TRANSFER LEARNING ON MULTIFIDELITY DATA

Dong H. Song & Daniel M. Tartakovsky*

Department of Energy Resources Engineering, Stanford University, Stanford, CA 94305, USA

*Address all correspondence to: Daniel M. Tartakovsky, Department of Energy Resources Engineering, Stanford University, Stanford, CA 94305, USA,
E-mail: tartakovsky@stanford.edu

Original Manuscript Submitted: 4/27/2021; Final Draft Received: 10/2/2021

Neural networks (NNs) are often used as surrogates or emulators of partial differential equations (PDEs) that describe the dynamics of complex systems. A virtually negligible computational cost of such surrogates makes them an attractive tool for ensemble-based computation, which requires a large number of repeated PDE solutions. Since the latter are also needed to generate sufficient data for NN training, the usefulness of NN-based surrogates hinges on the balance between the training cost and the computational gain stemming from their deployment. We rely on multifidelity simulations to reduce the cost of data generation for subsequent training of a deep convolutional NN (CNN) using transfer learning. High- and low-fidelity images are generated by solving PDEs on fine and coarse meshes, respectively. We use theoretical results for multilevel Monte Carlo method to guide our choice of the numbers of images of each kind. We demonstrate the performance of this multifidelity training strategy on the problem of estimation of the distribution of a quantity of interest, whose dynamics is governed by a system of nonlinear PDEs (parabolic PDEs of multiphase flow in heterogeneous porous media) with uncertain/random parameters. Our numerical experiments demonstrate that a mixture of a comparatively large number of low-fidelity data and smaller number of high-fidelity data provides an optimal balance of computational speed-up and prediction accuracy. The former is reported relative to both CNN training on high-fidelity images only and Monte Carlo solution of the PDEs. The latter is expressed in terms of both the Wasserstein distance and the Kullback–Leibler divergence.

KEY WORDS: *encoder-decoder, multifidelity, multiphase flow, neural network, shock, surrogate models, transfer learning, uncertainty quantification*

1. INTRODUCTION

Machine learning techniques, especially neural networks (NNs), have pervaded every facet of human activity and have permeated into the field of scientific computing. In the latter setting, NNs are used to approximate highly nonlinear and irregular functions (Friedman et al., 2001), solve (ordinary and partial) differential equations [e.g., Lee and Kang (1990), Lagaris et al. (1998), Fuks and Tchelepi (2020), among many others] and construct cheap surrogates for ensemble-based computation (e.g., Mo et al., 2019a; Raissi et al., 2019). Examples of the latter include inverse modeling (Mo et al., 2019b; Zhou and Tartakovsky, 2021), data assimilation (Tang et al., 2020), and uncertainty quantification (Tripathy and Bilionis, 2018; Zhu et al., 2019).

2689–3967/22/$35.00

A typical ensemble-based computation of practical significance involves repeated solutions of (coupled, nonlinear) partial-differential equations (PDEs)

$$\mathcal{N}(\mathbf{u};\boldsymbol{\theta}) = g(\mathbf{x},t;\boldsymbol{\theta}), \qquad (\mathbf{x},t) \in D \times (0,T], \tag{1}$$

which describe the spatiotemporal evolution of (a set of) state variables $\mathbf{u}(\mathbf{x},t)$ in the computational domain D over simulation time horizon $(0,T]$. Multiple solutions of Eq. (1)—for different values of the inputs $\boldsymbol{\theta}(\mathbf{x},t)$ that parameterize the differential operator $\mathcal{N}$, the source function g, and auxiliary functions in the initial and/or boundary conditions—are required because these values are known at best in terms of their distributions, which are either inferred from data or provided by the expert. The high computational cost of solving Eq. (1) numerically often precludes one from generating enough samples to obtain meaningful statistics of $\mathbf{u}(\mathbf{x},t)$ or the derived quantities of interest. A surrogate of Eq. (1) carries a negligible cost, making possible ensemble-based computation with arbitrarily small sampling error.

Alternative strategies for surrogate construction include polynomial chaos expansions (Xiu, 2010), Kriging or Gaussian processes (Couckuyt et al., 2014), polynomial regression (Montgomery and Evans, 2018), tensor-product splines (Hwang and Martins, 2018), and random forests (Breiman, 2001). The current popularity of NN-based surrogates (Mo et al., 2019a; Raissi et al., 2019) is grounded in the scalability and approximation capabilities of deep NNs (Friedman et al., 2001; Tripathy and Bilionis, 2018). Regardless of the surrogate type, the training of a surrogate requires a large number of solutions of Eq. (1) for different combinations of parameter values $\boldsymbol{\theta}$. Advanced computer architectures, e.g., CUDA-compatible graphics processing units (GPUs) and tensor processing units (TPUs), are almost a necessity to train a large NN. A combined cost of training-data acquisition and NN training can be so large as to negate the benefits of the NN.

This observation suggests that the practical utility of an NN as a surrogate model hinges on one's ability to dramatically reduce the cost of its construction. We rely on multifidelity simulations to reduce the cost of data generation for subsequent training of a deep convolutional NN (CNN) using transfer learning. High- and low-fidelity images are generated by solving Eq. (1) on fine and coarse meshes, respectively. A fine mesh is defined by the need to resolve the spatiotemporal variability of the model's inputs $\boldsymbol{\theta}$ and outputs $\mathbf{u}$; the resulting high-fidelity simulation carries a high computational cost. Lower-fidelity solutions of Eq. (1), obtained on coarser meshes on which appropriately homogenized inputs $\boldsymbol{\theta}_{\text{hom}}$ are defined, are cheaper to compute but less accurate. We train a CNN on a mixture of these multifidelity data, using the theoretical results for multilevel Monte Carlo (MLMC) (Giles, 2008; Heinrich, 1998, 2001; Taverniers et al., 2020) to guide our choice of the numbers of solutions $\mathbf{u}(\mathbf{x},t)$ of each kind. Similar to MLMC (Müller et al., 2013; Peherstorfer, 2019), the varying fidelity (aka "levels") of predictions of $\mathbf{u}$ can be achieved not only by solving Eq. (1) on different meshes, but also by replacing Eq. (1) with its cheaper-to-compute counterparts. For example, the multiphase flow equations used as the computational testbed in this study can be replaced with the cheaper-to-solve Richards equation and Green–Ampt equation (Sinsbeck and Tartakovsky, 2015; Yang et al., 2020), each of which encapsulates progressively simplified physics. We leave this aspect of NN training on multifidelity data for a follow-up study.

The idea of using multifidelity data in the context of NNs is not unique to this study. For example, Geneva and Zabaras (2020) trained surrogate models using low-fidelity simulations as a conditional input. Meng and Karniadakis (2020) built fully connected NN surrogates by training different networks to handle the low- and-high fidelity data. The novelty of our approach is to deploy transfer learning on multifidelity data to train different parts of a single network.

Section 2 contains a brief description of our CNN and the workflow for its training on multifidelity of data. The performance of this algorithm is tested on a system of nonlinear parabolic PDEs governing multiphase flow in a heterogeneous porous medium with uncertain properties, which are formulated in Section 3. In Section 4, we demonstrate the accuracy and computational efficiency of the CNN-based surrogate used to quantify predictive uncertainty of Eq. (1) in terms of the distribution of a quantity of interest. The main conclusions drawn from this study are presented in Section 5.

2. DEEP CONVOLUTIONAL NEURAL NETWORKS

While many flavors of NNs can be used as a surrogate for a PDE-based model like Eq. (1), we choose CNNs because of their proven ability to model complex nonlinear phenomena and the negligible cost of their forward pass. To be specific, we select the CNN with encoder-decoder architecture (Mo et al., 2019b), which has previously been used for single-phase (Mo et al., 2019a) and multiphase (Mo et al., 2019b) flow problems in the context of uncertainty quantification. The presence of dense-blocks differentiates the selected architecture from other encoder-decoders such as the one found in U-Net (Ronneberger et al., 2015). The dense blocks have connections between nonadjacent layers to enhance the flow of information through a network (Mo et al., 2019a). The encoder-decoder architecture is ideally suited for training on multifidelity data, as detailed in Section 2.1.

The CNN-based surrogate is set up as an image-to-image regression model (Zhou and Tartakovsky, 2021). To train and test the network, we use the parameter values $\boldsymbol{\theta}(\mathbf{x}_i)$ in N_{el} elements $\{\mathbf{x}_i\}_{i=1}^{N_{\text{el}}}$ of a numerical grid as input and the discretized solution $\mathbf{u}(\mathbf{x}_i, t_k)$ of PDE (1) at N_{ts} time steps $\{t_k\}_{k=1}^{N_{\text{ts}}}$ as output. To facilitate the generalizibility of the trained CNN to unseen sets of the input $\boldsymbol{\theta}(\mathbf{x}_i)$, i.e., to ensure that the CNN is not overfitted to a particular choice of $\boldsymbol{\theta}(\mathbf{x}_i)$, the training data comprises a large number N_{train} of the solutions $\mathbf{u}$ obtained for N_{train} realizations $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{N_{\text{train}}}\}$ of the input $\boldsymbol{\theta}$. The loss function

$$\mathcal{L}(\mathbf{w}) = \sum_{m=1}^{N_{\text{train}}} \sum_{i=1}^{N_{\text{el}}} \sum_{k=1}^{N_{\text{ts}}} |\mathbf{u}(\mathbf{x}_i, t_k; \boldsymbol{\theta}_m) - \hat{\mathbf{u}}_{ik}(\mathbf{w}; \boldsymbol{\theta}_m)| + \lambda \sum_{n=1}^{N_{\text{w}}} w_n^2, \tag{2}$$

consists of two parts. The first represents the L_1-norm discrepancy between the state variables $\mathbf{u}$ predicted by solving PDE (1), $\mathbf{u}(\mathbf{x}_i, t_k)$ and estimated by the CNN, $\hat{\mathbf{u}}_{ik}(\mathbf{w})$, with N_{w} weights $\mathbf{w} = (w_1, \ldots, w_{N_{\text{w}}})^\top$. The L_2-norm regularization prevents overfitting by penalizing large weights $\mathbf{w}$ associated complex models; the regularization parameter λ determines how much regularization penalty is applied. The CNN training consists of finding a set of weights $\mathbf{w}^\star$ that minimizes $\mathcal{L}$.

2.1 Transfer Learning

The construction of CNN-based generalizable surrogates, which are capable of making predictions for realizations of $\boldsymbol{\theta}(\mathbf{x})$ not seen during training, requires a large number of PDE solutions, N_{train}, e.g., $N_{\text{train}} \sim 1500$, was used by Mo et al. (2019a) and Zhou and Tartakovsky (2021) to train the encoder-decoder CNNs similar to ours. When a single PDE solution is expensive, the costs associated with large N_{train} can be large to the point where CNN training becomes unfeasible. To alleviate this problem, we use both multifidelity data and transfer learning (Donahue et al., 2014). The latter is a technique that uses an NN trained for one task as the starting point

for a different NN being trained for a new task. Transfer learning has been implemented for face detection (Jiang and Learned-Miller, 2017), generation of image description (Karpathy and Fei-Fei, 2015), and construction of physics-informed NNs (Haghighat et al., 2021), among other applications.

Let HFS and LFS data refer to the solutions of Eq. (1), $\mathbf{u}(\mathbf{x}_i, t_k)$, obtained on the fine ($N_{\text{el}} = N_{\text{el}}^{\text{HFS}}$) and coarse ($N_{\text{el}} = N_{\text{el}}^{\text{LFS}}$ with $N_{\text{el}}^{\text{LFS}} < N_{\text{el}}^{\text{HFS}}$) meshes, respectively. If N_{w} in Eq. (2) denotes the number of weights in the CNN trained on the HFS data, then our implementation of transfer learning starts with the construction of a CNN composed of N_{LFS} ($N_{\text{LFS}} < N_{\text{w}}$) weights $\mathbf{w}_{\text{LFS}} = (w_1, \ldots, w_{N_{\text{LFS}}})^\top$ trained on the LFS data. Then, the HFS data are used to train the desired high-resolution CNN, i.e., to determine the remaining weights $\mathbf{w}_{\text{HFS}} = (w_{N_{\text{LFS}}+1}, \ldots, w_{N_{\text{w}}})^\top$. This transfer learning strategy is depicted in Fig. 1 and detailed in the following.

2.2 Workflow for CNN Training on Multifidelity Data

Our strategy for CNN training on multifidelity data consists of three phases (Fig. 1), each of which results in a CNN denoted by M_i ($i = 1, 2, 3$). During Phase 1, the CNN M_1 with the $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$ output is trained on the LFS data. In Phase 2, the CNN M_2 with $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$ output is constructed by adding an additional layer with the weights $\mathbf{w}_{\text{HFS}}$, which are trained on the HFS data while keeping the original weights $\mathbf{w}_{\text{LFS}}$ fixed. Phase 3 consists of fine-tuning the CNN M_2 by allowing all the weights $\mathbf{w} = \{\mathbf{w}_{\text{LFS}}, \mathbf{w}_{\text{HFS}}\}$ to update during the training on the same HFS data. The numerical experiments reported in Sections 3 and 4 demonstrate that this transfer learning strategy significantly reduces the number of high-resolution PDE solutions.

The workflow of our approach is provided below (see Appendix A for the corresponding pseudocode).

Phase 1: Train a CNN M_1, with $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$ output, on the LFS data.

State 1.1: Initialize the transfer learning by employing the encoder-decoder CNN of Mo et al. (2019b), M_{init} with $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$ output, whose N_{w} weights $\mathbf{w}$ are set to `PyTorch` defaults.

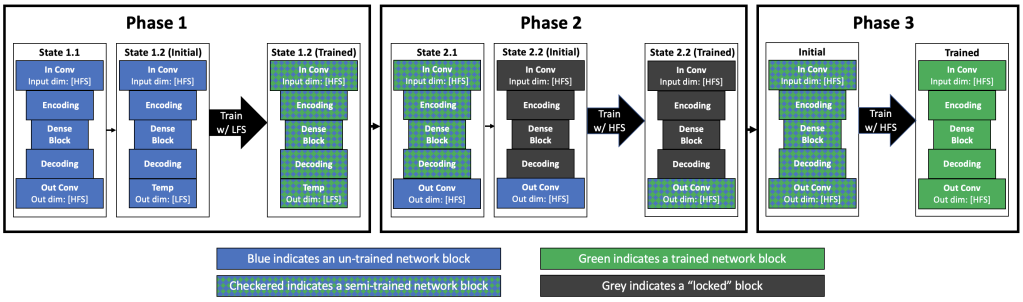

FIG. 1: Workflow for CNN training on multifidelity data. Phase 1 returns a low-resolution CNN trained on the LFS data. Phase 2 supplements that network with an additional layer whose weights are determined from the HFS data, producing a high-resolution CNN. In Phase 3, the latter is fine-tuned by allowing all the weights to vary during the training on the same HFS data. Appendix A provides a pseudocode for all three phases.

State 1.2: Train the CNN M_1 on the LFS data. The starting point is a CNN obtained from M_{init} by replacing its last layer L_{last}, which has N_{HFS} weights $\mathbf{w}_{\text{HFS}}$, with a temporary convolution layer L_{temp}. The latter is composed of weights $\mathbf{w}_{\text{temp}}$ and makes the output of M_1 match the dimensions of the LFS data, $[N_{\text{ts}} \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}]$. Then, the weights of M_1, $\mathbf{w}_{\text{phase1}} = \{\mathbf{w}_{\text{HFS}}, \mathbf{w}_{\text{temp}}\}$ are trained on the LFS data by minimizing Eq. (2).

Phase 2: Train a CNN M_2, with N_{w} weights $\mathbf{w} = \{\mathbf{w}_{\text{LFS}}, \mathbf{w}_{\text{HFS}}\}$ (of which N_{LFS} weights are locked) and $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$ output, on the HFS data.

State 2.1: Build a CNN from M_1 by replacing its layer L_{temp} with the layer L_{last} and discarding L_{temp}. The modified CNN has weights $\mathbf{w} = \{\mathbf{w}_{\text{LFS}}, \mathbf{w}_{\text{HFS}}\}$, among which weights $\mathbf{w}_{\text{LFS}}$ have been updated by data and $\mathbf{w}_{\text{HFS}}$ have not been updated by data.

State 2.2: Train the resulting CNN M_2 on the HFS data by minimizing Eq. (2) over the weights $\mathbf{w}_{\text{HFS}}$ of layer L_{last}, while keeping the remaining weights $\mathbf{w}_{\text{LFS}}$ fixed at their values in M_1.

Phase 3: Train a CNN M_3 on the HFS data by allowing all weights $\mathbf{w}$ of M_2 to vary during the minimization.

Because the bulk of the CNN M_3 training is carried out on the LFS data, this procedure is more efficient than CNN training solely on HFS data. The predictive capability of the trained M_2 is only sometimes similar to that of the trained M_3; Phase 3 improves the performance if changes are needed in the layers which were locked during Phase 2.

3. COMPUTATIONAL EXAMPLE: MULTIPHASE FLOW

Numerical solution of problems involving multiphase flow in porous media is notoriously difficult because of the high degree of nonlinearity and stiffness of the governing PDEs. Each forward solve of these PDEs is so expensive that it is uncommon; e.g., uncertainty quantification efforts in petroleum engineering have been based on as few as three model runs. This high cost and numerical complexity make the multiphase flow equations a challenging testbed for ensemble-based simulations.

We consider horizontal flow of two incompressible and immiscible fluids, with viscosities μ_1 and μ_2, in a heterogeneous, incompressible, and isotropic porous medium D. The latter is characterized by porosity ϕ and intrinsic permeability k. The porosity is assumed to be constant at $\phi = 0.25$, and intrinsic permeability $k(\mathbf{x})$ is treated as a random variable. The time domain t is between zero and a specified terminal time T. Mass conservation of the ℓth fluid phase ($\ell =$ 1,2) implies

$$\phi \frac{\partial S_\ell}{\partial t} + \nabla \cdot \mathbf{v}_\ell + q_\ell = 0, \qquad \mathbf{x} \equiv (x_1, x_2)^\top \in D, \quad t \in [0, T], \tag{3a}$$

where $S_\ell(\mathbf{x}, t)$ is the phase saturation constrained by $S_1 + S_2 = 1$; q_ℓ is the source/sink term; and the macroscopic velocity $\mathbf{v}_\ell(\mathbf{x}, t)$ is described by the generalized Darcy law

$$\mathbf{v}_\ell = -k \frac{k_{r\ell}}{\mu_\ell} \nabla P_\ell. \tag{3b}$$

The relative permeability for the ℓth phase, $k_{r\ell}$, varies with the phase saturation, $k_{r\ell} = k_{r\ell}(S_\ell)$, in accordance with the Brooks–Corey constitutive model (Corey, 1954). Following Taverniers et al. (2020) and many others, we neglect the capillary forces, i.e., we assume pressure within the two phases to be equal, $P_1 = P_2 \equiv P(\mathbf{x}, t)$; this is a common assumption in applications to reservoir engineering and carbon sequestration.

The two-dimensional computational spatial domain D is a 150 $\times$ 150 m square (Fig. 2) with impermeable bottom (Γ_b or $x_2 = 0$) and top (Γ_t or $x_2 = 150$ m) boundaries; Dirichlet conditions are imposed along the left (Γ_l or $x_1 = 0$) and right (Γ_r or $x_1 = 150$ m) boundaries:

$$\frac{\partial P}{\partial x_2} = 0, \quad \mathbf{x} \in \Gamma_\mathrm{b} \cup \Gamma_\mathrm{t}; \quad P = 10.2 \ \text{ and } \ S_1 = 1.0, \quad \mathbf{x} \in \Gamma_\mathrm{l}; \quad P = 10.1, \quad \mathbf{x} \in \Gamma_\mathrm{r}; \tag{4a}$$

here and below, the pressure P is expressed in MPa. Initial conditions are

$$P(\mathbf{x}, 0) = 10.1, \qquad S_1(\mathbf{x}, 0) = 0, \qquad \mathbf{x} \in D. \tag{4b}$$

All the model parameters, except for the intrinsic permeability $k(\mathbf{x})$, are assumed to be constant and known with certainty. The uncertain permeability $k(\mathbf{x})$ is modeled as a second-order stationary random field, such that $Y(\mathbf{x}) = \ln k$ is multivariate Gaussian with mean $\langle Y \rangle = 0$, variance $\sigma_Y^2 = 2.0$, and an exponential two-point covariance $C(\mathbf{x}, \mathbf{y}) = \sigma_Y^2 \exp(-|\mathbf{x} - \mathbf{y}|/\lambda_Y)$ with the correlation length $\lambda_Y = 19$ m. We use a truncated Karhunen–Loéve expansion with $p = 31$ terms to represent $Y(\mathbf{x})$ (Taverniers et al., 2020). A representative realization of the resulting permeability field is shown in Fig. 2 for the 128 $\times$ 128 mesh.

Equations (3)–(4) are approximated using a finite volume scheme in space and implicit Euler scheme in time, yielding a highly nonlinear algebraic system (Aziz, 1979). Adaptive time-stepping is implemented to advance the solution in time. At each time step, the nonlinear algebraic system is solved through Newton–Raphson (NR) iterations with the modified Appleyard update dampening (Appleyard et al., 1981) that improves the convergence of NR iterations by capping the maximum saturation update to a specified limit. For the νth iteration and the ith cell of volume V_i, the convergence criteria are

$$\max_i \left| \Delta t \left(\frac{r_{\ell,i}}{\phi V_i} \right) \right| < \epsilon_1, \quad \max_i |P_i^{(\nu+1)} - P_i^{(\nu)}| < \epsilon_2, \quad \max_i |S_{\ell,i}^{(\nu+1)} - S_{\ell,i}^{(\nu)}| < \epsilon_3, \tag{5}$$

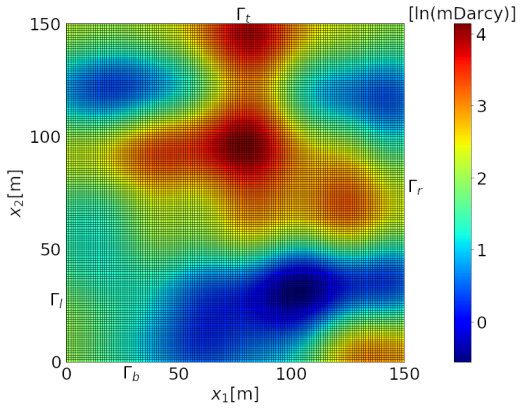

FIG. 2: A representative realization of log permeability field $Y = \ln k$ on the 128 $\times$ 128 grid, which is used in high-fidelity simulations. Permeability k is expressed in mDarcy.

where $r_{\ell,i}$ is the residual of the mass balance of phase ℓ, Δt is the time step, the relative residual norm $\epsilon_1 = 10^{-6}$, the maximum pressure update $\epsilon_2 = 10^{-3}$, and the maximum saturation update $\epsilon_3 = 10^{-2}$.

3.1 Upscaling of Permeability

Multifidelity data are generated by solving Eqs. (3)–(4) on progressively coarsened grids: the 128 $\times$ 128 and 64 $\times$ 64 grids are used for HFS and LFS, respectively. The spatial discretizations of these HFS and LFS ($\Delta x = 1.17$ and 2.34 m, respectively) are sufficient to capture the randomness in permeability fields. The latter rests on the "rule of thumb" requirement that Δx be such that $4\Delta x \leq \lambda_Y$, i.e., that a numerical mesh should have at least four elements of length Δx per correlation length λ_Y [e.g., Ye et al. (2004) and references therein]. Our HFS and LFS satisfy this requirement, since we use $\lambda_Y = 19$ m.

This grid coarsening must be accomplished by upscaling (coarsening) of the realizations of the random permeability $\hat{k}$ which are initially generated at the finest scale (Fig. 2). Among alternative upscaling strategies (Boso and Tartakovsky, 2018; Paleologos et al., 1996; Tartakovsky and Neuman, 1998), we select the one proposed by Durlofsky (2005) because of its computational simplicity. It turns a scalar permeability field defined on the fine (128 $\times$ 128) mesh into its upscaled tensorial (anisotropic) counterpart whose off-diagonal components are 0 and the diagonal components are computed as the distance-weighted arithmetic mean perpendicular to the direction of flow and the distance-weighted harmonic mean in the direction of flow.

3.2 Data Acquisition

Multifidelity training data come in the form of $N_{\text{ts}} = 16$ temporal snapshots of the saturation $S_1(\mathbf{x}, t)$ computed by solving Eqs. (3)–(4) on the $N_{\text{el}} \times N_{\text{el}}$ grids with $N_{\text{el}} = 128 \equiv N_{\text{el}}^{\text{HFS}}$ and $64 \equiv N_{\text{el}}^{\text{LFS}}$. Figure 3 shows examples of such images, corresponding to the permeability field in Fig. 2. The permeability fields on the finest mesh, $[1 \times N_{\text{HFS}} \times N_{\text{HFS}}]$, are used as the input θ for all CNNs. The size of the CNN, $[N_{\text{ts}} \times N_{\text{el}} \times N_{\text{el}}]$, depends on the size of the training data.

The numerical solutions of Eqs. (3)–(4) are obtained using a `Matlab`-based multiphase flow simulator on a computer with an Intel Core i7-4790 3.6GHz processor and 64GB of RAM. The

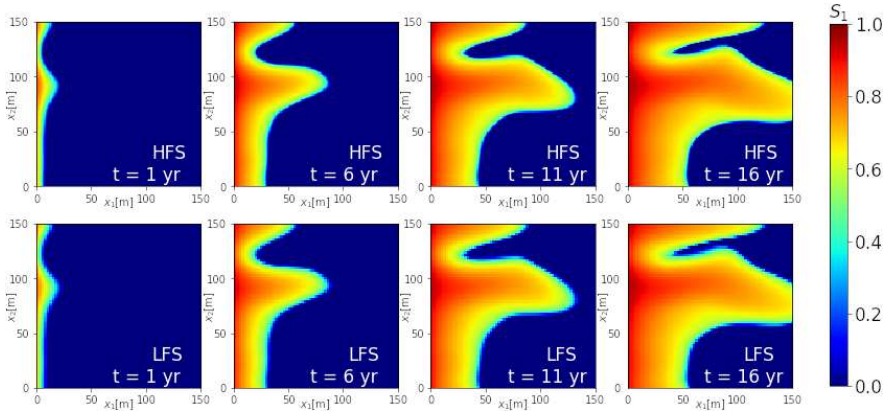

FIG. 3: Temporal snapshots of saturation $S_1(\mathbf{x}, t)$ computed with (top-row) HFS and (bottom-row) LFS for the permeability field $k(\mathbf{x})$ in Fig. 2

computation time for each HFS data point is 219.13 and 37.13 s for each LFS data point. The time needed to generate a data set is henceforth referred to as the *data-generation budget*.

3.3 CNN Training

Table 1 describes the CNN architecture used in the implementation of our approach (see Fig. 1). The model implementation and training is done using `PyTorch` and other open source packages. The computations were carried out on the Stanford Mazama high-performance computing cluster. The allocated computing resources include Intel Xenon Gold 6126 CPU (2.6 GHz), 60GB RAM, and Nvidia V100 GPU with 16GB vRAM. (Although available, multicores were not used for this work.)

The key hyper-parameters affecting the CNN performance are the learning rate (LR), the weight decay (WD), the factor (F), and the minimum learning rate (mLR). The LR and WD are parameters of the Adam optimizer (Kingma and Ba, 2014), and the F and mLR are parameters of the `ReduceLROnPlateau` scheduler. The CNN training involves many more hyper-parameters, but we use their default values in `PyTorch`. The the regularization parameter λ is specified through WD following the implementation of Loshchilov and Hutter (2017). Further information on the hyper-parameters, schedulers, and optimizers can be found in the `PyTorch` documentation (Paszke et al., 2019).

The hyper-parameters used by Mo et al. (2019b) in a similar CNN architecture serve as an initial guess for the hyper-parameter optimization. The search is iterated through variations of LR, WD, F, and mLR, in this order. Once an acceptable value of a hyper-parameter was found, the search moved to the next hyper-parameter. A robust grid search may yield a more optimal set of hyper-parameters. The search required 100 HFS, with each training pass taking about 0.65 hours to complete, when 200 epochs were used. It took 7.2 training-hours to find functional hyper-parameters (12 training passes), and a considerably smaller wall-clock time because this task was parallelized across several GPU nodes. We selected the hyper-parameter values yielding

TABLE 1: Model block description and the input and output dimensions of each model block. In our numerical experiments, the number of time steps is $N_{\text{ts}} = 16$; the number of elements in fine and coarse meshes is $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}} = 128 \times 128$ and $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}} = 64 \times 64$, respectively; the number of elements in the output of the dense block is $N_{\text{dense}} = 32$; and the number of channels in each of the seven layers of the CNN is $n_1 = 64$, $n_2 = 344$, $n_3 = 172$, $n_4 = 652$, $n_5 = 326$, $n_6 = 606$, and $n_7 = 303$

Layer	Input	Output
Input: permeability field k	$1 \times N_{\text{HFS}} \times N_{\text{HFS}}$	
Convolution 1	$n_1 \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	$n_2 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Dense block (encoding)	$n_2 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$n_3 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Convolution 2	$n_3 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$
Dense block	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$
Convolution Transpose 1	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_6 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Dense block (decoding)	$n_6 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$n_7 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Convolution Transpose 2	$n_7 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$N_{\text{ts}} \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$
Output: saturation map $\hat{S}$	$N_{\text{ts}} \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	

the smallest root mean square error (RMSE) on the HFS test data (Fig. 4). These values are used as a starting point in the hyper-parameter optimization for multifidelity transfer learning. Then, the LR and epochs at each phase (Section 2.2) are modified to minimize the RMSE on the corresponding test data. The resulting hyper-parameter values are shown in Table 2.

4. RESULTS

Once trained (in this example, on 573 LFS and 100 HFS, which took 12 hours to generate), the CNN surrogate provides an accurate approximation of the PDE solution on the fine mesh (Fig. 5), even for such highly nonlinear problems as Eq. (3) that exhibit sharp dynamic fronts. A forward pass of the CNN surrogate is on the order of a second, whereas a fine-mesh PDE solution takes nearly 220 s. This two-orders-of-magnitude speedup makes CNN surrogates an invaluable tool for uncertainty quantification (UQ) (Section 4.2).

4.1 Model Performance

We compare the relative performance of the CNN trained on multifidelity data and the CNNs trained on either HFS data or LFS data, in terms of both accuracy (RMSE on test data) and computational cost. We also investigate the effect of varying the amount of HFS and LFS data for a given computational budget of 12 hours.

To train the high-resolution (128×128 output) CNN solely on the LFS (64×64) data, the latter have to be downscaled to match the dimensions. We do so by taking the Kronecker product of a 64×64 LFS image and a 2×2 matrix of 1 s. The transformed LFS data have the desired dimensions, while containing the same information as the original image. The test data are composed of HFS images (PDE solutions on fine mesh) that were not used for CNN training. Figure 6 exhibits the RMSEs on test data of the CNNs trained on high-, low-, and multifidelity

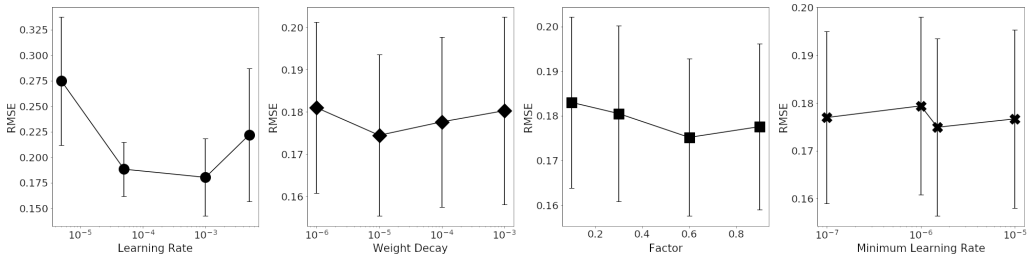

FIG. 4: Hyper-parameter performance in the neighborhood of optimum hyper-parameter set in terms of the root mean square error (RMSE) for the test data. Unless labeled as the x-axis variable, all plots correspond to LR $= 5 \times 10^{-5}$, WD $= 1 \times 10^{-5}$, F $= 0.6$, and mLR $= 5 \times 10^{-6}$. Each data point represents the mean and standard deviation of 10 training sessions.

TABLE 2: Learning rates and epochs used at each phase

	Learning rate	**Epochs**
Phase 1	5×10^{-4}	170
Phase 2	5×10^{-5}	150
Phase 3	10^{-5}	100

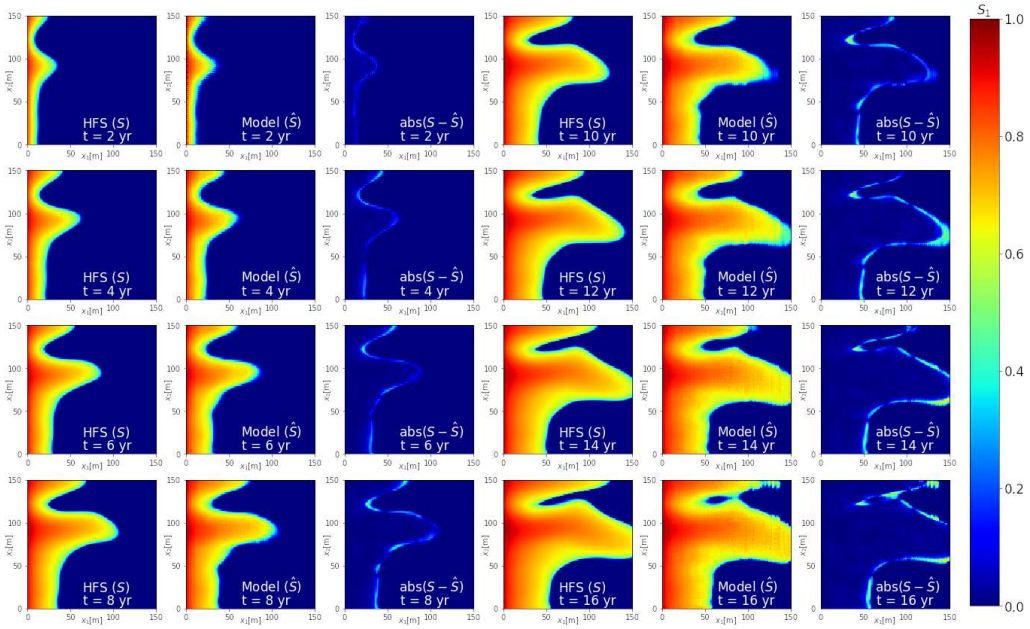

FIG. 5: Temporal snapshots of the saturation maps $S_1(\mathbf{x}, t)$ for the permeability field $k(\mathbf{x})$ in Fig. 2. These are generated with either HFS of the PDE model in Eqs. (3) and (4) (labeled as S in the first and fourth columns) or the CNN surrogate (labeled as $\hat{S}$ in the second and fifth columns). The third and sixth columns display the absolute difference between the two predictions, $|S - \hat{S}|$.

data as function of the computational budget; each point in these graphs represents an average over 10 repetitions of training and is accompanied by error bars (the standard deviation).

The left plate of Fig. 6 reveals that, if the data-generation budget does not exceed 20 hours, the CNN trained on the LFS data outperforms its HFS-trained counterpart in terms of RMSE. That is because such budgets do not allow for generation of sufficient amounts of HFS data. As the budget increases, the error of the LFS data precludes the RMSE of the CNN trained on such data from dropping below 0.125 while the RMSE of the HFS-trained CNN continues to decrease. This finding is reminiscent of the cost-constrained selection between high- and low-fidelity models in the context of ensemble-based simulations (Sinsbeck and Tartakovsky, 2015; Yang et al., 2020). This figure also demonstrates that, for a relatively small budget of 12 hours, the use of multifidelity data yields the CNN whose RMSE is appreciably smaller that those of the CNNs trained on either HFS data or LFS data.

An optimal mix of the HFS and LFS data is investigated in the right plate of Fig. 6. The multifidelity training was conducted five times for each HFS/LFS ratio, with random selection of LFS/HFS from a larger pool of data. At the empirically optimal mix of 573 LFS and 100 HFS, five of our experiments yield RMSE values of 0.097, 0.12, 0.098, 0.105, and 0.110. Two of the five CNNs trained on 12 hours, worth of these multifidelity data achieve lower RMSEs than the RMSE of 0.099 for the CNN trained on 79 hours, worth of HFS data. This LFS/HFS ratio lies near the range, 1.5–5.5, suggested for multilevel Monte Carlo method (Taverniers et al., 2020). Based on the theoretical results and the numerical experiments presented here, we recommend the LFS/HFS ratio of 5 as a suitable initial guess. For the data-generation budget of 12 hours, a mix dominated by the LFS data results in a CNN whose RMSE on test data exceeds 1.0 (beyond

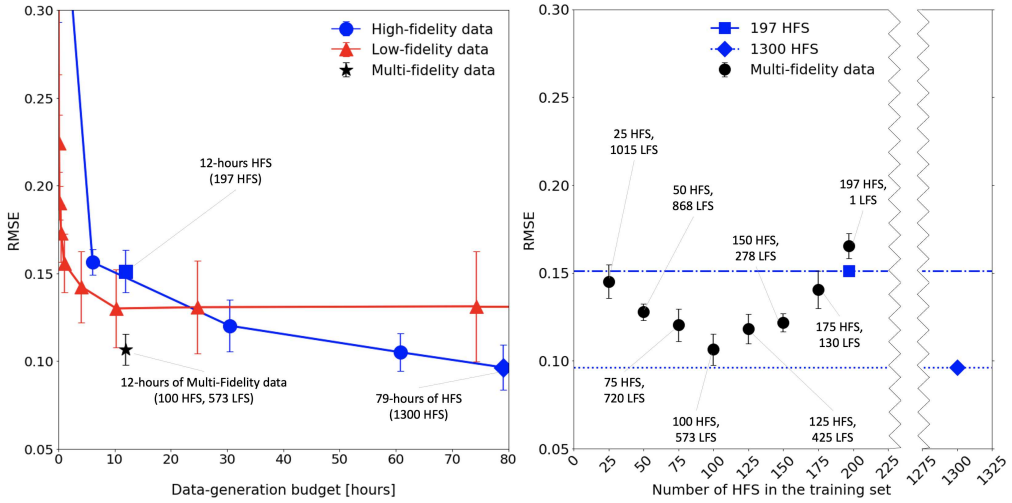

FIG. 6: RMSE on test data for the alternative CNN training strategies. It is plotted as function of the budget allocated for data generation (left) and the number of PDE solutions on the fine mesh used to generate HFS data (right). Each RMSE point in these graphs represents an average over 10 iterations of training and is accompanied by error bars (the standard deviation). The left plate provides RMSE for the CNNs trained on high-fidelity (blue circles), low-fidelity (red triangles), or multifidelity (black star) data. The latter corresponds to the CNN trained on an optimal (the lowest RMSE) mix of high- and low-fidelity data for a set budget of 12 hours; it is contrasted with the RMSE of the CNN trained on the HFS data generated within the same budget (blue square). The black circles in the right plate represent RMSE of the CNN trained on the multifidelity data sets, in which the number of HFS varies while the data-generation budget is fixed at 12 hours. Also shown there are RMSEs of the CNNs trained on 12 hours (dot-dashed line) and 79 hours (dotted line) of HFS.

the scale of Fig. 6), which indicates that the network's last Convolution Transpose 2 layer is not meaningfully trained.

4.2 CNN Surrogates for Uncertainty Quantification

Finally, we investigate the utility of our CNN surrogates for uncertainty quantification. A quantity of interest is the breakthrough time, T_{break}, at the $x_1 = 100$ m plane (Fig. 2), with the term *breakthrough* defined as the saturation of the invading phase (S_1) exceeding 0.15. Given uncertainty in intrinsic permeability $k(\mathbf{x})$, a solution of Eq. (3) and, hence, predictions of T_{break} are given in terms of their cumulative distribution functions (CDFs) or probability density functions (PDFs).

Figure 7 exhibits the CDF and PDF of T_{break} alternatively computed with HFS and LFS Monte Carlo and with the CNN trained on the multifidelity data. The distributions obtained via Monte Carlo method consisting of 292 hours of HFS are treated as ground truth. The distributions obtained from 24 hours of LFS involve a sufficient number of samples for the error to be attributable solely to the low resolution, i.e., to the discretization errors in solving PDEs. The numbers of HFS samples generated during either 6 or 12 hours of simulations are insufficient for the Monte Carlo method to converge, leading to the appreciable errors in estimation of PDF and CDF of T_{break}. The CNN trained on multifidelity data yields accurate estimates of these quantities, while requiring only 12 hours of data generation.

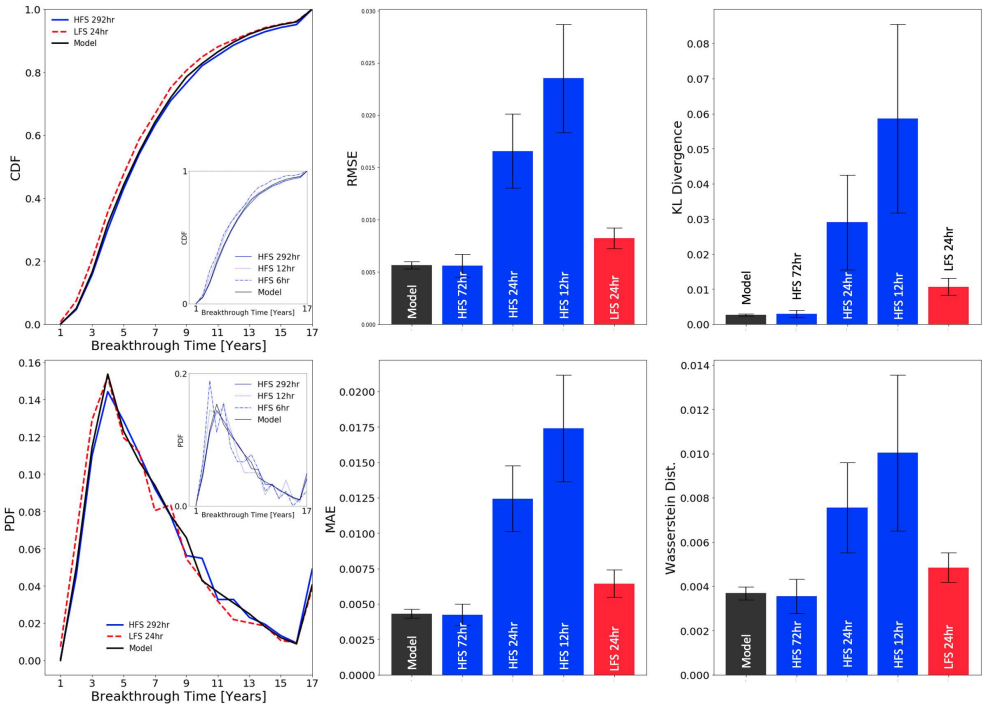

FIG. 7: Left: The converged CDF (top) and PDF (bottom) of breakthrough time is calculated using MC simulations of HFS, LFS, and the CNN surrogate model. The CDF and PDF calculated from varying amounts of HFS are displayed on the subplots. Bar plots: RMSE (middle-top), MAE (middle-bottom), KL divergence (right-top), and Wasserstein distance (right-bottom) from PDF calculated using CNN model, HFS, and LFS.

In addition to visual comparison, the alternative strategies for estimation of the distributions of T_{break} are compared in terms of RMSE, mean absolute error (MAE), the Kullback–Leibler (KL) divergence, and the Wasserstein distance. The UQ task was repeated 50 times, with Fig. 7 displaying the mean and standard deviation of these measures of discrepancy. We found 3200 forward passes of the CNN to be sufficient for the CDF/PDF estimates to converge; this UQ task took about 10 minutes, whereas an equivalent HFS Monte Carlo method takes 194 hours. By every discrepancy measure, the CNN estimates outperform the converged LFS Monte Carlo method and are at least as accurate as the HFS Monte Carlo method using 72 hours of data. Likewise, the CNN estimates are vastly more accurate than the HFS Monte Carlo of a similar data-generation budget.

5. CONCLUSIONS

We proposed a transfer learning-based approach to train a CNN on multifidelity (e.g., multi-resolution) data. High- and low-fidelity images were generated by solving a PDE on fine and coarse meshes, respectively. The performance of our algorithm was tested on a system of nonlinear parabolic PDEs governing multiphase flow in a heterogeneous porous medium with uncertain (random) permeability. A quantity of interest (QoI) in this example is the PDF or CDF of the breakthrough time of an invading fluid. Our analysis leads to the following major conclusions.

1. CNN surrogates trained on multifidelity data provide an accurate approximation of the PDE solution on the fine mesh, even for highly nonlinear problems that exhibit sharp dynamic fronts. A forward pass of the CNN surrogate is two orders of magnitude faster than a PDE solution on the fine-mesh. This speedup makes CNN surrogates an invaluable tool for ensemble-based computation of the PDF/CDF of a QoI.

2. CNN training on multifidelity data reduces the data-generation budget 7-fold relative to to CNN training on HFS data alone. If the budget is relatively small, the CNN trained on the LFS data is more accurate than its HFS-trained counterpart. As the budget increases, the opposite is true. This finding is reminiscent of the cost-constrained selection between high- and low-fidelity models in the context of ensemble-based simulations.

3. For a small data-generation budget (12 hours, in our example), the CNN trained on multifidelity data exhibits an appreciably smaller RMSE on test data than the CNNs trained on either HFS or LFS data. Performance of the multifidelity CNN depends on the ratio between HFS and LFS in the training set. Theoretical studies on the multilevel Monte Carlo method can be used to guide the selection of an optimal mix of low- and high-fidelity data.

4. The CNN trained on multifidelity data is largely insensitive to the discretization error of LFS. CNN-derived estimates of the PDF and CDF of the QoI are close to those of the converged high-fidelity Monte Carlo method, but the former are three orders of magnitude faster to obtain than the latter.

The computational efficiency and accuracy of CNN training on multifidelity data depend crucially on the HFS/LFS ratio. We relied on the theoretical results for MLMC as an empirical guide for the selection of this ratio; a detailed theoretical and/or experimental investigation of an optimal HFS/LFS ratio is left for the future. Another direction for subsequent studies is the use of multiple physical models of different complexity, rather than a single model solved on different numerical grids, to generate multifidelity data.

ACKNOWLEDGMENTS

This research was supported in part by Air Force Office of Scientific Research under Award No. FA9550-21-1-0381; by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award No. DE-AR0001202; and by a gift from Total.

REFERENCES

Appleyard, J.R., Cheshire, I.M., and Pollard, R.K., Special Techniques for Fully Implicit Simulators, *Proc. of the European Symposium on Enhanced Oil Recovery*, Bournemouth, UK, pp. 395–408, 1981.

Aziz, K., *Petroleum Reservoir Simulation*, New York: Applied Science Publishers, 1979.

Boso, F. and Tartakovsky, D.M., Information-Theoretic Approach to Bidirectional Scaling, *Water Resour. Res.*, vol. **54**, no. 7, pp. 4916–4928, 2018.

Breiman, L., Random Forests, *Mach. Learn.*, vol. **45**, no. 1, pp. 5–32, 2001.

Corey, A.T., The Interrelation between Gas and Oil Relative Permeabilities, *Producers Month.*, vol. **19**, no. 1, pp. 38–41, 1954.

Couckuyt, I., Dhaene, T., and Demeester, P., SooDACE Toolbox: A Flexible Object-Oriented Kriging Implementation, *J. Mach. Learn. Res.*, vol. **15**, pp. 3183–3186, 2014.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T., Decaf: A Deep Convolutional Activation Feature for Generic Visual Recognition, *Proc., 31st Int. Conf. on Machine Learning*, pp. 647–655, 2014.

Durlofsky, L.J., Upscaling and Gridding of Fine Scale Geological Models for Flow Simulation, *8th Int. Forum on Reservoir Simulation*, vol. 2024, Iles Borromees, Stresa, Italy, pp. 1–59, 2005.

Friedman, J., Hastie, T., and Tibshirani, R., *The Elements of Statistical Learning*, 1st ed., New York: Springer, 2001.

Fuks, O. and Tchelepi, H., Limitations of Physics Informed Machine Learning for Nonlinear Two-Phase Transport in Porous Media, *J. Mach. Learn. Model. Comput.*, vol. **1**, no. 1, pp. 19–37, 2020.

Geneva, N. and Zabaras, N., Multifidelity Generative Deep Learning Turbulent Flows, *Found. Data Sci.*, vol. **2**, no. 4, pp. 391–428, 2020.

Giles, M.B., Multilevel Monte Carlo Path Simulation, *Oper. Res.*, vol. **56**, no. 3, pp. 607–617, 2008.

Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R., A Physics-Informed Deep Learning Framework for Inversion and Surrogate Modeling in Solid Mechanics, *Comput. Meth. Appl. Mech. Eng.*, vol. **379**, p. 113741, 2021.

Heinrich, S., Monte Carlo Complexity of Global Solution of Integral Equations, *J. Complexity*, vol. **14**, no. 2, pp. 151–175, 1998.

Heinrich, S., Multilevel Monte Carlo Methods, *Int. Conf. on Large-Scale Scientific Computing*, Sozopol, Bulgaria, pp. 58–67, 2001.

Hwang, J.T. and Martins, J.R.R.A., A Fast-Prediction Surrogate Model for Large Datasets, *Aerospace Sci. Tech.*, vol. **75**, pp. 74–87, 2018.

Jiang, H. and Learned-Miller, E., Face Detection with the Faster R-CNN, *2017 12th IEEE Int. Conf. on Automatic Face & Gesture Recognition (FG 2017)*, Washington, DC, pp. 650–657, 2017.

Karpathy, A. and Fei-Fei, L., Deep Visual-Semantic Alignments for Generating Image Descriptions, *Proc., IEEE Conf. on Computer Vision and Pattern Recognition*, Boston, pp. 3128–3137, 2015.

Kingma, D.P. and Ba, J., Adam: A Method for Stochastic Optimization, 2014. arXiv: 1412.6980

Lagaris, I.E., Likas, A., and Fotiadis, D.I., Artificial Neural Networks for Solving Ordinary and Partial Differential Equations, *IEEE Trans. Neural Networks*, vol. **9**, no. 5, pp. 987–1000, 1998.

Lee, H. and Kang, I.S., Neural Algorithm for Solving Differential Equations, *J. Comput. Phys.*, vol. **91**, no. 1, pp. 110–131, 1990.

Loshchilov, I. and Hutter, F., Decoupled Weight Decay Regularization, 2017. arXiv: 1711.05101

Meng, X. and Karniadakis, G.E., A Composite Neural Network That Learns from Multifidelity Data: Application to Function Approximation and Inverse PDE Problems, *J. Comput. Phys.*, vol. **401**, p. 109020, 2020.

Mo, S., Zabaras, N., Shi, X., and Wu, J., Deep Autoregressive Neural Networks for High-Dimensional Inverse Problems in Groundwater Contaminant Source Identification, *Water Resour. Res.*, vol. **55**, no. 5, pp. 3856–3881, 2019a.

Mo, S., Zhu, Y., Zabaras, N., Shi, X., and Wu, J., Deep Convolutional Encoder-Decoder Networks for Uncertainty Quantification of Dynamic Multiphase Flow in Heterogeneous Media, *Water Resour. Res.*, vol. **55**, no. 1, pp. 703–728, 2019b.

Montgomery, D.C. and Evans, D.M., Second-Order Response Surface Designs in Computer Simulation, *Aerospace Sci. Tech.*, vol. **75**, pp. 74–87, 2018.

Müller, F., Jenny, P., and Meyer, D.W., Multilevel Monte Carlo for Two Phase Flow and Buckley-Leverett Transport in Random Heterogeneous Porous Media, *J. Comput. Phys.*, vol. **250**, pp. 685–702, 2013.

Paleologos, E.K., Neuman, S., and Tartakovsky, D.M., Effective Hydraulic Conductivity of Bounded, Strongly Heterogeneous Porous Media, *Water Resour. Res.*, vol. **32**, no. 5, pp. 1333–1341, 1996.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., Pytorch: An Imperative Style, High-Performance Deep Learning Library, *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Red Hook, NY: Curran Associates, Inc., pp. 8024–8035, 2019.

Peherstorfer, B., Multifidelity Monte Carlo Estimation with Adaptive Low-Fidelity Models, *SIAM/ASA J. Uncert. Quant.*, vol. **7**, no. 2, pp. 579–603, 2019.

Raissi, M., Perdikaris, P., and Karniadakis, G.E., Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations, *J. Comput. Phys.*, vol. **378**, pp. 686–707, 2019.

Ronneberger, O., Fischer, P., and Brox, T., U-Net: Convolutional Networks for Biomedical Image Segmentation, *Int. Conf. on Medical Image Computing and Computer-Assisted Intervention*, Munich, Germany, pp. 234–241, 2015.

Sinsbeck, M. and Tartakovsky, D.M., Impact of Data Assimilation on Cost-Accuracy Tradeoff in Multifidelity Models, *SIAM/ASA J. Uncert. Quant.*, vol. **3**, no. 1, pp. 954–968, 2015.

Tang, M., Liu, Y., and Durlofsky, L.J., A Deep-Learning-Based Surrogate Model for Data Assimilation in Dynamic Subsurface Flow Problems, *J. Comput. Phys.*, p. 109456, 2020.

Tartakovsky, D.M. and Neuman, S.P., Transient Effective Hydraulic Conductivities under Slowly and Rapidly Varying Mean Gradients in Bounded Three-Dimensional Random Media, *Water Resour. Res.*, vol. **34**, no. 1, pp. 21–32, 1998.

Taverniers, S., Bosma, S.B., and Tartakovsky, D.M., Accelerated Multilevel Monte Carlo with Kernel-Based Smoothing and Latinized Stratification, *Water Resour. Res.*, vol. **56**, no. 9, p. e2019WR026984, 2020.

Tripathy, R.K. and Bilionis, I., Deep UQ: Learning Deep Neural Network Surrogate Models for High Dimensional Uncertainty Quantification, *J. Comput. Phys.*, vol. **375**, pp. 565–588, 2018.

Xiu, D., *Numerical Methods for Stochastic Computations: A Spectral Method Approach*, Princeton, NJ: Princeton University Press, 2010.

Yang, L., Wang, P., and Tartakovsky, D.M., Resource-Constrained Model Selection for Uncertainty Propagation and Data Assimilation, *SIAM/ASA J. Uncert. Quant.*, vol. **8**, no. 3, pp. 1118–1138, 2020.

Ye, M., Neuman, S.P., Guadagnini, A., and Tartakovsky, D.M., Nonlocal and Localized Analyses of Conditional Mean Transient Flow in Bounded, Randomly Heterogeneous Porous Media, *Water Resour. Res.*, vol. **40**, no. 5, p. W05104, 2004.

Zhou, Z. and Tartakovsky, D.M., Markov Chain Monte Carlo with Neural Network Surrogates: Application to Contaminant Source Identification, *Stoch. Environ. Res. Risk Assess.*, vol. **35**, no. 3, pp. 639–651, 2021.

Zhu, Y., Zabaras, N., Koutsourelakis, P.S., and Perdikaris, P., Physics-Constrained Deep Learning for High-Dimensional Surrogate Modeling and Uncertainty Quantification without Labeled Data, *J. Comput. Phys.*, vol. **394**, pp. 56–81, 2019.

APPENDIX A. PSEUDOCODE FOR TRAINING SURROGATE MODEL WITH MULTIPLE LEVELS OF DATA

Algorithm 1: CNN training for given data

Input : Starting model (M_{in}); training data ($\text{data}_{\text{train}}$); test data ($\text{data}_{\text{test}}$); Number of phases ($N_p$); Epochs ($\text{eps}$)
Output : Best output model (M_{out})
Procedure:
for $i = 1, \ldots, N_p$ **do**
  for $j = 1, \ldots, \text{eps}$ **do**
    Train M_{in} using $\text{data}_{\text{train}}$;
    Compute $\text{RMSE}_{\text{test}}$ using $\text{data}_{\text{test}}$;
  end
  Set $\text{RMSE}_{\text{check}}$ as mean of last 10 $\text{RMSE}_{\text{test}}$;
  if $\text{RMSE}_{\text{check}} < \text{RMSE}_{\text{best}}$ **then**
    Set M_{out} as M_{in};
    Set $\text{RMSE}_{\text{best}}$ as $\text{RMSE}_{\text{check}}$;
  end
end

Algorithm 2: Phase 1: Training using LFS

Input : Original model (M_{orig}); Convolution Transpose 2 layer from M_{orig} ($L_{\text{conv,transpose2}}$); Temporary convolution layer in order to match LFS output dimensions (L_{temp}); LFS training data ($\text{data}_{\text{train,LFS}}$); LFS test data ($\text{data}_{\text{test,LFS}}$); Number of Phase 1 iterations (N_{phase1}); Epoch in Phase 1 ($\text{eps}_{\text{phase1}}$)
Output : CNN trained on LFS ($M_{1,\text{best}}$)
Procedure:
Set M_{mod1} by removing $L_{\text{conv,transpose2}}$ from M_{orig};
Set M_1 by attaching L_{temp} to the end of M_{mod1};
Train $M_{1,\text{best}}$ using Algorithm 1 (inputs: $M_{\text{in}} = M_1$, $N_{\text{phase}} = N_{\text{phase1}}$, $\text{eps} = \text{eps}_{\text{phase1}}$, $\text{data}_{\text{test}} = \text{data}_{\text{test,LFS}}$, $\text{data}_{\text{train}} = \text{data}_{\text{train,LFS}}$);

Algorithm 3: Phase 2: Initial training using HFS

Input : Model from Phase 1 ($M_{1,\text{best}}$); Convolution Transpose 2 layer from M_{orig} ($L_{\text{conv,transpose2}}$); Temporary convolution layer in order to match LFS output dimensions (L_{temp}); HFS training data ($\text{data}_{\text{train,HFS}}$); HFS test data ($\text{data}_{\text{test,HFS}}$); Number of Phase 2 iterations (N_{phase2}); Epoch in Phase 2 ($\text{eps}_{\text{phase2}}$)
Output : CNN trained on LFS and HFS ($M_{2,\text{best}}$)
Procedure:
Set M_{mod2} by removing L_{temp} from $M_{1,\text{best}}$, and lock all weights;
Set M_2 by attaching $L_{\text{conv,transpose2}}$ to the end of M_{mod2} (the weights of $L_{\text{conv,transpose2}}$ remain unlocked);
Train $M_{2,\text{best}}$ using Algorithm 1 (inputs: $M_{\text{start}} = M_2$, $N_{\text{phase}} = N_{\text{phase2}}$, $\text{eps} = \text{eps}_{\text{phase2}}$, $\text{data}_{\text{test}} = \text{data}_{\text{test,HFS}}$, $\text{data}_{\text{train}} = \text{data}_{\text{train,HFS}}$);

Algorithm 4: Phase 3: Final training using HFS

Input : Model from Phase 2 ($M_{2,\text{best}}$); HFS training data ($\text{data}_{\text{train,HFS}}$); HFS test data ($\text{data}_{\text{test,HFS}}$); Number of Phase 3 iterations (N_{phase3}); Epoch in Phase 3 ($\text{eps}_{\text{phase3}}$)

Output : Fine tuned CNN trained on LFS and HFS ($M_{3,\text{best}}$)

Procedure:

Set M_3 by unlocking all weights in $M_{2,\text{best}}$;

Train $M_{3,\text{best}}$ using Algorithm 1 (inputs: $M_{\text{start}} = M_3$, $N_{\text{phase}} = N_{\text{phase3}}$, $\text{eps} = \text{eps}_{\text{phase3}}$, $\text{data}_{\text{test}} = \text{data}_{\text{test,HFS}}$, $\text{data}_{\text{train}} = \text{data}_{\text{train,HFS}}$);

Algorithm 5: Training surrogate model on multiple scales of data

Input : HFS training data ($\text{data}_{\text{train,HFS}}$); HFS test data ($\text{data}_{\text{test,HFS}}$); LFS training data ($\text{data}_{\text{train,LFS}}$); LFS test data ($\text{data}_{\text{test,LFS}}$); Number of Phase 1 iterations (N_{phase1}); Number of Phase 2 iterations (N_{phase2}); Number of Phase 3 iterations (N_{phase3}); Epoch in Phase 1 ($\text{eps}_{\text{phase1}}$); Epoch in Phase 2 ($\text{eps}_{\text{phase2}}$); Epoch in Phase 3 ($\text{eps}_{\text{phase3}}$)

Output : Surrogate model on the high-fidelity scale (M_3)

Procedure:

Set and initialize M_{orig} as original model as described by Table 1;

Set $L_{\text{conv,transpose2}}$ as "Convolution Transpose 2" layer from M_{orig};

Set L_{temp} as temporary convolution layer in order to match LFS output dimensions ($16 \times 64 \times 64$);

Train M_1 using LFS data via Algorithm 2;

Train M_2 using HFS data via Algorithm 3;

Train M_3 using HFS data via Algorithm 4.