Journal of Computational Physics 436 (2021) 110296

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks

Shengze Cai [a,*,1], Zhicheng Wang [a,1], Lu Lu [b], Tamer A. Zaki [c], George Em Karniadakis [a,*]

[a] Division of Applied Mathematics, Brown University, Providence, RI, 02912, USA
[b] Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA
[c] Department of Mechanical Engineering, Johns Hopkins University, Baltimore, MD, 21218, USA

ARTICLE INFO

Article history:
Available online 22 March 2021

Keywords:
Mutiphysics
Multiscale modeling
Deep learning
Data assimilation
Operator approximation
DeepONet

ABSTRACT

Electroconvection is a multiphysics problem involving coupling of the flow field with the electric field as well as the cation and anion concentration fields. Here, we use electroconvection as a benchmark problem to put forward a new data assimilation framework, the DeepM&Mnet, for simulating multiphysics and multiscale problems at speeds much faster than standard numerical methods using pre-trained neural networks. We first pre-train DeepONets that can predict independently each field, given general inputs from the rest of the fields of the coupled system. DeepONets can approximate nonlinear operators and are composed of two sub-networks, a *branch net* for the input fields and a *trunk net* for the locations of the output field. DeepONets, which are extremely fast, are used as building blocks in the DeepM&Mnet and form constraints for the multiphysics solution along with some sparse available measurements of any of the fields. We demonstrate the new methodology and document the accuracy of each individual DeepONet, and subsequently we present two different DeepM&Mnet architectures that infer accurately and efficiently 2D electroconvection fields for unseen electric potentials. The DeepM&Mnet framework is general and can be applied for building any complex multiphysics and multiscale models based on very few measurements using pre-trained DeepONets in a "plug-and-play" mode.

1. Introduction

Recently, deep learning techniques have been introduced in modeling diverse fluid mechanics problems [1–3]. A promising direction is focused on the surrogate modeling of complex fluid systems, which are typically described by partial differential equations (PDEs). For example, Raissi et al. [4] applied physics-informed neural networks (PINNs) to obtain accurate solutions of a wide class of PDEs. The governing equations are embedded in PINNs and form the constraints of the solutions. Similar approaches for scientific machine learning have been proposed in a number of recent works [5–7]. Following the framework of [4], a number of learning-based methods were proposed to simulate different types of fluid mechanics problems [8–11]. In addition to solving forward problems, neural networks are also used to identify the unknown coefficient

* Corresponding authors.
E-mail addresses: shengze_cai@brown.edu (S. Cai), george_karniadakis@brown.edu (G.E. Karniadakis).
[1] The authors contributed equally to this work.

https://doi.org/10.1016/j.jcp.2021.110296
0021-9991/

values of PDEs from data [12–16]. A typical example is to learn the Reynolds number of Navier-Stokes equations based on some measurements of flow [17,18]. This is a data-driven inverse problem, which is not easy to address by conventional computational fluid mechanics (CFD) methods.

Overall, the recent applications of deep learning to physics modeling are based on the universal approximation theorem stating that neural networks (NNs) can be used to approximate any continuous function [19]. However, there are other approximation theorems stating that a neural network can approximate accurately any continuous nonlinear functional [20] or operator (a mapping from a function to another function) [21]. Based on the universal approximation theorem for operator, Lu et al. [22] proposed a specific network architecture, the deep operator network (DeepONet), to learn both explicit as well as implicit operators, e.g., in the form of PDEs. DeepONet, which is designed based on rigorous theory, is composed of two sub-networks, the *branch net* for the input function and the *trunk net* for the coordinate to evaluate the output function. In [22], the authors presented theoretical analysis and demonstrated various examples to show the high accuracy and high efficiency of DeepONets. Inspired by the work of [22], here we first pose the related question: "Can DeepONet learn the mapping between two coupled state variables for complex fluid systems?" If this is the case, we ask: "Can the pre-trained DeepONets be combined in a simplified data assimilation framework, using a single NN, to accelerate significantly modeling of multiphysics and multiscale flow problems?"

In this paper, our first goal is to demonstrate the effectiveness of DeepONets for multiphysics and multiscale problems in fluid mechanics. To this end, we use electroconvection as a benchmark problem, which describes electrolyte flow driven by an electric voltage and involves multiphysics, i.e., the mass, momentum and cation/anion transport as well as electrostatic fields [23]. Electroconvection is observed in many electrochemical and microfluidic systems that usually have ion selective interfaces. Since the development of the theoretical formulations of electroconvection by [24,25], which were later confirmed by experiment in [26], a considerable number of numerical studies has been published [27–29,23]. More recently, several 3D interesting simulations have been presented in [30,31]. The main bottleneck of the simulation of electroconvection is the computational cost to resolve the electric double layer (EDL), the thickness of which is characterized by a dimensionless parameter, namely the Debye length (ϵ). In the aforementioned references, $\epsilon = 10^{-3}$ is generally used, which is small enough to simulate the unsteady flow patterns that exist in real experiments. Using a smaller Debye length (e.g., $\epsilon = 10^{-4}$) is not necessary as the main flow quantities (e.g., mean flow rate) are weakly sensitive to ϵ when it is small [30]. However, the flow of electroconvection will become steady when a larger value of Debye length (e.g., $\epsilon = 10^{-2}$) is employed. Here, we aim to present a proof-of-concept for a new deep learning framework and we only consider the steady-state electronvection (i.e., $\epsilon = 10^{-2}$) as a benchmark multiphysics problem in this paper.

To apply DeepONets to electroconvection, we train DeepONets to predict each field independently by giving inputs from the rest of the fields. Upon supervised training with labeled data, the DeepONets can predict very accurately and efficiently the target field. There are some existing learning-based methods that can perform similar input-output mappings. In particular, the most common algorithm is the convolutional neural networks (CNNs), which are used to learn mapping between two variables when the input and output functions are treated as images [32–35]. For example, Kim and Lee [35] used a CNN to predict the wall-normal heat flux from the wall-shear stresses and pressure. However, this kind of approach is limited to problems where the input and output are represented by 2D maps with equispaced grids. On the contrary, it is not necessary for DeepONets to include all the grid values inside the computational domain. The training data for DeepONets can be sparsely and randomly distributed, making it agnostic to the geometry.

In addition to the application of DeepONets, another objective in this paper is to put forward a new framework, the DeepM&Mnet, for simulating general multiphysics and multiscale problems. To use the DeepONets for prediction of an independent condition, at least one of the electroconvection fields is required. However, this is not realistic in general. Therefore, we develop the DeepM&Mnet, which can integrate several pre-trained DeepONets with a few measurements from any of the fields to produce the full fields of the coupled multiphysics system. To this end, we develop two architectures of DeepM&Mnet. One of them, the parallel DeepM&Mnet, is illustrated in Fig. 1. In the context of DeepM&Mnet, a neural network, which takes the spatial coordinates as inputs, is applied to approximate the multiphysics solutions. In other words, the neural network is the surrogate model of the multiphysics electroconvection, where the pre-trained DeepONets are used as building blocks and form the constraints for the solutions. Such proposed DeepM&Mnet can be considered as a framework assimilating the data measurements and the physics-related operators. We note that there are methods, such as PDE-constrained optimization with adjoint- or ensemble-variational techniques, that can address similar problems in diverse applications of fluid mechanics [36–41]. However, the DeepM&Mnet framework is much more efficient compared to them since it avoids solving the PDE system numerically, which is required in conventional data assimilation methods.

The paper is organized as follows. In Section 2, we introduce the numerical simulation of the electroconvection problem and the data preparation. In Section 3, we introduce the concept of DeepONet and train several DeepONets to predict separately each field accurately. In Section 4, we propose two architectures of DeepM&Mnet, namely the parallel DeepM&Mnet and series DeepM&Mnet, and then evaluate the performance of these new networks. Finally, we conclude the paper in Section 5. We also present some details on the implementations of DeepONet and DeepM&Mnet, e.g., the choice of the training loss, regularization issues, etc., in the appendices.

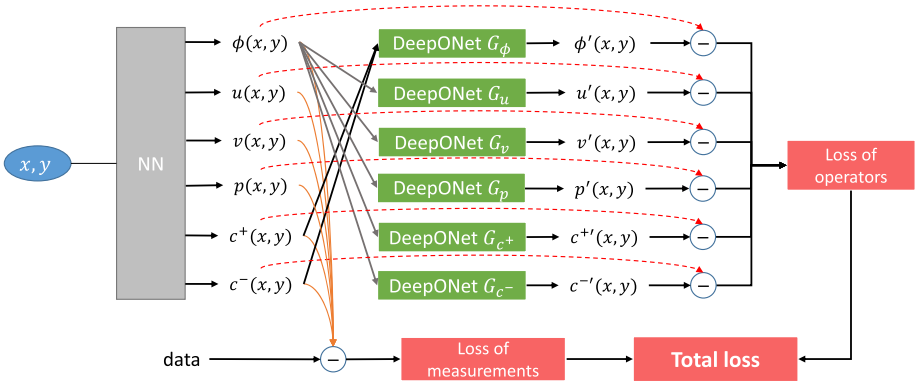


Fig. 1. DeepM&Mnet for 2D electroconvection problem: schematic of the parallel architecture. Here, the neural network (NN), which is used to approximate the multiphysics solutions, is a fully-connected network with trainable parameters. The pre-trained DeepONets are fixed (not trainable) and they are considered as the constraints of the NN outputs.

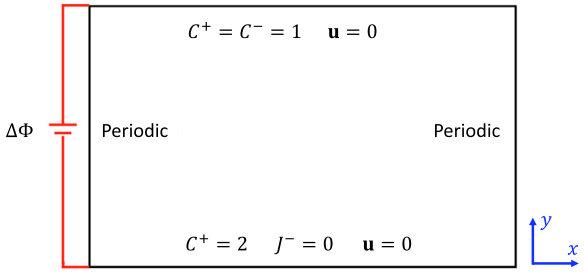


Fig. 2. Simulation domain and boundary conditions for 2D electroconvection.

2. Numerical simulation of electroconvection and data generation

We developed an eletroconvection solver using the high-order spectral element method [42]. Following the numerical study in [27], here the governing equations in dimensionless form, including the Stokes equations, the electric potential and the ion transport equations, are given as follows:

$$\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \nabla^2 \mathbf{u} + \mathbf{f_e}, \tag{1a}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1b}$$

$$-2\epsilon^2 \nabla^2 \phi = \rho_e, \tag{1c}$$

$$\frac{\partial c^{\pm}}{\partial t} = -\nabla \cdot \mathbf{J}^{\pm}, \tag{1d}$$

$$\mathbf{J}^{\pm} = c^{\pm}\mathbf{u} - \nabla c^{\pm} \mp c^{\pm} \nabla \phi. \tag{1e}$$

Here, $\mathbf{u}$ is the velocity vector, p is the pressure field and ϕ is the electric potential. Moreover, c^+ and c^- are the cation and anion concentrations, respectively. Also, $\rho_e = z(c^+ - c^-)$ is the free charge density with ionic valence $z = 1$, $f_e = -\kappa \rho_e \nabla \phi / 2\epsilon^2$ is the electrostatic body force, where ϵ is the Debye length; κ is the electrohydrodynamic coupling constant and its value is 0.5 throughout this paper.

We consider electroconvection between an ion-selective surface and a stationary reservoir subject to an applied voltage $\Delta\Phi$. As shown in the Fig. 2, the simulation domain is a rectangle of size $[-3, 3] \times [0, 1]$. On the top surface ($y = 1$), no-slip and no-penetration wall boundary condition ($\mathbf{u} = 0$) for the Stokes equations, constant ion concentrations ($c^+ = c^- = 1$) and constant potential ($\phi = \Delta\Phi$) are imposed. The bottom surface ($y = 0$) is also a solid wall, and it is assumed to be impermeable to anions ($J^- = 0$), but permeable to cations whose concentration is maintained at $c^+ = 2$. The potential on the bottom surface is $\phi = 0$. Periodic boundary conditions are used on both the right ($x = 3$) and left ($x = -3$) boundaries. The electroconvection solver is developed based on our spectral element library NekTar, where the spectral element is used to discretize the governing equations in space and the second-order stiffly-stable scheme is employed for time advancing [42].

To validate the spectral element electroconvection solver, we first compare our simulation results with those in the literature. Following [27], we performed nine cases of 2D simulations at a fixed Debye length $\epsilon = 10^{-3}$, while $\Delta\Phi = 25$, 30, 35, 40, 50, 55, 80, 100 and 120, respectively, where ϵ and $\Delta\Phi$ are the key parameters determining the flow of eletroconvection system. The computational domain consists of 48×64 quadrilateral elements, which are uniformly distributed in

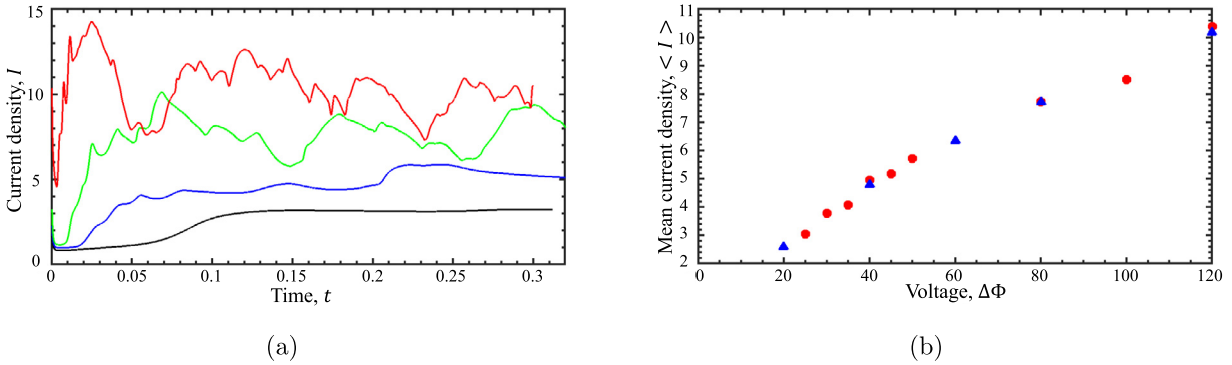


Fig. 3. Spectral element simulation results of the current density (I) on the top wall and validation against the simulations of [27]. (a) x-averaged I as a function of time (t): black line: applied voltage $\Delta\Phi = 25$; blue line: $\Delta\Phi = 40$; green line: $\Delta\Phi = 80$; red line: $\Delta\Phi = 120$. (b) Time and x-averaged $< I >$ as a function of $\Delta\Phi$: red circles: NekTar simulation; blue triangles: simulation results of [27]. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the x direction, but clustered in the vicinity of the walls. The smallest element size in the y direction is 0.0011. Same as the procedure in [27], the initial condition is generated by solving the governing equations without flow, and randomly perturbing the resulting concentration fields locally by 1%. Moreover, the time step is $\delta t = 5 \times 10^{-7}$, and 4 spectral element modes (3^{rd} order polynomials) and 7 quadrature points (per direction) are used in all the validation cases. All the simulations stopped after $t > 0.3$ and convergence was verified. The collection of the statistical samples began at $t = 0.15$ when the current on the top surface entered into a statistically stationary state. Fig. 3 shows the comparisons of current spectral element solutions to those of [27]. Same as the finding in [27], it can be observed from Fig. 3(a) that the current (I) on the top surface is steady at low voltage $\Delta\Phi = 25$, it becomes unsteady when $\Delta\Phi = 40$, and it shows chaotic behavior as the $\Delta\Phi$ is increased further. Quantitatively, the time averaged current $< I >$ is in good agreement with the results of [27], as shown in Fig. 3(b).

The simulations described above with $\epsilon = 10^{-3}$, which mostly involve unsteady flows for different $\Delta\Phi$, are used to validate our CFD solver. However, as mentioned before, we mainly focus on the steady-state electroconvection problem and use it to put forward the new data assimilation framework in this paper. To this end, we performed simulations at $\epsilon = 10^{-2}$ with $\Delta\Phi$ varying in the range $\Delta\Phi \in [5, 75]$ systematically, which form the training data for deep learning. The flow is steady under these conditions. Note that the computational domain is the same as that used in the validation, but the number of elements is reduced to 32×32. Also since a bigger ϵ is used, the smallest element size in y direction is increased to 0.0082, the time step is increased to $\delta t = 10^{-5}$, the number of spectral element modes is 3 and the number of quadrature points in each element in each direction is 5. All the simulations stopped at $t = 2.5$ when the time is sufficient long for the current on the top surface to reach steady state. For each simulation condition, we save the 2D snapshots of ϕ, u, v, p and $c^{\pm}$ at time instant $t = 2.5$, which will be used as the training data.

3. DeepONets for 2D electroconvection

3.1. The building blocks: DeepONets

In this section, we describe the DeepONets that will be used as the building blocks in DeepM&Mnets. The DeepONet was proposed in [22] for learning general nonlinear operators, including different types of PDEs. Let G be an operator mapping from a space of functions to another space of functions. In this study, G is represented by a DeepONet, which takes inputs composed of two components: a function U and the location points (x, y), and outputs $G(U)(x, y)$. Note that the selection of input and output functions for a DeepONet is problem-dependent. In this paper, we design six independent DeepONets according to the physics of electroconvection, which can be divided into two classes. The first one takes the electric potential $\phi(x, y)$ as input and predicts the velocity, pressure and concentration fields (u, v, p, c^+, c^-), which are denoted as G_u, G_v, G_p, G_{c^+} and G_{c^-}, respectively. These operators are proposed based on the steady-state formulations of Equ. (1a) and (1e), i.e., the time derivatives of the equations are zero. In this case, the electric potential specifies the force term, which determines the transport patterns of the flow and the ion. The second one, represented by G_ϕ, is used to predict ϕ by using the cation and anion concentrations, which is determined by Equ. (1c) where ϕ can be solved when c^+ and c^- are given. The definitions of these DeepONets are illustrated in Table 1.

In this paper, we apply the "unstacked" architecture for DeepONet, which is composed of a branch network and a trunk network. DeepONets are implemented in DeepXDE [43], a user-friendly Python library designed for scientific machine learning. The schematic diagrams of the proposed networks are illustrated in Fig. 4. In this framework, the trunk network takes the coordinates (x, y) as input and outputs $[t_1, t_2, \cdots, t_p]^T \in \mathbb{R}^p$. In addition, the input function, which is represented by m discrete values (e.g., $[\phi(x_1, y_1), \cdots, \phi(x_m, y_m)]^T$), is fed into the branch network. Then the two vectors from the branch and trunk nets are merged together via a dot product to obtain the output function value. We use the fully-connected

Table 1
DeepONets for 2D electroconvection: definitions of the 2D operators. We propose these operators based on the governing equations of the investigated problem. For example, the electric potential ϕ can be solved when c^+ and c^- are given, according to Equ. (1c).

DeepONets	Input function	Output function
G_ϕ	$c^+(x,y), c^-(x,y)$	$\phi(x,y)$
G_u	$\phi(x,y)$	$u(x,y)$
G_v	$\phi(x,y)$	$v(x,y)$
G_p	$\phi(x,y)$	$p(x,y)$
G_{c^+}	$\phi(x,y)$	$c^+(x,y)$
G_{c^-}	$\phi(x,y)$	$c^-(x,y)$

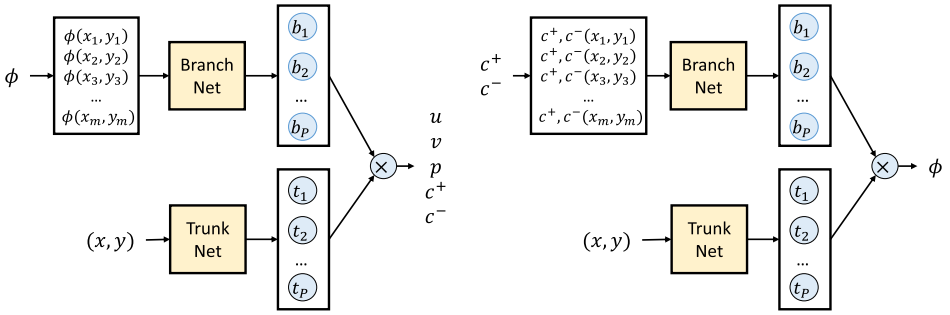


Fig. 4. DeepONets for 2D electroconvection: schematics of the DeepONet architectures. Left: using the electric potential to predict the velocity, pressure, cation and anion concentrations, i.e., $G_{u,v,p,c^+,c^-} : \phi \to [u, v, p, c^+, c^-]$; right: using the concentrations to predict the electric potential, i.e., $G_\phi : [c^+, c^-] \to \phi$.

Table 2
DeepONets for 2D electroconvection: architectures and loss functions of the DeepONets. MSE: mean squared error; MAPE: mean absolute percentage error.

DeepONets	Branch depth	Branch width	Trunk depth	Trunk width	Loss
G_ϕ	2	[300, 200]	3	[100, 200, 200]	MSE
G_u	2	[200, 200]	3	[100, 200, 200]	MAPE
G_v	2	[200, 200]	3	[100, 200, 200]	MAPE
G_p	2	[200, 200]	3	[100, 200, 200]	MSE
G_{c^+}	2	[200, 200]	3	[100, 200, 200]	MSE
G_{c^-}	2	[200, 200]	3	[100, 200, 200]	MSE

networks as the sub-networks (i.e., trunk and branch nets) in this study. The numbers of hidden layers and the number of neurons per layer of these sub-networks are given in Table 2.

The DeepONets are trained by minimizing a loss function, which measures the difference between the labels and NN predictions. In general, the mean squared error (MSE) is applied:

$$\textbf{MSE} = \frac{1}{N}\sum_{i=1}^{N}(V_i - \hat{V}_i)^2, \tag{2}$$

where V represents the predicting variable, V_i and $\hat{V}_i$ are the labeled data and the prediction, respectively; N is the number of training data. Alternatively, the mean absolute percentage error (MAPE) is also considered in the paper:

$$\textbf{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|V_i - \hat{V}_i|}{|V_i| + \eta}, \tag{3}$$

where η is a small number to guarantee the stability when $V_i = 0$. The MSE loss works well in most cases, while the MAPE loss is better for the case where the output has a large range of function values. In this paper, we apply the MSE loss to the training of G_{ϕ,p,c^+,c^-} and apply the MAPE loss for $G_{u,v}$. A comparison between MSE and MAPE applied to G_u is demonstrated in Appendix A, showing the advantage of MAPE loss in dealing with multiscale data.

3.2. Training of DeepONets

The training of DeepONets requires datasets with labeled outputs. We apply NekTar to simulate the 2D steady state fields of electroconvection problem. The computational domain is defined as Ω: $x \in [-1, 1]$, $y \in [0, 1]$. As mentioned in Section 2,

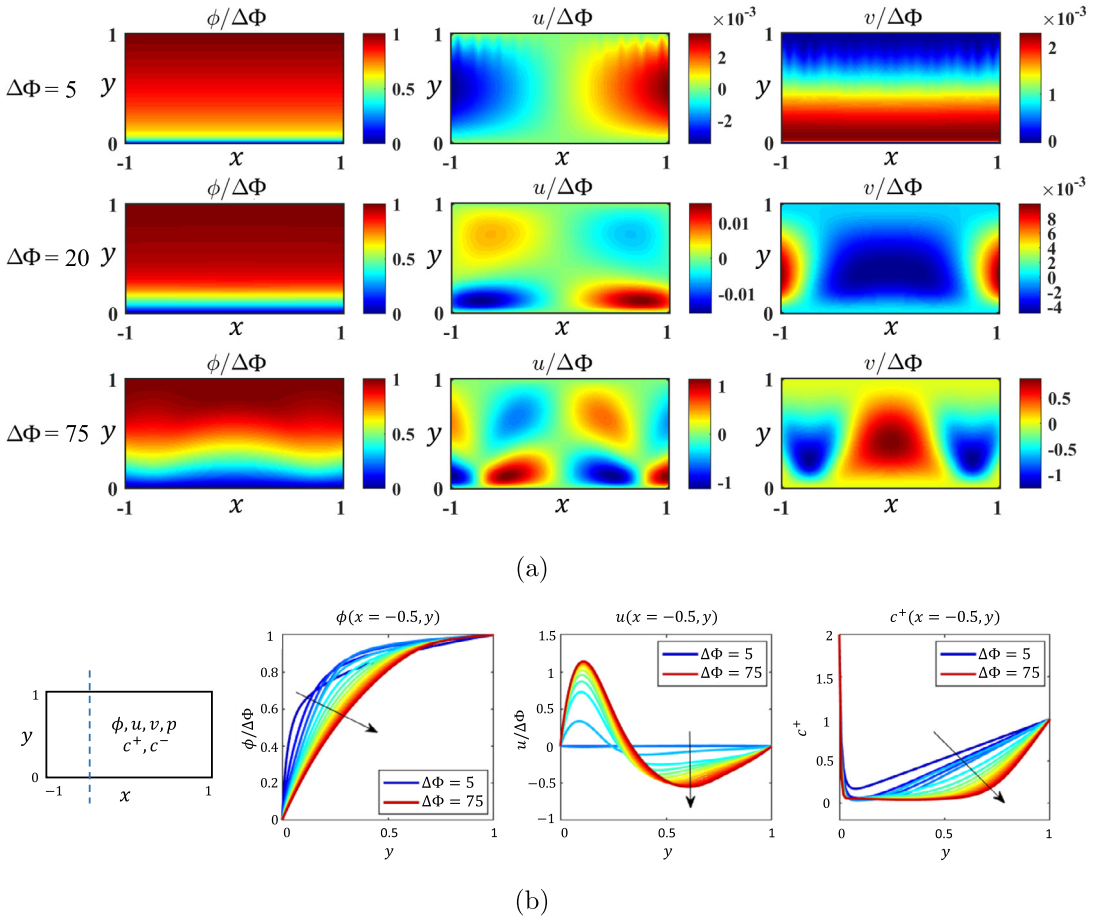


Fig. 5. DeepONets for 2D electroconvection: demonstration of the training data. (a) 2D steady fields of ϕ, u and v under various boundary conditions, i.e., $\Delta\Phi = 5, 20, 75$; (b) 1D profiles of $\phi(x=-0.5, y)$, $u(x=-0.5, y)$ and $c^+(x=-0.5, y)$ for 15 training states with $\Delta\Phi = 5, 10, \ldots, 75$. The arrow represents the increasing direction of $\Delta\Phi$.

different steady-state patterns can be produced by modifying the electric potential difference $\Delta\Phi$ between the boundaries. The flow fields of $\phi(x, y)$, $c^+(x, y)$, $c^-(x, y)$ and $u(x, y)$, $v(x, y)$, $p(x, y)$ are collected. By modifying the boundary conditions of ϕ, namely using $\Delta\Phi = 5, 10, \ldots, 75$, we generate 15 steady states for this electroconvection problem. The 2D snapshots at various values of $\Delta\Phi$ and some 1D profiles of $\phi(x=-0.5, y)$, $u(x=-0.5, y)$, $c^+(x=-0.5, y)$ are demonstrated in Fig. 5. Note that throughout this paper, the data are dimensionless and normalized for enhanced stability in the DeepONet training. In particular,

$$\phi = \phi/\Delta\Phi, \quad u = u/\Delta\Phi, \quad v = v/\Delta\Phi, \quad p = p \times 2\epsilon^2, \quad c^{\pm} = c^{\pm},$$

where $\Delta\Phi$ and ϵ are *a priori* known. As shown in Fig. 5, we can observe that the flow pattern varies significantly with different $\Delta\Phi$. Moreover, the magnitude of normalized velocity ranges from the order of 10^{-4} to 10^{0}, showing the multiscale nature of this electroconvection problem.

For each 2D input field, we have 21×11 uniformly-distributed sensors to represent the function. As for the corresponding output fields, we randomly select 800 data points in the space for each state variable. In this context, we have $N = 15 \times 800 = 12000$ training data points in all, where one data item is a triplet. For example, for the DeepONet G_u, one data item is $[\phi, (x, y), u(x, y)]$. It is worth noting again, the training data, which are not necessarily located on an equispaced grid, can be selected randomly in the computational domain. Moreover, it is possible to include the sparse experimental measurements from sensors in the dataset. These advantages show the flexibility when preparing the data for DeepONets training. We also use NekTar to generate fields under two additional conditions, namely $\Delta\Phi = 13.4$ and $\Delta\Phi = 62.15$, which are not included in the training dataset and are used for testing and validation.

To train the DeepONets, we apply the Adam optimizer with a small learning rate 2×10^{-4}, and the networks are trained over 500,000 iterations. The activation function of the neural network is ReLU. The losses of the training data and testing data during training process are presented in Fig. 6, where we can find the losses all converge to small values. Note that we

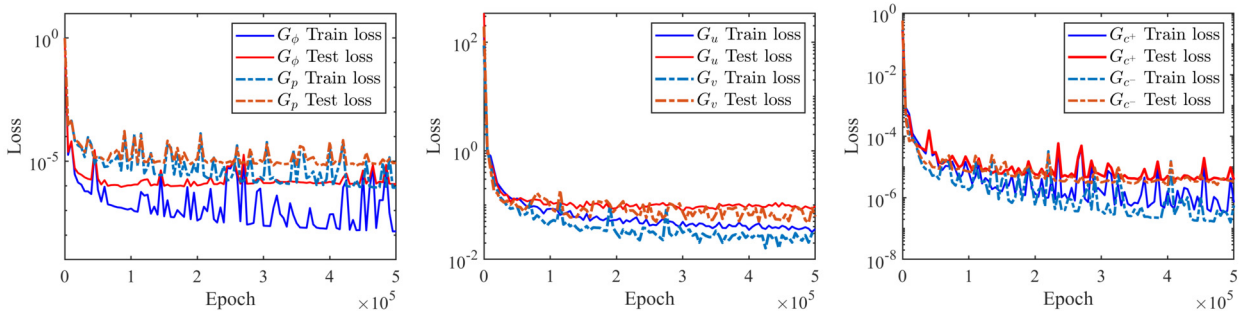


Fig. 6. DeepONets for 2D electroconvection: training and testing losses of G_{ϕ,u,v,p,c^+,c^-}. Note that we use MSE loss for training G_{ϕ,p,c^+,c^-} and MAPE loss for $G_{u,v}$, thus the magnitudes of the loss functions are different.

apply MAPE loss function to $G_{u,v}$, and thus the magnitude of their losses is different from the others. Upon training, these DeepONets can predict all fields accurately when the input functions are given.

3.3. Testing of DeepONets

Here, we demonstrate the predictions of the well-trained DeepONets for the new case (not included in the training set) corresponding to $\Delta\Phi = 62.15$. The results of G_u, G_v, G_p, G_{c^+} and G_{c^-} are shown in Fig. 7. In this figure, the input function $\phi(x, y)$, which is represented by a set of values at the sensor locations, as shown in the left column, is fed into the networks. We find that the DeepONet predictions are in good agreement with the fields generated by NekTar simulation. The MSEs for different state variables are on the order of $\mathcal{O}(10^{-6} - 10^{-5})$. On the other hand, for testing the DeepONet G_ϕ, the inputs c^+ and c^- are provided. The prediction of ϕ is illustrated in Fig. 8.

The prediction errors for both $\Delta\Phi = 13.4$ and $\Delta\Phi = 62.15$ are given in Table 3. In addition to MSEs, we also compute the relative L_2-norm errors, namely:

$$\epsilon_V = \| V - \hat{V} \|_2 / \| V \|_2 . \tag{4}$$

As shown, the relative errors of G_{ϕ,p,c^+,c^-} are less than 1%, indicating that the predictions of these networks are very accurate. We find that in the case of $\Delta\Phi = 13.4$, the relative L_2 errors of the velocity fields (u and v) are much larger than the others. The reason is that the maximal velocity value is almost zero ($\mathcal{O}(10^{-4})$) in the case of $\Delta\Phi = 13.4$, thus the relative error is very sensitive to a minor difference. In fact, the predictions of DeepONets G_u and G_v are very close to the references, which can be seen from the small MSEs in the table. However, this is not the case for $\Delta\Phi = 62.15$ since the magnitude of velocity is much larger, resulting in a larger MSE and a smaller relative error. To reduce the effect of this multiscale nature, we apply the MAPE loss when training G_u and G_v. A comparison between the MSE and MAPE applied to G_u is demonstrated in Appendix A.

We should note that once the DeepONets have been well trained, the prediction is a simple network evaluation task that can be performed very efficiently. In the 2D electroconvection problem, the training of a DeepONet requires approximately 2 hours. However, the well-trained DeepONets take less than 1 second (on any common CPU/GPU) to predict the 2D fields when the input functions are provided. The speedup of DeepONets prediction versus the NekTar simulation for independent conditions is about 1,000 folds.

4. DeepM&Mnet for 2D electroconvection

In order to use the pre-trained DeepONets, the input function should be given. For example, the proper electric potential $\phi(x, y)$ should be provided to G_u to obtain the u-velocity. However, this is not realistic in general. Therefore, we develop a new framework called DeepM&Mnet, which allows us to infer the full fields of the coupled electroconvection problem when only a few measurements for any of the state variables are available. In the context of DeepM&Mnet, a neural network is used to approximate the solutions of the electroconvection, while the pre-trained DeepONets are applied as the constraints of the solutions. In the following, we introduce two architectures of the proposed DeepM&Mnet in Section 4.1 and present the evaluation results in Section 4.2 as well as the discussion in Section 4.3.

4.1. DeepM&Mnet architectures

4.1.1. Parallel DeepM&Mnet architecture

A schematic diagram of the parallel DeepM&Mnet architecture is illustrated in Fig. 1. In this context, a fully-connected network with trainable parameters is used to approximate the coupling solutions. This is an ill-posed problem since only a few measurements are available. Therefore, regularization is required to avoid overfitting. In this work, the pre-trained

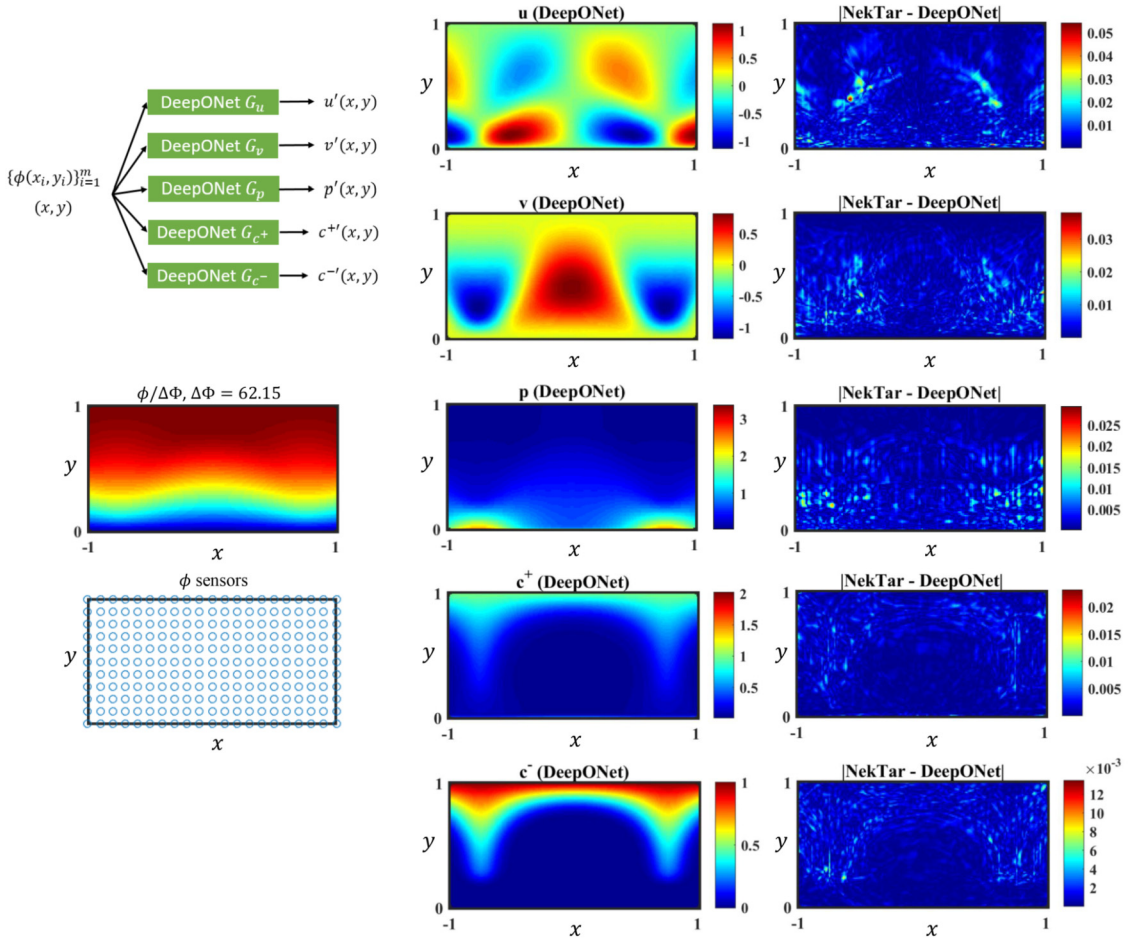


Fig. 7. DeepONets (upper left) for 2D electroconvection: predictions of u, v, p, c^+ and c^- (from top to bottom) using DeepONets G_u, G_v, G_p, G_{c^+} and G_{c^-} for a new case corresponding to $\Delta\Phi = 62.15$. Note that ϕ, the input of these networks, is represented by a set of values, and (u, v, p, c^+, c^-) are the outputs of five independent DeepONets. For comparison, we generate the reference fields by using NekTar simulations and plot the absolute errors (right column).

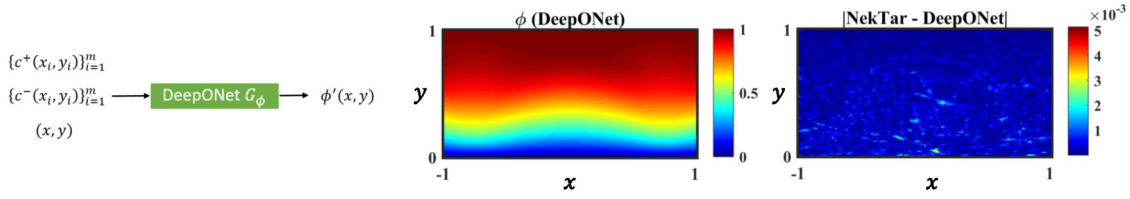


Fig. 8. A DeepONet for 2D electroconvection: predictions of ϕ using the DeepONet G_ϕ in the case of $\Delta\Phi = 62.15$. Note that c^+ and c^- (represented by discrete values) are the inputs of this network, and $\phi(x, y)$ is the output. For comparison, we generate the reference field by using NekTar simulation and plot the absolute error (right plot).

DeepONets are applied to deal with this problem. The pre-trained DeepONets are fixed (not trainable) and considered as the constraints of the NN outputs, which can be seen in Fig. 1. The neural network, which takes (x, y) as inputs and outputs $(\phi, u, v, p, c^+, c^-)$, is trained by minimizing the following loss function:

$$\arg\min_\theta \mathcal{L} = \lambda_d \mathcal{L}_{data} + \lambda_o \mathcal{L}_{op} + \lambda_r \mathcal{L}_2(\theta), \tag{5}$$

where λ_d, λ_o and λ_r are the weighting coefficients of the loss function; $\mathcal{L}_2(\theta) = \| \theta \|_2^2$ is the L_2 regularization of the trainable parameters θ, which can help avoid overfitting and stabilize the training process[2]; and

[2] The influence of the L_2 regularization term for DeepM&Mnet is investigated in Appendix B.

Table 3
DeepONets for 2D electroconvection: MSEs and relative L_2-norm errors of the DeepONet predictions for two testing examples. In the case of $\Delta\Phi = 13.4$, the relative L_2-norm errors of the velocities (u and v) are much larger than the others because the magnitude of the velocity in this case is almost zero ($\mathcal{O}(10^{-4})$).

DeepONets	$\Delta\Phi = 13.4$		$\Delta\Phi = 62.15$	
	MSE	relative L_2 error	MSE	relative L_2 error
G_ϕ	3.85×10^{-6}	0.23%	1.53×10^{-7}	0.05%
G_u	1.92×10^{-9}	8.54%	3.89×10^{-5}	1.65%
G_v	2.55×10^{-9}	10.01%	1.51×10^{-5}	0.86%
G_p	5.96×10^{-6}	0.92%	1.01×10^{-5}	0.52%
G_{c^+}	6.59×10^{-6}	0.49%	1.29×10^{-6}	0.31%
G_{c^-}	4.20×10^{-6}	0.36%	3.64×10^{-6}	0.45%

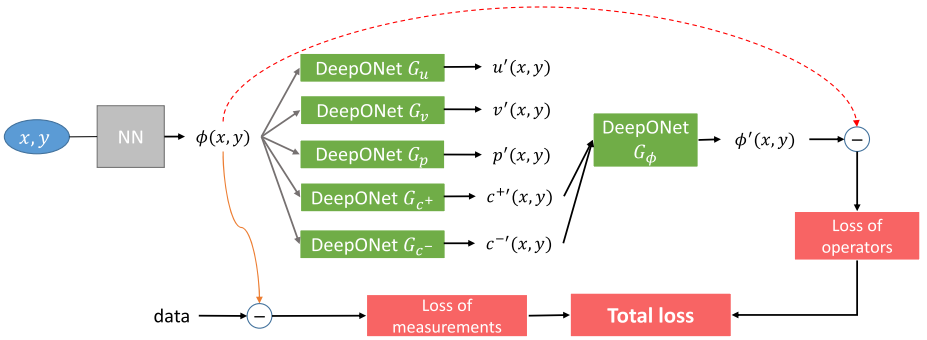


Fig. 9. DeepM&Mnet for 2D electroconvection problem: schematic of the series architecture. Here, the neural network (NN), which is used to approximate only one solution (i.e., ϕ), is a fully-connected network with trainable parameters. The other variables (i.e., u, v, p, c^+, c^-) are hidden outputs in this framework. The pretrained DeepONets are fixed (not trainable) and they are considered as the constraints of the NN outputs. The input of each DeepONet involves the input function (for the branch net) and the coordinates of the output function (for the trunk net). For simplicity, the coordinates feeding into each DeepONet are not shown in the figure.

$$\mathcal{L}_{data} = \sum_{V \in \{\phi, u, v, p, c^+, c^-\}} \frac{1}{N_d} \sum_{i=1}^{N_d} (V(x^i, y^i) - V_{data}(x^i, y^i)),$$

$$\mathcal{L}_{op} = \sum_{V \in \{\phi, u, v, p, c^+, c^-\}} \frac{1}{N_{op}} \sum_{i=1}^{N_{op}} (V(x^i, y^i) - V'(x^i, y^i)),$$

where $\mathcal{L}_{data}$ is the data mismatch and $\mathcal{L}_{op}$ is the difference between the neural network outputs and the DeepONets outputs. V can be any variables of the investigated solutions $(\phi, u, v, p, c^+, c^-)$; $V(x^i, y^i)$ accounts for the output of the fully-connected network, while $V'(x^i, y^i)$ is the output of DeepONet. N_d and N_{op} denote the number of measurements for each variable and the number of points for evaluating the operators, respectively. Here, we would like to add some comments on DeepM&Mnet. First, it is not necessary to have measurements for every variable. For example, if measurements are only available for ϕ, the DeepONets G_u, G_v, G_p, G_{c^+} and G_{c^-} can provide constraints for the other variables and guide the NN outputs to the correct solutions. Second, in the framework of parallel DeepM&Mnet (Fig. 1), we have not only the outputs of the fully-connected network $V \in \{\phi, u, v, p, c^+, c^-\}$, but also the outputs of DeepONets $V' \in \{\phi', u', v', p', c^{+'}, c^{-'}\}$. Ideally, V and V' should converge to the same values. However, there is bias between V and V' due to the approximation error of the pre-trained DeepONets and the optimization error of the neural network. Moreover, we note that the input of each DeepONet involves the input function (for the branch net) and the coordinates of the output function (for the trunk net). For simplicity, the coordinates feeding into each DeepONet are not shown in Fig. 1.

4.1.2. Series DeepM&Mnet architecture

In addition to the parallel architecture, we also propose a series architecture of DeepM&Mnet, which is illustrated in Fig. 9. In the series setup, the fully-connected network is only used to approximate ϕ. The other variables (i.e., u, v, p, c^+, c^-) are the hidden outputs in this framework and given by the DeepONets based on the result of ϕ. Moreover, with the pre-trained DeepONet G_ϕ, we can generate ϕ' from $(c^{+'}, c^{-'})$. The loss function is similar to (5). However, here we assume that we only have the measurements of ϕ, and $\mathcal{L}_{op}$ only contains the ϕ operator, thus:

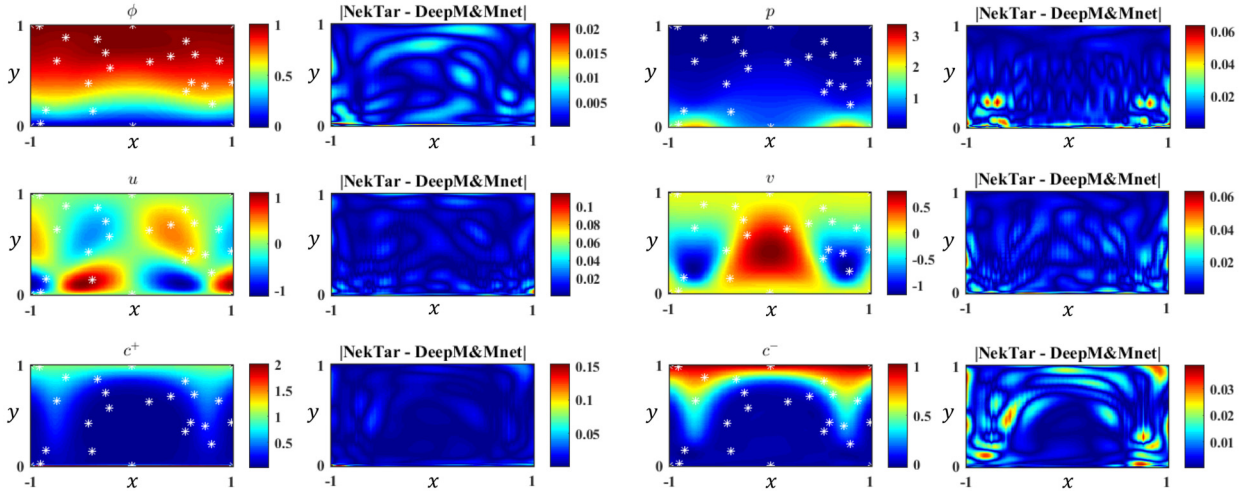


Fig. 10. DeepM&Mnet for 2D electroconvection problem: results of parallel architecture. In this case, 20 measurements of each field inside the domain are provided, which are marked with $*$. Left: results of the NN outputs (i.e., ϕ, u, v, p, c^+, c^-), right: point-wise difference.

$$\mathcal{L}_{data} = \frac{1}{N_d}\sum_{i=1}^{N_d}(\phi(x^i, y^i) - \phi_{data}(x^i, y^i)),$$

$$\mathcal{L}_{op} = \frac{1}{N_{op}}\sum_{i=1}^{N_{op}}(\phi(x^i, y^i) - \phi'(x^i, y^i)).$$

Different from the parallel architecture, in the series DeepM&Mnet, V only represents ϕ, while $V' \in \{\phi', u', v', p', c^{+'}, c^{-'}\}$. This framework shows that given a few measurements of ϕ, the neural network can produce the full field of ϕ. All other fields can be obtained by the pre-trained DeepONets inside the loop.

4.2. DeepM&Mnet testing results

We first investigate the performance of DeepM&Mnet for the 2D electroconvection problem, where the data is generated by NekTar with $\Delta\Phi = 62.15$. This corresponds to an input not seen by the pre-trained DeepONets, which are the building blocks of DeepM&Mnet. First, 20 measurements of each field inside the domain and six boundary points are provided for the parallel architecture. The neural network contains 6 hidden layers and 100 neurons per layer. The hyperbolic tangent function is applied as the activation function. The neural network is trained by the Adam optimizer for 60,000 iterations with a learning rate 5×10^{-4}. For the loss function, we set $\lambda_d = 10$, $\lambda_o = 1$ and $\lambda_r = 10^{-4}$. To evaluate the operator loss, we randomly choose 1000 points inside the domain, namely $N_{op} = 1000$. The resulting fields of one typical training process are demonstrated in Fig. 10, where point-wise differences between the NN outputs and the NekTar simulations are also given. The relative L_2-norm errors of V are: $\epsilon_\phi = 0.47\%, \epsilon_u = 2.83\%, \epsilon_v = 1.95\%, \epsilon_p = 1.66\%, \epsilon_{c^+} = 2.18\%, \epsilon_{c^-} = 2.53\%$. In addition, we can obtain the solutions from the DeepONets outputs V', where the relative errors are: $\epsilon_{\phi'} = 0.54\%, \epsilon_{u'} = 1.26\%, \epsilon_{v'} = 0.54\%, \epsilon_{p'} = 0.93\%, \epsilon_{c^{+'}} = 0.67\%, \epsilon_{c^{-'}} = 0.48\%$. The DeepONets outputs are different from the NN outputs due to the approximation error of the pre-trained DeepONets and the optimization error of the neural network. However, we cannot conclude which one (V or V') is always better since such network training involves randomness. In our specific case, the electric potential ϕ seems to be easier to approximate than the other fields (i.e., $\epsilon_\phi < \epsilon_{u,v,p,c^+,c^-}$). Hence, the DeepONets, using ϕ as input, also have higher potential to provide the fields ($u', v', p', c^{+'}, c^{-'}$) with better accuracy.

Next, we also investigate the effect of the number of sensors (N_{data}). Fig. 11 shows the convergence of the relative L_2-norm errors with respect to the number of measurements. For comparison, we perform data fitting using a standard neural network for the same problem, in which the operator loss $\mathcal{L}_{op}$ is not considered, namely $\mathcal{L} = \lambda_d\mathcal{L}_{data} + \lambda_r\mathcal{L}_2(\theta)$. We can see from the figure that without the DeepONet constraints, the neural network requires hundreds of sensors to obtain good accuracy (less than 10%) for every state variable. However, only 10 measurements of each field are sufficient for DeepM&Mnet to achieve the same accuracy. Note that the sensors are randomly distributed inside the domain. Therefore, we perform 5 independent training processes with different sensor locations to reduce the effect of sensor selection, and then compute the mean errors and standard deviations. We find that the deviations (shown in shaded region in the figure) are mostly very small for all the fields inferred by DeepM&Mnet, indicating the robustness of the proposed algorithm.

Moreover, we also demonstrate the influence of the number of operator evaluation points, namely N_{op}. We use 20 measurements for each condition. The relative L_2-norm errors are demonstrated in Fig. 12, where we find that with more evaluation points for DeepONets, the results are more accurate since there are more constraints on the solutions (NN

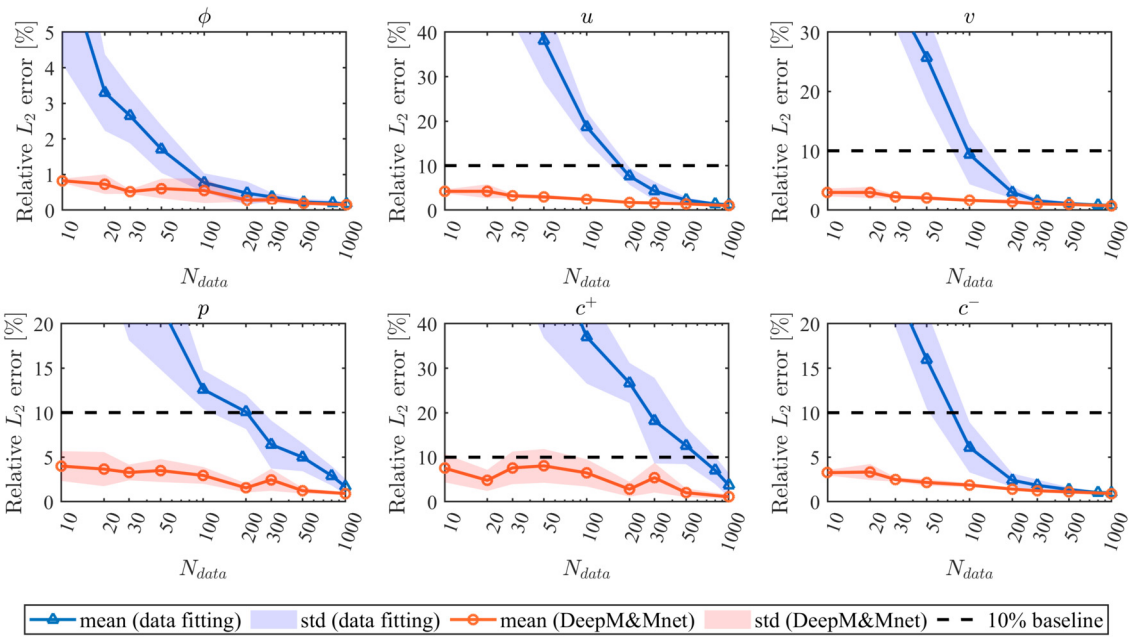


Fig. 11. DeepM&Mnet for 2D electroconvection problem: relative L_2-norm errors with respect to the number of measurements (N_{data}). The DeepM&Mnet results are from the neural network (namely $V \in \{\phi, u, v, p, c^+, c^-\}$) of the parallel architecture. For each setting, 5 independent training processes with randomly-distributed sensors are applied. The mean errors and the standard deviations are computed.

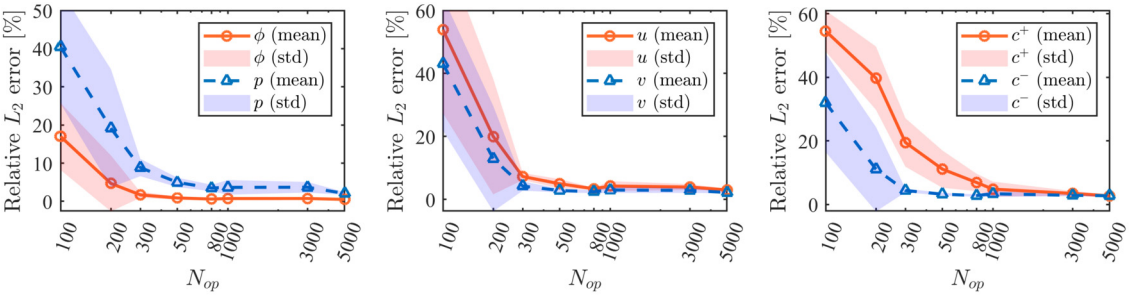


Fig. 12. DeepM&Mnet for 2D electroconvection problem: relative L_2-norm errors with respect to the number of operator evaluation points (N_{op}). The DeepM&Mnet results are from the neural network (namely $V \in \{\phi, u, v, p, c^+, c^-\}$) of the parallel architecture. For each setting, 5 independent training processes with randomly-distributed sensors are applied. The mean errors and the standard deviations are computed.

outputs). Similarly, 5 independent training processes are applied and the mean errors and standard deviations are given in the plots. A small value of N_{op} results in a significantly large standard deviation since the DeepONets constraints are weak and data overfitting phenomenon will occur.

For testing the *series architecture* of DeepM&Mnet, only the data of ϕ is given. An example of the resulting fields is shown in Fig. 13. In this case, 20 measurements of ϕ are provided and $N_{op} = 1000$; the neural network consists of 6 layers and 100 neurons per layer. The results of the other fields can be generated from the pre-trained DeepONets. As shown in the figure, the outputs are very consistent with the simulation. The relative L_2-norm errors are: $\epsilon_\phi = 1.45\%$ for the NN output and $\epsilon_{\phi'} = 0.75\%, \epsilon_{u'} = 4.14\%, \epsilon_{v'} = 1.33\%, \epsilon_{p'} = 0.93\%, \epsilon_{c^{+\prime}} = 1.31\%, \epsilon_{c^{-\prime}} = 0.95\%$ for the DeepONets outputs.

Comparison between two DeepM&Mnets. In the assessment mentioned above, we use sensors for all the state variables in the parallel DeepM&Mnet. However, the data term in the parallel architecture's loss function, namely Equ. (5), can be still well-defined when there is only data of ϕ, which is consistent with the setup defined for series DeepM&Mnet. Herein, we perform a testing example to investigate the performance of parallel and series DeepM&Mnets by providing them with the same data measurements. First of all, a network size of 6×100 is employed for both architectures. The hyperparameters are given as: $\lambda_d = 10$, $\lambda_o = 1$, $\lambda_r = 10^{-4}$ and $N_{op} = 1000$. The neural network is trained for 60,000 iterations with a learning rate 5×10^{-4}. The convergence of the relative L_2-norm errors with respect to the number of measurements is shown in Fig. 14(a), where we can find that the series architecture performs better than the parallel one. Especially, when the data is limited ($N_{data} < 20$), the parallel DeepM&Mnet seems to be not well-defined and fails to infer the flow fields correctly. The discrepancy between two architectures becomes larger when a smaller network (3×50) is used, as shown in Fig. 14(b). In

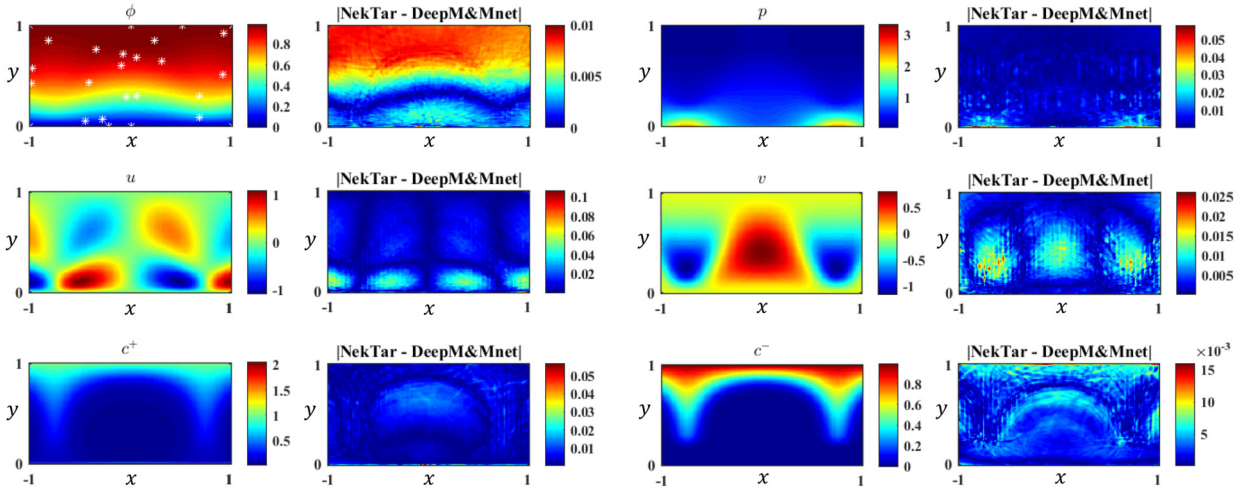
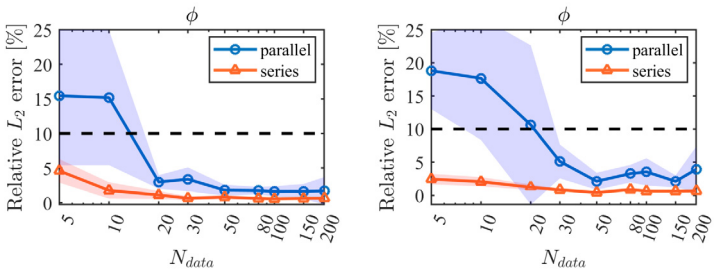


Fig. 13. DeepM&Mnet for 2D electroconvection problem: results of *series* architecture. In this case, 20 measurements of ϕ are provided, which are marked with $*$. Left: results of the DeepONets outputs (i.e., $\phi', u', v', p', c^{+'}, c^{-'}$), right: point-wise difference.



(a) NN size: 6×100 (b) NN size: 3×50

Fig. 14. DeepM&Mnet for 2D electroconvection problem: comparison between *parallel* and *series* architectures when only data of ϕ is available. Two network sizes are investigated: (a) 6×100 and (b) 3×50. The relative L_2-norm errors of ϕ are plotted, where we can find that when there is only data of ϕ used in DeepM&Mnets, the series architecture performs better than the parallel one in this electroconvection problem.

the parallel DeepM&Mnet, the neural network is approximating all the variables, while it is used only for ϕ in the series setup. Therefore, the series DeepM&Mnet is less sensitive to the network size.

In summary, we can conclude that the series DeepM&Mnet outperforms the parallel DeepM&Mnet when only the data of electric potential is available, although the parallel structure also provides acceptable results. However, we should note that the parallel DeepM&Mnet can utilize more information from data. We recommend to use parallel DeepM&Mnet when there are measurements of different variables. For better performance of the parallel DeepM&Mnet, a large neural network is recommended. Moreover, the series DeepM&Mnet requires a closed loop to construct the operator loss, e.g. $\phi \rightarrow (c^{+'}, c^{-'}) \rightarrow \phi'$, which is not always available when designing the DeepONets.

4.3. Discussion on DeepM&Mnets

4.3.1. DeepM&Mnets for unsteady flow

Although in this paper we only consider the steady state of electroconvection problem, we note that the DeepONets and DeepM&Mnets can be extended to handle the time-dependent case, where the physical quantities are functions of space and time (x, y, t). For example, a DeepONet, G_u, can be designed to take the time-dependent function $\phi(x, y, t)$ as input of the branch net and the coordinates (x, y, t) as input of the trunk net, which accounts for a mapping from an unsteady field to another. However, in this setup, the initial conditions of Equ. (1) should be fixed for all the training and testing cases. Otherwise, the pre-trained DeepONet will not work if the initial conditions change. In order to address this problem and make the DeepONet appropriate for different initial conditions, the initial conditions can be also fed into the branch net along with $\phi(x, y, t)$. Then, using the data generated with various initial conditions and various potential fields, the pre-trained DeepONet can be used to predict the unsteady flow. Such a problem setup can be found in [44], where the DeepONet was used to predict the bubble growth dynamics by considering the initial bubble size as input. Moreover, the pre-trained unsteady DeepONets can be incorporated into the DeepM&Mnets, where the input involves (x, y, t) as well. By

doing so, the DeepM&Mnets are able to infer the spatio-temporal flow fields using the data measurements collected at different time steps.

4.3.2. DeepM&Mnets vs. conventional data assimilation methods

The results shown above indicate that DeepM&Mnets can infer full fields of electroconvection accurately by integrating pre-trained DeepONets with a very small number of measurements. Here, we discuss another two widely-used data assimilation methods, the adjoint- and ensemble-variational techniques, together with their respective costs and benefits, and the comparative performance of DeepM&Mnets. Similar to the DeepM&Mnets, data assimilation techniques define a cost function that is the difference between available observations and the model prediction. In conventional methods, the model is comprised of the governing PDEs that are expensive to solve numerically, while in DeepM&Mnets, the physics laws are represented by DeepONets which are much more efficient to evaluate. A key step in all approaches is to iteratively minimize the cost function using its gradient. However, there are some differences regarding the optimization procedure, which are summarized as follows:

The adjoint variational methods [36–38] can accurately compute the gradient of high-dimensional problems, where each iteration involves the evolution of the forward and dual models. As a result, their computational cost per iteration is at least twice that of a forward model, and they require significant storage because in nonlinear systems the adjoint equations feature the forward solution. Another important limitation is the requirement to have an accurate adjoint algorithm, which is not always available.

The ensemble methods [39–41] require forward evolution of an ensemble of control vectors in order to approximate the local landscape of the cost function and evaluate the gradient. While they do not require an adjoint model, the cost of each iteration is proportional to the size of the ensemble which itself is proportional to the dimension of the control vector.

The key advantage of DeepM&Mnet is that the PDEs are replaced with pre-trained DeepONets which is much more efficient per iteration. We have also demonstrated that the method is applicable to high-dimensional problems (multiscale and multiphysics) and does not require a dual model because the optimization relies on automatic differentiation. In this sense, the DeepM&Mnet migrates the cost to pre-training DeepONets, elevates the burden of adjoint using automatic differentiation and retains the benefit of dealing with complex systems. As mentioned previously in this paper, the speedup of DeepONets evaluation versus the CFD simulation for solving the forward steady-state electroconvection model is 1000 folds. With these "plug-and-play" networks, the training of a DeepM&Mnet takes about 20 minutes on a single processor (e.g., NVIDIA Quadro RTX 6000).

It is also important to note that the efficiency of data assimilation strategies is sensitive to the choice of the initial guess. The initial control vector is different in conventional methods (e.g. wall electric potential) versus DeepM&MNets (NN parameters). For these reasons, a detailed comparison of these approaches requires a separate effort, which is entirely dedicated to an exhaustive assessment for specifically designed test problems. This is beyond the scope of this work, but is worth investigating in the future.

5. Concluding remarks

In this paper, we first demonstrate the effectiveness of DeepONets for complex fluid systems using the multiphysics of electroconvection as a benchmark. We train several DeepONets to predict one of the coupled fields from the rest of the fields. In our example, we use 15 different conditions corresponding to a range of applied electric potentials to form the training dataset. Upon training, the DeepONets can predict all fields very accurately and efficiently for any condition when the input is given.

More importantly, in this paper, we propose a novel framework, the DeepM&Mnet, which allows to integrate the pre-trained DeepONets and a few measurements from any of the fields, to produce the full fields of the coupled system. In DeepM&Mnets, we use a neural network as the surrogate model of the multiphysics solutions, and use the pre-trained DeepONets as the constraints for the solutions. For both *parallel* and *series* DeepM&Mnets, we find that only a few measurements are sufficient to infer the full fields of the electroconvection, even if measurements are not available for all state variables. The DeepM&Mnet, which can be considered as a simple data assimilation framework, is more flexible and efficient than other conventional numerical methods in terms of dealing with such assimilation problem. Note that in order to use the DeepM&Mnets, the building blocks - the DeepONets - are required to be pre-trained with labeled data. However, as shown in the paper, preparing the training data is very flexible and the training can be done offline. Once the DeepONets have been trained and embedded in the DeepM&Mnet, it is straightforward to predict the solutions of a complex multiphysics and multiscale system when only a few measurements are available. More broadly, the results in this paper show that the new framework can be used for any type of multiphysics and multiscale problems, for example in hypersonics with finite chemistry, and we will report such results in future work.

CRediT authorship contribution statement

Shengze Cai: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Zhicheng Wang:** Conceptualization, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing. **Lu Lu:** Conceptualization, Methodology, Software, Writing – review & editing. **Tamer A. Zaki:** Conceptualization,

Table A.1
Prediction errors of G_u with MSE loss and MAPE loss.

G_u	$\Delta\Phi = 13.4$		$\Delta\Phi = 62.15$	
	MSE	relative L_2 error	MSE	relative L_2 error
MSE loss	5.41×10^{-8}	45.27%	1.10×10^{-5}	0.88%
MAPE loss	1.92×10^{-9}	8.54%	3.89×10^{-5}	1.65%

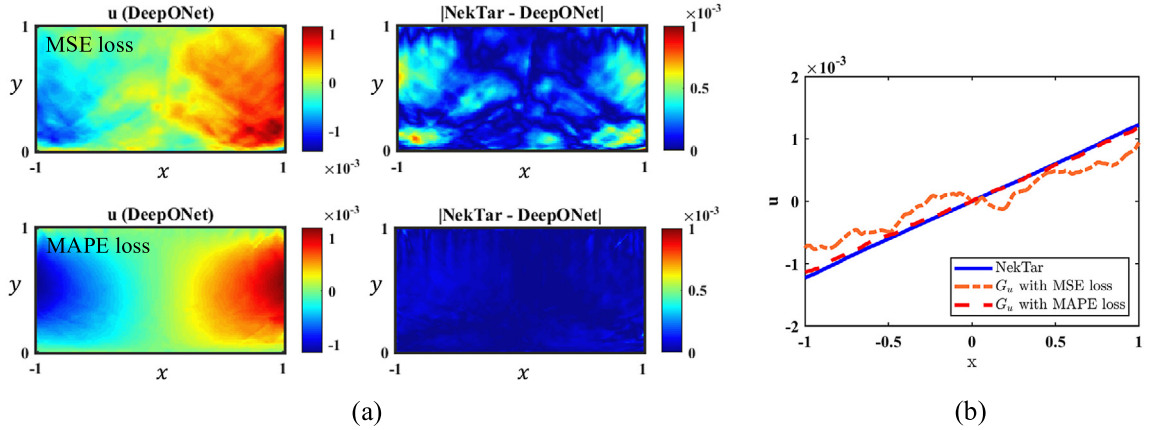


Fig. A.1. Predictions of G_u with different losses: (a) 2D fields and errors of u-velocity (the first row is G_u with MSE loss and the second is G_u with MAPE loss); (b) 1D profiles at $y = 0.5$. The testing example shown here is the case when $\Delta\Phi = 13.4$.

Funding acquisition, Methodology, Project administration, Supervision, Writing – review & editing. **George Em Karniadakis:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing.

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.

Acknowledgements

The authors acknowledge support from DARPA/CompMods HR00112090062 and DOE/PhILMs DE-SC0019453.

Appendix A. DeepONet training: MSE loss vs. MAPE loss

As mentioned in Section 3, we apply the MAPE loss to the DeepONets training for $G_{u,v}$ due to the large range of the velocity magnitude in the electroconvection problem. If we use a MSE loss for such multiscale prediction, the training loss and the back-propagation process will be dominated by those velocities with larger magnitudes, as the small-scale velocities contribute little to the NN optimization. Therefore, the MAPE loss is better for the DeepONets that are required to predict multiscale values.

A comparison between the MSE and MAPE losses applied to G_u is shown here. We learn from Fig. 5 that the u-velocity is a multiscale state variable. In particular, the magnitude of the velocity is approximately zero (10^{-4}) when $\Delta\Phi$ is small (e.g., $\Delta\Phi = 13.4$), while it is in the order of 10^0 when $\Delta\Phi$ is large (e.g., $\Delta\Phi = 62.15$). The prediction errors of G_u with MSE loss and MAPE loss are given in Table A.1. We can see that the G_u with MAPE loss performs much better than the one with MSE loss for $\Delta\Phi = 13.4$. Although for $\Delta\Phi = 62.15$, for G_u with MAPE loss we get larger errors, it is still very accurate as the relative L_2-norm error is only 1.65%. To demonstrate the superiority of the MAPE loss for G_u, we show the predicted fields with different losses in Fig. A.1. The DeepONet with MSE loss fails to predict the flow pattern at $\Delta\Phi = 13.4$ accurately, while the prediction of G_u with MAPE loss is very consistent with the NekTar high-fidelity simulation.

Appendix B. Effect of L_2 regularization for DeepM&Mnet

The loss function of DeepM&Mnet training, given in Equ. (5), is composed of a data term, an operator term and a L_2 regularization term for the trainable parameters. We note that the DeepM&Mnet is essentially a function parameterized by the weights and biases of neural network. The manifold of the parameter space contains a number of local minima,

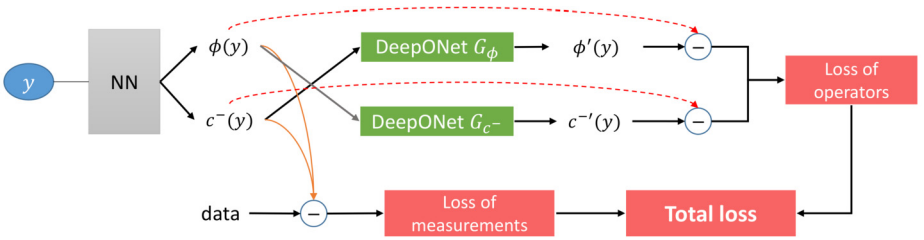


Fig. B.2. DeepM&Mnet for 1D electroconvection problem: schematic of the *parallel* architecture. Here, the neural network (NN) is used to approximate the 1D coupled solutions (i.e., ϕ and c^-). The pretrained DeepONets G_ϕ and G_{c^-} are fixed and form constraints for the NN outputs.

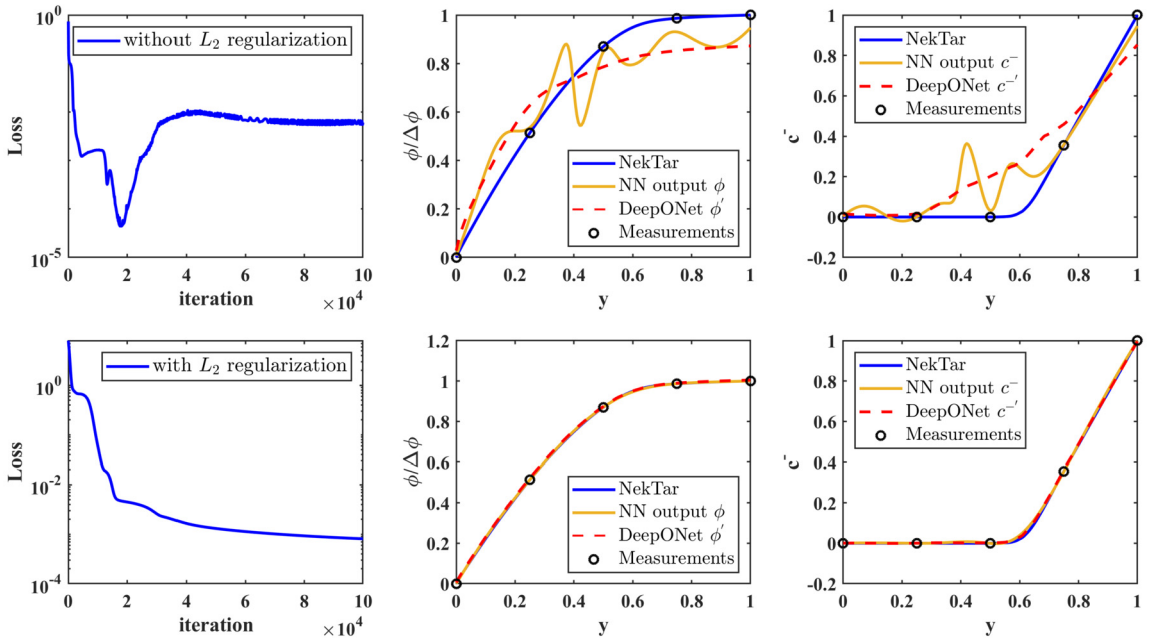


Fig. B.3. Parallel DeepM&Mnet for 1D electroconvection problem: results with different loss functions for $\Delta\Phi = 77.6$. First row: loss without L_2 regularization; second row: loss with L_2 regularization. Left: training loss; middle: solution of $\phi(y)$; right: solution of $c^-(y)$. Here, 5 measurements are used for each coupled field (i.e., ϕ and c^-). Overfitting is observed in the case without L_2 regularization.

some of which will lead to data overfitting. Although the operator loss, involving the pre-trained DeepONets, is viewed as a constraint of the approximating solution, it is not enough to guarantee a unique solution especially in the case when we only have a very small number of data, where the input functions of DeepONets (denoted by V) are not well defined (namely the number of data is greatly smaller than the sensors required for DeepONets input). Therefore, we apply the L_2 regularization, which is commonly used to avoid overfitting for data-driven methods and is easy to implement in the DeepM&Mnet framework. In this section, we present an example showing the effect of the L_2 regularization.

We use the 1D electroconvection as the benchmark here. For 1D electroconvection, the electric potential ϕ and the concentration c^- are functions of y. We train two independent DeepONets G_ϕ and G_{c^-}, where G_ϕ takes $c^-(y)$ as input and outputs $\phi(y)$, while G_{c^-} performs oppositely. We generate 24 training trajectories for the 1D electroconvection at various values of $\Delta\Phi$, namely $\Delta\Phi = 5, 10, \ldots, 120$. For the individual DeepONet, two hidden layers and 100 neurons per layer are applied for both branch net and trunk net. The input function is represented by 21 discrete values with equal space interval. Upon training, the DeepONets G_ϕ and G_{c^-} can predict one field from another very accurately.

Finally, we can construct the parallel DeepM&Mnet architecture, which is illustrated in Fig. B.2. A neural network (with 3 layers and 50 neurons per layer) is used to approximate the solutions of $\phi(y)$ and $c^-(y)$. We apply the proposed DeepM&Mnet to infer the profiles in the case of $\Delta\Phi = 77.6$. The results with different loss functions are demonstrated in Fig. B.3. Five measurements, which are uniformly distributed along y direction, are used for each coupled field. We find that in the case where the L_2 regularization is not considered, the loss jumps suddenly during training to a bad minimum and the final solutions are overfitting to the measurements. Although the result is affected by the parameter initialization and the training scheme, the similar phenomenon is observed frequently when we perform ensemble simulations (5 times over 10 simulations). Intuitively, overfitting may be caused by large weights and biases in the neural network. Therefore, a L_2 regularization penalizing the magnitudes of parameters can help to avoid this. As shown in Fig. B.3, the DeepM&Mnet

with L_2 regularization in the loss function is much more robust and attains consistent results with good accuracy for all ten simulations with random initialization.

References

[1] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508.
[2] K. Duraisamy, G. Iaccarino, H. Xiao, Turbulence modeling in the age of data, Annu. Rev. Fluid Mech. 51 (2019) 357–377.
[3] D. Fan, L. Yang, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Reinforcement learning for bluff body active flow control in experiments and simulations, Proc. Natl. Acad. Sci. 117 (2020) 26091–26098.
[4] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[5] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[6] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510.
[7] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925.
[8] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, J. Fluid Mech. 861 (2019) 119–137.
[9] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations, Science 367 (2020) 1026–1030.
[10] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732.
[11] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951.
[12] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166.
[13] 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.
[14] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (2019) A2603–A2626.
[15] Y. Chen, L. Lu, G.E. Karniadakis, L. Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (2020) 11618–11633.
[16] A. Yazdani, L. Lu, M. Raissi, G.E. Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput. Biol. 16 (2020) e1007575.
[17] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (2017) e1602614.
[18] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561, 2017.
[19] K. Hornik, M. Stinchcombe, H. White, et al., Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[20] T. Chen, H. Chen, Approximations of continuous functionals by neural networks with application to dynamic systems, IEEE Trans. Neural Netw. 4 (1993) 910–918.
[21] 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.
[22] 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.
[23] A. Mani, K.M. Wang, Electroconvection near electrochemical interfaces: experiments, modeling, and computation, Annu. Rev. Fluid Mech. 52 (2020) 509–529.
[24] I. Rubinstein, B. Zaltzman, Electro-osmotically induced convection at a permselective membrane, Phys. Rev. E 62 (2000) 2238.
[25] B. Zaltzman, I. Rubinstein, Electro-osmotic slip and electroconvective instability, J. Fluid Mech. 579 (2007) 173–226.
[26] S. Rubinstein, G. Manukyan, A. Staicu, I. Rubinstein, B. Zaltzman, Direct observation of a nonequilibrium electro-osmotic instability, Phys. Rev. Lett. 101 (2008) 236101.
[27] C.L. Druzgalski, M.B. Andersen, A. Mani, Direct numerical simulation of electroconvective instability and hydrodynamic chaos near an ion-selective surface, Phys. Fluids 25 (2013) 110804.
[28] E. Karatay, C.L. Druzgalski, A. Mani, Simulation of chaotic electrokinetic transport: performance of commercial software versus custom-built direct numerical simulation codes, J. Colloid Interface Sci. 446 (2015) 67–76.
[29] E. Karatay, M.B. Andersen, M. Wessling, A. Mani, Coupling between buoyancy forces and electroconvective instability near ion-selective surfaces, Phys. Rev. Lett. 116 (2016) 194501.
[30] C.L. Druzgalski, A. Mani, Statistical analysis of electroconvection near an ion-selective membrane in the highly chaotic regime, Phys. Rev. Fluids 1 (2016) 073601.
[31] S.V. Pham, H. Kwon, B. Kim, J.K. White, G. Lim, J. Han, Helical vortex formation in three-dimensional electrochemical systems with ion-selective membranes, Phys. Rev. E 93 (2016) 033114.
[32] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.
[33] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. 64 (2019) 525–545.
[34] S. Cai, S. Zhou, C. Xu, Q. Gao, Dense motion estimation of particle images via a convolutional neural network, Exp. Fluids 60 (2019) 73.
[35] J. Kim, C. Lee, Prediction of turbulent heat transfer using convolutional neural networks, J. Fluid Mech. 882 (2020).
[36] Q. Wang, Y. Hasegawa, T.A. Zaki, Spatial reconstruction of steady scalar sources from remote measurements in turbulent flow, J. Fluid Mech. 870 (2019) 316–352.
[37] M. Wang, Q. Wang, T.A. Zaki, Discrete adjoint of fractional-step incompressible Navier-Stokes solver in curvilinear coordinates and application to data assimilation, J. Comput. Phys. 396 (2019) 427–450.
[38] M. Wang, T.A. Zaki, State estimation in turbulent channel flow from limited observations, arXiv preprint arXiv:2011.03711, 2020.
[39] S. Cai, E. Mémin, Y. Yang, C. Xu, Sea surface flow estimation via ensemble-based variational data assimilation, in: 2018 Annual American Control Conference (ACC), 2018, pp. 3496–3501.
[40] V. Mons, Q. Wang, T.A. Zaki, Kriging-enhanced ensemble variational data assimilation for scalar-source identification in turbulent environments, J. Comput. Phys. 398 (2019) 108856.

[41] D.A. Buchta, T.A. Zaki, Observation-infused simulations of high-speed boundary layer transition, arXiv preprint arXiv:2011.08265, 2020.
[42] G.E. Karniadakis, S. Sherwin, Spectral/hp Element Methods for Computational Fluid Dynamics, Oxford University Press, 2005.
[43] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: a deep learning library for solving differential equations, SIAM Rev. 63 (2021) 208–228.
[44] 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.