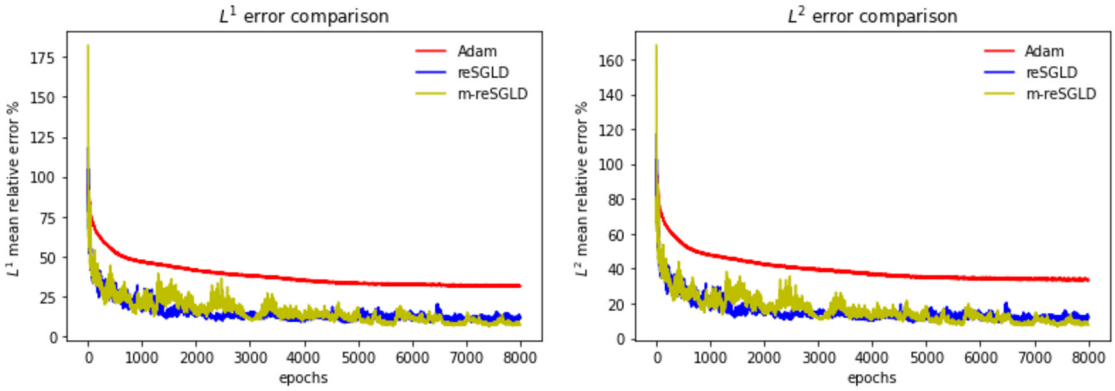

Fig. 18. Convergence results for the diffusion reaction example with noise $\mathcal{N}(0, 0.1^2)$. The e_1 errors are presented on the left while e_2 errors are shown on the right. The average errors after the burn in are presented in the Table 5.

Table 5
Diffusion reaction example with noise $\mathcal{N}(0, 0.1^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	31.954806	33.850071
reSGLD	11.670668	12.043854
m-reSGLD	9.040757	9.404453

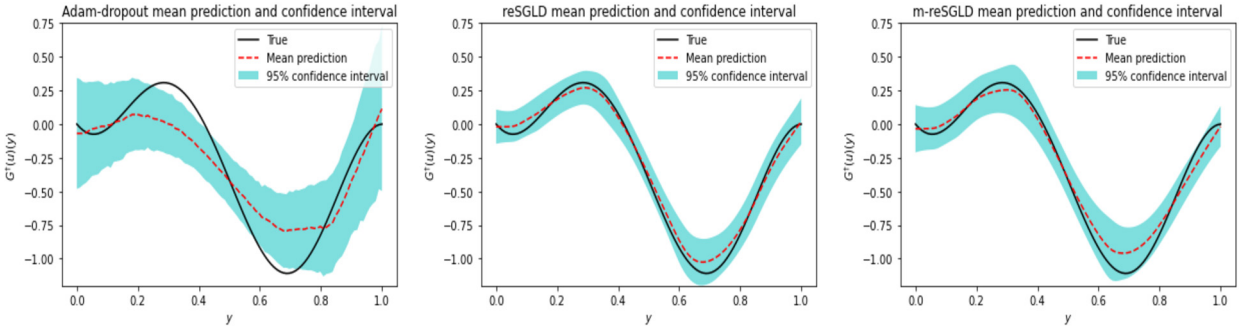

Fig. 19. Confidence intervals for the diffusion reaction example with noise $\mathcal{N}(0, 0.1^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, e_3 of Adam is 68%, for reSGLD is 100%, and for m-reSGLD is 100%.

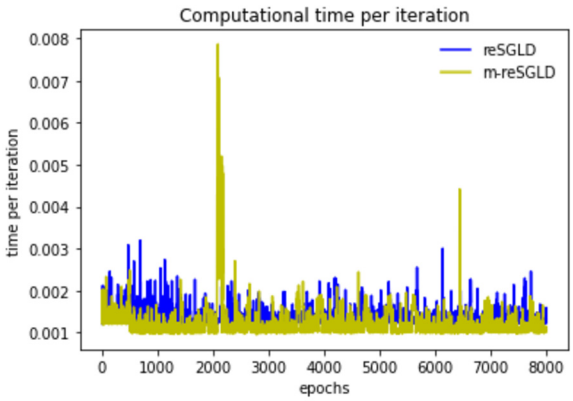

Fig. 20. Diffusion reaction example with noise $\mathcal{N}(0, 0.1^2)$. Computer running time per iteration for reSGLD and m-reSGLD. The average computation time of m-reSGLD is around 81.5% of reSGLD framework.

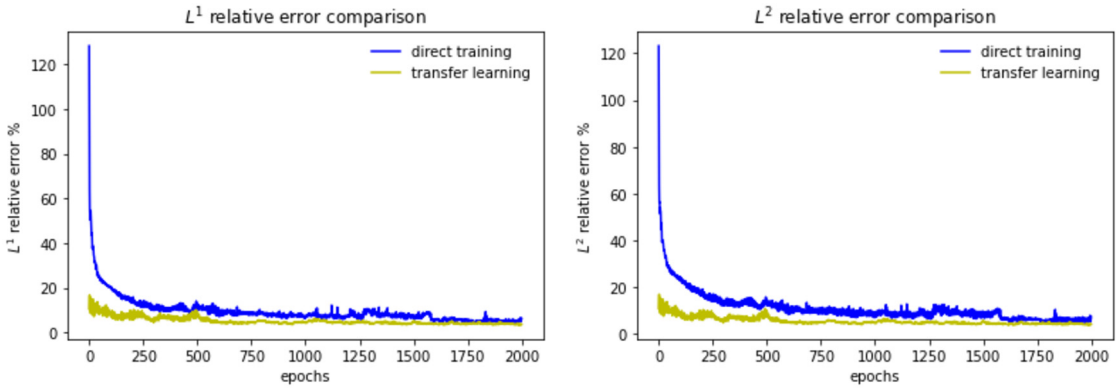

Fig. 21. Convergence results for the diffusion reaction example using direct training and transfer learning. The e_1 errors are presented on the left while e_2 errors are shown on the right. The average errors are presented in the Table 6.

Table 6
Transfer learning for diffusion-reaction system with different parameter $k' = -0.005$. Bayesian DeepONet is initialized using the trained model from increased noise scenario (see Section 5.2.2). The new dataset has small noise $\mathcal{N}(0, 0.075^2)$.

	e_1	e_2
Direct training	9.65037	11.1515
Transfer learning	5.251359	5.60123

The proposed transfer learning strategy trains the Branch Net of the target Bayesian DeepONet for 2000 epochs starting from the source DeepONet. We compare the results with a Bayesian DeepONet trained for 2000 epochs starting from random initialization (i.e., using direct training). Fig. 21 shows the convergence of the predictive error e_1 and e_2 for the transfer learning and direct training strategy. The results clearly illustrate the advantages of transfer learning. It reduces the training/recalibration time needed to reach improved mean predictive performance. Finally, Table 6 shows the average of the predictive errors for transfer learning. We observe that transfer learning greatly outperforms direct training with a fraction of the computational training cost.

To conclude this section, we would like to remark that for all examples, the training strategies for Bayesian DeepONets, reSGLD, and m-reSGLD, converge much faster than Adam, with improved predictive accuracy. The proposed Bayesian DeepONet can also handle the noise in the training data and capture the true solutions within the estimated confidence intervals. The performance of the DeepONet trained with Adam, on the other hand, deteriorates for the increased noise scenario. In addition, the benefit of the accelerated training regime of m-reSGLD is that it saves up to 25% of the computational cost of reSGLD. Such savings are achieved without deteriorating predictive performance. In fact, in some of the examples, m-reSGLD (remarkably) has even better mean predictive accuracy than reSGLD.

6. Conclusion

In this work, we study the problem of approximating the solution operator of parametric PDEs using the Deep Operator Network (DeepONet) framework and noisy data. To this end, we propose a Bayesian DeepONet based on replica exchange Langevin diffusion. The replica exchange algorithm can escape local minima (using one particle to explore and another to exploit the loss function landscape of DeepONets). Moreover, replica exchange enables the proposed framework to provide (1) improved training convergence in noisy scenarios and (2) (remarkably) enhanced mean predictive capability compared to vanilla DeepONets trained using state-of-the-art gradient-based optimization (e.g., Adam). We also demonstrate that the proposed framework effectively estimates the uncertainty of approximating solution trajectories of parametric PDEs. To reduce the computational cost of the replica, due to the doubled computational cost of using two particles, we propose (1) an accelerated replica exchange algorithm that randomly chooses to train one of the DeepONet sub-networks while keeping the other fixed and (2) a transfer learning strategy that disseminates training results to Bayesian DeepOnets that approximate PDEs with different parameter values using data with different levels of noise. In our future work, we will study the theoretical aspect of the accelerated replica exchange framework and empirically/theoretically extend the proposed framework to the scenario where the data has non-Gaussian noise.

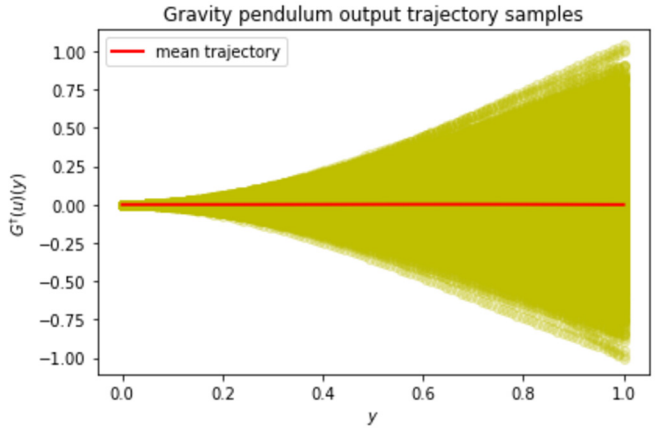

Fig. A.22. Distribution of the operator targets $G^\dagger(u)(y)$ obtained from the distribution of inputs depicted in Fig. 6 and used to train the gravity pendulum.

CRediT authorship contribution statement

CM and ZZ conceived the mathematical models, implemented the methods, designed the numerical experiments, interpreted the results, and wrote the paper. GL supported this study and reviewed the final manuscript. All authors gave final approval for publication.

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.

Data availability

No data was used for the research described in the article.

Acknowledgements

The authors gratefully acknowledge the support of the National Science Foundation (DMS-1555072, DMS-2053746, and DMS-2134209), and Brookhaven National Laboratory Subcontract 382247, and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0021142 and DE-SC0023161.

Appendix A. Experiment: gravity-pendulum ode system

In this experiment, we use the proposed frameworks to train a DeepONet that approximates the solution operator of the following gravity pendulum with external force,

$$\frac{ds_1}{dt} = s_2, \ t \in [0, 1], \tag{A.1a}$$

$$\frac{ds_2}{dt} = -k \sin(s_1) + u(t), \tag{A.1b}$$

with an initial condition $(s_1(0), s_2(0)) = (0, 0)$, and $k = 1$. We simulate the discretized external force $u(t)$ using a mean-zero GRF with RBF kernel (length-scale parameter $l = 0.2$) and $m = 100$ sensors. We illustrate in Fig. A.22 the distribution of the true operator targets associated with the input samples u and used to train the gravity pendulum system.

Small noise

We start by verifying the performance of the proposed frameworks using a small noise scenario. In particular, let us assume the noise ϵ_i for the target outputs follows the normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.01^2)$. We train the DeepONet for $8,000$ epochs using the three frameworks. Fig. A.23 shows the convergence results for the errors e_1 and e_2 during training. And Table A.7 presents the average of these errors after the burn-in. The results show that the proposed frameworks converge much faster during training and provide better mean predictive performance than the state-of-the-art optimization algorithm Adam.

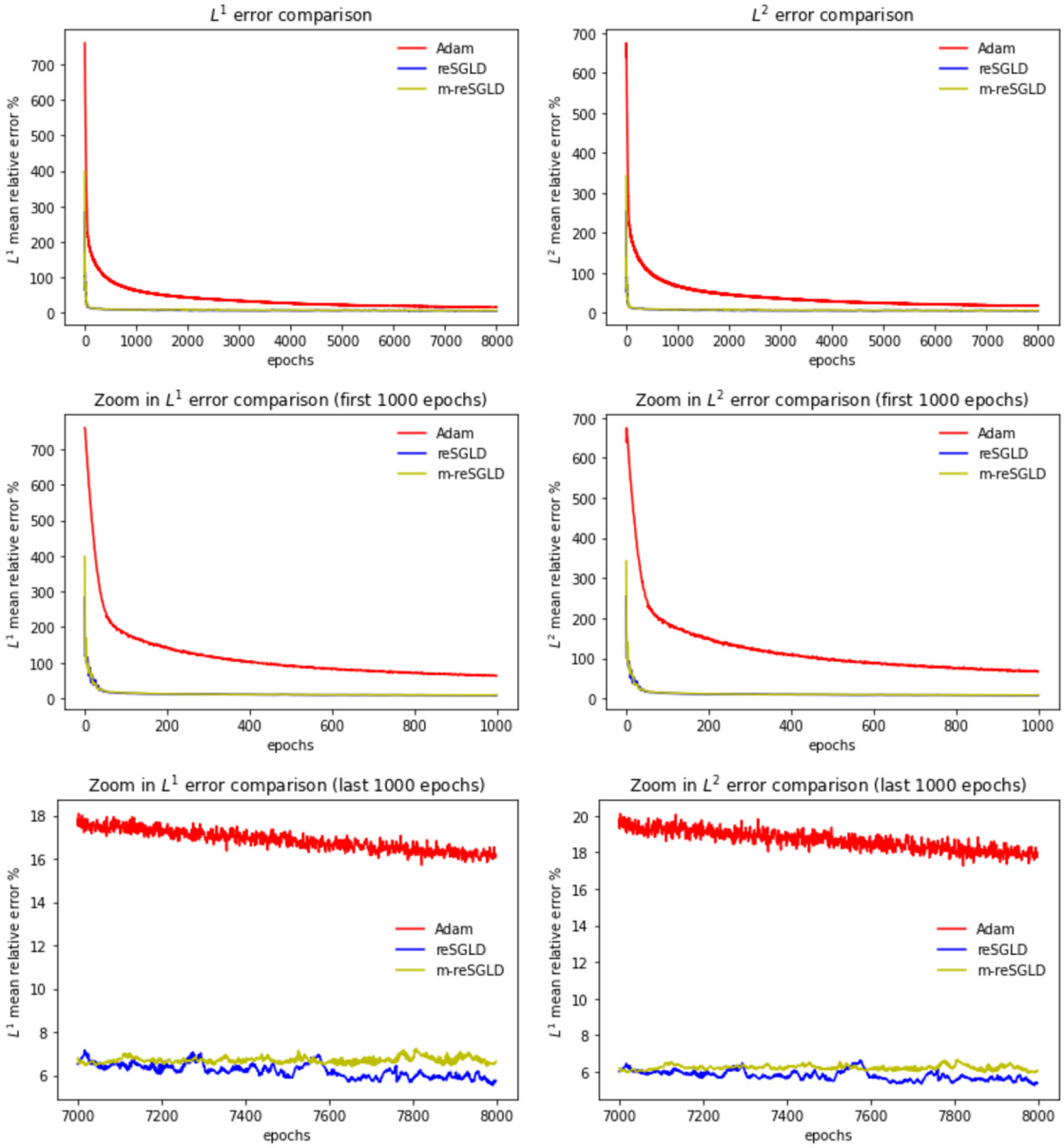

Fig. A.23. Convergence results for the gravity-pendulum with noise $\mathcal{N}(0, 0.01^2)$. The e_1 errors are shown on the left while e_2 errors are shown on the right. Top row: entire history. Middle row: zoom in the first $1,000$ epochs. Last row: zoom in the last $1,000$ epochs. The average errors after burn-in are presented in Table A.7.

Table A.7
Gravity-pendulum example with noise $\mathcal{N}(0, 0.01^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	16.8571	18.6582
reSGLD	6.2147	5.7965
m-reSGLD	6.6997	6.2231

To test how well the proposed frameworks estimate the uncertainty, we select at random one test trajectory and plot the estimated confidence intervals in Fig. A.24. Fig. A.24 shows that the proposed replica exchange frameworks capture the true test trajectory within their confidence interval, i.e., $e_3 = 100\%$. Adam with dropout, on the other hand, fails to capture the

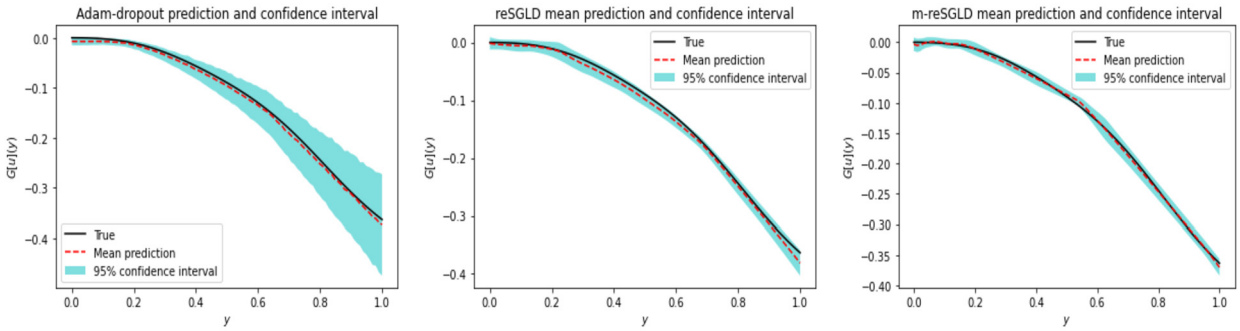

Fig. A.24. Confidence intervals for the gravity pendulum example with noise $\mathcal{N}(0, 0.01^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, the error e_3 for Adam is 71%, for reSGLD is 100%, and for m-reSGLD is 100%.

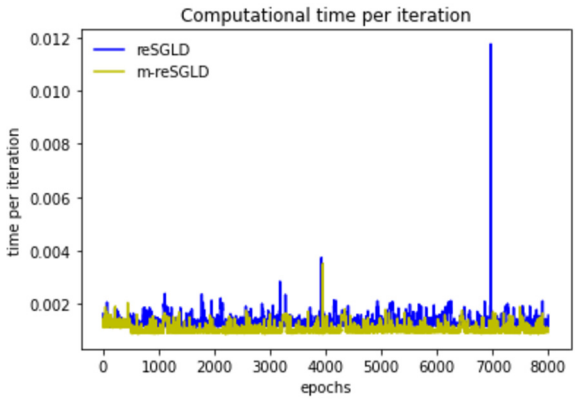

Fig. A.25. Gravity pendulum example with noise $\mathcal{N}(0, 0.01^2)$. Computer running time per iteration for reSGLD and m-reSGLD. The average computation time of m-reSGLD is around 79.06% of reSGLD framework.

whole true trajectory ($e_3 = 71\%$), and also overestimates the predictive uncertainty by producing a wider confidence band in some regions.

We also compare the computational cost of reSGLD and m-reSGLD, and we plot in Fig. A.25 the per-iteration running time for both frameworks. We observe that the per-iteration time for m-reSGLD is shorter than that for reSGLD. Thus, the computational cost for m-reSGLD represents 79.06% of the computational cost for reSGLD. Such a reduction is achieved by m-reSGLD without compromising mean predictive accuracy (see Table A.7).

Increased noise

In this section, we verify the performance of the DeepONets when the noise variance for the operator targets increases. In particular, we assume the noise follows the normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.05^2)$. Fig. A.26 and Table A.8 detail the convergence errors for this increased noise scenario. Similar to the anti-derivative example, the convergence results deteriorate when the noise increases. However, the proposed frameworks still provide improved training convergence and mean predictive performance compared with Adam.

The constructed confidence intervals for the three frameworks in this increased noise scenario are depicted in Fig. A.27. We observe that the proposed frameworks have a wider confidence band. This is expected since the increase in noise variance leads to larger uncertainty. Adam with dropout, however, seems to not change its estimate for this increased noise scenario. Furthermore, compared to Adam ($e_3 = 77\%$), we note that the proposed frameworks capture fairly well the true trajectory with their estimated confidence intervals (reSGLD: $e_3 = 93\%$ and m-reSGLD: $e_3 = 100\%$).

We conclude this gravity pendulum example by illustrating in Fig. A.28 the per-iteration time used by reSGLD and m-reSGLD. The results show that the computational cost of m-reSGLD corresponds to 80.8% of the computational cost of reSGLD. As before, m-reSGLD achieves such a reduction without compromising the mean predictive performance.

Appendix B. Experiment: advection-diffusion system

In our final experiment, we verify the effectiveness of the proposed Bayesian DeepONet using the following advection-diffusion system with periodic boundary conditions,

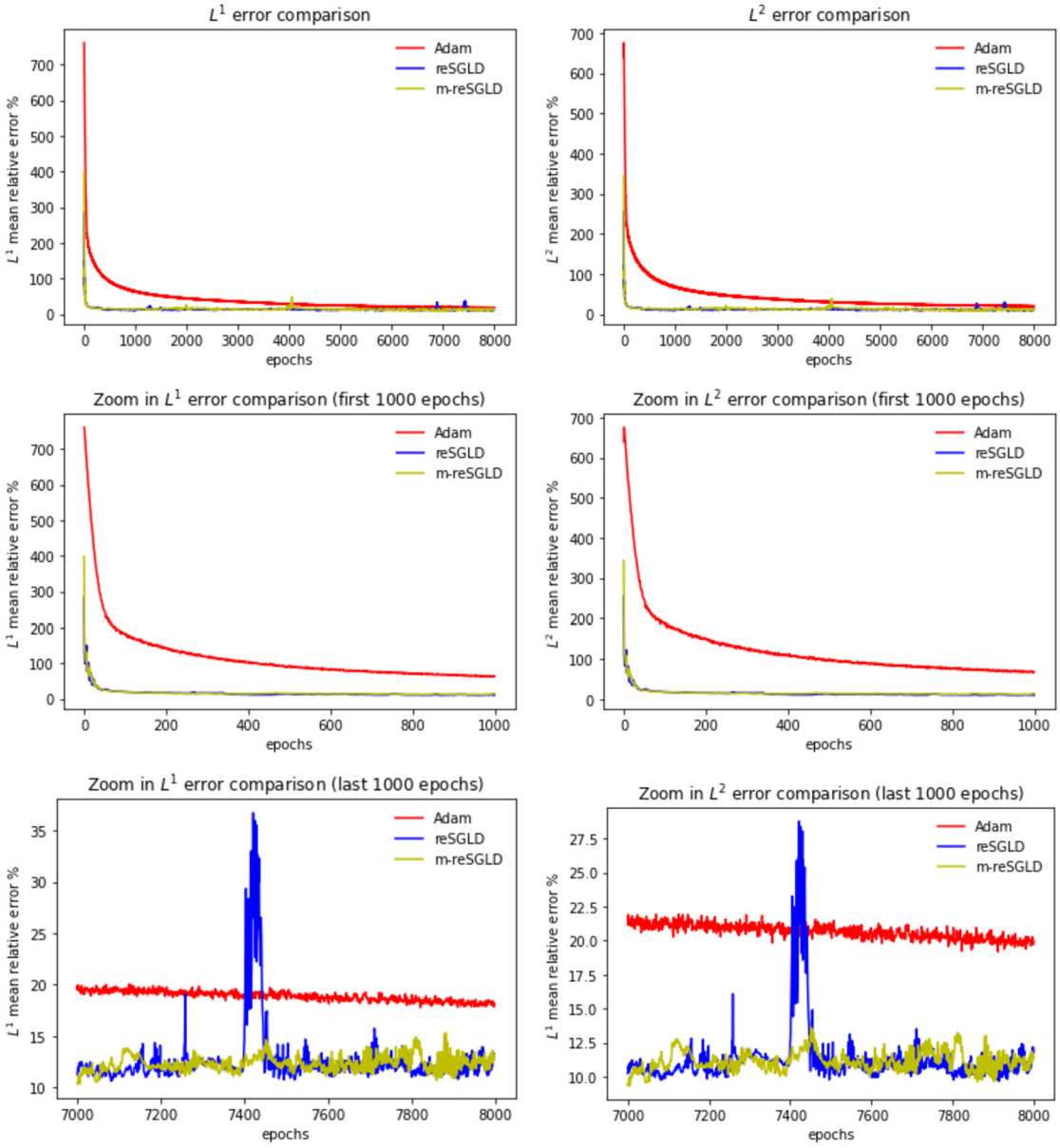

Fig. A.26. Convergence results for the gravity-pendulum with increased noise $\mathcal{N}(0, 0.05^2)$. The e_1 errors are shown on the left while e_2 errors are shown on the right. Top row: entire history. Middle row: zoom in the first $1,000$ epochs. Last row: zoom in the last $1,000$ epochs. The mean errors after the burn-in are presented in Table A.8.

Table A.8
Gravity-pendulum example with increased noise $\mathcal{N}(0, 0.05^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	18.8935	20.6669
reSGLD	12.6139	11.2885
m-reSGLD	12.2996	11.11955

$$\frac{\partial s}{\partial t} + \frac{\partial s}{\partial x} - D\frac{\partial^2 s}{\partial x^2} = 0, \qquad x \in \Omega, t \in [0, 1], \tag{B.1a}$$

$$s(x, 0) = u(x), \tag{B.1b}$$

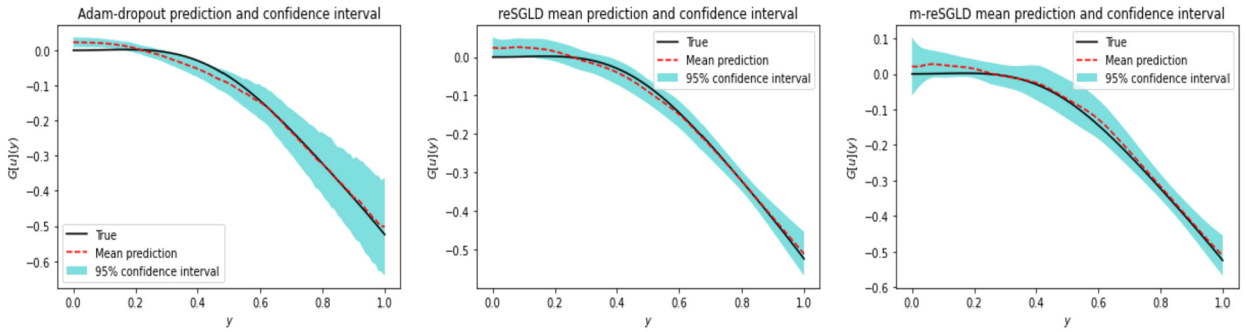

Fig. A.27. Confidence intervals for the gravity pendulum example with noise $\mathcal{N}(0, 0.05^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, the error e_3 for Adam is 77%, for reSGLD is 93%, and for m-reSGLD is 100%.

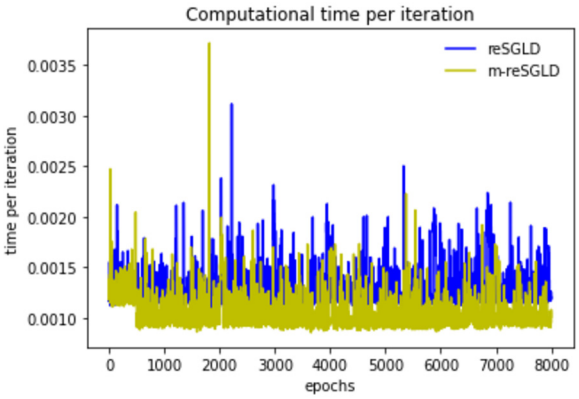

Fig. A.28. Gravity pendulum example with noise $\mathcal{N}(0, 0.05^2)$. Computer running time per iteration for reSGLD and m-reSGLD. The average computation time of m-reSGLD is around 80.8% of reSGLD framework.

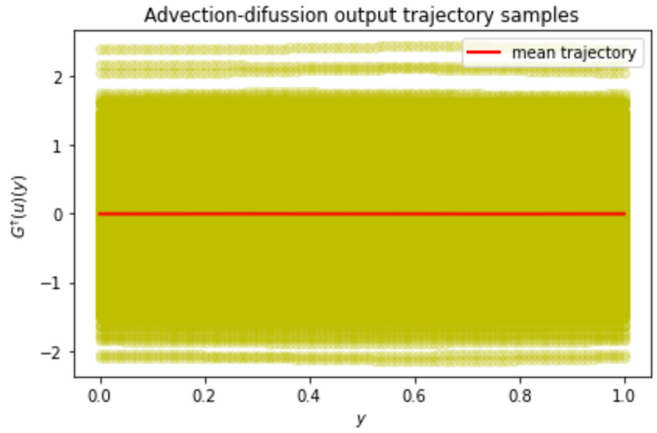

Fig. B.29. Distribution of the operator targets $G^\dagger(u)(y)$ obtained from the distribution of inputs depicted in Fig. 6 and used to train the advection-diffusion example.

where $\Omega = [0, 1]$, and $D = 0.1$. Here $u(x)$ is an initial condition sampled from the GRF with an RBF kernel of length scale $l = 0.2$ and discretized using $m = 100$ sensors. Fig. B.29 shows the distribution of the true operator targets associated with the input samples u.

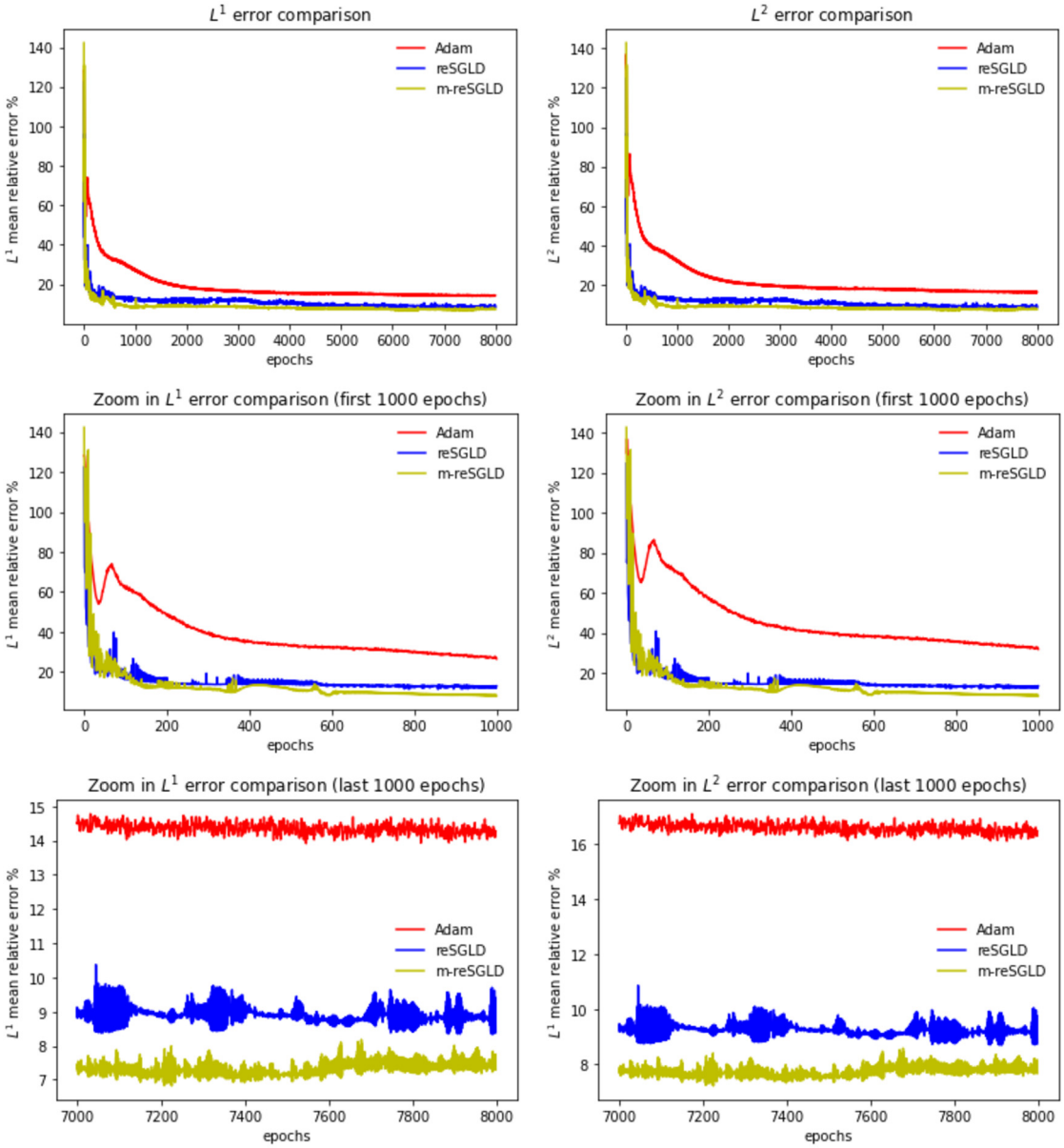

Fig. B.30. Convergence results for the advection-diffusion example with noise $\mathcal{N}(0, 0.01^2)$. The e_1 errors are shown on the left while e_2 errors are shown on the right. Top row: entire history. Middle row: zoom in the first $1,000$ epochs. Last row: zoom in the last $1,000$ epochs. The average errors after the burn in are presented in the Table B.9.

Small noise

In the small noise scenario, we assume the training operator targets are corrupted by a noise that follows the normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.01^2)$. We train the DeepONets for $8,000$ epochs. Fig. B.30 shows the convergence errors e_1 and e_2 and Table B.9 presents the average of these errors after $7,000$ burn-in epochs. From Fig. B.30 and Table B.9, we conclude that reSGLD and m-reSGLD have faster training converge and better mean predictive performance than Adam.

To illustrate how well our Bayesian DeepONet estimates the uncertainty, we select one test trajectory and plot, in Fig. B.31, the estimated confidence interval by the three frameworks. We observe that the three methods capture the true trajectory ($e_3 = 100\%$). Adam with dropout, however, seems to overestimate the uncertainty, yielding a wider confidence band.

We conclude this section by plotting (see Fig. B.32) the per-iteration time consumed by the proposed frameworks reSGLD and m-reSGLD. We observe that The average computation time of m-reSGLD is around 79.4% of the reSGLD framework. The

Table B.9
Advection-diffusion example with noise $\mathcal{N}(0, 0.01^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	14.3524	16.5843
reSGLD	8.9257	9.2734
m-reSGLD	7.3409	7.7345

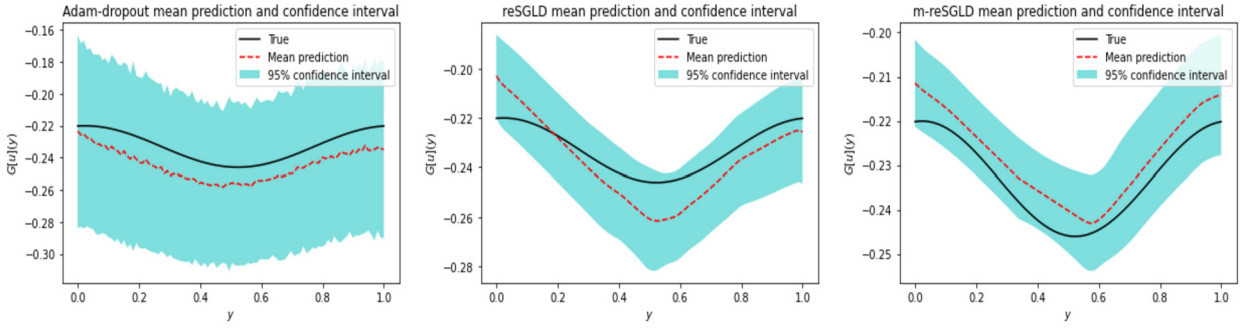

Fig. B.31. Confidence intervals for the advection-diffusion example with noise $\mathcal{N}(0, 0.01^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, the error e_3 for Adam is 100%, for reSGLD is 100%, and for m-reSGLD is 100%.

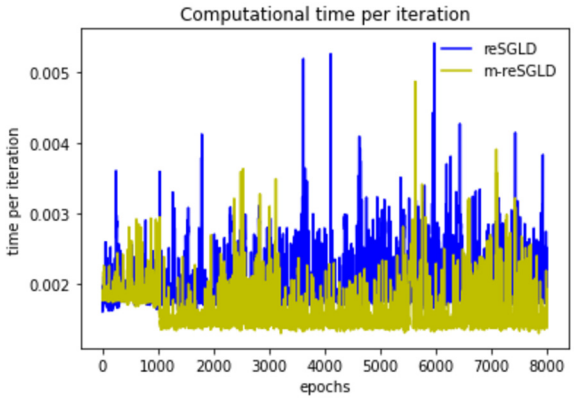

Fig. B.32. Advection-diffusion example with noise $\mathcal{N}(0, 0.01^2)$. Computer running time periteration for m-reSGLD and reSGLD. The average computation time of m-reSGLD is around 79.4% of reSGLD framework.

proposed accelerated training regime m-reSGLD achieves such a reduction even with improved mean predictive performance (see Table B.9).

Increased noise

Finally, we verify the performance of the proposed Bayesian DeepONet when the variance of the noise for the operator targets increases. In particular, for the advection-diffusion example, we assume the increased noise follows the normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.1^2)$. Fig. B.33 and Table B.10 detail the convergence errors for this increased noise scenario. Similar to our previous experiments, the proposed frameworks present improved training convergence and better mean predictive performance than Adam. This shows that the proposed frameworks handle noise more effectively.

We verify next how well the DeepONets capture uncertainty in this increased noise scenario. To this end, we select a test trajectory randomly and plot the estimated confidence intervals in Fig. B.34. The results show that, as expected, the proposed frameworks are more uncertain in this increased noise scenario, capturing the whole true test trajectory ($e_3 = 100\%$). On the other hand, Adam with dropout seems to provide the same uncertainty estimate as for the small noise scenario.

We also plot the per-iteration time used by re-SGLD and m-reSGLD in Fig. B.35. We observe that m-reSGLD is more efficient; it consumes only 80.2% of the computational cost of re-SGLD.

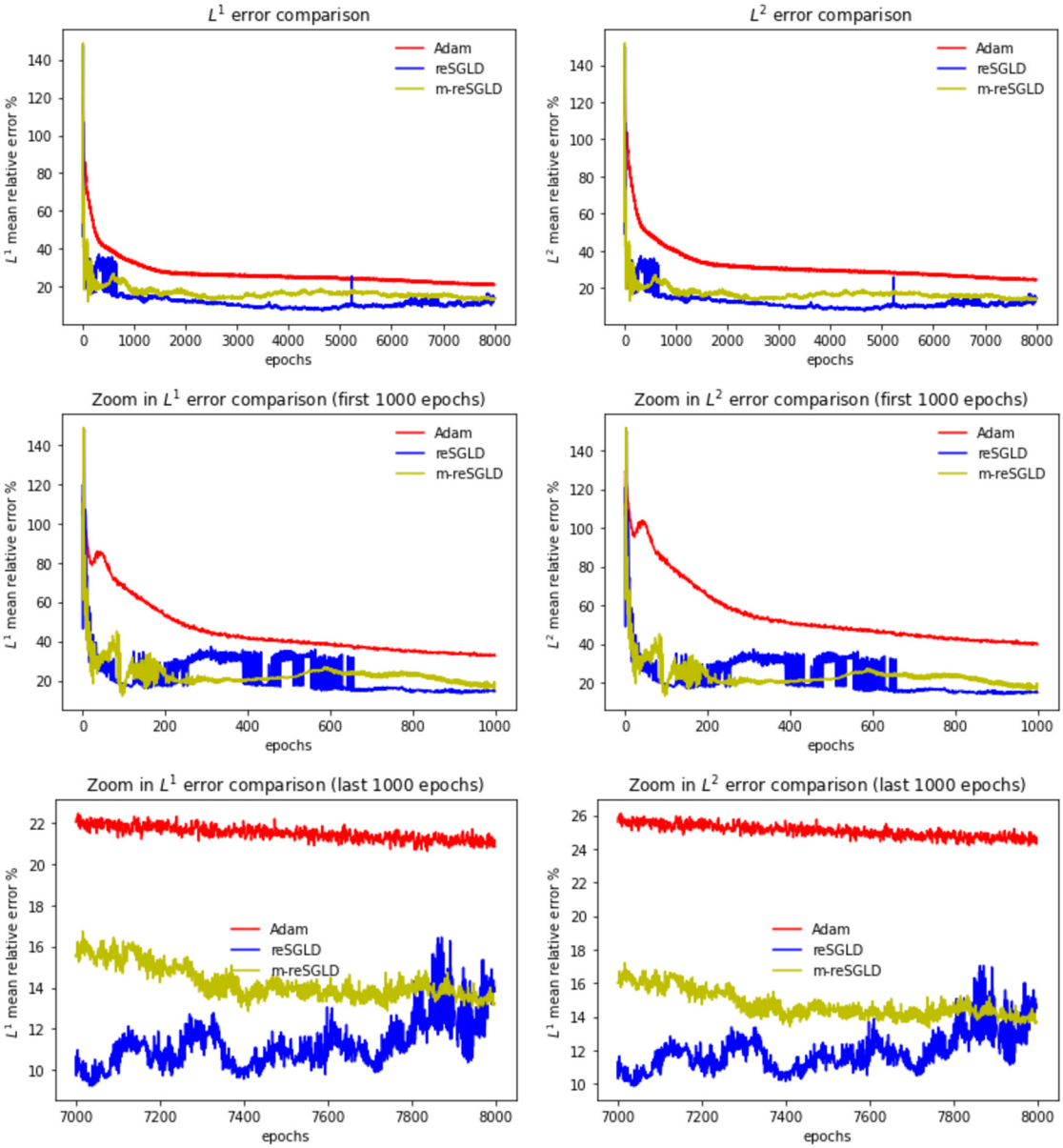

Fig. B.33. Convergence results for the advection-diffusion with increased noise $\mathcal{N}(0, 0.1^2)$. The e_1 errors are shown on the left while e_2 errors are shown on the right. Top row: entire history. Middle row: zoom in the first $1,000$ epochs. Last row: zoom in the last $1,000$ epochs. The average errors after the burn in are presented in the Table B.10.

Table B.10
Advection-diffusion example with increased noise $\mathcal{N}(0, 0.1^2)$. The mean errors after burn-in for all three frameworks.

frameworks	e_1	e_2
Adam	21.5141	25.0701
reSGLD	11.2455	11.9178
m-reSGLD	14.2553	14.7162

Appendix C. M-ensemble analysis

In this section, we select 100 test trajectories and analyze how well the proposed Bayesian DeepONet predicts the uncertainty (for the diffusion-reaction system) as a function of the ensemble size M. Fig. C.36 illustrates that the uncertainty prediction for both Bayesian frameworks deteriorates when $M \leq 2000$. For larger values of M, the performance always

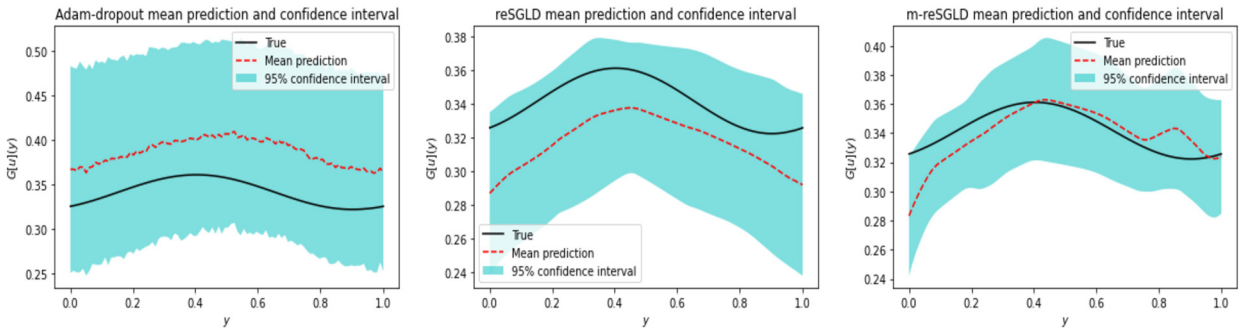

Fig. B.34. Confidence intervals for the advection-diffusion example with noise $\mathcal{N}(0, 1^2)$. Left: Adam. Middle: reSGLD. Right: m-reSGLD. In this example, the error e_3 for Adam is 100%, for reSGLD is 100%, and for m-reSGLD is 100%.

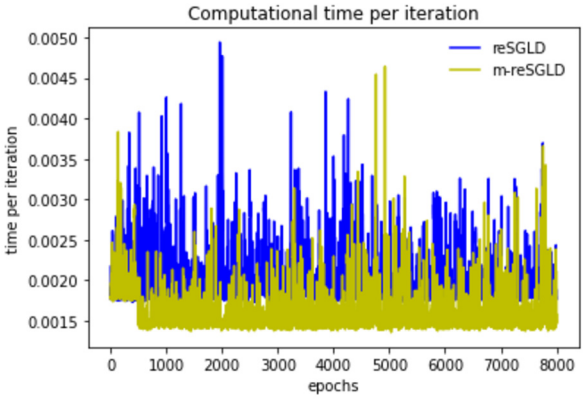

Fig. B.35. Advection-diffusion example with noise $\mathcal{N}(0, 0.1^2)$. Computer running time per iteration for m-reSGLD and reSGLD. The average computation time of m-reSGLD is around 80.2% of reSGLD framework.

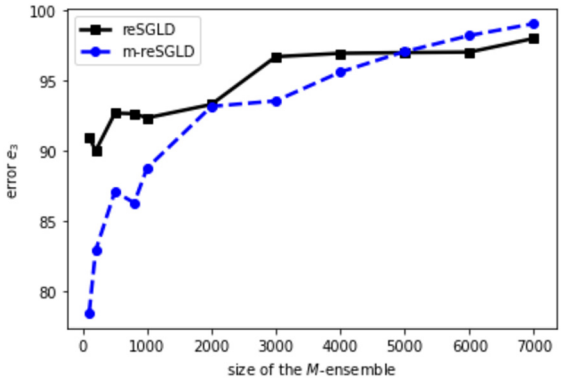

Fig. C.36. The fraction of the true solution trajectories captured by the constructed confidence interval (error e_3) as a function of the size M of the ensemble.

improves. However, if the ensemble size increases, the computational cost of inference also increases. As a result, in all our experiments, we selected $M = 2000$, which balances uncertainty prediction with inference cost.

References

[1] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model reduction and neural networks for parametric pdes, arXiv preprint, arXiv:2005.03180, 2020.

[2] S. Cai, Z. Wang, L. Lu, T.A. Zaki, G.E. Karniadakis, Deepm&mnet: inferring the electroconvection multiphysics fields based on operator approximation by neural networks, J. Comput. Phys. 436 (2021) 110296.

[3] 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 (1995) 911–917.
[4] T. Chen, E. Fox, C. Guestrin, Stochastic gradient Hamiltonian Monte Carlo, in: International Conference on Machine Learning, PMLR, 2014, pp. 1683–1691.
[5] Y. Chen, J. Chen, J. Dong, J. Peng, Z. Wang, Accelerating nonconvex learning via replica exchange Langevin diffusion, arXiv preprint, arXiv:2007.01990, 2020.
[6] B. Chetverushkin, E. Chung, Y. Efendiev, S.M. Pun, Z. Zhang, Computational multiscale methods for quasi-gas dynamic equations, J. Comput. Phys. 440 (2021) 110352.
[7] T.S. Chiang, C.R. Hwang, S.J. Sheu, Diffusion for global optimization in r^n, SIAM J. Control Optim. 25 (1987) 737–753.
[8] L. Chizat, E. Oyallon, F. Bach, On lazy training in differentiable programming, arXiv preprint, arXiv:1812.07956, 2018.
[9] E. Chung, Y. Efendiev, W.T. Leung, S.M. Pun, Z. Zhang, Multi-agent reinforcement learning accelerated mcmc on multiscale inversion problem, arXiv preprint, arXiv:2011.08954, 2020.
[10] E. Chung, Y. Efendiev, S.M. Pun, Z. Zhang, Computational multiscale methods for parabolic wave approximations in heterogeneous media, arXiv preprint, arXiv:2104.02283, 2021.
[11] E. Chung, W.T. Leung, S.M. Pun, Z. Zhang, A multi-stage deep learning based algorithm for multiscale model reduction, J. Comput. Appl. Math. 394 (2021) 113506.
[12] W. Deng, Q. Feng, L. Gao, F. Liang, G. Lin, Non-convex learning via replica exchange stochastic gradient mcmc, in: International Conference on Machine Learning, PMLR, 2020, pp. 2474–2483.
[13] S. Du, J. Lee, H. Li, L. Wang, X. Zhai, Gradient descent finds global minima of deep neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 1675–1685.
[14] S.S. Du, X. Zhai, B. Poczos, A. Singh, Gradient descent provably optimizes over-parameterized neural networks, arXiv preprint, arXiv:1810.02054, 2018.
[15] Y. Efendiev, W.T. Leung, G. Lin, Z. Zhang, Hei: hybrid explicit-implicit learning for multiscale problems, arXiv preprint, arXiv:2109.02147, 2021.
[16] S. Fresca, L. Dede, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes, J. Sci. Comput. 87 (2021) 1–36.
[17] Y. Gal, Z. Ghahramani, Dropout as a bayesian approximation: representing model uncertainty in deep learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1050–1059.
[18] C.R. Hwang, Laplace's method revisited: weak convergence of probability measures, Ann. Probab. (1980) 1177–1182.
[19] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, Eur. J. Appl. Math. 32 (2021) 421–435.
[20] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: learning maps between function spaces, arXiv preprint, arXiv:2108.08481, 2021.
[21] I.M.S. Leite, J.D.M. Yamim, L.G.d. Fonseca, The deeponets for finance: an approach to calibrate the Heston model, in: EPIA Conference on Artificial Intelligence, Springer, 2021, pp. 351–362.
[22] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, G.E. Karniadakis, Operator learning for predicting multiscale bubble growth dynamics, J. Chem. Phys. 154 (2021) 104118.
[23] G. Lin, Y. Wang, Z. Zhang, Multi-variance replica exchange stochastic gradient mcmc for inverse and forward bayesian physics-informed neural network, arXiv preprint, arXiv:2107.06330, 2021.
[24] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.
[25] M. Welling, Y.W. Teh, Bayesian learning via stochastic gradient Langevin dynamics, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), Citeseer, 2011, pp. 681–688.
[26] D. Zhang, L. Lu, L. Guo, G.E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.
[27] Y. Zhang, P. Liang, M. Charikar, A hitting time analysis of stochastic gradient Langevin dynamics, in: Conference on Learning Theory, PMLR, 2017, pp. 1980–2022.
[28] Z. Zhang, E.T. Chung, Y. Efendiev, W.T. Leung, Learning algorithms for coarsening uncertainty space and applications to multiscale simulations, Mathematics 8 (2020) 720.