On the geometry transferability of the hybrid iterative numerical solver for differential equations

Adar Kahana · Enrui Zhang · Somdatta Goswami · George Karniadakis · Rishikesh Ranade · Jay Pathak

Received: 31 October 2022 / Accepted: 11 January 2023 / Published online: 2 February 2023
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2023

Abstract
The discovery of fast numerical solvers prompted a clear and rapid shift towards iterative techniques in many applications, especially in computational mechanics, due to the increased necessity for solving very large linear systems. Most numerical solvers are highly dependent on the problem geometry and discretization, facing issues when any of these properties change. The newly developed Hybrid Iterative Numerical Transferable Solver (HINTS) combines a standard solver with a neural operator to achieve better performance, focusing on a single geometry at a time. In this work, we explore the “T” in HINTS, i.e., the geometry transferability properties of HINTS. We first propose to directly employ HINTS built for a specific geometry to a different but related geometry without any adjustments. In addition, we propose the integration of an operator level transfer learning with HINTS to even further improve the convergence of HINTS on new geometries and discretizations. We conduct numerical experiments for a Darcy flow problem and a plane-strain elasticity problem. The results show that both the direct application of HINTS and the transfer learning enhanced HINTS are able to accurately solve these problems on different geometries. In addition, using transfer learning, HINTS is able to converge to machine zero even faster than the direct application of HINTS.

1 Introduction
Numerical simulations play a crucial role in scientific and engineering applications such as mechanics of materials and structures [1–7], bio-mechanics [8], fluid dynamics [9–11], etc. The simulation approach is based on solving linear/nonlinear partial differential equations (PDEs). The efficiency and accuracy of a simulation approach is always comparable to conflict partners. This means that the quest for a more efficient numerical solver frequently results in lesser numerical accuracy. In engineering simulations, the main factor is to have an acceptable accuracy with feasible computational requirements for both memory utilization and computing time. Furthermore, the solution process must be stable and dependable. Therefore, determining an appropriate approach for the problem at hand is crucial, and usually determines the outcome of a simulation.

In traditional numerical solvers like the finite element method [1], we often reduce the complex differential equations defining the physical system to a system of linear equations of the form: $\mathbf{K} \{u\} = \{f\}$, where $\mathbf{K}$ is referred as the stiffness matrix; $\{f\}$ is the force vector and $\{u\}$ is the set of unknowns. A simple, yet not recommended way to solve for the set of unknown is to use the direct method by inverting the stiffness matrix and multiplying it with a force vector. However, the direct method fails in cases of large degrees of freedom (the stiffness matrix is in the order of a few millions) and/or the stiffness matrix is sparse. At this juncture, the iterative solvers come to the rescue. We start with an initial guess for $\{u\}$, and gradually progress towards the true solution for $\{u\}$. The solver iterates until a reasonable solution that meets the stopping criterion (typically an error tolerance value) is obtained. Iterative solvers are appropriate for large computing problems because they can frequently be parallelized more efficiently using algorithms. However, proper pre-conditioning of the stiffness matrix is a mandatory requirement. The solution’s high oscillatory component can be solved efficiently using a dense mesh and few steps with a simple iterative method like Jacobi iteration or Gauss-Seidel.
The GS method. It suffers from divergence for non-symmetric and indefinite systems over a coarse grid, as well as slow convergence associated with low-frequency eigen modes, restricting its application for large scale linear systems.

Recent advances in deep learning in addition to the developments in computational power have provided the means to employ neural networks as efficient approximators for PDEs [12–15]. Their compositional character differs from the traditional additive form of trial functions in linear function spaces, where PDE solution approximations are built using Galerkin, collocation, or finite volume approaches [1,11]. Their computational parametrization via statistical learning and large-scale optimization approaches makes them increasingly suitable for solving nonlinear and high-dimensional PDEs. However, the neural network often learns the low-frequency eigen modes, and tends to avoid the high-frequency modes. This phenomenon referred to as spectral bias is observed in numerous applications of neural networks.

In one of the recent works [16], we proposed an efficient approach to integrate synergistically the iterative solvers with deep neural operators to exploit the merits of both the solvers in turn and to overcome the challenges of either of them. The approach acronym as HINTS, improves the convergence of the solution across the spectrum of eigen modes by leveraging the spectral bias of a deep neural operator. As observed in the seminal work of HINTS, the solution is flexible with regards to the computational domain and is transferable to different discretizations. In this work, we investigate the transferability properties, i.e., the “T” in HINTS, with respect to domain adaptation. Specifically, the information from a model trained on a specific domain (source), is employed to infer the solution on a different but closely related domain (target).

Additionally, we integrate HINTS with the seminal work of operator level transfer learning [17] to improve the convergence rate of HINTS on the target domain. In this scenario, we use a small number of labelled data to fine tune the target model using samples from the target domain. The model is initialized with the learnt parameters of the source model and is trained under a hybrid loss function, comprised of a regression loss and the Conditional Embedding Operator Discrepancy (CEOD) loss, used to measure the divergence between conditional distributions in a Reproducing Kernel Hilbert Space (RKHS). The target model is trained only for the deeper layers, acknowledging the widely accepted fact that the shallow layers are responsible for capturing the more general features.

As a summary, we investigate the capability of HINTS trained on a source domain to operate on a target domain with the following two approaches:

- **Direct application** of HINTS to an unseen target domain without retraining the DeepONet;

- **Usage of transfer learning** to fine-tune the HINTS with limited data associated with the target domain.

This is illustrated in Fig. 1.

In this manuscript, we have considered the Darcy’s model on a L-shaped domain (source), and the same domain with a circular or a triangular inclusion (target), and the linear elasticity model on a square domain (source) and a square domain with a circular inclusion (target). The remainder of the manuscript is organized as follows. In Sect. 2, we review the existing traditional numerical methods for solving linear systems, along with the recently popular deep learning solvers. In Sect. 3, we present the methodology of HINTS followed by the integration of the transfer learning approach. The experiments carried out to show the efficiency of domain adaptation with and without the transfer learning are presented in Sect. 4. Finally, we summarize our observations and provide concluding remarks in Sect. 5.

### 2 Related work

**Classical numerical solvers of ODEs/PDEs** Over the last few decades, many studies have been conducted for developing advanced numerical solvers mainly for approximating the solutions of ODEs and PDEs. Classical methods are the Jacobi and GS methods, which were proposed in the 19th century. Since then, although many advanced algorithms have been proposed, most of the state-of-the-art solvers still use...
some versions of these two algorithms. The current leaders of the field of numerical solvers are the family of MultiGrid (MG) methods [18,19], where one uses a set of discretizations to solve the system. It is common to use Jacobi or GS smoothing inside the MG iterations. Therefore, constraints such as positive definiteness that apply for the Jacobi and GS algorithms, also affect MG as well. There are methods to overcome this, such as the shifted Laplacian method [20], but these suffer from other disadvantages. The solvers compete for the lowest number of iterations needed for convergence, and also other properties such as physical time per iteration, parallelization capabilities, and more. However, in some cases the solvers may diverge (as an example, approximating the solution of an indefinite system), and solvers that are robust and can solve all types of problems are sought after.

**Machine learning-based numerical solvers** Many researchers have been investigating the use of AI when designing efficient numerical solvers. Some focus on creating an AI based solver for solving PDEs, replacing the numerical solvers. A notable example is the Physics-Informed Neural Networks [12,14,15], where one trains a network to infer the solution of the PDE in a domain, without the need to assemble a linear system and solve, nor use a complex meshing algorithm. Another direction is to enhance numerical solvers using AI (which is the main focus of this paper). Most recent studies aim to replace components of the MG algorithms with AI based components, for example training a neural network to replace the restriction and prolongation operations [21,22]. Others try to achieve a better performing preconditioner using learning [23].

**DeepONet** Another notable advancement in the field of AI is the invention of operator learning methods [24–26]. In this work, in particular, we focus on the DeepONet [25], and use this for the operator learning. In contrast to standard Machine Learning (ML) tasks, where one seeks to approximate a function that can connect input data to output data, in operator learning one seeks a mapping between a family of functions and another family of functions that satisfies an operator, hence the name deep operator learning. After the network has been trained and the operator has been learned, one does not need to either re-train nor solve the system again for new conditions or parameters, but rather infer the solution using the trained system for the new problem definition, which dramatically lowers the computational cost. In addition, in the case of DeepONet, the learned operator does not depend on a discretization and can be used to infer the solution on any given discretization, meaning that the DeepONet can be used for different meshes without re-training. However, changing the scenario (the equation, the dimensions, the boundary conditions, etc.), requires training a DeepONet for the new scenario. In the past years, DeepONet has been applied to diverse engineering problems [13,27–29], showing extraordinary capabilities.

**HINTS** A new method to enhance numerical solvers using operator learning has been proposed under the name HINTS: Hybrid Numerical Iterative Transferable Solver (see “Appendix A” and [16]). The idea of HINTS is to use an iterative solver such as Jacobi or GS, and replace some of the iterations with a DeepONet trained to receive the problem parameters and infer the solution. This DeepONet can also be used to receive the problem parameters and the residual at the current iteration and produce the correction term for the solution, so it can be applied in a similar way to the numerical solvers. Experiments show that using the existing numerical solvers, they tend to face slow convergence due to their difficulty to smooth the error for low frequency modes (where for the high modes they operate well), while the DeepONet excels in smoothing the low modes errors (but may fail for the high modes). Using both the numerical methods and the DeepONet, uniform convergence of all modes is achieved and the solvers converge to machine precision much faster. HINTS showed promising results on many tasks, and also a lot of potential for extensions, and in this paper we discuss a very important extension of HINTS that mechanics simulations may benefit from - the transferability of HINTS to new geometries and discretizations.

**Transfer learning** The idea of transfer learning is to leverage the parameters of a model trained with sufficiently large labelled dataset to infer information on a related task with few labelled datasets and minimal training. In [17] (briefly summarized in “Appendix B”), an operator level transfer learning was proposed to lower the computational costs of training a DeepONet (from Scratch) for related tasks. In the categorization of TL approaches, one popular classification is based on the consistency between the distributions of source and the target input (or feature) spaces and output (or label) spaces. The shift between the source and target data distributions is considered the major challenge in modern TL. One type of distribution shifts include conditional shift, where the marginal distribution of source and target input data remains the same while the conditional distributions of the output differ (i.e., $P(x_t) = P(x_s)$ and $P(y_t|x_t) \neq P(y_s|x_s)$) and covariate shift, where the opposite occurs (i.e., $P(x_t) \neq P(x_s)$ and $P(y_t|x_t) = P(y_s|x_s)$). In this work, we have employed the TL model proposed in [17] to work on conditional distribution discrepancy between the domains. In this work, the authors have reported that the target model essentially requires a smaller dataset to fine tune, which can thus be done in significantly less time.

**3 Method**

Without any loss of generality, we consider a family of PDEs defined in a domain $\Omega$: 

$$
\frac{\partial}{\partial t} u(x,t) = \nabla \cdot (a(x,t) \nabla u(x,t)) + f(x,t) \quad (x \in \Omega, t > 0)
$$

$$
\frac{\partial}{\partial n} u(x,t) = g(x,t) \quad (x \in \partial \Omega, t > 0)
$$

$$
u(x,0) = u_0(x) \quad (x \in \Omega)
$$
Fig. 2 Mesh discretization for the geometries considered in the Darcy’s problem: a source domain, b–c target domains considered for domain adaptation

\[ L_x(u; k) = f(x), \quad x \in \Omega \]  
\[ B_x(u) = g(x), \quad x \in \partial \Omega, \]  

where \( L_x \) is a differential operator, \( B_x \) is a boundary operator, \( k = k(x) \) parameterizes \( L_x, f(x) \) and \( g(x) \) are the forcing terms, and \( u = u(x) \) is the solution of the given PDE. With a well-trained DeepONet embedded, a HINTS is capable of solving for \( u \) corresponding to \( k \) and \( f \), i.e., it captures the solution operator \( G \) of the family of PDEs specified by Eqs. (1) and (2):

\[ G : k, f \mapsto u \text{ s.t. Equation 1 and Equation 2 are satisfied.} \]  

For detailed information on the implementation of HINTS, readers are suggested to refer to “Appendix A” and [16].

Herein, we first construct a source HINTS, i.e., a HINTS with DeepONet trained offline for Eqs. (1) and (2) defined in the source domain \( \Omega = \Omega^S \). This is conducted by using \( N_S \) labelled source data, \( \mathcal{D}_S = \{ k_i, f_i, u_i \}_{i=1}^{N_S} \). In the DeepONet architecture, the branch network is a convolution neural network comprising of channels, each taking as input one of the mapping functions. The trunk network takes as inputs the spatial location of the points in the domain \( \Omega^S \). This DeepONet is trained with a standard regression loss (relative mean squared error) to obtain the optimized weights and biases of the source network, \( \theta^S \). We mention that training the DeepONet is a costly procedure that takes hours and requires large computational resource, but it is an offline process that needs to be done only once. Next, we build HINTS by assembling the DeepONet and the numerical solver (e.g., GS). We discretize \( \Omega^S \) with triangular elements. The simulation starts by assuming an initial solution of the dependent variable and at every iteration of HINTS it adopts either the pre-decided fixed relaxation method or the pre-trained DeepONet to approximate the solution. The iterative solver and DeepONet is chosen based on a pre-decided ratio. Among the available numerical iterative solvers, we have employed the GS approach.

With the source HINTS properly constructed, we now consider transferring it to become a target HINTS, i.e. a HINTS for solving the class of PDEs (Eqs. 1, 2) defined in a target domain \( \Omega = \Omega^T \), with the following two methods:

**Direct application: use source DeepONet directly in target HINTS** To do this, the first approach is to directly apply the source HINTS for inference in the target domain \( \Omega^T \). Specifically, when implementing the target HINTS (i.e., HINTS for the target domain), the trained DeepONet from the source HINTS is directly invoked in the workflow of the target HINTS. This methodology involves using nodes from a different mesh in the DeepONet trunk.

**Transfer learning: fine-tune the DeepONet in target HINTS** In the second approach, we fine-tune the trained DeepONet from the source HINTS with a small number of labelled samples from the target domain. We generate \( N_T \) labelled data, \( \mathcal{D}_T = \{ k_i, f_i, u_i \}_{i=1}^{N_T} \) on the target domain, where \( N_S \gg N_T \). In the examples presented in this work, \( N_T \approx 0.01 \times N_S \). We initialize a target model (having the same architecture as the source model), with the learnt parameters of the source model and fine tune the model by training the fully connected layers of the convolution neural network and the last layer of the trunk net under a hybrid loss function. The hybrid loss function reads as:

\[ \mathcal{L}(\theta^T) = \lambda_1 \mathcal{L}_r(\theta^T) + \lambda_2 \mathcal{L}_{CEOD}(\theta^T), \]

where \( \lambda_1 = 1 \) and \( \lambda_2 \gg \lambda_1 \) are trainable coefficients, which determine the importance of the two loss components during the optimization process [30]. In Eq. (4), while the first term is a standard regression loss, the second term ensures the agreement between the conditional distributions of the target data. For details on the construction of the \( \mathcal{L}_{CEOD}(\theta^T) \), readers are suggested to refer to [17]. While employing the HINTS algorithm for the target domain, the source DeepONet is replaced by the target DeepONet.

Even though the HINTS solution is transferable between related domains, and integrating the transfer learning approach would essentially mean an additional training time, we argue that the convergence rate of a transfer-learning inte-
grated HINTS solution is in faster than the direct application (see Sect. 5 for statistical analysis). It is worth noting that the target model is trained with much less iterations and uses less data, thus it takes only a few minutes and is also considered an offline step, done only once. We mention that there is potential for performing the transfer learning even better, and producing a more accurate target DeepONet, but in this work we did not aim for achieving this. We focus on showing that the transfer learning mechanism works and produces even faster convergence for HINTS.

4 Computational experiments

In this section, we explore our method using two benchmark problems. The first problem involves the flow in heterogeneous porous media (Darcy’s model) on a two-dimensional L-shaped domain (source). The target domains considered in this case are the same L-shaped domain with a circular and a triangular cutout. In the next problem, we have considered a plane-strain elasticity problem in a square domain (source). The target domain considered in this case is the same square domain with a central circular cutout. In both the examples, the DeepONet is trained using the Adam optimizer [31]. The implementation has been carried out using the PyTorch framework [32]. For both the examples we initialize the DeepONet parameters using Xavier initialization. Details on the data generation, network architecture, such as number of layers, number of neurons in each layer, and the activation functions are provided with each example.

4.1 Darcy flow

In the first example we consider the Darcy’s flow on a L-shaped domain. The problem is defined on a L-shaped domain (source) as:

\[ \nabla \cdot (k(x) \nabla u(x)) + f(x) = 0, \quad x = (x, y) \in \Omega^S, \]
\[ u(x) = 0, \quad x \in \partial \Omega^S. \]

where \( k(x) \) is a spatially varying hydraulic conductivity, \( u(x) \) is the hydraulic head, and \( f(x) \) is a spatially varying force vector. We use triangular elements to discretize the L-shaped domain, \( \Omega^S_L \). The spatial discretization is shown in Fig. 2a.

Training the source HINTS

In this example, the goal is to learn the operator of the system in Eqs. (5) and (6), which maps the random conductivity field and the random force field to the output hydraulic head, i.e., \( k(x), f(x) \rightarrow u(x) \). To generate multiple samples of \( [k(x), f(x)] \) for training the source DeepONet, we use the Gaussian Random Field (GRF) with standard deviations of 0.3 (for \( k(x) \)) and 0.1 (for \( f(x) \)) and a correlation length of 0.1 (for \( k(x) \) and \( f(x) \)). We implemented a standard numerical solver to calculate \( u(x) \) corresponding to \( [k(x), f(x)] \). We generate \( N_S = 51,000 \) training samples and \( N^{test}_S = 9,000 \) testing samples.

The branch network of the DeepONet is a combination of a CNN (input dimension \( 31 \times 31 \), number of channels \( [2, 40, 60, 100, 180] \), kernel size \( 3 \times 3 \), stride 2, the number of channels of the input 2 comes from the concatenation of \( k(x) \) and \( f(x) \)) and a fully-connected network (dimensions \( [180, 80, 80] \)). The dimension of the trunk network (fully-connected network) is \([2, 80, 80, 80]\). We train the DeepONet for 25,000 epochs with a fixed learning rate of \( 1e^{-4} \), and a mini-batch size of 10,000. For the branch network we employ the ReLU activation function and for the trunk network, we use Tanh activation. The last layers of both the branch and the trunk networks have linear activation functions. The source model converges with a mean relative error of 4% on the test dataset. Training this model took roughly 8h using the NVIDIA A6000 Graphical Processing Unit (GPU), a step that is done offline and only once for this problem. The DeepONet is trained to learn the operator and does not require re-training for each equation.

The trained DeepONet is plugged into the HINTS algorithm and the HINTS iterations are executed until the error of the solution reaches machine zero. An example of such iterative solution is given in Fig. 3. When inspecting the solution, we focus mainly on the convergence efficiency of the solution to machine zero. For that, we observe the norm of the error per iteration, as demonstrated in Fig. 3a. The founding idea of HINTS is based on the uniform convergence of all the eigenmodes, which is shown in Fig. 3b. To illustrate the solution of the problem at hand, we show Fig. 3c, which is the solution at the last iteration (after convergence). We also show the error between the approximate solution and the exact solution (obtained by directly solving the system instead of using an iterative method) in Fig. 3d. Note that the scale of the error is machine precision limit, which means the desired convergence has been achieved.

Solving problems in the target domain

Next, to investigate the domain adaptation capabilities, we consider the following two tasks:

- **Task 1** From a L-shaped domain to a L-shaped domain with a circular cutout. The target domain is defined as: \( \Omega^T_L = \Omega_L \setminus \{(x, y) \mid (x−0.25)^2+(y−0.25)^2 \leq 0.15 \}. \)

- **Task 2** From a L-shaped domain to a L-shaped domain with a triangular cutout. The target domain is defined as: \( \Omega^T_L = \Omega_L \setminus \{(x, y) \mid (x, y) \in \Delta((0.2, 0.1), (0.6, 0.4), (0.3, 0.4)). \}

For both tasks, we impose zero Neumann boundary conditions on the cutout. The discretization of the target domains
are shown in Fig. 2b, c. While the circular cutout has a smooth boundary and is easier to approximate, the triangular cutout has locations of singularity, and hence imposes a more challenging scenario for domain adaptation.

**Direct application** To begin with, we first investigate the domain adaptation capabilities of source DeepONet to make extrapolated approximations on target domain discretization for target HINTS. Precisely, the source DeepONet is directly employed and is iterated with the relaxation methods to approximate the solution of the dependent variable on the target discretization. On the target domains, we define \( k(x) \), and \( f(x) \) by setting the input function values to be zero for points within the cutouts. The convergence of the solution is attributed to the generalization ability of DeepONet. The results for the two target tasks are presented in Fig. 4. We observe that for Task1, convergence to machine precision is obtained in 182 iterations. For Task2, even though much slower (takes longer than 300 iterations), we do achieve convergence as well. We attribute the slow convergence to the three vertices of the triangle, which are singular points and are considered more difficult to handle. The HINTS was able to operate on this geometry and show good performance.
Fig. 4 Direct application of HINTS for domain adaptation. The results presented here are the plots of the target domain when HINTS has been directly used for inference without any re-training. Uniform convergence of the modes is observed as shown in the bottom left images for \( \Omega_L^1 \) and \( \Omega_L^2 \).

Transfer learning As an alternative to the above approach, we propose to fine-tune the DeepONet with a small number of samples from the target domain. To that end, we employ the operator level transfer learning approach proposed in [17], where the target DeepONet is initialized with the weights and biases of the source DeepONet. To fine tune the target DeepONet, only the fully-connected layers of the branch network and the last layer of the trunk network are re-trained with a hybrid loss function that takes into account the difference in the conditional distribution of the source and the target.

To train the target DeepONet for each of the two tasks, we generate \( N_T = 500 \) random samples of \( k(x) \) and \( f(x) \) and obtain the corresponding solution \( u(x) \). The target DeepONet is trained for 10,000 iterations. The time it takes to train the target DeepONet is a few minutes on the NVIDIA A6000 GPU, process that is also done offline and once. When compared to the initial training of the network (from scratch), fine-tuning is faster. Once the target model is trained, we employ target DeepONet with the iterative solver on Target HINTS for the target domain.

Finally, a comparative study is carried out based on the number of iterations each approach takes to converge to machine precision. We randomly select a sample from the target dataset and compare the convergence of the error (both the norm of the error and the mode errors) over iterations, between the direct application of HINTS and the transfer learning HINTS. This is shown in Fig. 5. In addition, a comparison of the convergence rate that includes the standard GS solver is shown in Fig. 6. We conclude that the HINTS solution is transferable to different domain geometries, and integrating the transfer-learning approach to fine tune the source DeepONet on the target domain results in a 25% faster convergence without any loss of accuracy for this example.

4.2 Linear elasticity problem

In the second example, we consider a plane-strain linear elasticity problem defined on a square domain. The governing equations are:

\[
\begin{align*}
\varepsilon_{ij} &= \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \\
\sigma_{ij} &= \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}, \\
\frac{\partial \sigma_{ij}}{\partial x_j} + f_i &= 0,
\end{align*}
\]  

(7)

where subscripts \( i, j \in \{1, 2\} \) refer to the two in-plane directions, \( u_i \) is the displacement in the \( i \) direction, \( \varepsilon_{ij} \) is the strain component, \( \sigma_{ij} \) is the stress component, and \( f_i \) is the body force in the \( i \) direction. The Lamé parameters, \( \mu = \frac{E}{2(1+v)} \) and \( \lambda = \frac{Ev}{(1+v)(1-2v)} \) describe the mechanical properties of
Fig. 5 Convergence of the direct application of HINTS (left) and the transfer learning HINTS (right), done for the L-shaped Darcy problem with a circular cutout. The top figures show the error norm convergence and the bottom figures show the error norms of specific modes.

Fig. 6 Convergence of the different methods for the L-shaped Darcy problem with a circular cutout. Comparison of the convergence of GS (without HINTS, shown in blue line), HINTS-GS (orange line) and the transfer learning HINTS-GS (green line) in terms of error decay. (Color figure online)

Fig. 7 Illustration of the meshes used for the elasticity numerical experiments:
- (a) Square mesh ($\Omega^S$).
- (b) Square mesh with a circular hole ($\Omega^{T^2}$).

The material, where $E$ and $\nu$ are the Young’s modulus and the Poisson’s ratio, respectively. In this example, we consider the square domain, $\Omega^S = [0, 1] \times [0, 1]$ as the source domain. The four edges of the square domain are clamped (i.e., $u = 0$ on $\partial\Omega^S$). The discretization of the domain is presented in Fig. 7a.
Training the source HINTS

The goal of the elasticity problem is to learn the operator of the system described by Eq. (7) and the clamped boundary condition, which maps randomly generated spatially varying Young’s modulus field, \( E(x) \) and randomly varying force field, \( f(x) \) to the displacement field, \( u(x) \). In this example, we follow the same data generation approach as discussed in Sect. 4.1 (using the same parameters for the GRF based data generation of \( E(x) \) and \( f(x) \) as the ones used for generating \( k(x) \) and \( f(x) \) in the Darcy example), and the model was trained with \( N_S = 85,000 \) samples and tested with \( N_{S_{test}} = 15,000 \) samples. The branch net inputs two channels (\( E(x) \) and \( f(x) \)), and we adopt the same architecture for the convolution modules as discussed in the previous example. The fully-connected network following the convolution modules has a dimension [256, 160]. The dimension of the trunk network is [2, 128, 128, 128, 160]. The network is trained for 25,000 epochs to achieve roughly 4% relative error, indicating a sufficiently well-trained network. Training takes roughly 10 h using the NVIDIA A6000 GPU for this setup.

Solving problems in the target domain

Now, we employ the trained source HINTS to solve the elasticity problem on the target domain. We define the following task:

- **Task 3** From a square domain to a square domain with a circular cutout. The target domain is: \( \Omega^{T3} = \Omega^S \setminus \{(x, y) \mid (x - 0.5)^2 + (y - 0.5)^2 \leq 0.15\} \). The boundary of the cutout is traction-free.

The discretization of the target domains is shown in Fig. 7. To begin with, we first employ the source DeepONet to infer on the target domain. In this scenario, no additional training is carried out on the target model. The HINTS algorithm is employed on the target domain for the iterative solver and use the source DeepONet to approximate the solution of displacement. The results presented in Fig. 8a show that using HINTS as is we can converge to machine precision after 205 iterations on the target domain.
As discussed in the previous example, we now integrate the operator level transfer learning algorithm with the HINTS model. In this setup, to train the target model, we generate samples of the input function by appending zeros to the function values with the circular cutout. The target model is trained with \( N_T = 100 \) samples, where the model is initialized with the optimized parameters of the source model and while training all the layers except the fully-connected layers of the branch network and the last layer of the trunk network are frozen. Training in this case also takes just a few minutes using the NVIDIA A6000 GPU. The fine tuned target DeepONet is replaced in the HINTS algorithm to generate the solution for the target domain. The results obtained using transfer learning integrate HINTS are presented in Fig. 8b. In this setup, we observe convergence to machine precision after 169 iterations, a 22.5% improvement over the previous setup of employing HINTS with the source DeepONet to approximate solution for the target domain.

### 5 Summary

In this work, we explored the transferability of HINTS on different computational domains for differential equations. Specifically, we considered two methods: (1) directly employ the HINTS for an equation defined in an unseen domain; (2) adopt the transfer operator learning to fine-tune the HINTS trained on the source domain for the unseen/target domain with limited data. Through presenting the results for Darcy flow and linear elasticity, we demonstrate the effectiveness of the two methods based on HINTS on fast, accurate solutions of differential equations. In particular, HINTS with transfer learning, by leveraging both the knowledge from the source domain and the target domain, converges even faster than using the direct application of HINTS on the target geometry. Despite the faster convergence, it comes with a price that one still needs a small dataset on the target domain. While we have focused on a specific instance of \( k(x) \) (\( E(x) \) for elasticity) and \( f(x) \) in Sect. 4, the performance is consistent for the entire test dataset. Table 1 shows the mean, median and standard deviation (STD) of the number of iterations needed for convergence for 100 different cases in the test dataset.

The capability of the direct application of HINTS for unseen geometries is, to some extent, rather surprising. Seemingly, a DeepONet trained for a fixed geometry (e.g., L-shaped domain) should not be effective on another geometry (e.g., L-shaped domain with a cutout) that is not included in the training dataset. We attribute the functionality of HINTS for unseen geometry to the following two factors: (1) DeepONet simply needs to provide an approximate solution within HINTS, while the task of achieving accuracy is accomplished by the embedded numerical solver; (2) the differential equation defined on the unseen geometry, for the examples that we consider, is similar to the equation defined on the original geometry but with an input function \( k(x) \) (\( E(x) \) for elasticity) defined in an extended domain. For (1), intuitively, the prediction error of the DeepONet caused by the mismatch between the original and the new geometry depends on the difference between the two geometries. Within a reasonable degree of similarity between the two geometries, DeepONet can still decrease the errors of the low-frequency modes. For (2), using the case of Darcy flow as an example, it may be shown that the differential equation defined in the L-shaped domain excluding the cutout is equivalent to the same equation defined in the L-shaped domain, where \( (a) \) within the cutout \( k(x) \) is simply padded with zero, and \( (b) \) the boundary condition at the cutout boundary is zero Neumann boundary condition. Technically, our approach of padding \( k(x) \) inside the cutout with zeros conforms with such equivalence. Therefore, generalizing the L-shaped domain into a new geometry (L-shaped domain with a circle or triangular cutout) is transformed into the generalization of \( k(x) \) from GRF in the training dataset into an unseen \( k(x) \), where it is from GRF outside the cutout but equals to zero inside the cutout. In view of this, we anticipate that this method works for any cutout shape, as long as the outer boundary is fixed. If we want to change the outer boundary (for example, from an L-shaped domain to a square), the transfer learning HINTS approach will be able to handle the scenario. However, the direct application of HINTS may struggle, and this is an open research topic to be investigated thoroughly in the future.

### Appendix A: HINTS

The Hybrid Iterative Numerical Transferable Solver (HINTS) is the basis of the work presented in the main text of this paper. Here we briefly review the key components of HINTS. The main proposition of the HINTS is to synergistically use both a standard solver such as Jacobi or Gauss-Seidel, and a Deep Operator Network (DeepONet). The DeepONet con-

### Table 1 Summary of the results for the two benchmark problems

| Problem  | Method       | Mean  | Median | STD  |
|----------|--------------|-------|--------|------|
| Darcy    | GS           | 403   | 405    | 34.62|
|          | HINTS-GS     | 165   | 164    | 19.03|
|          | Transferred HINTS-GS | 157   | 162    | 15.75|
| Elasticity| GS           | 1029  | 1020   | 64.41|
|          | HINTS-GS     | 257   | 253    | 34.56|
|          | Transferred HINTS-GS | 176   | 173    | 15.68|

Mean, median and standard deviation of the number of iterations it takes for each method to converge to machine precision. The statistical measures were computed for 100 target samples of each one of the Darcy and elasticity problems. The best performing method is clearly the Transferred HINTS-GS, as shown in bold.
sists of a branch (in this case a convolutional neural network) and a trunk (in this case a fully-connected neural network). The DeepONet aims to fit an operator, mapping a family of functions to another family of functions, by learning the projection between the families. The branch learns the coefficients of the projection while the trunk learns the basis. By using Einstein summation we get the projection itself, which is a function in the output space. This can be formulated by:

\[ G(u)(y) \approx \sum_{k=1}^{p} b_k(u) \cdot t_k(y), \]

where \( b_k(u) \) are the branch elements and \( t_k(y) \) are the trunk elements. Since the branch and the trunk are neural networks we can also write:

\[ G(u)(y) \approx \sum_{k=1}^{p} \sum_{i=1}^{n} c^k_i g^N(u; \Theta^{(k,i)}) f^N(y; \theta^{(k)}), \]

where \( f^N, g^N \) are neural networks operating on discretized inputs \( u, y \) using parameters \( \Theta, \theta \) respectively, and \( c^k_i \) are a set of constants. In this formulation the DeepONet supports more than one branch (stacked DeepONet), precisely \( n \) branches, that can handle different sets of inputs with different sets of weights. The input function \( u \) is evaluated on a set of points (sampled).

It is well known that the standard solvers (Jacobi and GS) excel in smoothing the high frequency mode errors, but struggle with the low frequency mode errors. In addition, the spectral bias of neural network is a phenomenon causing the networks to learn the lower frequencies faster than the higher frequencies. Exploiting the spectral bias, the HINTS alternates between applying a standard solver iteration and a DeepONet iteration, so that the former is smoothing the higher frequency modes and the latter is smoothing the lower frequency modes. A schematic of the HINTS is given in Fig. 9, presenting the entire process of using the HINTS.

We now present the method for the case of Darcy flow using HINTS-GS (Gauss-Seidel). The pseudo code from the original HINTS paper [16] is shown below.

We start by discretizing the system to the form \( A^h u^h = f^h \), seeking the solution \( u^h \). Throughout the iterative solution process, a proportion of the Gauss-Seidel iterations is replaced by invoking the trained DeepONet, which predicts a correction to the approximate solution based on the current residual of the linear system. This proportion is set by \( n_r \), so that the iterations numbers that satisfy \( k_{ii} \mod n_r = 0 \) utilize the DeepONet, and the rest use GS. When using the DeepONet, we first apply a reverse discretization step to calculate the function-form residual based on the given vector-form residual. Then, the material properties \( k(x) \) and the function form residual \( r(x) \) are used as inputs to the DeepONet, which predicts the correction term. We add this correction to the current approximate solution and continue to the next iteration. Complete details of the HINTS algorithm are included in [16]. In addition, the seminal paper of the HINTS presents eigenmode analysis that shows which frequency modes the DeepONet can handle, and how the optimal ratio between DeepONet application and GS application is chosen. The analysis is verified using empirical tests for the Darcy problem and Helmholtz problem in one, two and three dimensions.

**Algorithm 1 HINTS-GS**

function HINTS_GS\( k(x), f(x), \mathrm{PDE} \)

\[ A^h, f^h = \text{DISCRETIZE}(k(x), f(x), \mathrm{PDE}) \rightarrow \text{solve} \ v^h = (A^h)^{-1} f^h \]

\[ v^h \leftarrow 0^h \]

\[ k_{ii} \leftarrow 1 \]

while \( k_{ii} \leq n_h \) and not converged do

\[ r^h \leftarrow f^h - A^h v^h \]

if \( k_{ii} \mod n_h = 0 \) then  // condition for invoking DeepONet

\[ r(x) \leftarrow \text{REVERSE_DISCRETIZE}(r^h) \]

\[ \delta v^h = \delta v(x) \leftarrow \text{DEEPONET}(k(x), r(x)) \]

\[ v^h \leftarrow v + \delta v^h \]

else

\[ v^h \leftarrow \text{GAUSS_SEIDEL}(A^h, f^h, v^h) \]

end if

\[ k_{ii} \leftarrow k_{ii} + 1 \]

end while

return \( v^h = v(x) \)

end function

**Appendix B: Transfer learning scenarios**

Approaches based on conventional machine learning techniques frequently encounter significant difficulties when used in real-world applications. Collecting a sizable amount of labeled training data for a specific problem is costly or sometimes even impossible. Transfer learning (TL), which can improve learning capacity in a target domain by transferring knowledge from different but related domains, has been successful in handling such bottleneck of machine learning techniques in a variety of applications. Under this powerful paradigm of TL, various learning methodologies have been proposed, based on diverse hypothesis pertaining to the relationship between source and target domains, including covariate shift; prior probability shift; sample selection bias; conditional shift; and so on. For problems in the domain of scientific machine learning involving the learning of PDEs governing engineering problems, we focus on conditional shift across domains, where the source and target domains have different conditional distributions, while sharing the same marginal distribution, i.e., \( P(x_s) = P(x_t) \) and \( P(y_s|x_s) \neq P(y_t|x_t) \).
In this regression setting, we aim to approximate the mapping,
\[ f : x \in \mathbb{R}^{D_{\text{in}}} \rightarrow y \in \mathbb{R}^{D_{\text{out}}} \],
based on a training dataset of \( N \) input-output pairs, \( (X = \{x_1, ..., x_N\}, Y = \{y_1, ..., y_N\}) \), and achieve the lowest possible predictive error, where the dimensionality of the inputs and the outputs are denoted by \( D_{\text{in}} \) and \( D_{\text{out}} \), respectively. Suppose that a surrogate \( f_S \) is learned on a source domain, i.e., under specified conditions a dataset with \( N_s \) sufficient labeled data is generated \( D_s = \{(x_{si}, y_{si})\}_{i=1}^{N_s} \). Now consider a second target domain corresponding to different but related problem condition like different geometrical domains, boundary conditions, model parameters, etc., that has only few available labeled target data and there exists a conditional shift in the output space of the target domain with respect to the source domain. Training an independent surrogate model for the target domain with scarce data can lead to overfitting. In this work, we have employed an operator level TL for PDEs under conditional shift proposed in [17]. In this section we have put forward the constituents of the TL method.

The main idea behind the TL work can be elaborated with the following steps:

1. Train a source model with sufficient labeled data (i.e., model evaluations) from a source domain under a standard regression loss, \( \mathcal{L}_r(\theta^S) \), such as the relative \( L_2 \) error

\[
\mathcal{L}(\theta^S) = \mathcal{L}_r(\theta^S) = \frac{\|f_S(x^c) - y^i\|_2}{\|y^i\|_2},
\]

where \( \| \cdot \|_2 \) denotes the standard Euclidean norm and \( f_S(x^c) \), \( y^i \) are the prediction and reference responses, respectively.

2. Initialize the target model with the learned variables from the source domain.

3. Fine-tune the target model employing a hybrid loss function which is comprised of a regression loss and the Conditional Embedding Operator Discrepancy (CEOD) loss [33], used to measure the divergence between conditional distributions in a reproducing kernel Hilbert space (RKHS). The hybrid loss function, \( \mathcal{L}(\theta^T) \) reads as:

\[
\mathcal{L}(\theta^T) = \lambda_1 \mathcal{L}_r(\theta^T) + \lambda_2 \mathcal{L}_{\text{CEOD}}(\theta^T)
\]

\[
= \lambda_1 \frac{\|f_T(x^{IL}) - y^{IL}\|_2}{\|y^{IL}\|_2} + \lambda_2 \frac{\|\hat{C}_{Y\mid X}^{IL} - \hat{C}_{Y\mid X}^{IL}\|_{HS}^2}{\|\hat{C}_{XX}\|_{HS}},
\]

where \( \lambda_1 = 1 \) and \( \lambda_2 \gg \lambda_1 \) are trainable coefficients, which determine the importance of the two loss components during the optimization process and \( f_T(x^{IL}) \) and \( y^{IL} \) are the surrogate model prediction and the ground truth, respectively, of the labeled data on the target domain. In Eq. (9), \( \hat{C}_{Y\mid X}^{IL} \) and \( \hat{C}_{Y\mid X}^{IL} \) are defined as:

\[
\hat{C}_{Y\mid X} = \hat{C}_{YX} \hat{C}_{XX}^{-1},
\]

where \( \hat{C}_{YX} \) and \( \hat{C}_{XX} \) are the cross-covariance and self-covariance operators as defined by Song et al. [34], respectively.

4. While training the target model, only the deeper task-specific layers are fine-tuned while others remain frozen with constant parameters. To that end, we adapt to the concept prevalent in computer vision, where it is accepted that the convolutional layers are for learning...
the general features, while fully-connected layers are for task-specific learning. Hence, we fine-tune the fully-connected network of the branch CNN and the last layer of the trunk net, to allow for sufficient expressivity during the training of the target model while maintaining the low training cost.

The Conditional Embedding Operator Discrepancy (CEOD) is a measure the divergence between conditional distributions and is defined as:

\[
\left\| \tilde{C}_{Y,p\mid x_{p}} - \tilde{C}_{Y,q\mid x_{q}} \right\|_{HS}^{2} = \text{Tr}\left( (K_{X_{p}X_{p}} + \lambda N_{1}I)^{-1}K_{Y_{p}Y_{p}}(K_{X_{p}X_{p}} + \lambda N_{1}I)^{-1}K_{X_{p}X_{p}} \right) \\
+ \text{Tr}\left( (K_{X_{q}X_{q}} + \lambda N_{2}I)^{-1}K_{Y_{q}Y_{q}}(K_{X_{q}X_{q}} + \lambda N_{2}I)^{-1}K_{X_{q}X_{p}} \right) \\
- 2\text{Tr}\left( (K_{X_{p}X_{p}} + \lambda N_{1}I)^{-1}K_{Y_{p}Y_{q}}(K_{X_{q}X_{q}} + \lambda N_{2}I)^{-1}K_{X_{q}X_{p}} \right),
\]

where \( p \) and \( q \) denote two datasets, which in the context of Eq. (9) are the target labeled, \( \mathcal{L} \) and target unlabeled, \( \mathcal{U} \) data and \( K_{X_{i}X_{j}}(i, j) = k(x_{i}, x_{j}) \) is the Gram matrix calculated with a Gaussian kernel \( k \).

References

1. Hughes TJR (2012) The finite element method: linear static and dynamic finite element analysis. Courier Corporation, North Chelmsford
2. Simo JC, Hughes TJR (2006) Computational inelasticity, vol 7. Springer, Berlin
3. Hughes TJR, Cottrell JA, Bazilevs Y (2005) Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput Methods Appl Mech Eng 194(39–41):4135–4195
4. Jing L, Hudson JA (2002) Numerical methods in rock mechanics. Springer, Berlin
5. Rappaz M, Bellet M, Deville MO, Snyder R (2003) Numerical modeling in materials science and engineering. Springer, Berlin
6. Goswami S, Anitescu C, Rabczuk T (2019) Adaptive phase field modeling in materials science and engineering. Springer, Berlin
7. Hughes TJR, Cottrell JA, Bazilevs Y (2005) Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput Methods Appl Mech Eng 194(39–41):4135–4195
8. Goswami S, Li DS, Rego BV, Latorre M, Humphrey JD, Karniadakis GE (2022) Neural operator learning of heterogeneous mechanobiological insults contributing to aortic aneurysms. arXiv preprint. arXiv:2208.03780
9. Patera AT (1984) A spectral element method for fluid dynamics: laminar flow in a channel expansion. J Comput Phys 54(3):468–488
10. Kim J, Moin P, Moser R (1987) Turbulence statistics in fully developed channel flow at low Reynolds number. J Fluid Mech 177:133–166
11. Cockburn B, Karniadakis GE, Shu C-W (2012) Discontinuous Galerkin methods: theory, computation and applications, vol 11. Springer, Berlin
12. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
13. Goswami S, Yin M, Yue Yu, Karniadakis GE (2022) A physics-informed variational deepenon for predicting crack path in quasi-brittle materials. Comput Methods Appl Mech Eng 391:114587
14. Goswami S, Anitescu C, Chakraborty S, Rabczuk T (2020) Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theor Appl Fract Mech 106:102447
15. Zhang E, Dao M, Karniadakis GE, Suresh S (2022) Analyses of internal structures and defects in materials using physics-informed neural networks. Sci Adv 8(7):eabk6044
16. Zhang E, Kahana A, Turkel E, Ranade R, Pathak J, Karniadakis GE (2022) A hybrid iterative numerical transferable solver (HINTS) for PDEs based on deep operator network and relaxation methods. arXiv preprint. arXiv:2208.13273
17. Goswami S, Kontolati K, Shields MD, Karniadakis GE (2022) Deep transfer operator learning for partial differential equations under conditional shift. Nat Mach Intell 4:1155–1164
18. Briggs WL, Henson VE, McCormick SF (2000) A multigrid tutorial. SIAM, Philadelphia
19. Bramble JH (2019) Multigrid methods. Chapman and Hall/CRC, Boca Raton
20. van Gijzen MB, Erlangga YA, Vuik C (2007) Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian. SIAM J Sci Comput 29(5):1942–1958
21. Moore NS, Cyr E, Sieferl C (2022) Learning an algebraic multi-grid interpolation operator using a modified graphnet architecture. Technical report. Sandia National Laboratories (SNL-NM), Albuquerque
22. Luz I, Galun M, Maron H, Basri R, Yavneh I (2020) Learning algebraic multigrid using graph neural networks. In: International conference on machine learning. PMLR, pp 6489–6499
23. Götz M, Anzt H (2018) Machine learning-aided numerical linear algebra: convolutional neural networks for the efficient preconditioner generation. In: IEEE/ACM 9th workshop on latest advances in scalable algorithms for large-scale systems (scalA). IEEE, pp 49–56
24. Goswami S, Bora A, Yu Y, Karniadakis GE (2022) Physics-informed neural operators. arXiv preprint. arXiv:2207.05748
25. Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE (2021) Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nat Mach Intell 3(3):218–229
26. Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A, Anandkumar A (2021) Fourier neural operator for parametric partial differential equations. In: Proceedings of the international conference on learning representations
27. Zhang E, Spronck B, Humphrey JD, Karniadakis GE (2022) G2net: relating genotype and biomechanical phenotype of tissues with deep learning. PLoS Comput Biol 18(10):e1010660
28. Kahana A, Zhang Q, Gleyzer L, Karniadakis GE (2022) Function regression using spiking deeponet. arXiv preprint. arXiv:2205.10130
29. Yin M, Zhang E, Yu Y, Karniadakis GE (2022) Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems. In: Computer methods in applied mechanics and engineering, pp 115027
30. Kontolati K, Goswami S, Shields MD, Karniadakis GE (2022) On the influence of over-parameterization in manifold based surrogates and deep neural operators. arXiv preprint. arXiv:2203.05071
31. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint. arXiv:1412.6980
32. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch:
an imperative style, high-performance deep learning library. Adv
Neural Inf Process Syst 32:8026–8037
33. Liu X, Li Y, Meng Q, Chen G (2021) Deep transfer learning for
conditional shift in regression. Knowl Based Syst 227:107216
34. Song L, Huang J, Smola A, Fukumizu K (2009) Hilbert space
embeddings of conditional distributions with applications to
dynamical systems. In: Proceedings of the 26th Annual interna-
tional conference on machine learning, pp 961–968

Publisher’s Note Springer Nature remains neutral with regard to jurisdic-
tional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds
exclusive rights to this article under a publishing agreement with the
author(s) or other rightsholder(s); author self-archiving of the accepted
manuscript version of this article is solely governed by the terms of such
publishing agreement and applicable law.