Journal of Petroleum Science and Engineering 192 (2020) 107273

Contents lists available at ScienceDirect

Journal of Petroleum Science and Engineering

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

Deep-learning-based surrogate model for reservoir simulation with time-varying well controls

Zhaoyang Larry Jin *, Yimin Liu, Louis J. Durlofsky

Department of Energy Resources Engineering, Stanford University, Stanford, CA, 94305, United States of America

ARTICLE INFO

Keywords:
Reservoir simulation
Reduced-order model
Deep learning
Physics-informed neural network
Auto-encoder
Embed-to-control
E2C

ABSTRACT

A new deep-learning-based reduced-order modeling (ROM) framework is proposed for application in subsurface flow simulation. The reduced-order model is based on an existing embed-to-control (E2C) framework and includes an auto-encoder, which projects the system to a low-dimensional subspace, and a linear transition model, which approximates the evolution of the system states in low dimension. In addition to the loss function for data mismatch considered in the original E2C framework, we introduce a physics-based loss function that penalizes predictions that are inconsistent with the governing flow equations. The loss function is also modified to emphasize accuracy in key well quantities of interest (e.g., fluid production rates). The E2C ROM is shown to be analogous to an existing ROM, POD-TPWL, which has been extensively developed for subsurface flow simulation. The new ROM is applied to oil-water flow in a heterogeneous reservoir, with flow driven by nine wells operating under time-varying control specifications. A total of 300 high-fidelity training simulations are performed in the offline stage, and the network training requires 10-12 minutes on a Tesla V100 GPU node. Online (runtime) computations achieve speedups of over a factor of 1000 relative to full-order simulations. Extensive test case results, with well controls varied over large ranges, are presented. Accurate ROM predictions are achieved for global saturation and pressure fields at particular times, and for injection and production well responses as a function of time. Error is shown to increase when 100 or 200 (rather than 300) training runs are used to construct the E2C ROM. Overall the E2C ROM is shown to provide reliable predictions with levels of perturbations in the well controls that are much larger than those used with existing POD-TPWL treatments. The current model is however limited to 2D systems, and the required number of training simulations is much larger than that for POD-based ROMs.

1. Introduction

Reservoir simulation is widely applied to model and manage subsurface flow operations. However, due to the nonlinear nature of the governing equations and the multiscale character of the geological description, computational costs can be high, especially when highly resolved models are used. Computational demands can become prohibitive when simulation tools are applied for optimization, uncertainty quantification, and data assimilation, in which case thousands of simulation runs may be required.

Reduced-order models (ROMs) have been developed and applied to accelerate flow predictions in a variety of settings. Our goal in this work is to develop a new deep-learning-based reduced-order modeling procedure. Following the embed-to-control framework, the approach introduced here is comprised of a linear transition model and an auto-encoder (AE, also referred to as encoder–decoder). An encoder–decoder architecture is used to achieve dimension reduction by constructing the mapping to and from the low-dimensional representation. The AE component is a stack of multiple convolutional neural network (CNN) layers and dense feed-forward layers. The linear transition model represents the step-wise evolution of the system states with multiple linear feed-forward layers. The E2C procedure is constructed to predict key well quantities, such as time-varying production and injection rates and/or bottom-hole pressure (BHPs), as well as global pressure and saturation fields, in oil-water reservoir simulation problems.

ROM methodologies have received a large amount of attention in recent years. These procedures typically involve an offline (train-time) component, where training runs are performed and relevant solution information is processed and saved, and an online (test-time) component, where new (test) runs are performed. A popular category of methods is proper-orthogonal-decomposition-based (POD-based) ROMs, in which POD is applied to enable the low-dimensional representation of solution unknowns in the online computations. These approaches also require

* Corresponding author.
E-mail addresses: zjin@stanford.edu (Z.L. Jin), yiminliu@stanford.edu (Y. Liu), lou@stanford.edu (L.J. Durlofsky).

https://doi.org/10.1016/j.petrol.2020.107273
Received 11 November 2019; Received in revised form 15 March 2020; Accepted 5 April 2020
Available online 10 April 2020

the projection of the system of equations to low dimension (this projection is also referred to as constraint reduction). Galerkin projection and least-squares Petrov–Galerkin projection are the two approaches typically used for this step.

A treatment of solution nonlinearity is also required, and there have been a number of treatments for this within the context of POD-based ROMs. One effective approach is Gauss–Newton with approximated tensors or GNAT, which also uses POD for state reduction and least-squares Petrov–Galerkin projection. GNAT was developed by Carlberg et al. (2011), and has since been used for structural and solid mechanics (Zahr et al., 2017), electromechanics (Amsallem et al., 2012), and computational fluid dynamics (Carlberg et al., 2013). GNAT represents a generalization of the discrete empirical interpolation method (DEIM) (Chaturantabut and Sorensen, 2010), and the two methods (GNAT and POD-DEIM) have been applied in a number of studies involving subsurface flow simulation (Yoon et al., 2016; Yang et al., 2016; Efendiev et al., 2016; Tan et al., 2019; Jiang and Durlofsky, 2019; Florez and Gildin, 2019). A radial basis function (RBF) multidimensional interpolation method has also been used to treat nonlinearity in the low-dimensional space represented by POD, and the resulting procedure is referred to as the POD-RBF method (Xiao et al., 2015; Kostorz et al., 2019). Trajectory piecewise linearization, originally introduced by Rewienski and White (2003), entails linearization around 'nearby' training solutions. POD-TPWL has been widely applied for subsurface flow simulations involving oil-water, oil-gas compositional, CO_2 storage, and coupled flow-geomechanics systems (Cardoso and Durlofsky, 2010; He et al., 2011; He and Durlofsky, 2014, 2015; Jin and Durlofsky, 2018; Jin et al., 2020). Trehan and Durlofsky (2016) extended POD-TPWL to include a quadratic term, which gives a trajectory piecewise quadratic (POD-TPWQ) procedure.

The recent success of deep learning in image processing has inspired the rapid development of algorithms for subsurface modeling that make use of deep neural networks. These methods have been applied for geological parameterization, uncertainty quantification, and surrogate/reduced-order modeling. For geological parameterization and uncertainty quantification, Lee et al. (2018) applied a distance-based clustering framework, in which models that are close in terms of distance are grouped. Distance is determined based on a parameterization of the reservoir models using a stacked AE. Efficient uncertainty quantification was then achieved by simulating only one model in each group. The results from all groups were taken to represent the uncertainty range of the entire ensemble.

Canchumuni et al. (2019) generated new geological realizations from randomized low-dimensional latent variables using a variational auto-encoder (VAE). A VAE (Kingma and Welling, 2013) entails a convolutional encoder–decoder neural network architecture similar to the AE, where the encoder component projects a high-dimensional distribution into a low-dimensional random vector, with each element following an independent Gaussian distribution. The decoder acts as the inverse of the encoder and projects the sampled Gaussian-distributed random variables back to the high dimension. Laloy et al. (2018) achieved a similar goal using a generative adversarial network (GAN), where the projection to high dimension is determined by training two adversarial neural networks (known as the generator and the discriminator). Liu et al. (2019) and Liu and Durlofsky (2020) extended principal component analysis (PCA) based representations to a CNN-PCA procedure. This approach applied the 'fast neural style transfer' algorithm (Johnson et al., 2016) to represent complex geological models characterized by multipoint spatial statistics, and was shown to enable more efficient data assimilation. Zhu and Zabaras (2018) formulated surrogate modeling as an image-to-image regression, and constructed a Bayesian deep convolutional neural network for geological uncertainty quantification. Subsequently, Mo et al. (2019) extended this model to handle multiphase flow problems, and further improved performance by introducing additional physical constraints.

Recent developments involving the use of deep-learning techniques in ROMs indicate great potential for such approaches. Lee and Carlberg (2020) introduced an improved GNAT procedure by replacing POD with AE. The resulting method was applied to a one-dimensional dynamic Burgers' equation and a two-dimensional quasi-static chemically reacting flow problem, with the boundary conditions in the test runs different from those in the training runs. Kani and Elsheikh (2019) developed a deep residual recurrent neural network (DR-RNN) procedure, which employed RNN to approximate the low-dimensional residual functions for the governing equations in a POD-DEIM procedure. The resulting ROM was then applied for uncertainty quantification in a two-dimensional small-scale oil-water system with the distribution of porosity in the test runs perturbed from that of the training runs. Zhang et al. (2019) used a fully-connected network to replace the Newton iterations in a POD-DEIM procedure. The method was used to predict well responses in a two-dimensional oil-water problem, in which combinations of well controls and permeability fields for test runs were different from those of the training simulations. Though improvements in accuracy were achieved by all of the above approaches relative to the 'standard' implementations, all of these developments were within existing ROM settings; i.e., none adopted an end-to-end deep-learning framework.

Other researchers have developed ROM methodologies that represent more of a departure from existing approaches. Wang et al. (2018), for example, used the long-short-term-memory (LSTM) RNN (Gers et al., 1999) to approximate flow dynamics in a low-dimensional subspace constructed by POD. Subsequently, Gonzalez and Balajewicz (2018) replaced the POD step with AE for the low-dimensional representation. Both of these approaches, however, were applied on relatively simple problems, where the only differences between online and offline simulation runs were the initial conditions of the systems (boundary conditions were identical). In the subsurface flow equations, wells appear as localized source/sink terms, which essentially act as 'internal' boundary conditions. The ability to vary well settings (by well settings here we mean time-varying rates or BHPs for each well in the model) between offline and online computations is an essential feature for ROMs used in oil production optimization and related areas. Thus the above implementations may not be directly applicable for these problems.

Temirchev et al. (2020) constructed a similar ROM, representing the reservoir states in low dimension with VAE. They tested this in combination with either linear regression, LSTM, or gated recurrent units (GRU) for dynamic simulation, with the best results achieved with GRU. A GRU (Chung et al., 2014) is an RNN that is similar to an LSTM RNN, but GRUs have simpler structures. The relative error with GRU was, however, reported to be relatively large in some validation scenarios, which might pose problems for applications such as well control optimization. This study nonetheless provides a useful assessment of several potential approaches within a VAE setting. Temirchev et al. (2019) also devised a 'neural-differential-equation-based' ROM, and applied it for a 3D synthetic benchmark test model. Tang et al. (2019) introduced a deep-learning-based surrogate model with convolutional and recurrent neural networks to predict flow responses for new geomodels. Well controls were not changed between training and testing runs. This model, referred to as 'recurrent R-U-Net,' was applied successfully within a history matching workflow.

Many of the existing approaches are purely data driven and do not take the underlying governing equations into (direct) consideration. A number of methods have, however, been applied to incorporate physical constraints into deep neural networks. Raissi et al. (2019) introduced a physics-informed deep learning framework (later referred to as physics-informed neural network or PINN) that used densely connected feed-forward neural networks. In PINN, the residual functions associated with the governing partial differential equations (PDEs) are introduced into the loss function of the neural network. Zhu et al.

(2019) extended this PDE-constraint concept to a deep flow-based generative model (GLOW Kingma and Dhariwal, 2018), and constructed a surrogate model for uncertainty quantification using residuals of the governing equations rather than simulation outputs. Watter et al. (2015) proposed an embed-to-control (E2C) framework, in the context of robotic planning systems, to predict the evolution of system states using direct sensory data (images) and time-varying controls as inputs. The E2C framework combines a VAE, which is used as both an inference model to project the system states to a low-dimensional subspace, and a generative model to reconstruct the prediction results at full order, with a linear transition model. The latter approximates the evolution of low-dimensional states based on the time-varying control inputs.

In this paper, we develop a deep-learning framework for reduced-order modeling of subsurface flow systems based on the E2C model (Watter et al., 2015) and the aforementioned physics-informed treatments (Raissi et al., 2019; Zhu et al., 2019). Two key modifications of the existing E2C model are introduced. Specifically, we simplify the VAE to an AE to achieve better accuracy for deterministic test cases, and we incorporate a comprehensive loss function that introduces both PDE-based physical constraints and improves accuracy for well production and injection quantities. The latter treatment is important for improving the accuracy of well rates, which are essential in oil production optimization procedures. Because we are considering a supervised learning problem with labeled data (input and output pairs), the way we introduce the physical constraints is different from the approaches of Raissi et al. (2019) and Zhu et al. (2019), where the PDE residuals were used in the loss function during the training process. Interestingly, our E2C procedure is analogous to existing POD-TPWL methodologies, and we discuss the relationships between the two approaches in some detail.

This paper proceeds as follows. In Section 2, we present the governing equations for subsurface oil-water flow and then briefly describe the POD-TPWL ROM. In Section 3, the E2C formulation is presented, and the correspondences between E2C and POD-TPWL are highlighted. We present results for a two-dimensional oil-water problem in Section 4. Test cases involve the specification of different time-varying well settings, as would be encountered in an optimization problem. We also present a detailed error assessment for several key quantities. We conclude with a summary and suggestions for future work in Section 5.

Supplementary Material for this paper, available online, includes the detailed architectures for the encoder and decoder used in the E2C model, performance comparisons between an auto-encoder, variational auto-encoder and uncertainty auto-encoder (UAE) (Grover and Ermon, 2018), E2C ROM results for two additional test cases, and a Nomenclature defining the main variables used in this work.

2. Governing equations and POD-TPWL ROM

In this section, we present the equations for oil-water flow. We then provide an overview of the POD-TPWL ROM for this problem, which will allow us to draw analogies with the E2C ROM.

2.1. Governing equations

The governing equations for immiscible oil-water flow derive from mass conservation for each component combined with Darcy's law for each phase. The resulting equations, with capillary pressure effects neglected, are

$$\frac{\partial}{\partial t}(\phi S_j \rho_j) - \nabla \cdot (\lambda_j \rho_j \mathbf{k} \nabla p) + \sum_w \rho_j q_j^w = 0, \tag{1}$$

where subscript j ($j = o, w$ for oil and water) denotes fluid phase. The geological characterization is represented in Eq. (1) through porosity ϕ and the permeability tensor $\mathbf{k}$, while the interactions between rock and fluids are specified by the phase mobilities λ_j, where $\lambda_j = k_{rj}/\mu_j$, with k_{rj} the relative permeability of phase j and μ_j the viscosity of phase j. Other variables are pressure p and phase saturation S_j (these are the primary solution variables), time t, and phase density ρ_j. The q_j^w term denotes the phase source/sink term for well w. This oil-water model is completed by enforcing the saturation constraint $S_o + S_w = 1$. Because the system considered in this work is horizontal (in the x–y plane), gravity effects are neglected.

The oil and water flow equations are discretized using a standard finite-volume formulation, and their solutions are computed for each grid block. In this work, we use Stanford's Automatic Differentiation-based General Purpose Research Simulator, AD-GPRS (Zhou, 2012), for all flow simulations. Let n_b denote the number of grid blocks in the model. The flow system is fully defined through the use of two primary variables, p and S_w, in each grid block, so the total number of variables in the system is $2n_b$. We define $\mathbf{x}_t = [\mathbf{p}_t^T, \mathbf{S}_t^T]^T \in \mathbb{R}^{2n_b}$ to be the state vector for the flow variables at a specific time step t, where $\mathbf{p}_t \in \mathbb{R}^{n_b}$ and $\mathbf{S}_t \in \mathbb{R}^{n_b}$ denote the pressure and saturation in every grid block at time step t.

The set of nonlinear algebraic equations representing the discretized fully implicit system can be expressed as:

$$\mathbf{g}(\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{u}_{t+1}) = \mathbf{0}, \tag{2}$$

where $\mathbf{g} \in \mathbb{R}^{2n_b}$ is the residual vector (set of nonlinear algebraic equations) we seek to drive to zero, the subscript t indicates the current time level and $t + 1$ the next time level, and $\mathbf{u}_{t+1} \in \mathbb{R}^{n_w}$ designates the well control variables, which can be any combination of time-varying bottom-hole pressures (BHPs) or well rates. Here n_w denotes the number of wells in the system. In this work we operate production wells under BHP specifications and injection wells under rate specifications. Our treatments are general in this regard, and other control settings could also be applied. Although we apply AD-GPRS for the reference flow simulations, we use a standalone Peaceman well model (Peaceman, 1978) in conjunction with the AD-GPRS pressure and saturation results to compute well rates or BHPs from the well-block states. We proceed in this manner to ensure full consistency in the well treatment between the full-order simulator and the E2C ROM.

Newton's method is typically used to solve the full-order discretized nonlinear system defined by Eq. (2). This requires constructing the sparse Jacobian matrix of dimension $2n_b \times 2n_b$, and then solving a linear system of dimension $2n_b$, at each iteration for every time step. Solution of the linear system is often the most time-consuming part of the simulation. As will be explained later, both POD-TPWL and the deep-learning-based E2C ROM avoid the test-time construction and solution of this high-dimensional system.

2.2. POD-TPWL formulation

Many deep-learning-based models involve treatments that are not directly analogous to those used in existing ROMs, which were developed based on the underlying PDEs and numerical discretizations. Rather, these new approaches often involve machine-learning methods that derive from image classification, language recognition, or other non-PDE-based applications. Our E2C ROM is somewhat different in this regard, because its three main components are analogous to treatments used in an existing ROM (POD-TPWL) that has been extensively applied for subsurface flow. We believe it is worthwhile to discuss the correspondences between the POD-TPWL and E2C ROMs, since the analogies between the two approaches may enable insight or suggest approaches for some of the detailed treatments.

To enable this discussion, we first provide a high-level overview of POD-TPWL for reservoir simulation. For full details on recent POD-TPWL implementations, please see He and Durlofsky (2014, 2015), Jin and Durlofsky (2018) and Jin et al. (2020). Note that, although we discuss the conceptual similarities between these approaches, we will not present any POD-TPWL results in this paper. This is because the number of training runs we use for E2C (100–300) is much more than is compatible with existing POD-TPWL frameworks (which use,

e.g., 3–5 training runs). More specifically, the so-called point-selection strategies used in the POD-TPWL linearization step would have to be reformulated in order to accommodate 100 or more training runs. This would entail the development of new treatments along with extensive testing, both of which are beyond the scope of this paper.

During the offline (pre-processing) POD-TPWL stage, the training simulation runs are performed using a full-order simulator (AD-GPRS in this work). The goal here is to predict test-time results with varying well control sequences. Therefore, during training runs, we apply different well control sequences $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{N_{\text{ctrl}}}] \in \mathbb{R}^{n_w \times N_{\text{ctrl}}}$, where $\mathbf{u}_k \in \mathbb{R}^{n_w}$, $k = 1, \ldots, N_{\text{ctrl}}$, contains the settings (rates or BHPs) for all wells at control step k, and N_{ctrl} denotes the total number of control steps in a training run. There are many fewer control steps than time steps in a typical simulation (in our examples we have 20 control steps and around 100 time steps). State variables in all grid blocks (referred to as snapshots) and derivative matrices are saved at each time step in the training runs. At test-time, simulations with control sequences that are different from those of the training runs are performed. Information saved from the training runs is used to (very efficiently) approximate test solutions.

POD-TPWL entails (1) projection from a high-dimensional space to a low-dimensional subspace, (2) linear approximation of the dynamics in the low-dimensional subspace, and (3) projection back to the high-dimensional space. A projection matrix $\mathbf{\Phi} \in \mathbb{R}^{2n_b \times l_\xi}$ is constructed based on the singular value decomposition (SVD) of the solution snapshot matrices (these snapshot matrices contain full-order solutions at all time steps in all training runs). Given $\mathbf{\Phi}$, the high-dimensional states $\mathbf{x} \in \mathbb{R}^{2n_b}$ can be represented in terms of the low-dimensional variable $\boldsymbol{\xi} \in \mathbb{R}^{l_\xi}$ using

$$\mathbf{x} \approx \mathbf{\Phi}\boldsymbol{\xi}, \tag{3}$$

where l_ξ is the dimension of the reduced space, with $l_\xi \ll n_b$. Note that in practice, the SVD and subsequent projections are performed separately for the pressure and saturation variables. Because $\mathbf{\Phi}$ is orthonormal, we also have $\boldsymbol{\xi} = \mathbf{\Phi}^T\mathbf{x}$.

Before discussing the POD-TPWL approximation in low-dimensional space, we first show the linearization in high dimension. Following He and Durlofsky (2015), the TPWL formulation (with the POD representation for states, $\mathbf{x} = \mathbf{\Phi}\boldsymbol{\xi}$, applied to the right-hand side) can be expressed as

$$\mathbf{J}_{i+1}\hat{\mathbf{x}}_{t+1} = \mathbf{J}_{i+1}\mathbf{\Phi}\boldsymbol{\xi}_{i+1} - [\mathbf{A}_{i+1}\mathbf{\Phi}(\boldsymbol{\xi}_t - \boldsymbol{\xi}_i) + \mathbf{B}_{i+1}(\mathbf{u}_{t+1} - \mathbf{u}_{i+1})], \tag{4}$$

where

$$\mathbf{J}_{i+1} = \frac{\partial \mathbf{g}^{i+1}}{\partial \mathbf{x}^{i+1}} \in \mathbb{R}^{2n_b \times 2n_b}, \quad \mathbf{A}_{i+1} = \frac{\partial \mathbf{g}^{i+1}}{\partial \mathbf{x}^{i}} \in \mathbb{R}^{2n_b \times 2n_b}, \quad \mathbf{B}_{i+1} = \frac{\partial \mathbf{g}^{i+1}}{\partial \mathbf{u}^{i+1}} \in \mathbb{R}^{2n_b \times n_w}. \tag{5}$$

Here the subscripts t and $t+1$ denote time steps in the test run, while the subscripts i and $i+1$ designate time steps in the training simulations. Note that Eq. (4) differs slightly from the expressions in He and Durlofsky (2015) since the time step designations are now subscripted, for consistency with the embed-to-control equations shown later. The variable $\boldsymbol{\xi}_t$ is the projection of the true (high-order) solution of Eq. (2) at time step t. The variable $\hat{\mathbf{x}}_{t+1} \in \mathbb{R}^{2n_b}$ is distinct from $\mathbf{x}_{t+1}$, in that it represents the full-order variable at time step $t+1$ approximated through linearization instead of via solution of the full-order system (Eq. (2)). From here on, we will use variables without 'hats' to denote the true high-order solution (e.g., $\mathbf{x}$) or the true solution projected with matrix $\mathbf{\Phi}$ (e.g., $\boldsymbol{\xi} = \mathbf{\Phi}^T\mathbf{x}$). And, we will use variables with 'hats' ($\hat{\mathbf{x}}$ and $\hat{\boldsymbol{\xi}}$) to designate solutions approximated (either reconstructed or predicted, as will be explained in detail later) by the ROM. The variables $\mathbf{u}_{t+1}, \mathbf{u}_{i+1} \in \mathbb{R}^{n_w}$ are the well settings at time step $t+1$ and $i+1$ — these are prescribed by the user or specified by an optimization algorithm.

Applying the POD representation on the left-hand side and constraint reduction (projection) on both sides of Eq. (4), the solution approximation in low-dimensional space, after some rearrangement, is given by

$$\hat{\boldsymbol{\xi}}_{t+1} = \boldsymbol{\xi}_{i+1} - (\mathbf{J}^r_{i+1})^{-1}[\mathbf{A}^r_{i+1}(\boldsymbol{\xi}_t - \boldsymbol{\xi}_i) + \mathbf{B}^r_{i+1}(\mathbf{u}_{t+1} - \mathbf{u}_{i+1})], \tag{6}$$

with the reduced derivative matrices defined as

$$\mathbf{J}^r_{i+1} = (\mathbf{\Psi}_{i+1})^T\mathbf{J}_{i+1}\mathbf{\Phi}, \quad \mathbf{A}^r_{i+1} = (\mathbf{\Psi}_{i+1})^T\mathbf{A}_{i+1}\mathbf{\Phi}, \quad \mathbf{B}^r_{i+1} = (\mathbf{\Psi}_{i+1})^T\mathbf{B}_{i+1}. \tag{7}$$

Here $\mathbf{J}^r_{i+1} \in \mathbb{R}^{l_\xi \times l_\xi}$, $\mathbf{A}^r_{i+1} \in \mathbb{R}^{l_\xi \times l_\xi}$ and $\mathbf{B}^r_{i+1} \in \mathbb{R}^{l_\xi \times n_w}$. The matrix $\mathbf{\Psi}_{i+1}$ denotes the constraint reduction matrix at time step $i+1$. The variable $\hat{\boldsymbol{\xi}}_{t+1} \in \mathbb{R}^{l_\xi}$ represents the reduced variable approximated through linearization at time step $t+1$.

During the online stage (test-time), we do not know $\boldsymbol{\xi}_t$ (the projected true solution of Eq. (2) at time step t). Rather, we have $\hat{\boldsymbol{\xi}}_t$, the reduced variable approximated through linearization at time step t (computed from Eq. (6) at the previous time step). Therefore, at test-time, Eq. (6) becomes

$$\hat{\boldsymbol{\xi}}_{t+1} = \boldsymbol{\xi}_{i+1} - (\mathbf{J}^r_{i+1})^{-1}[\mathbf{A}^r_{i+1}(\hat{\boldsymbol{\xi}}_t - \boldsymbol{\xi}_i) + \mathbf{B}^r_{i+1}(\mathbf{u}_{t+1} - \mathbf{u}_{i+1})]. \tag{8}$$

Note that $\hat{\boldsymbol{\xi}}_t$ now appears on the right-hand side instead of $\boldsymbol{\xi}_t$. At test-time, the training 'point,' around which linearization is performed (this point defines i and $i+1$), is determined using a 'point-selection' procedure. This point selection depends on $\hat{\boldsymbol{\xi}}_t$ (see He et al., 2011; Jin and Durlofsky, 2018 for details), so the reduced derivative matrices $\mathbf{J}^r_{i+1}$, $\mathbf{A}^r_{i+1}$ and $\mathbf{B}^r_{i+1}$ can all be considered to be functions of $\hat{\boldsymbol{\xi}}_t$. In the last step of POD-TPWL, the approximated solutions are projected back to the full-order space through application of $\hat{\mathbf{x}} = \mathbf{\Phi}\hat{\boldsymbol{\xi}}$.

Each of the above-mentioned steps in POD-TPWL can be viewed in terms of an optimization, as we now consider. The projection matrix $\mathbf{\Phi}$ is constructed using the POD procedure. This has the well-known property that the resulting basis matrix minimizes a projection error e_{proj}, defined as

$$e_{\text{proj}} = \|\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^{\text{T}}\mathbf{x}\|_2^2, \tag{9}$$

where $\mathbf{x} \in \mathbb{R}^{2n_b}$ is the full-order state variable.

In addition, as discussed by He and Durlofsky (2015), the constraint reduction error can be defined as

$$e_{\text{cr}} = \|\hat{\mathbf{x}} - \mathbf{\Phi}\hat{\boldsymbol{\xi}}\|_{\mathbf{\Theta}}^2, \tag{10}$$

where $\hat{\mathbf{x}}$ corresponds to the solution $\hat{\mathbf{x}}_{t+1}$ in Eq. (4) (before constraint reduction is applied); this variable was denoted as $\mathbf{x}_2$ in He and Durlofsky (2015). The variable $\hat{\boldsymbol{\xi}}$ corresponds to the solution $\hat{\boldsymbol{\xi}}_{t+1}$ in Eq. (6) (after constraint reduction is applied) and was expressed as $\boldsymbol{\xi}_3$ in He and Durlofsky (2015). The notation $\|\cdot\|_{\mathbf{\Theta}}$ is a norm defined as $\|\mathbf{e}\|_{\mathbf{\Theta}} = \sqrt{\mathbf{e}^{\text{T}}\mathbf{\Theta}\mathbf{e}}$, with $\mathbf{e} \in \mathbb{R}^{2n_b}$ and $\mathbf{\Theta} \in \mathbb{R}^{2n_b \times 2n_b}$, where $\mathbf{\Theta}$ is a symmetric positive definite matrix. The optimal constraint reduction matrix $\mathbf{\Psi}$ can be determined by minimizing the constraint reduction error, i.e.,

$$\mathbf{\Psi} = \arg\min_{\mathbf{\Psi}} e_{\text{cr}}. \tag{11}$$

If the matrix $\mathbf{\Theta}$ is defined as $\mathbf{J}^{\text{T}}\mathbf{J}$ then, following Eqs. 21 through 27 in He and Durlofsky (2015), we arrive at the least-squares Petrov–Galerkin projection, i.e.,

$$\mathbf{\Psi} = \mathbf{J}\mathbf{\Phi}. \tag{12}$$

This treatment, which as we see is optimal in a particular norm, is now routinely used in POD-TPWL.

The remaining aspect of POD-TPWL to be considered is point selection. Different point-selection strategies have been used for different applications, and these typically include a heuristic component. These procedures entail the minimization of a 'distance' metric, which quantifies the distance (in an application-specific sense) between the current test point and a large set of training-run points. Thus, this step also entails an optimization. As we will see, these POD-TPWL component

optimizations correspond to the loss function minimization that will be applied in the embed-to-control framework. A key difference, however, is that in the E2C framework all of the steps are optimized together, rather than separately as in POD-TPWL.

3. Embed-to-control formulation

In this section, we develop an embed-to-control ROM that includes physical constraints. Analogies to POD-TPWL are established for the various E2C components. The E2C model presented here generally follows that developed by Watter et al. (2015), though several important modifications are introduced, as will be discussed below.

3.1. E2C overview

The embed-to-control framework entails three processing steps: an encoder or inference model that projects the system variables from a high-dimensional space to a low-dimensional subspace (referred to here as the latent space), a linear transition model that approximates system dynamics in low-dimension, and a decoder or generative model that projects solutions back to high-dimensional (full-order) space. The E2C framework originally proposed by Watter et al. (2015) used a VAE architecture for both the encoder and decoder procedures, which allowed them to account for uncertainty in predictions. In the formulation here, the VAE architecture is reduced to an auto-encoder (AE) architecture, since we are considering deterministic systems. We performed limited numerical experimentation (some of the results from these experiments are presented in Supplementary Material) and found that AE was more accurate than VAE for our application. This may be because the complexity of the network has not reached the point where over-fitting is a significant issue. If we use deeper networks, as might be required if we wish to use the E2C ROM for production optimization under uncertainty, another comparison between AE and VAE should be performed to determine the preferable architecture.

We note that the AE architecture is commonly used for semantic segmentation (Ronneberger et al., 2015), where each pixel of the image is associated with a class label, and for depth prediction (Eigen et al., 2014), where the 3D geometry of a scene is inferred from a 2D image. In the context of subsurface flow simulation, AE architectures have been used to construct surrogate simulation models as an image-to-image regression. In this case the input images are reservoir properties (e.g., permeability field) and the outputs are state variables (Zhu and Zabaras, 2018; Mo et al., 2019).

Fig. 1 displays the overall workflow for our embed-to-control model. The pressure field $\mathbf{p}_i \in \mathbb{R}^{n_b}$ is the only state variable shown in this illustration (the subscript i, distinct from t, denotes the time steps in a training run), though our actual problem also includes the saturation field $\mathbf{S}_i \in \mathbb{R}^{n_b}$. Additional state variables would appear in more general settings (e.g., displacements if a coupled flow-geomechanics model is considered).

Box 1 in Fig. 1 displays pressure snapshots $\mathbf{p}_i \in \mathbb{R}^{n_b}, i = 1, \ldots, N_s$ in the full-order space, where N_s is the total number of snapshots. The notation Q_ϕ^{enc} in Funnel 2 denotes the encoder, which projects the full space into a latent space, with ϕ representing all of the 'learnable' parameters in the encoder. By learnable parameters we mean, in general, the set of parameters within the deep-learning-based ROM that are determined in the offline training step. This training is accomplished by minimizing an appropriate loss function. As discussed later, there are learnable parameters associated with the encoder, decoder, and linear transition components of the ROM. The variable $\mathbf{z}_i^p \in \mathbb{R}^{l_z}$ in Box 3 is the latent variable for pressure, with l_z the dimension of the latent space.

In Box 3, the test simulation results are approximated in the latent space with a linear transition model. The variable $\mathbf{z}_0^p \in \mathbb{R}^{l_z}$ denotes the initial latent state for a test run, and $\mathbf{u}_t \in \mathbb{R}^{n_w}, t = 1, \ldots, N_{\text{ctrl}}$ designates the control sequence for a test run, with n_w the number of wells (as noted previously), the subscript t indicates time step in the test run, and N_{ctrl} is the number of control steps in the test run. The linear transition model $\hat{Q}_\psi^{\text{trans}}$ (ψ denotes the learnable parameters) takes $\mathbf{z}_0^p \in \mathbb{R}^{l_z}$ and $\mathbf{u}_t \in \mathbb{R}^{n_w}$ as input, and outputs $\mathbf{z}_t^p \in \mathbb{R}^{l_z}, t = 1, \ldots, N_{\text{te}}$ sequentially, where N_{te} is the total number of time steps in a test run. The decoder P_θ^{dec} (indicated by Funnel 4, with θ representing all of the learnable parameters in the decoder) then projects the variable $\mathbf{z}_t^p$ back to the full-order state $\mathbf{p}_t \in \mathbb{R}^{n_b}$, as shown in Box 5.

We reiterate that the embed-to-control ROM incorporates the control variable $\mathbf{u}_t \in \mathbb{R}^{n_w}$ naturally in the framework. This will be evident in Section 3.3, where we describe the linear transition model. This is an important distinction relative to the AE-LSTM-based ROM developed by Gonzalez and Balajewicz (2018), where system controls were not included in the model.

In the following subsections, the three main components of the embed-to-control framework, the encoder, the linear transition model, and the decoder, will be discussed in detail. A loss function with physical constraints, along with E2C implementation details, will also be presented.

3.2. Encoder component

The encoder provides a low-dimensional representation of the full-order state variables. In contrast to the original embed-to-control implementation by Watter et al. (2015), here we adopt an AE instead of a VAE architecture. With this treatment only the mean values of the latent variables are estimated, not the variances. Also, we do not require a sampling process in the latent space. At train-time, the encoder can be simply expressed as

$$\mathbf{z}_t = Q_\phi^{\text{enc}}(\mathbf{x}_t), \tag{13}$$

where Q_ϕ^{enc} represents the encoder (this notation appears in Fig. 1). The variable $\mathbf{x}_t \in \mathbb{R}^{2n_b}$ is the full-order state variable at time step t, and $\mathbf{z}_t \in \mathbb{R}^{l_z}$ is the corresponding latent variable, with l_z the dimension of the latent space.

In the examples presented later, we consider a 2D 60×60 oil-water model (which means the full-order system is of dimension 7200), and we set $l_z = 50$. This value of l_z was determined based on numerical experiments, where the goal was to assure that the dimension of the subspace was high enough to accurately represent the physics, while still low enough for efficient computation. More specifically, the initial value of l_z was set consistent with values used in POD-TPWL (Jin and Durlofsky, 2018; Jin et al., 2020), where l_z is typically on the order of 100. Then, l_z was reduced gradually, with the accuracy of the E2C ROM monitored. Non-negligible reduction in E2C accuracy was observed for $l_z < 50$; thus we use $l_z = 50$ in this work. Note that the appropriate value for l_z is expected to be somewhat case-dependent. Cross-validation was also conducted to verify that the trained network did not lead to over-fitting.

Note that Eq. (13) is analogous to Eq. (3) in the POD-TPWL procedure, except the linear projection in POD is replaced by a nonlinear projection Q_ϕ^{enc} in the encoder. Following the convention described earlier, we use variables without a 'hat' to denote (projected) true solutions of Eq. (2), which are available from training runs. Variables with a hat designate approximate solutions provided by the test-time ROM.

The detailed layout of the encoder in the E2C model is presented in Fig. 2. During training, sequences of pressure and saturation snapshots are fed through the encoder network, and sequences of latent state variables $\mathbf{z}_t \in \mathbb{R}^{l_z}$ are generated. The encoder network used here is comprised of a stack of four encoding blocks, a stack of three residual convolutional (resConv) blocks, and a dense layer. The encoder in Fig. 2 is more complicated (i.e., it contains resConv blocks and has more convolutional layers) compared to those used in Watter et al. (2015). A more complicated structure may be needed here because, compared to the prototype planning tasks addressed in Watter et al. (2015) (e.g, cart–pole balancing, and three-link robotic arm planning), proper

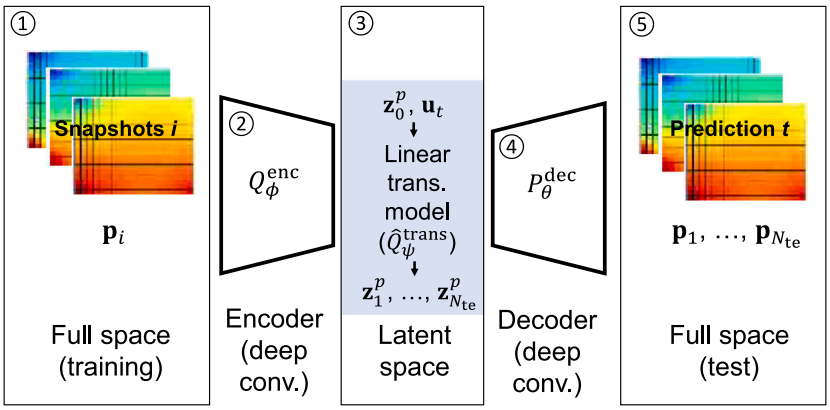


Fig. 1. Embed-to-control (E2C) overview.

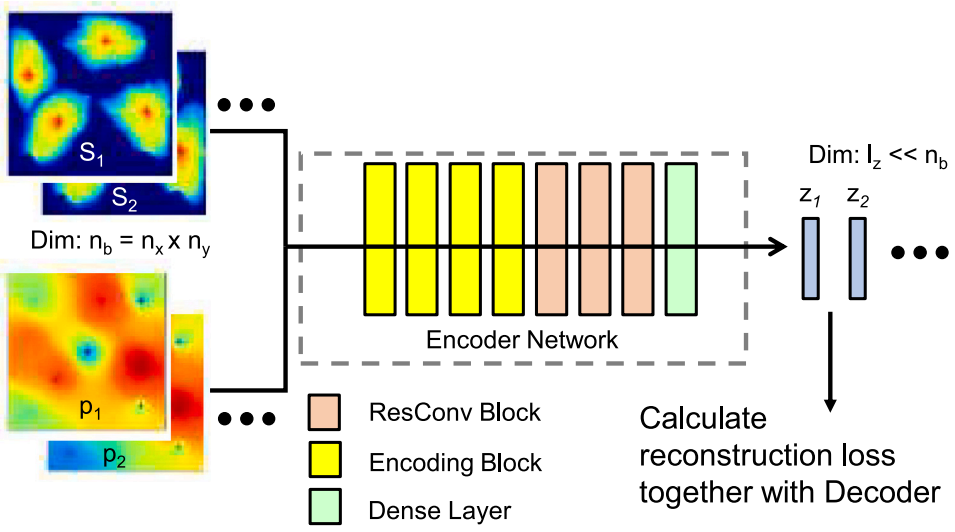


Fig. 2. Encoder layout.

representation of PDE-based pressure and saturation fields requires feature maps from a deeper network.

Similar to the CNN-PCA proposed by Liu et al. (2019), which uses the filter operations in CNN to capture the spatial correlations that characterize geological features, the embed-to-control framework uses stacks of convolutional filters to represent the spatial distribution of the pressure and saturation fields determined by the underlying governing equations. Earlier implementations with AE/VAE-based ROMs (Lee and Carlberg, 2020; Gonzalez and Balajewicz, 2018) have demonstrated the potential of convolutional filters to capture such fields in fluid dynamics problems. Thus, our encoder network is mostly comprised of these convolutional filters (in the form of two-dimensional convolutional layers, i.e., conv2D layer, LeCun et al., 1998). More detail on the encoder network is provided in Table 1 in Supplementary Material.

The input to an encoding block is first fed through a convolution operation, which can also be viewed as a linear filter. Following the expression in Liu et al. (2019), the mathematical formulation of linear filtering is

$$F_{i,j}(\mathbf{x}) = \sum_{p=-n}^{n} \sum_{q=-n}^{n} \mathbf{w}_{p,q} \mathbf{x}_{i+p,j+q} + b, \tag{14}$$

where $\mathbf{x}$ is the input state map, subscripts i and j denote x and y coordinate direction indices, $\mathbf{w}$ represents the weights of a linear filter (template) of size $(2n+1)\times(2n+1)$, $F_{i,j}(\mathbf{x})$ designates the filter response map (i.e., feature map) for $\mathbf{x}$ at spatial location (i, j), and b is a scalar parameter referred to as bias. Note that there are typically many filters associated with a conv2D layer, and the filter response map, which collects all of these operations, is thus a third-order tensor. The output filter response maps are then passed through a batch normalization (batchNorm) layer (Ioffe and Szegedy, 2015), which applies normalization operations (shifts the mean to zero and rescales by the standard deviation) for each subset of training data. A batchNorm operation is a crucial step in the efficient training of deep neural networks, since it renders the learning process less sensitive to parameter initialization, which means a larger initial learning rate can be used. The nonlinear activation function ReLU (rectified linear unit, max(0, x)) (Glorot et al., 2011) is applied on the normalized filter response maps to give a final response (output) of the encoder block. This nonlinear response is referred to as the 'activation' of the encoding block. The conv2D-batchNorm-ReLU architecture (with variation in ordering) is a standard processing step in CNNs.

The learnable parameters in an encoding block include the collection of weights $\mathbf{w}$ and bias terms b in Eq. (14) for all of the filters in conv2D layers, and the shifting and scaling parameters in batchNorm. These parameters are determined during training by minimizing the loss function (defined later). The ReLU layer does not involve learnable parameters. The collection of learnable parameters in all the encoding blocks, resConv blocks, and dense layers in the encoder (which is about 2.42×10^6 total parameters), is represented by ϕ in Eq. (13). An illustration of the encoding block structure is provided in Fig. 1(a) in Supplementary Material.

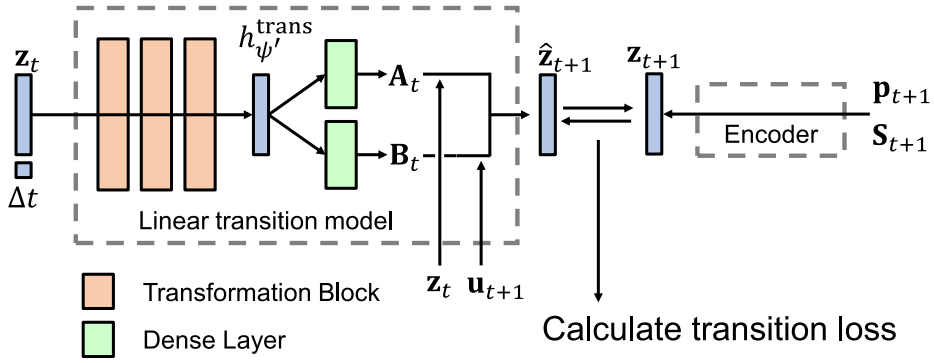


Fig. 3. Linear transition model layout.

To properly incorporate feature maps capable of representing the spatial pressure and saturation distributions, as determined by the underlying governing equations, a deep neural network with many stacks of convolutional layers is required. Deep neural networks are, however, difficult to train, mostly due to the vanishing-gradient issue (Glorot and Bengio, 2010). By this we mean that gradients of the loss function with respect to the model parameters (weights of the filters) become vanishingly small, which negatively impacts training. He et al. (2016) addressed this issue by creating an additional identity mapping, referred to as resNet, that bypasses the nonlinear layer. Following the idea of resNet, we add a stack of resConv blocks to the encoder network to deepen the network while mitigating the vanishing-gradient issue. The nonlinear layer in the resConv block still generally follows the conv2D-batchNorm-ReLU architecture. See Fig. 1(d) in Supplementary Material for a depiction of the resConv block.

Similar to that of the encoding block, the output of resConv blocks is a stack of low-dimension feature maps. This stack of feature maps is 'flattened' to a vector (which is still a relatively high-dimensional vector due to the large number of feature maps), and then input to a dense layer. A dense (fully-connected) layer is simply a linear projection that maps a high-dimensional vector to a low-dimensional vector.

The overall architecture of the encoder network used here differs from that constructed by Zhu and Zabaras (2018) in three key aspects. First, resNet is used in our encoder while they used denseNet (Huang et al., 2017) to mitigate the vanishing-gradient issue. Another key distinction is that the encoder (and the decoder) in Zhu and Zabaras (2018) do not include the dense layer at the end, which means the encoder outputs a stack of feature maps at the end. A large number of feature maps (i.e., a tall but relatively thin third-order tensor) would be too high-dimensional for the sequential linear operations subsequently performed by our linear transition model. Finally, Zhu and Zabaras (2018) adopted a U-Net (Ronneberger et al., 2015) architecture, while our E2C model uses a different architecture.

The encoder (and decoder) in the embed-to-control ROM is analogous to the POD representation used in POD-TPWL. As noted earlier, the basis matrix $\boldsymbol{\Phi}$ constructed via SVD of the snapshot matrices has the feature that it minimizes e_{proj} in Eq. (9). In the context of the encoder, a reconstruction loss $\mathcal{L}_{\text{R}}$, which is similar to e_{proj} for POD, is computed. Conceptually, the 'best' Q_{ϕ}^{enc} is found by minimizing $\mathcal{L}_{\text{R}}$. However, as mentioned earlier, the optimization applied for the embed-to-control model involves all three processing steps considered together, so $\mathcal{L}_{\text{R}}$ is not minimized separately.

3.3. Linear transition model

The linear transition model evolves the latent variable from one time step to the next, given the controls. Fig. 3 shows how the linear transition model is constructed and evaluated during the offline stage (train-time). The inputs to the linear transition model include the latent variable for the current state $\mathbf{z}_t \in \mathbb{R}^{l_z}$, the current step control $\mathbf{u}_{t+1} \in \mathbb{R}^{n_w}$, and time step size Δt. The model outputs the predicted latent state for the next time step $\hat{\mathbf{z}}_{t+1} \in \mathbb{R}^{l_z}$. We reiterate that $\hat{\mathbf{z}}_{t+1}$ represents the output of the linear transition model. The structure of the linear transition model, which generally follows that in Watter et al. (2015), is comprised of a stack of three transformation (trans) blocks and two dense layers. The trans block follows a dense-batchNorm-ReLU architecture (dense represents a dense layer), which is considered a standard processing step for fully-connected networks. The trans block architecture is shown in Fig. 1(c) in Supplementary Material. The variables $\mathbf{z}_t$ and Δt are first fed into the trans blocks. The final activation vector of the trans blocks, $h_{\psi'}^{\text{trans}}$, is then used to construct the linearization matrices $\mathbf{A}_t \in \mathbb{R}^{l_z \times l_z}$ and $\mathbf{B}_t \in \mathbb{R}^{l_z \times n_w}$ through two separate dense layers. Matrices $\mathbf{A}_t$ and $\mathbf{B}_t$ are then combined with the latent variable for the current state $\mathbf{z}_t$ and current step control $\mathbf{u}_{t+1}$ to predict the latent variable at the next time step $\hat{\mathbf{z}}_{t+1}$.

The optimization applied to determine the parameters for the linear transition model is again analogous to a key step in POD-TPWL. In POD-TPWL, the goal is essentially to minimize the difference between the predicted reduced state $\hat{\boldsymbol{\xi}}^{t+1}$ and the projected true state $\boldsymbol{\xi}^{t+1}$. This is achieved, in part, by determining the optimal constraint reduction matrix $\boldsymbol{\Psi}$, as described in Eqs. (10) and (11). Given this optimal $\boldsymbol{\Psi}$ matrix, the matrices appearing in the POD-TPWL evolution equation (Eq. (6)), namely $\mathbf{J}_{i+1}^{r}$, $\mathbf{A}_{i+1}^{r}$ and $\mathbf{B}_{i+1}^{r}$, are all fully defined. As discussed earlier, point-selection represents another (heuristic) optimization that appears in POD-TPWL. Similarly, in the embed-to-control formulation, a transition loss $\mathcal{L}_{\text{T}}$ is computed by comparing $\hat{\mathbf{z}}_{t+1}$ with $\mathbf{z}_{t+1}$, where $\hat{\mathbf{z}}_{t+1}$ is the output from the linear transition model, and $\mathbf{z}_{t+1}$ is the state projected by the encoder at time step $t+1$. The transition loss contributes to the total loss function, which is minimized during the offline stage.

The linear transition model at train-time can also be represented as

$$\hat{\mathbf{z}}_{t+1} = \hat{Q}_{\psi}^{\text{trans}}(\mathbf{z}_t, \mathbf{u}_{t+1}, \Delta t), \tag{15}$$

where Δt is the time step size, the function $\hat{Q}_{\psi}^{\text{trans}}$ is the linear transition model as previously defined (ψ denotes all of the associated learnable parameters, of which there are about 2.35×10^5 in total), and $\hat{\mathbf{z}}_{t+1} \in \mathbb{R}^{l_z}$ denotes the latent variable at $t+1$ predicted by the linear transition model. To be more specific, Eq. (15) can be expressed as

$$\hat{\mathbf{z}}_{t+1} = \mathbf{A}_t(\mathbf{z}_t, \Delta t)\mathbf{z}_t + \mathbf{B}_t(\mathbf{z}_t, \Delta t)\mathbf{u}_{t+1}, \tag{16}$$

where $\mathbf{A}_t \in \mathbb{R}^{l_z \times l_z}$ and $\mathbf{B}_t \in \mathbb{R}^{l_z \times n_w}$ are matrices. Consistent with the expressions in Watter et al. (2015), these matrices are given by

$$\text{vec}[\mathbf{A}_t] = \mathbf{W}_A h_{\psi'}^{\text{trans}}(\mathbf{z}_t, \Delta t) + \mathbf{b}_A, \tag{17}$$

$$\text{vec}[\mathbf{B}_t] = \mathbf{W}_B h_{\psi'}^{\text{trans}}(\mathbf{z}_t, \Delta t) + \mathbf{b}_B, \tag{18}$$

where vec denotes vectorization, so $\text{vec}[\mathbf{A}_t] \in \mathbb{R}^{(l_z^2)\times 1}$ and $\text{vec}[\mathbf{B}_t] \in \mathbb{R}^{(l_z n_w)\times 1}$. The variable $h_{\psi'}^{\text{trans}} \in \mathbb{R}^{n_{\text{trans}}}$ represents the final activation output after three transformation blocks (which altogether are referred

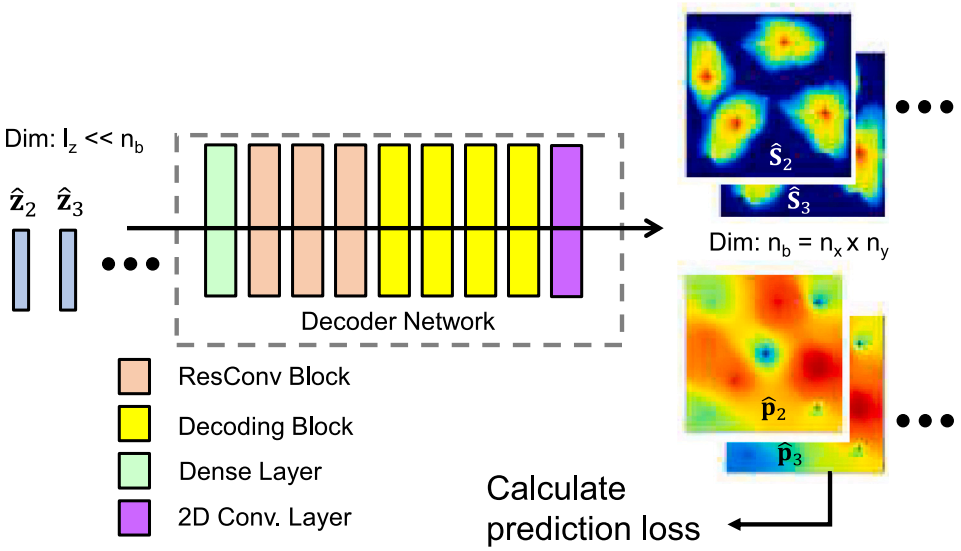


Fig. 4. Decoder layout.

to as the transformation network). The ψ' in Eqs. (17) and (18) is a subset of ψ in Eq. (15), since the latter also includes parameters outside the transformation network. Here $\mathbf{W}_A \in \mathbb{R}^{l_z^2 \times n_{\text{trans}}}$, $\mathbf{W}_B \in \mathbb{R}^{(l_z n_w) \times n_{\text{trans}}}$, $\mathbf{b}_A \in \mathbb{R}^{(l_z^2) \times 1}$, and $\mathbf{b}_B \in \mathbb{R}^{(l_z n_w) \times 1}$, where n_{trans} denotes the dimension of the transformation network. We set $n_{\text{trans}} = 200$ in the model tested here.

During the online stage (test-time) the linear transition model is slightly different, since the latent variable fed into the model ($\hat{\mathbf{z}}_t \in \mathbb{R}^{l_z}$) is predicted from the last time step. Therefore, at test-time, Eq. (16) becomes

$$\hat{\mathbf{z}}_{t+1} = \mathbf{A}_t(\hat{\mathbf{z}}_t, \Delta t)\hat{\mathbf{z}}_t + \mathbf{B}_t(\hat{\mathbf{z}}_t, \Delta t)\mathbf{u}_{t+1}. \tag{19}$$

Note the only difference is that $\mathbf{z}_t$ on the right-hand side of Eq. (16) is replaced by $\hat{\mathbf{z}}_t$ in Eq. (19).

The test-time formulation of the linear transition model is directly analogous to the linear representation step in POD-TPWL. In POD-TPWL, since the training step i (and thus $i+1$) is determined based on the point-selection calculation involving $\hat{\boldsymbol{\xi}}_t$, the matrices appearing in the online expression (Eq. (8)) can be considered to be functions of $\hat{\boldsymbol{\xi}}_t$. After some reorganization, Eq. (8) can then be written as

$$\hat{\boldsymbol{\xi}}^{t+1} = \mathbf{A}_t^{\text{TPWL}}(\hat{\boldsymbol{\xi}}_t)\hat{\boldsymbol{\xi}}_t + \mathbf{B}_t^{\text{TPWL}}(\hat{\boldsymbol{\xi}}_t)\mathbf{u}_{t+1} + \mathbf{c}_t^{\text{TPWL}}, \tag{20}$$

where

$$\begin{aligned} &\mathbf{A}_t^{\text{TPWL}} = -(\mathbf{J}_r^{i+1})^{-1}\mathbf{A}_r^{i+1}, \quad \mathbf{B}_t^{\text{TPWL}} = -(\mathbf{J}_r^{i+1})^{-1}\mathbf{U}_r^{i+1}, \\ &\mathbf{c}_t^{\text{TPWL}} = -\mathbf{A}_t^{\text{TPWL}}\boldsymbol{\xi}^i - \mathbf{B}_t^{\text{TPWL}}\mathbf{u}^{i+1} + \boldsymbol{\xi}^{i+1}. \end{aligned} \tag{21}$$

Thus we see that Eq. (19) for the online stage of the embed-to-control formulation is of the same form as Eq. (20) for the online stage of POD-TPWL. The key difference is that matrices $\mathbf{A}_t$ and $\mathbf{B}_t$ in E2C are determined by a deep-learning model instead of being constructed from derivative matrices from training runs. The vector $\mathbf{c}_t$ does not appear in the E2C formulation, since this representation does not entail expansion around nearby solutions.

Note that the transition loss used here involves pairs of time steps (t and $t+1$) rather than the full sequence. Approaches of this type can potentially lead to error accumulation over the full simulation period. As will be demonstrated later (in the results), error accumulation is not significant for the cases considered here, though this is something that should be monitored. Alternate treatments for the transition loss, in which the loss function is defined over the entire simulation period, are presented in Kani and Elsheikh (2019). Using approaches of this type, error propagation over time can be effectively controlled. The formulation in Kani and Elsheikh (2019) is, however, for cases with fixed well settings. It is not clear if this approach can be applied directly for cases with time-varying well controls, as are considered here.

3.4. Decoder component

The decoder is similar to the encoder and can be represented as

$$\hat{\mathbf{x}}_t = P_\theta^{\text{dec}}(\mathbf{z}_t), \tag{22}$$

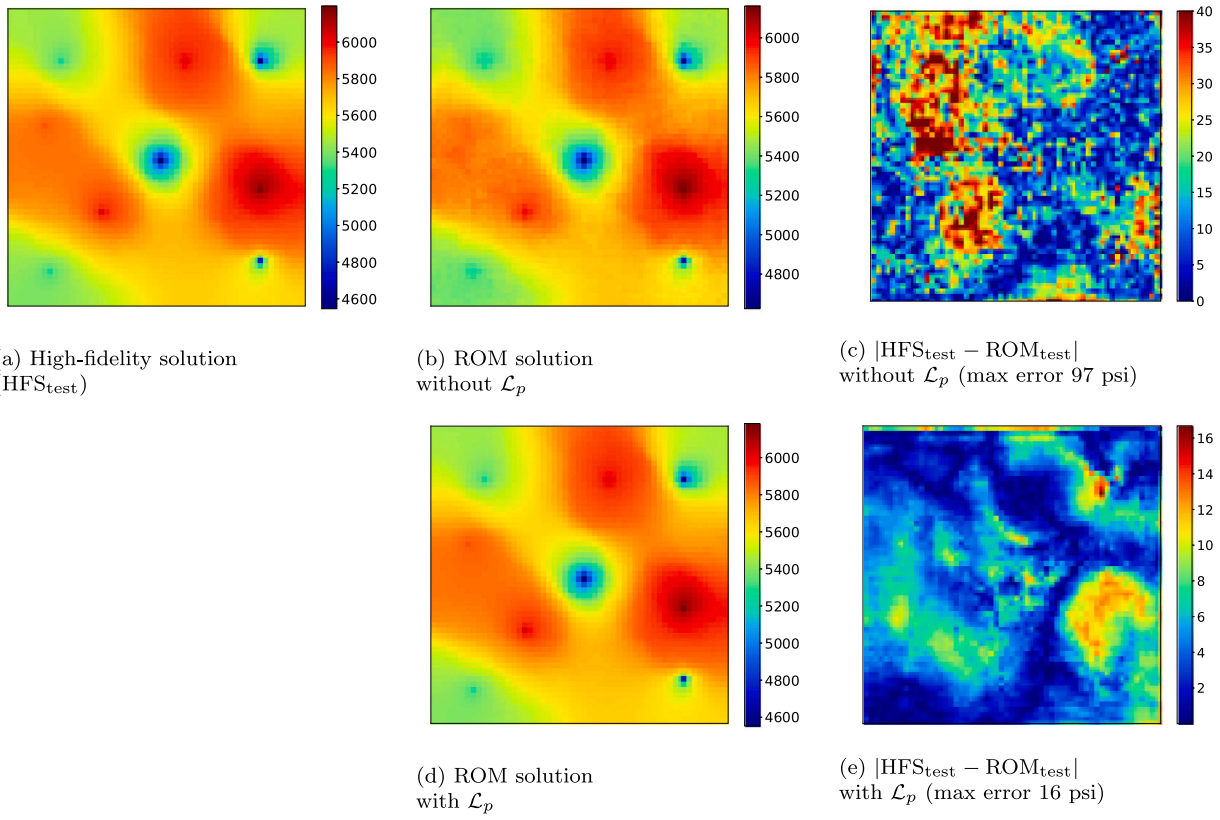
where P_θ^{dec} is the decoder as previously defined. The variable $\hat{\mathbf{x}}_t \in \mathbb{R}^{2n_b}$ denotes the reconstructed state variable at time step t (which is distinct from the high-fidelity state variable $\mathbf{x}_t \in \mathbb{R}^{2n_b}$ from the training snapshots), though the input to the decoder $\mathbf{z}_t \in \mathbb{R}^{l_z}$ is the latent variable determined from the encoding of $\mathbf{x}_t$. If the input is instead $\hat{\mathbf{z}}_{t+1} \in \mathbb{R}^{l_z}$, which is the latent variable predicted at time step $t+1$ by the linear transition model, Eq. (22) becomes

$$\hat{\mathbf{x}}_{t+1} = P_\theta^{\text{dec}}(\hat{\mathbf{z}}_{t+1}), \tag{23}$$

where $\hat{\mathbf{x}}_{t+1}$ is the predicted state variable at time step $t+1$. Note that Eq. (22) only appears in the train-time procedure (to compute reconstructed states), while Eq. (23) has the same form at both train-time and test-time.

The detailed structure of the decoder is shown in Fig. 4. Latent variables predicted by the linear transition model (at time step $t+1$) are fed to the decoder network as input, and the predicted high-dimensional states are output. The architecture of the decoder is analogous to that of the encoder except it is in reversed order (which is not surprising since the decoder is conducting the inverse operation). The decoder here is comprised of a dense layer, a stack of three resConv blocks, a stack of four decoding blocks, and a conv2D layer. The dense layer converts a low-dimensional latent vector to a stack of feature maps (after reshaping). The feature maps are expanded while going through stacks of resConv blocks and decoding blocks. The spatial distributions of the pressure and saturation fields are sequentially 'extracted' from the feature maps as we proceed downstream in the decoder. The conv2D layer at the end converts the expanded feature maps to pressure and saturation fields as the final outputs. More detail on the decoder is provided in Table 2 in Supplementary Material. The layout of the decoding block is shown in Fig. 1(b) of Supplementary Material.

To determine the learnable parameters θ in the decoder, of which there are about 2.60×10^6 in total, a prediction loss $\mathcal{L}_{\text{PD}}$ is minimized (along with the other losses) in the offline process. More details on this optimization will be presented later.



(a) High-fidelity solution ($\mathrm{HFS}_{\mathrm{test}}$)

(b) ROM solution without $\mathcal{L}_p$

(c) $|\mathrm{HFS}_{\mathrm{test}} - \mathrm{ROM}_{\mathrm{test}}|$ without $\mathcal{L}_p$ (max error 97 psi)

(d) ROM solution with $\mathcal{L}_p$

(e) $|\mathrm{HFS}_{\mathrm{test}} - \mathrm{ROM}_{\mathrm{test}}|$ with $\mathcal{L}_p$ (max error 16 psi)

Fig. 5. Pressure field predictions with and without $\mathcal{L}_p$ (all colorbars in units of psi).

3.5. Loss function with physical constraints

We have described each of the components of the embed-to-control framework. We now explain how the model parameters are determined during the offline stage. The parameters for the embed-to-control framework are ϕ, ψ, and θ for the encoder, linear transition model, and decoder, respectively. The objective function to be minimized is the total loss function that quantifies the overall performance of the model in predicting the output state variables.

We have briefly introduced the reconstruction loss ($\mathcal{L}_{\mathrm{R}}$), the linear transition loss ($\mathcal{L}_{\mathrm{T}}$), and the prediction loss ($\mathcal{L}_{\mathrm{PD}}$), which comprise major components of the total loss function. To be more specific, the reconstruction loss for a training data point i can be expressed as

$$(\mathcal{L}_{\mathrm{R}})_i = \{\|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2\}_i, \tag{24}$$

where $i = 1, \ldots, N_t$, with N_t denoting the total number of data points generated in the training runs. Note that $N_t = N_s - n_{\mathrm{train}}$, where N_s is the total number of snapshots in the training runs and n_{train} is the number of training simulations performed. Here N_t and N_s differ because, for a training simulation containing N_{tr} snapshots, only $N_{\mathrm{tr}}-1$ data points can be collected (since pairs of states, at sequential time steps, are required). The variable $\mathbf{x}_t$ is the state variable at time step t from a training simulation, and $\hat{\mathbf{x}}_t = P_\theta^{\mathrm{dec}}(\mathbf{z}_t) = P_\theta^{\mathrm{dec}}(Q_\phi^{\mathrm{enc}}(\mathbf{x}_t))$ denotes the states reconstructed by the encoder and decoder.

The linear transition loss for training point i is similarly defined as

$$(\mathcal{L}_{\mathrm{T}})_i = \{\|\mathbf{z}_{t+1} - \hat{\mathbf{z}}_{t+1}\|_2^2\}_i, \tag{25}$$

where $\mathbf{z}_{t+1} = Q_\phi^{\mathrm{enc}}(\mathbf{x}_{t+1})$ is the latent variable encoded from the full-order state variable at $t+1$, and the variable $\hat{\mathbf{z}}_{t+1} = \hat{Q}_\psi^{\mathrm{trans}}(\mathbf{z}_t, \mathbf{u}_{t+1}, \Delta t)$ denotes the latent variable predicted by the linear transition model. Finally, the prediction loss for training point i is defined as

$$(\mathcal{L}_{\mathrm{PD}})_i = \{\|\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1}\|_2^2\}_i, \tag{26}$$

where $\mathbf{x}_{t+1}$ designates the state variable at time step $t+1$ from the training simulations, and $\hat{\mathbf{x}}_{t+1} = P_\theta^{\mathrm{dec}}(\hat{\mathbf{z}}_{t+1})$ represents the full-order state variable predicted by the ROM. The data mismatch loss is the sum of these losses averaged over all training data points,

$$\mathcal{L}_d = \frac{1}{N_t}\sum_{i=1}^{N_t}(\mathcal{L}_{\mathrm{R}})_i + (\mathcal{L}_{\mathrm{PD}})_i + \lambda(\mathcal{L}_{\mathrm{T}})_i, \tag{27}$$

where λ is a weight term.

The ROM as described up to this point is a purely data driven model, i.e., the goal of the model is to minimize the pixel-wise difference between the E2C output and the high-fidelity solution (the HFS is taken as the 'true' reference solution). Physical behavior is, to some extent, inferred by E2C from the input pressure and saturation snapshots, but it is not explicitly enforced. If the ROM is trained using the loss function $\mathcal{L}_d$ given in Eq. (27), unphysical effects can, however, be observed. This is illustrated in Fig. 5, where we show predictions for the pressure field at a particular time (the problem setup will be described in detail in Section 4). The high-fidelity solution is shown in Fig. 5(a), and the E2C pressure field based solely on $\mathcal{L}_d$ appears in Fig. 5(b). Although the two results are visually similar, the difference map in Fig. 5(c) indicates that the E2C result is not sufficiently smooth, and relatively large errors appear at some spatial locations. This could have a significant impact on well rate predictions, which are an essential ROM output.

To address this issue, we combine the loss for data mismatch with a loss function based on flow physics. Specifically, we seek to minimize the inconsistency in flux between each pair of adjacent grid blocks. Extra weight is also placed on key well quantities. We consider both reconstruction (at time step t) and prediction (at time step $t+1$). Thus we define the physics-based loss for each data point, $(\mathcal{L}_p)_i$, as

$$\begin{aligned}(\mathcal{L}_p)_i =& \{\|\mathbf{k}\cdot[(\nabla\mathbf{p}_t - \nabla\hat{\mathbf{p}}_t)_{\mathrm{recon}} + (\nabla\mathbf{p}_{t+1} - \nabla\hat{\mathbf{p}}_{t+1})_{\mathrm{pred}}]\|_2^2\}_i \\ &+ \gamma\{\|(\mathbf{q}_t^w - \hat{\mathbf{q}}_t^w)_{\mathrm{recon}} + (\mathbf{q}_{t+1}^w - \hat{\mathbf{q}}_{t+1}^w)_{\mathrm{pred}}\|_2^2\}_i.\end{aligned} \tag{28}$$

Here $\mathbf{p}_t, \mathbf{p}_{t+1} \in \mathbb{R}^{n_b}$ are the pressure fields at time steps t and $t+1$ from the training data, which are components of the state variables $\mathbf{x}_t$

and $\mathbf{x}_{t+1}$, and $\hat{\mathbf{p}}_t, \hat{\mathbf{p}}_{t+1} \in \mathbb{R}^{n_b}$ represent the ROM pressure reconstruction (at time step t, defined after Eq. (24)) and prediction (at time step $t+1$, defined after Eq. (26)). The variables $\mathbf{q}_t^w, \mathbf{q}_{t+1}^w \in \mathbb{R}^{n_w}$ are well quantities from the training data, and $\hat{\mathbf{q}}_t^w, \hat{\mathbf{q}}_{t+1}^w \in \mathbb{R}^{n_w}$ are well quantities reconstructed (at time step t) and predicted (at time step $t+1$) by the ROM. Recall that n_w is the total number of wells. The variable γ is a parameter that defines the weights for well-data loss in loss function $\mathcal{L}_p$. The pressure gradients in Eq. (28) are computed via numerical finite difference. The additional computation associated with these terms is negligible.

The terms on the right hand side of Eq. (28) correspond to the flux and source terms in Eq. (1). In the examples in this paper, we specify rates for injection wells and BHPs for production wells. With this specification, the loss on injection rates is zero. The key quantity to track for production wells is the well-block pressure for each well. This is because production rate is proportional to the difference between wellbore pressure (BHP in this case, which is specified) and well-block pressure. The proportionality coefficient is the product of phase mobility λ_j and the Peaceman well index (Peaceman, 1978), which depends on permeability, block dimensions and wellbore radius. Because overall well rate in this case is largely impacted by well-block pressure, we set the second term on the right-hand side of Eq. (28) to $\gamma' \|\mathbf{p}_j^w - \hat{\mathbf{p}}_j^w\|_2^2$, where $\mathbf{p}_j^w \in \mathbb{R}^{n_p}$ and $\hat{\mathbf{p}}_j^w \in \mathbb{R}^{n_p}$ $(j = t, t+1)$ denote the true and ROM well-block pressures, and n_p is the number of production wells. Here γ' is a modified weight that accounts for the well index.

The physics-based loss function is computed by averaging $(\mathcal{L}_p)_i$ over all data points, i.e.,

$$\mathcal{L}_p = \frac{1}{N_t} \sum_{i=1}^{N_t} (\mathcal{L}_p)_i. \tag{29}$$

Combining the loss for data mismatch with this physics-based loss, the total loss function becomes

$$\mathcal{L} = \mathcal{L}_d + \alpha \mathcal{L}_p, \tag{30}$$

where α is a weight term. Through limited numerical experimentation, we found $\alpha = 0.033$ and $\gamma' = 20$ to be appropriate values for these parameters. The E2C ROM prediction for the pressure field at a particular time, using the total loss function $\mathcal{L}$, is shown in Fig. 5(d), and the difference map appears in Fig. 5(e). We see that the ROM prediction is noticeably improved when $\mathcal{L}_p$ is included in the loss function. Specifically, the maximum pressure error is reduced from 97 psi to 16 psi, and the resulting field is smoother (and thus more physical). This demonstrates the benefit of incorporating physics-based losses into the E2C ROM. We note that the (global) flux-loss terms in Eq. (28) contribute more to this error reduction than the well-block-loss terms.

3.6. E2C implementation and training details

To train the E2C model, we use a data set $D = \{(\mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_{t+1})_i\}, i = 1, \ldots, N_t$, containing full-order states and corresponding well controls, where N_t is the total number of training run data points. In the examples in this paper, we simulate a total of 300 training runs. As discussed earlier, this is many more than are used with POD-TPWL (where we typically simulate three or five training runs), but we expect a much higher degree of robustness with E2C. By this we mean that E2C can provide accurate results over a large range of control specifications, rather than over a limited range as in POD-TPWL.

Part of the reason we use a large number of full-order training simulations with the E2C ROM is that the pressure and saturation solutions (snapshots) are extracted at only 20 time steps in each simulation run. These time steps correspond to control steps (changes in well controls). Thus we set $N_{\text{ctrl}} = N_{\text{tr}} = N_{\text{te}} = 20$. This treatment accelerates training and focuses ROM predictions on quantities of interest at time steps when the controls are changing. With 300 training runs, this provides a total number of data points of $N_t = 5700$. It is possible, however, that fewer total training runs could be used if we extracted solutions at more time steps. This might have the downside of limiting the variability in the training data, so a balance between the number of runs and the number of solutions extracted from each run must be established. This issue should be considered in future work. We note finally that the training runs are completely independent of one another, so they can be performed in parallel if a large cluster or cloud computing is available.

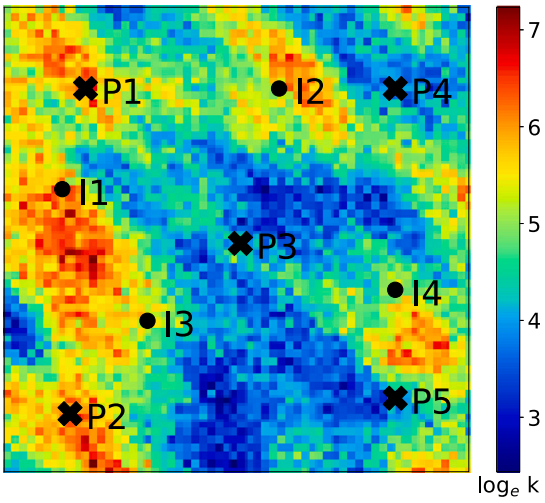


Fig. 6. Log-permeability field and well locations.

The gradient of the total loss function with respect to the model parameters (ϕ, ψ, θ) is calculated via back-propagation through the embed-to-control framework. The adaptive moment estimation (ADAM) algorithm is used for this optimization, as it has been proven to be effective for optimizing deep neural networks (Kingma and Ba, 2014). The rate at which the model parameters are updated at each iteration is controlled by the learning rate l_r. Here we set $l_r = 10^{-4}$.

Normalization is an important data preprocessing step, and its appropriate application can improve both the learning process and output quality. For saturation we have $S \in [0, 1]$, so normalization is not required. Pressure and well data, including control variables, are normalized. Normalized rate q^0, and pressure (both grid-block pressure and BHP) p^0, are given by

$$q^0 = \frac{q - q_{\min}}{q_{\max} - q_{\min}}, \quad p^0 = \frac{p - p_{\min}}{p_{\max} - p_{\min}}. \tag{31}$$

Here q denotes simulator rate output in units of m^3/day, $q_{\max}$ and $q_{\min}$ are the upper and lower injection-rate bounds, p is either grid-block pressure or production-well BHP (units of psi or bar), $p_{\min}$ is the lower bound on BHP, and $p_{\max}$ is 1.1 times the highest field pressure observed (the factor of 1.1 ensures essentially all data fall within the range).

The workflows for the offline and online components of the E2C ROM are summarized in Algorithm 1. In terms of timing, each full-order training simulation requires about 60 seconds to run on dual Intel Xeon ES-2670 CPUs (24 cores). Our E2C ROM is implemented using Keras (Chollet et al., 2015) with TensorFlow (Abadi et al., 2015) backend. The offline training process (excluding training simulation runtime) takes around 10–12 min on a Tesla V100 GPU node (exact timings depend on the memory allocated, which can vary from 8–12 GB). The model is applied on 100 test runs, which will be discussed in detail in the following section. Nearly all of the test results presented are based on the use of 300 training runs, though we also present summary error statistics using 100 and 200 training runs. Offline training for these cases requires about the same amount of time as for 300 training runs, except for the direct savings in the full-order training simulations.

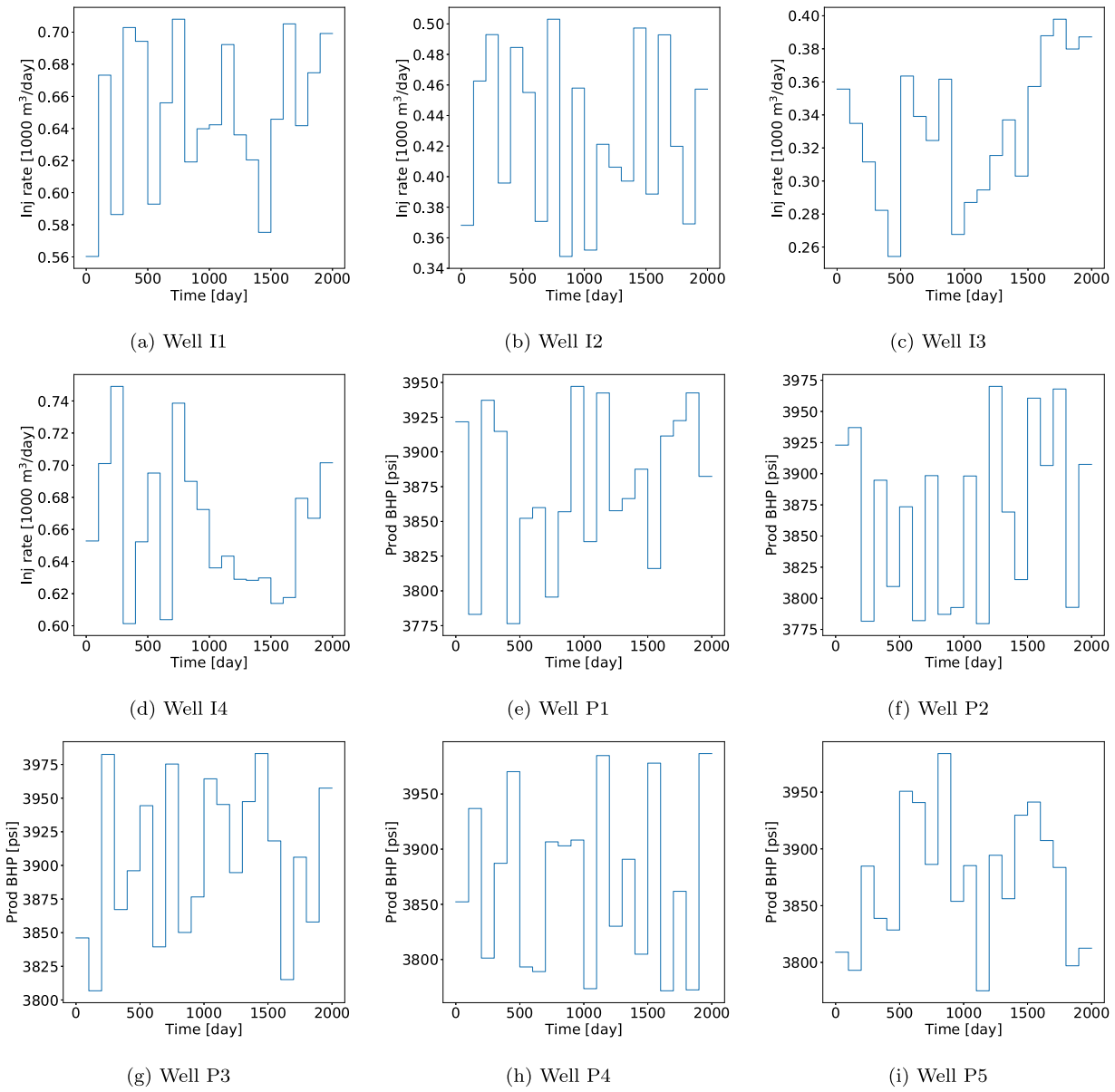


Fig. 7. Test Case 1: well controls.

4. Results using embed-to-control ROM

In this section, we describe the model setup for the oil-water simulations and we present simulation results for the deep-learning-based ROM. One of the test cases is considered in detail in this section; results for two additional test cases are provided in Supplementary Material. In this section we also present summary error results for all 100 test cases.

4.1. Model setup

The geological model, in terms of the log-permeability field, is shown in Fig. 6. The locations of the four injection wells and five production wells are also displayed. The reservoir model contains 60 × 60 (total of 3600) grid blocks, with each block of dimensions 50 m×50 m×10 m. The correlation structure of the log-permeability field is characterized by an exponential variogram model, with maximum and minimum correlation lengths of ∼1000 m and ∼500 m, and an azimuth of 45°. The arithmetic mean permeability is 158 mD, and the standard deviation of log-permeability is 0.88. Permeability is taken to be isotropic, and porosity is set to a constant value of 0.2.

The relative permeability functions are given by

$$k_{ro}(S_w) = k^0_{ro}\left(\frac{1-S_w-S_{or}}{1-S_{wr}-S_{or}}\right)^a,\ k_{rw}(S_w) = k^0_{rw}\left(\frac{S_w-S_{wr}}{1-S_{wr}-S_{or}}\right)^b, \quad (32)$$

where $k^0_{ro} = 1.0$, $k^0_{rw} = 0.7$, $S_{or} = 0.3$, $S_{wr} = 0.1$, $a = 3.6$, and $b = 1.5$. Fluid densities are set to $\rho_o = 800$ kg/m^3 and $\rho_w = 1000$ kg/m^3, and viscosities are specified as $\mu_o = 0.91$ cp and $\mu_w = 0.31$ cp. Capillary pressure effects are neglected.

The initial pressure at the top of the reservoir is 4712 psi (325 bar), and the initial water saturation is 0.1. The total number of primary variables in the system is $3600 \times 2 = 7200$. The model is run for a total of 2000 days. This simulation time frame is the same for both the training and test runs. The injection wells are controlled by specifying time-varying water rates, and the production wells are controlled by specifying time-varying BHPs. The controls for both production wells and injection wells are altered every 100 days, which means there are 20 control periods. Therefore, we have a total of $9 \times 20 = 180$ control parameters over the entire simulation time frame. The range for the

Algorithm 1: E2C ROM procedures

Procedure: Offline procedure
Perform training simulations with given control settings;
Collect snapshots $\mathbf{x}_t$ and controls $\mathbf{u}_t$ and normalize with Eq. (31);
Construct training dataset $\mathcal{D}$;
for *each training epoch* **do**
  Feed the training dataset through E2C model (i.e., Q_ϕ, $\hat{Q}_\psi$, and P_θ defined in Eqs. (13), (16) and (22));
  Compute loss function $\mathcal{L}$ defined in Eq. (30);
  Get derivatives of $\mathcal{L}$ with respect to (ϕ, ψ, θ);
  Update (ϕ, ψ, θ);
end
Procedure: Online procedure
Construct E2C model with parameters (ϕ, ψ, θ) determined in the offline procedure;
Initialize predicted value $\hat{\mathbf{x}}_0 = \mathbf{x}_0$;
for $t = 1, \ldots, T$ *(final simulation time)* **do**
  Predict $\hat{\mathbf{x}}_{t+1}$ from $\hat{\mathbf{x}}_t$ with E2C model defined in Eqs. (13), (19) and (23);
end
Compute well quantities and perform any subsequent analysis;

injection rates is between 1500 and 6500 bbl/day (between 238 and 1033 m^3/day). This is a very large range for well operation compared with what is often considered with ROMs (Jin and Durlofsky, 2018; Jin et al., 2020). The range for production BHPs is 3770 to 3988 psi (between 260 and 275 bar).

The controls for the training and test runs are specified as follows. For each injection well, we randomly sample, from a uniform distribution between 2000 and 6000 bbl/day, a baseline injection rate q_w^{base}. Then, at each control period, we sample uniformly a perturbation q'_w over the range $[-500, 500]$ bbl/day. The rate for the control period is then prescribed to be $q_w^{\text{base}} + q'_w$. Producer BHPs at each control step are sampled uniformly over the range $[3770, 3988]$ psi. For production wells there is not a baseline BHP, and the settings from control step to control step are uncorrelated. This approach for specifying injection rates results in a wide range of solution behaviors (e.g., saturation distributions), since well-by-well injection varies considerably from run to run. This treatment also avoids the averaging effect that can occur if injection rates are not referenced to a baseline value q_w^{base}. Well specifications for a test case, generated using this procedure, are shown in Fig. 7.

We perform 300 training simulations to construct the E2C ROM, except where otherwise indicated. As discussed in previous papers (e.g., Jin and Durlofsky, 2018), the types of well schedules shown in Fig. 7 are intended to represent the well control profiles evaluated during optimization procedures, where the goal is to maximize oil production or profitability, or to minimize environmental impact or some measure of risk. We note finally that the dimension of the E2C latent space, l_z, is set to 50.

As discussed earlier, comparisons of E2C results with those from POD-TPWL are not performed in this work. This is because the range over which the well settings are varied, and the number of training runs performed, are very different between the two procedures, so direct comparisons are not feasible. In particular, existing POD-TPWL implementations typically involve 3–5 training runs, and relatively small perturbations in the test-run controls relative to those used in training runs (during optimization, retraining can be used with POD-TPWL if necessary). Algorithmic treatments, specifically the point-selection strategy used to determine the solution around which linearization is performed, would require detailed reformulation if large numbers of training runs were used with POD-TPWL. With the E2C ROM, by contrast, the controls span a much broader range, but 100–300 training runs are required. In fact, as a deep-learning-based ROM, E2C requires, and is fully compatible with, large numbers of training runs. Thus, although there are conceptual similarities, the two approaches are distinct in terms of their pre-processing demands and applicability range, which greatly limits our ability to perform direct comparisons with current implementations.

4.2. Results for test Case 1

In this section we present detailed results for a particular test case. These include well quantities (injection BHPs and production rates) and global quantities (pressure and saturation fields). The injection rate and BHP profiles for Test Case 1 are displayed in Fig. 7. Here we show the water rates for the four injection wells (Fig. 7(a)–(d)), and the BHPs for the five production wells (Fig. 7(e)–(i)).

We now assess the performance of the deep-learning-based ROM for this test case. The time-evolution of the global saturation field is first considered. The first column in Fig. 8 displays results for the saturation field at 200 days. In Fig. 8(a) the full-order saturation field (also referred to as the high-fidelity solution, HFS) is shown, and the corresponding E2C ROM result for saturation is presented in Fig. 8(d). The color scale indicates water saturation value (thus red denotes water). The close visual agreement between Fig. 8(a) and (d) suggests that the deep-learning-based ROM is able to provide accurate results for this quantity. The level of agreement between the two solutions is quantified in Fig. 8(g), where the difference between the HFS and ROM solutions is displayed. Note that the colorbar scale here is very different than that in Fig. 8(a) and (d). The error between the ROM and HFS results is clearly very small.

In order to better quantify the predictive ability of the E2C ROM, we introduce the concept of the 'closest training run.' We use this term to denote the specific training run, out of the 300 training runs performed, that most closely resembles (in a particular sense) the test case. The 'distance' between the test run and each of the training runs is quantified in terms of the Euclidean distance between their vectors of normalized control parameters, and the 'closest training run' (k^*) is the training run with the minimum distance. Specifically,

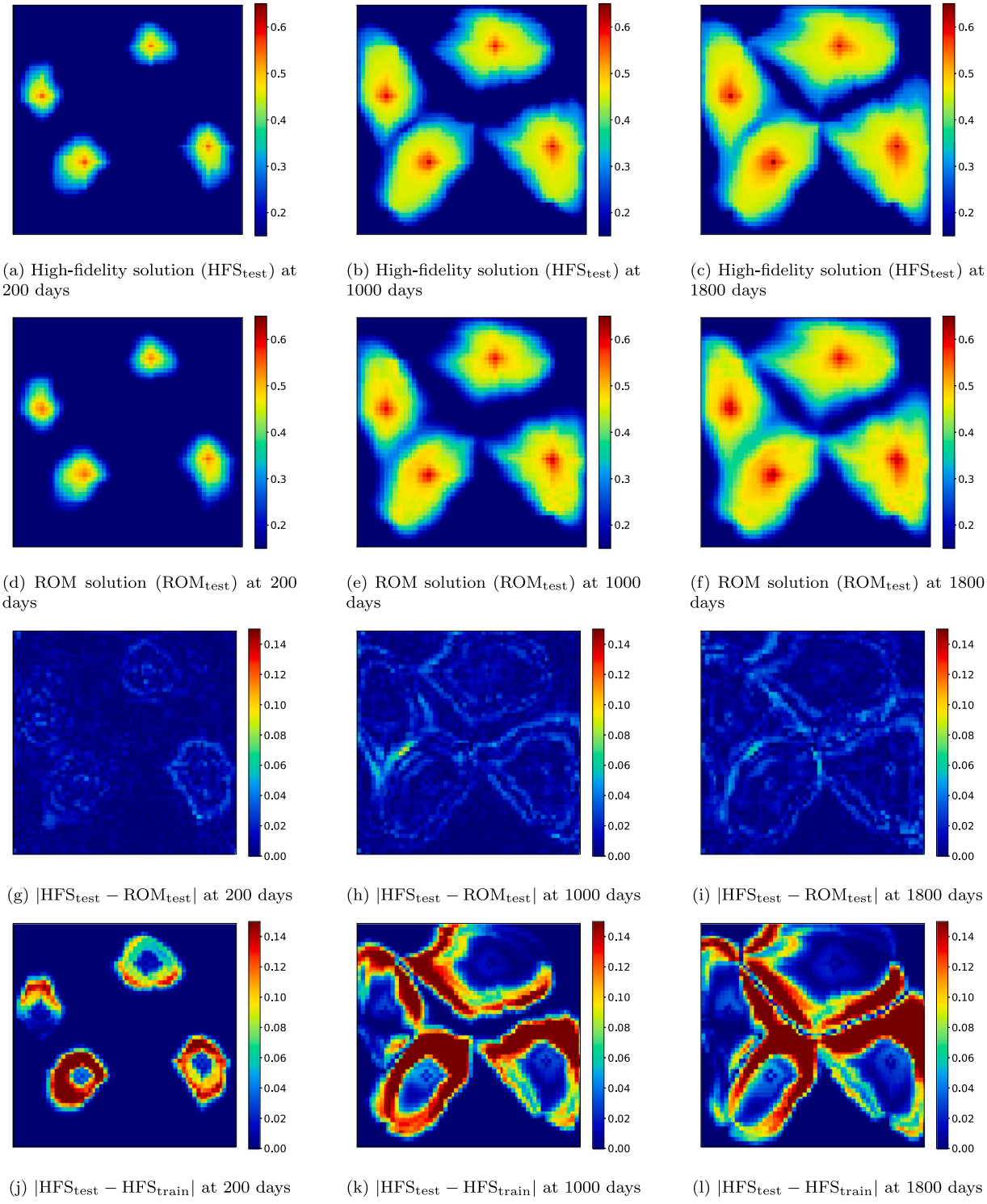
$$k^* = \arg\min_k \|\mathbf{U}^0_{\text{te}} - \mathbf{U}^0_{\text{tr},k}\|_2^2, \tag{33}$$

where $k = 1, \ldots, 300$, denotes the index for the training runs, $\mathbf{U}_{\text{te}} \in \mathbb{R}^{n_w \times N_{\text{ctrl}}}$ represents the control inputs for the test run, $\mathbf{U}_{\text{tr},k} \in \mathbb{R}^{n_w \times N_{\text{ctrl}}}$ indicates the control inputs for training run k, and the superscript 0 designates normalized pressures and rates in the controls, as per the normalizations in Eq. (31).

Eq. (33) provides a very approximate indicator of the 'closest training run.' This definition has the advantage of simplicity, though more involved (and computationally demanding) assessments would be expected to provide closer training solutions. These would, however, require the application of an approach along the lines of the point-selection procedure used in POD-TPWL (Jin and Durlofsky, 2018). This would entail computing a measure of distance over many time steps for each training run. Since we have 300 training runs here (as opposed to three or five with POD-TPWL), this would become very time consuming. Thus we apply the simple approach defined in Eq. (33), with the recognition that more sophisticated procedures could be devised.

We now return to the global saturation results. Fig. 8(j) shows the difference between the 'closest training run' (determined as we just described and simulated at high fidelity), and the test-case saturation fields. The colorbar scale is the same as in Fig. 8(g). The advantage of applying the deep-learning-based ROM is evident by comparing Fig. 8(g) and (j). More specifically, the error in Fig. 8(g) is about an order of magnitude less than the differences evident in Fig. 8(j).

The second and third columns in Fig. 8 display analogous results for the saturation fields at 1000 and 1800 days. The evolution of the



(a) High-fidelity solution (HFS_{test}) at 200 days

(b) High-fidelity solution (HFS_{test}) at 1000 days

(c) High-fidelity solution (HFS_{test}) at 1800 days

(d) ROM solution (ROM_{test}) at 200 days

(e) ROM solution (ROM_{test}) at 1000 days

(f) ROM solution (ROM_{test}) at 1800 days

(g) $|HFS_{test} - ROM_{test}|$ at 200 days

(h) $|HFS_{test} - ROM_{test}|$ at 1000 days

(i) $|HFS_{test} - ROM_{test}|$ at 1800 days

(j) $|HFS_{test} - HFS_{train}|$ at 200 days

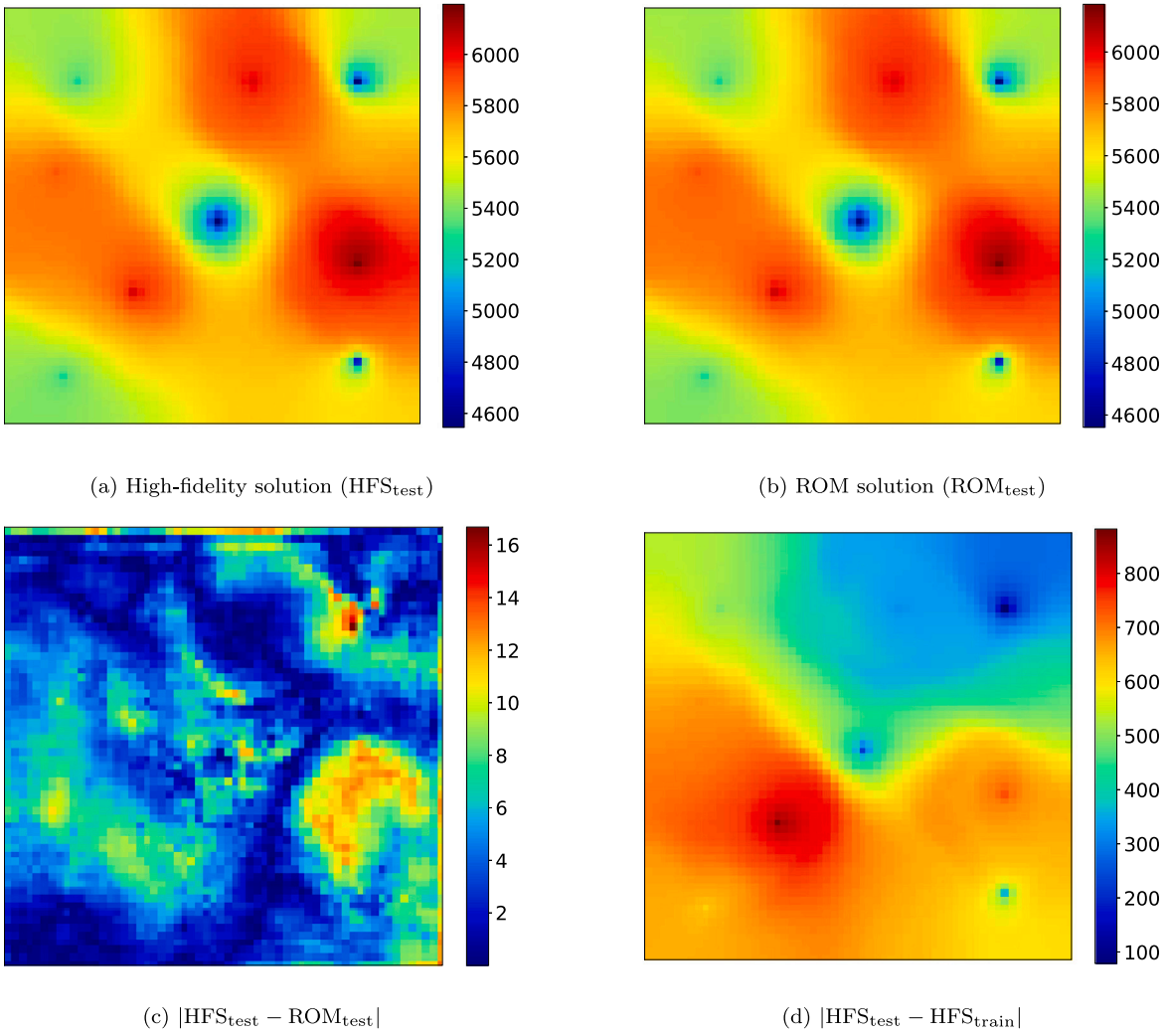
(k) $|HFS_{test} - HFS_{train}|$ at 1000 days

(l) $|HFS_{test} - HFS_{train}|$ at 1800 days

Fig. 8. Test Case 1: saturation field at 200, 1000 and 1800 days. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).

saturation field with time is apparent, and the deep-learning-based ROM solutions (Fig. 8(e) and (f)) are again seen to be in close visual agreement with the HFS (Fig. 8(b) and (c)). The error maps in Fig. 8(h) and (i) further quantify the accuracy of the deep-learning-based ROM. These errors are quite small compared with the difference maps between the 'closest training run' and the HFS, shown in Fig. 8(k) and (l), which further illustrates the effectiveness of the ROM.

Note that in the ROM solutions, we do observe some local (unphysical) extrema within the saturation plumes. These are perhaps most apparent in Fig. 8(f), in the two lower plumes. At the outer (bottom) edges of both plumes, we see small 'islands' of yellow pixels within a light green background. These local fluctuations correspond to nonmonotonic saturation profiles, which are not observed in the HFS (see Fig. 8(c)). Smaller fluctuations in similar regions can also be seen in Fig. 8(e). These unphysical extrema are a minor issue here since the



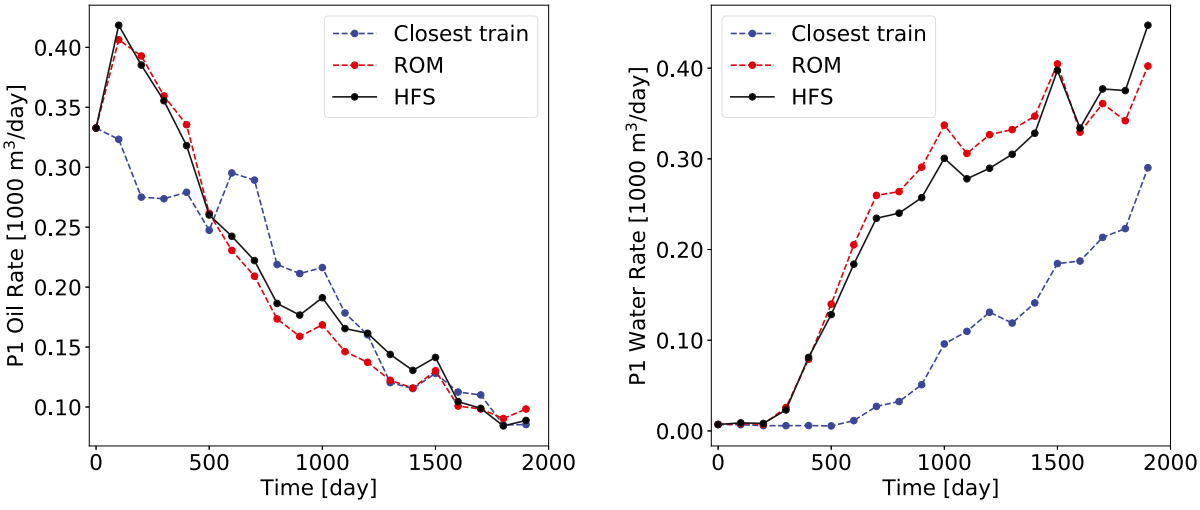
(a) High-fidelity solution ($\mathrm{HFS}_{\mathrm{test}}$)

(b) ROM solution ($\mathrm{ROM}_{\mathrm{test}}$)

(c) $|\mathrm{HFS}_{\mathrm{test}} - \mathrm{ROM}_{\mathrm{test}}|$

(d) $|\mathrm{HFS}_{\mathrm{test}} - \mathrm{HFS}_{\mathrm{train}}|$

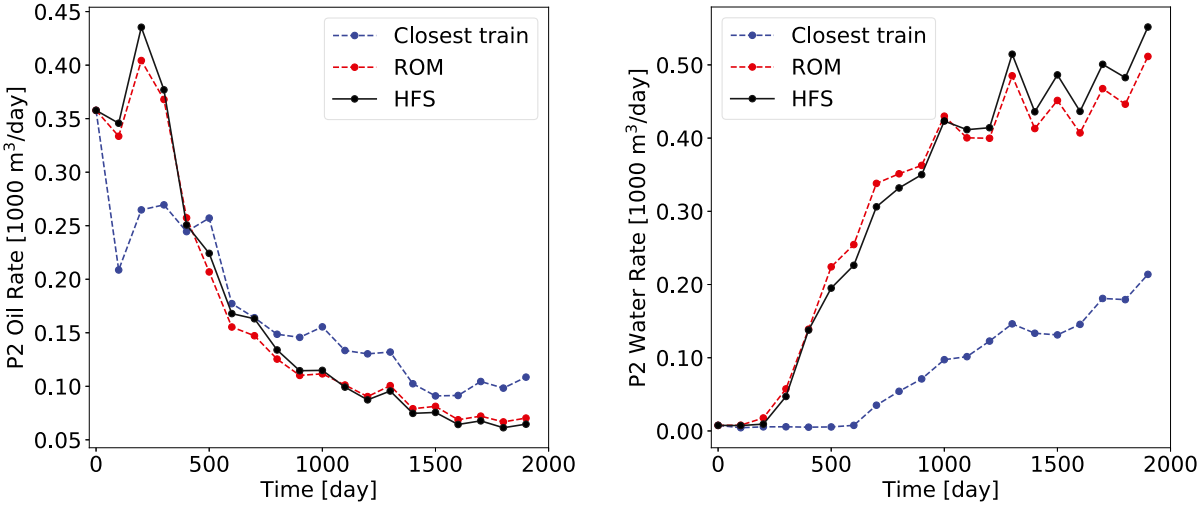
Fig. 9. Test Case 1: pressure field at 1000 days (all colorbars in units of psi).



(a) Oil rate

(b) Water rate

Fig. 10. Test Case 1: production rates for Well P1. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).



(a) Oil rate (b) Water rate

Fig. 11. Test Case 1: production rates for Well P2. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).

difference maps show small overall discrepancies between the ROM and high-fidelity solutions. In some cases, however, such fluctuations could be a cause for concern. A potential remedy for this would be to add a term to the physics-based loss function such that local extrema in saturation that are inconsistent with the governing flow equations are penalized.

The global pressure field at particular times is also of interest. In Fig. 9(a) and (b) we display the HFS and ROM pressure solutions at 1000 days. The close visual agreement suggests that the deep-learning-based ROM is able to provide accurate (and smooth) pressure predictions. Fig. 9(c) shows the error map for the ROM solution, where we see that errors are indeed very small. These errors are much less than those for the 'closest training run,' which are shown in Fig. 9(d).

In many subsurface flow applications the well responses are of primary interest. E2C ROM predictions for these quantities will now be assessed. Since in this problem we specify injection rates and production well BHPs, the quantities of interest are injection well BHPs and oil and water production rates. Figs. 10 and 11 display the phase flow rates for Wells P1 and P2, which are the wells contributing most to total field production. Fig. 12 shows the BHP responses for all four injection wells. In all figures the black curves represent the full-order (reference) HFS, the red curves are the deep-learning-based ROM results, and the blue curves are the results for the 'closest training run.' A high degree of accuracy between the ROM and HFS results is consistently observed. The level of agreement in these essential quantities is enhanced through the additional weighting placed on well-block quantities in loss function $\mathcal{L}_p$ (see Eq. (28)). Recall that we determine all well quantities using a standalone well model to ensure consistency between the HFS and ROM computations.

We provide results for two more examples (Test Cases 2 and 3) in Supplementary Material. These results corroborate our observations here; namely, that the deep-learning-based ROM is able to accurately predict both global saturation and pressure distributions and well quantities of interest.

Finally, we discuss the timings for the high-fidelity and ROM runs. The high-fidelity test cases take 60 seconds each to simulate using AD-GPRS on a node with dual Intel Xeon ES-2670 CPUs (24 cores). The full batch of 100 test cases can be evaluated using the E2C ROM in about 1.25 seconds on a Tesla V100 GPU node with 8 GB of memory allocated. A direct comparison thus indicates a speedup factor of 4800, though this value involves comparing a GPU to a CPU. If we instead use the same CPU for the online E2C ROM runs as was used for the full-order simulations (Intel Xeon ES-2670), it takes around 16 seconds for the 100 test runs. This corresponds to a speedup factor of 375, which is less than that achieved with the GPU, but still very substantial.

4.3. Results and error measurement for all test cases

In this section we assess the accuracy of the ROM results for the full ensemble of 100 test cases. We first consider field cumulative oil and water production, which are given by

$$Q_j = \int_0^T \sum_{w=1}^{n_p} q_j^w(t)\mathrm{d}t. \tag{34}$$

Here $j = o, w$ denotes the phase, n_p is the total number of production wells, T designates the total simulation time, and $q_j^w(t)$ represents the fluid rate for phase j at time step t.

In Fig. 13 we present crossplots of Q_o and Q_w for the HFS and ROM solutions for the 100 test cases. The three ×'s on each plot indicate the results for Test Cases 1, 2 and 3. It is evident that these cases are quite different in terms of Q_o and Q_w, and in this sense span the range of the 100 test cases. We see that the points in both plots fall near the 45° line, which demonstrates that our ROM solutions are in close agreement with the HFS. The results for Q_w in Fig. 13(b) indicate that the ROM under-predicts cumulative water production. The under-prediction is relatively small, however, as the range covered in this plot is narrow. Note also that a slight over-prediction for cumulative oil production is evident in Fig. 13(a).

The errors in Fig. 13 appear to be due, at least in part, to a trend toward the slight under-prediction of water saturation at production-well grid blocks at late time. This often occurs when water production rates are the highest. These errors could potentially be reduced by modifying the loss function used in training to include mismatch in production-block saturation values. This would, however, likely act to compromise ROM prediction accuracy for other quantities of interest. Since the errors in Fig. 13 are small, we did not pursue this line of investigation, but this could be a topic for future work.

We now introduce a number of error measures, which will be used to assess the general performance of the E2C ROM. These error metrics follow those used in Jin et al. (2020). The relative error for oil or water production rate, for a single production well p, is defined as:

$$e_j^p = \frac{\int_0^T \left| q_{\mathrm{ROM}}^{j,p}(t) - q_{\mathrm{HFS}}^{j,p}(t) \right| \mathrm{d}t}{\int_0^T \left| q_{\mathrm{HFS}}^{j,p}(t) \right| \mathrm{d}t}, \tag{35}$$

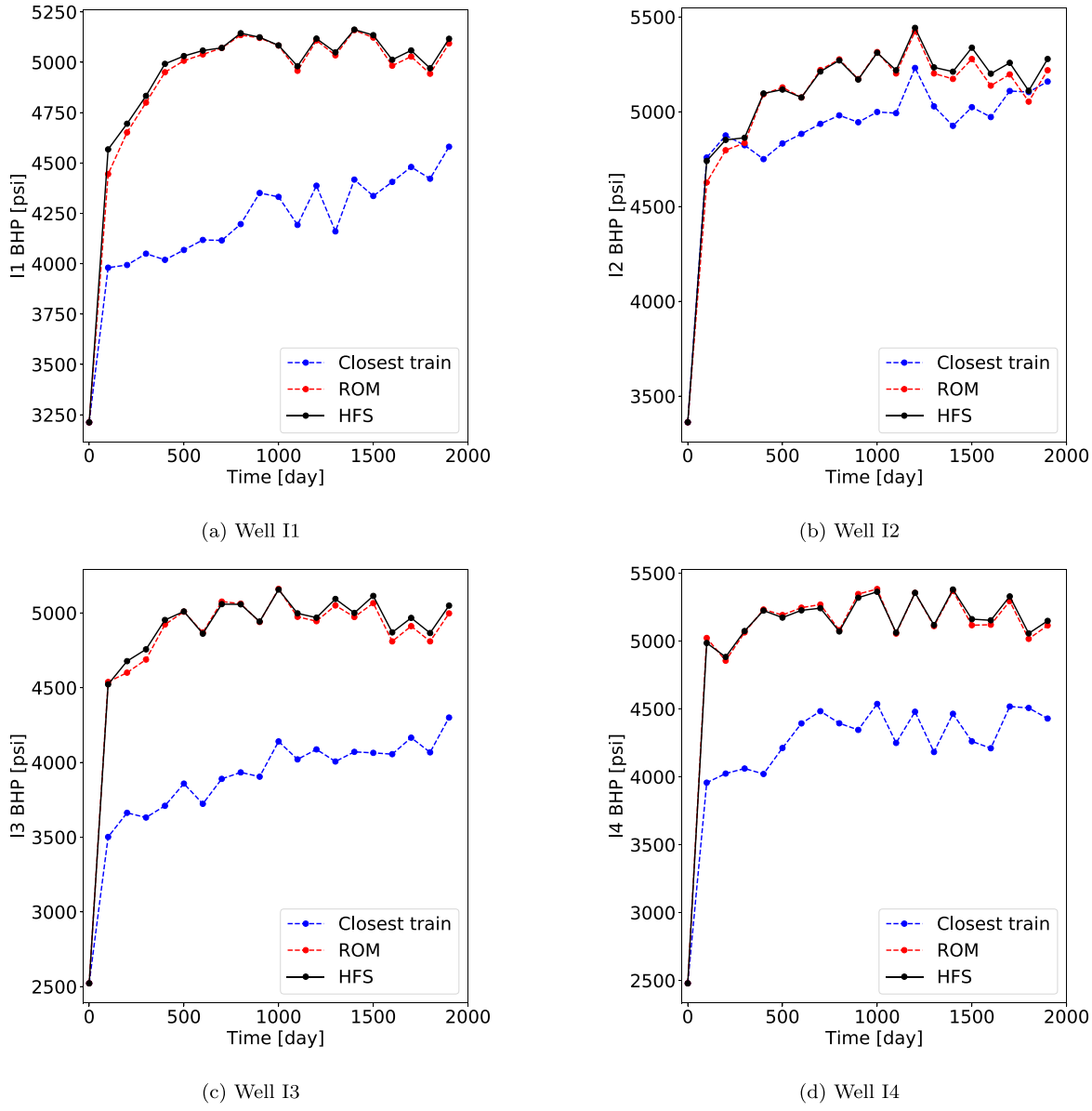


Fig. 12. Test Case 1: injection BHPs. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).

where $j = o, w$ is the fluid phase, $q^{j,p}(t)$ is the oil or water production rate at time t for production well p, the subscripts HFS and ROM denote the high-fidelity and ROM results, and T is the total simulation time. We define the error for overall production rate, E_r, in terms of e_o and e_w for all production wells, as:

$$E_r = \frac{1}{n_p}\sum_{p=1}^{n_p}(e_o^p + e_w^p), \tag{36}$$

where n_p is the total number of production wells. Similarly, the relative error in injection BHP for a single injection well i is defined as:

$$e_{\text{BHP}}^i = \frac{\int_0^T \left|p_{\text{ROM}}^{w,i}(t) - p_{\text{HFS}}^{w,i}(t)\right|\mathrm{d}t}{\int_0^T \left|p_{\text{HFS}}^{w,i}(t)\right|\mathrm{d}t}, \tag{37}$$

where $p^{w,i}(t)$ denotes the injection BHP at time t for injection well i. The overall injection well BHP error E_{BHP} is then given by:

$$E_{\text{BHP}} = \frac{1}{n_i}\sum_{i=1}^{n_i} e_{\text{BHP}}^i, \tag{38}$$

where n_i is the total number of injection wells.

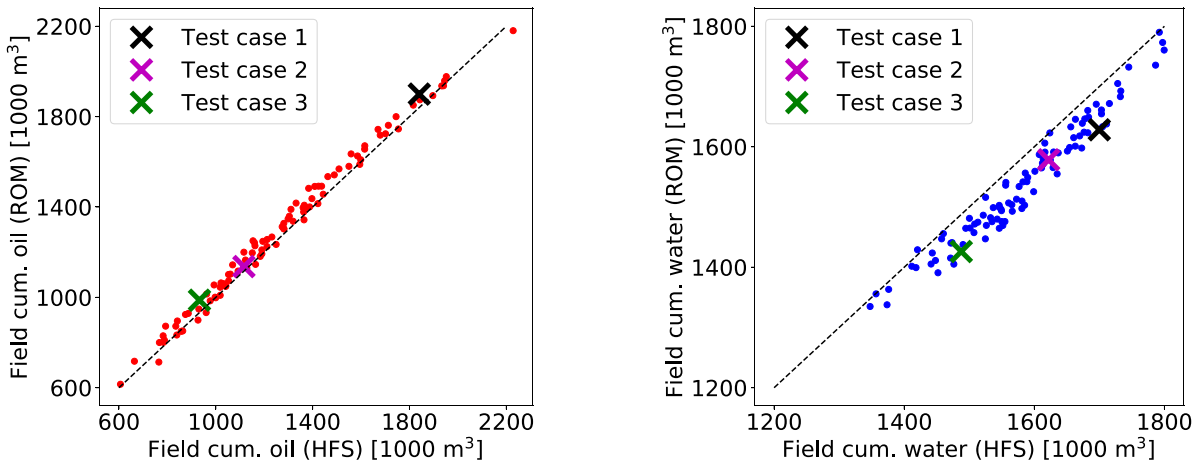
Error in global quantities is also of interest. We define global pressure and saturation error as:

$$E_v = \frac{\sum_{k=1}^{n_b}\int_0^T \left|v_{\text{ROM}}^k - v_{\text{HFS}}^k\right|\mathrm{d}t}{\sum_{k=1}^{n_b}\int_0^T \left|v_{\text{HFS}}^k\right|\mathrm{d}t}, \tag{39}$$

where v^k denotes the global variable of interest in grid block k (pressure p^k or saturation S^k), and n_b is the total number of grid blocks in the model.

These four error quantities are displayed as the red points in Fig. 14. We also evaluate these errors for the 'closest training run' for all test cases. In the plots, the points are ordered by increasing error for the 'closest training run' (blue points). Results for Test Cases 1, 2 and 3 are indicated in each plot. We see that the ROM errors are consistently very small, while the errors for the 'closest training run' are large in many cases. Interestingly, the ROM errors do not appear to depend on the error associated with the 'closest training run.' This is a desirable feature as it suggests a high degree of robustness in the E2C ROM.

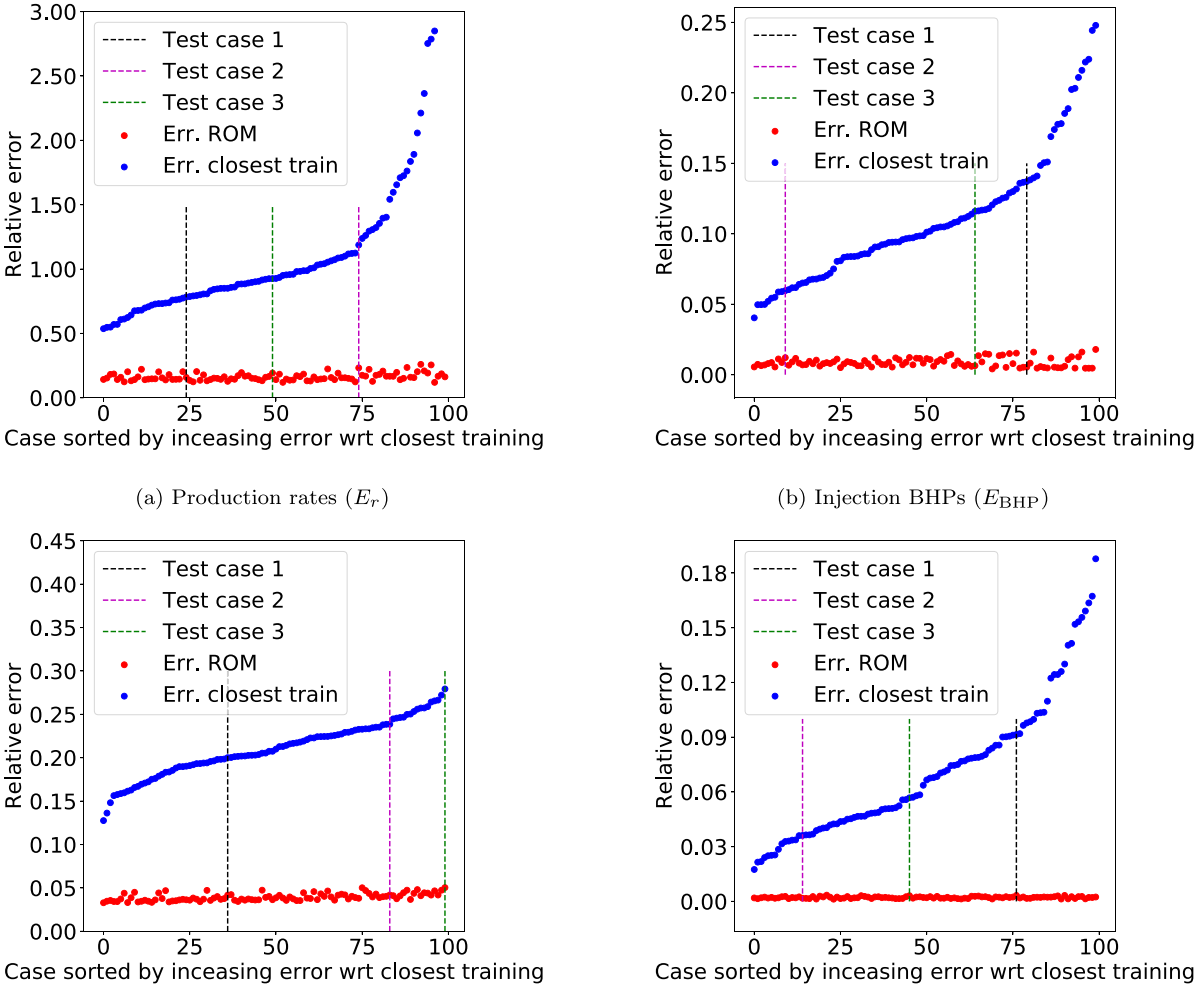
It might also be of interest to evaluate the E2C ROM in terms of mass balance error. This could theoretically be accomplished by



(a) Cumulative oil production

(b) Cumulative water production

Fig. 13. Cumulative oil and water production for all 100 test cases.



(a) Production rates (E_r)

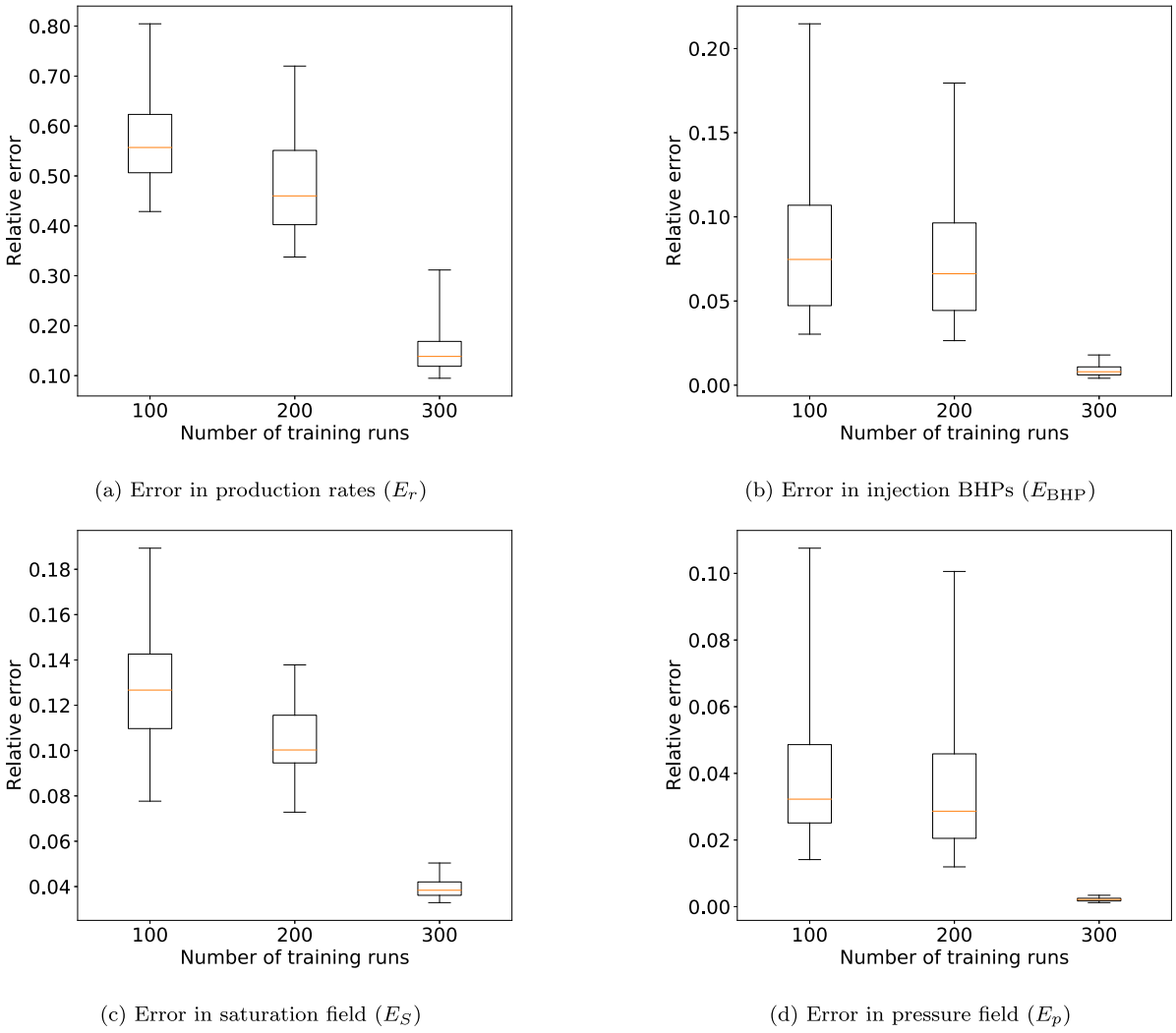
(b) Injection BHPs (E_{BHP})

(c) Saturation field (E_S)

(d) Pressure field (E_p)

Fig. 14. Errors for quantities for interest. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).

computing the net mass flux into each grid block and subtracting the mass accumulation (for well blocks, the source term would also enter this computation). This quantity should sum to zero for each block, and the degree of deviation from zero would indicate mass balance error. In our treatment, however, we predict pressure and saturation fields only at 20 particular time steps; i.e., every 100 days. If we try to compute



(a) Error in production rates (E_r)

(b) Error in injection BHPs (E_{BHP})

(c) Error in saturation field (E_S)

(d) Error in pressure field (E_p)

Fig. 15. ROM error with different numbers of training runs. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).

mass balance error using sequential solutions, this 100-day 'effective' time step is too large to provide meaningful results. In order to assess the mass balance error in our method, we would need to train (and then predict) for a significantly larger number of time steps. This would enable us to have a sufficiently small time step in the mass balance computation.

We now briefly consider the use of smaller numbers of training runs in the construction of the E2C ROM. For these cases we present only summary error results. Fig. 15 displays the four relative errors considered above, in terms of box plots, for 100, 200 and 300 training runs. In each box, the central orange line indicates the median error, and the bottom and top edges of the box show the 25th and 75th percentile errors. The 'whiskers' extending out from the boxes indicate the minimum and maximum errors. There is significant improvement in ROM accuracy as we proceed from 200 to 300 training runs. In future work it will be useful to establish approaches to determine the required number of training runs.

Because it is difficult to display the errors for Test Cases 1, 2 and 3 in the box plots in Fig. 15, we present them in Table 1. These results are with 300 training runs. Note that the average values for E_r, E_{BHP}, E_S and E_p across all 100 test cases are about 0.14, 0.02, 0.04 and 0.002, respectively. The error values for the three test cases shown in the table can be seen to represent a reasonable spread among the full set of test cases. It is of interest to observe that the four errors do not appear to be closely correlated within a particular test case. For example, in Test Case 1, E_p is in the 98th percentile, while E_{BHP} is in the 15th percentile.

Table 1
Errors and percentiles for test cases.

Case		E_r	E_{BHP}	E_S	E_p
Test Case 1	Error	0.11	0.0054	0.042	0.0033
	Percentile	37	15	70	98
Test Case 2	Error	0.19	0.0064	0.050	0.0031
	Percentile	96	29	99	95
Test Case 3	Error	0.16	0.012	0.042	0.0017
	Percentile	77	84	73	24

5. Concluding remarks

In this work, we introduced a deep-learning-based reduced-order modeling procedure for subsurface flow simulation. The procedure was adapted from the existing embed-to-control (E2C) procedure, though we introduced some key modifications relative to the formulation in Watter et al. (2015). Essentially, the ROM consists of an auto-encoder (AE) and a linear transition model. In our E2C formulation, an additional physics-based loss function was combined with the data-mismatch loss function to enhance consistency with the governing flow equations. Although it is based on deep-learning concepts and methods, the various E2C ROM steps were shown to be analogous to those used in the well-developed physics/numerics-based POD-TPWL ROM.

In most of our evaluations, we performed 300 training runs in the offline step. Excluding the run time for the training simulations, the offline model construction required 10-12 minutes for ROM training using a Tesla V100 GPU. Online (runtime) speedups of over a factor of 1000, relative to AD-GPRS full-order simulations, were observed for the case considered. Given the offline costs and online speedup, the use of this ROM is appropriate when many (related) simulation runs are required. This is the case in production optimization computations, data assimilation and uncertainty assessments (though in this work only a single geological model was considered).

The deep-learning-based ROM was tested on 2D oil-water reservoir simulation problems involving a heterogeneous permeability field. Large variations (relative to training runs) in injection and production well control settings were prescribed in the test cases. A total of 100 test cases were considered. ROM accuracy was assessed for key quantities of interest, including well injection BHPs, phase production rates, and global pressure and saturation fields. The E2C ROM was shown to be consistently accurate over the full set of test runs. ROM error was seen to be much lower than that for the 'closest training run' (appropriately defined). Error was found to increase, however, if 100 or 200 training runs were used instead of 300.

In future work, the E2C ROM should be extended to more complicated 3D problems and tested on realistic cases involving, e.g., 10^6 or more cells. Extension to 3D can be approached by replacing conv2D layers with conv3D layers. The online computation time for the 3D ROM will still be negligible compared to the full-order simulation model, but training times will increase with problem size. As noted in Tang et al. (2019), some of the training computations scale linearly with the number of grid blocks in the model, so training times for large 3D models could require many hours. Thus it will be useful to explore a range of network architectures to identify designs for which training can be accomplished in reasonable time. A systematic hyper-parameter tuning mechanism should also be established for more complicated (2D or 3D) models. The relationship between the best hyper-parameters, number of training runs, and the complexity of the model should be investigated.

Future investigations should also consider the use of the E2C ROM for production optimization computations. The use of the method with a range of optimization algorithms should be considered. Importantly, the E2C ROM is expected to be applicable for use with global as well as local optimization algorithms. This is in contrast to existing POD-based ROMs, which can only be expected to be accurate in more limited neighborhoods and are thus most suitable for local-search methods. It is also of interest to explore the potential of predicting flow responses with changing well locations. If this is successful, the ROM could be applied for well location optimization, or combined well location and control optimization problems.

Finally, it will be of interest to extend the ROM to systems with geological uncertainty. Although in this work we found an AE to outperform both a VAE and a UAE, as is demonstrated in Supplementary Material, it is possible that VAE or UAE could be preferable for problems involving geological uncertainty. This is because VAE and UAE can effectively reduce over-fitting for systems with uncertainty. Thus these treatments may be effective for these more general problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Zhaoyang Larry Jin: Conceptualization, Methodology, Data curation, Software, Investigation, Formal analysis, Visualization, Validation, Writing- original draft, Writing- review & editing. **Yimin Liu:** Methodology, Software, Writing- review & editing. **Louis J. Durlofsky:** Supervision, Project administration, Funding acquisition, Writing- review & editing.

Acknowledgments

We are grateful to the Stanford University Smart Fields Consortium (SFC) for partial funding of this work. We thank the Stanford Center for Computational Earth & Environmental Science (CEES) for providing the computational resources used in this study. We also thank Yuke Zhu and Aditya Grover for useful discussions, and Oleg Volkov for help with the AD-GPRS software.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.petrol.2020.107273.

The code is available at https://github.com/lonelysun1990/e2c_jpse.

Please see the data as "data.zip" in the google drive https://drive.google.com/drive/folders/1P-R6uNkzw4lbVjgOIoe42okom08MtAN7?usp=sharing.

References

Abadi, M., et al., 2015. TensorFlow: large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/.

Amsallem, D., Zahr, M.J., Farhat, C., 2012. Nonlinear model order reduction based on local reduced-order bases. Internat. J. Numer. Methods Engrg. 92 (10), 891–916.

Canchumuni, S.W., Emerick, A.A., Pacheco, M.A.C., 2019. History matching geological facies models based on ensemble smoother and deep generative models. J. Pet. Sci. Eng. 177, 941–958.

Cardoso, M., Durlofsky, L.J., 2010. Linearized reduced-order models for subsurface flow simulation. J. Comput. Phys. 229 (3), 681–700.

Carlberg, K., Bou-Mosleh, C., Farhat, C., 2011. Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. Internat. J. Numer. Methods Engrg. 86 (2), 155–181.

Carlberg, K., Farhat, C., Cortial, J., Amsallem, D., 2013. The GNAT method for nonlinear model reduction: effective implementation and application to computational fluid dynamics and turbulent flows. J. Comput. Phys. 242, 623–647.

Chaturantabut, S., Sorensen, D.C., 2010. Nonlinear model reduction via discrete empirical interpolation. SIAM J. Sci. Comput. 32 (5), 2737–2764.

Chollet, F., et al., 2015. Keras. GitHub, https://github.com/fchollet/keras.

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Efendiev, Y., Gildin, E., Yang, Y., 2016. Online adaptive local-global model reduction for flows in heterogeneous porous media. Computation 4 (2), 22.

Eigen, D., Puhrsch, C., Fergus, R., 2014. Depth map prediction from a single image using a multi-scale deep network. In: Advances in Neural Information Processing Systems, Montreal, Canada, pp. 2366–2374.

Florez, H., Gildin, E., 2019. Model-order reduction of coupled flow and geomechanics in ultra-low permeability ULP reservoirs (SPE paper 193911). In: SPE Reservoir Simulation Conference, Galveston, Texas, USA.

Gers, F.A., Schmidhuber, J., Cummins, F., 1999. Learning to forget: continual prediction with LSTM. In: 9th International Conference on Artificial Neural Networks, Edinburgh, UK.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks. In: Fourteenth International Conference on Artificial Intelligence and Statistics, Ft. Lauderdale, Florida, USA.

Gonzalez, F.J., Balajewicz, M., 2018. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. arXiv preprint arXiv:1808.01346.

Grover, A., Ermon, S., 2018. Uncertainty autoencoders: learning compressed representations via variational information maximization. arXiv preprint arXiv:1812.10539.

He, J., Durlofsky, L.J., 2014. Reduced-order modeling for compositional simulation by use of trajectory piecewise linearization. SPE J. 19 (05), 858–872.

He, J., Durlofsky, L.J., 2015. Constraint reduction procedures for reduced-order subsurface flow models based on POD-TPWL. Internat. J. Numer. Methods Engrg. 103 (1), 1–30.

He, J., Sætrom, J., Durlofsky, L.J., 2011. Enhanced linearized reduced-order models for subsurface flow simulation. J. Comput. Phys. 230 (23), 8313–8341.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, Nevada, USA.

Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely connected convolutional networks. In: Computer Vision and Pattern Recognition, Honolulu, USA.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Jiang, R., Durlofsky, L.J., 2019. Implementation and detailed assessment of a GNAT reduced-order model for subsurface flow simulation. J. Comput. Phys. 379, 192–213.

Jin, Z.L., Durlofsky, L.J., 2018. Reduced-order modeling of CO_2 storage operations. Int. J. Greenh. Gas Control 68, 49–67.

Jin, Z.L., Garipov, T., Volkov, O., Durlofsky, L.J., 2020. Reduced-order modeling of coupled flow and quasistatic geomechanics. SPE J. 25 (01), 326–346.

Johnson, J., Alahi, A., Li, F.-F., 2016. Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision, Amsterdam, The Netherlands.

Kani, J.N., Elsheikh, A.H., 2019. Reduced-order modeling of subsurface multi-phase flow models using deep residual recurrent neural networks. Transp. Porous Media 126 (3), 713–741.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kingma, D.P., Dhariwal, P., 2018. Glow: generative flow with invertible 1x1 convolutions. In: Advances in Neural Information Processing Systems, Montreal, Canada.

Kingma, D.P., Welling, M., 2013. Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114.

Kostorz, W., Muggeridge, A., Jackson, M., Moncorge, A., 2019. Non-intrusive reduced order modelling for reconstruction of saturation distributions (SPE paper 193831). In: SPE Reservoir Simulation Conference, Galveston, Texas, USA.

Laloy, E., Hérault, R., Jacques, D., Linde, N., 2018. Training-image based geostatistical inversion using a spatial generative adversarial neural network. Water Resour. Res. 54 (1), 381–406.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86 (11), 2278–2324.

Lee, K., Carlberg, K.T., 2020. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J. Comput. Phys. 404, 108973.

Lee, K., Lim, J., Ahn, S., Kim, J., 2018. Feature extraction using a deep learning algorithm for uncertainty quantification of channelized reservoirs. J. Pet. Sci. Eng. 171, 1007–1022.

Liu, Y., Durlofsky, L.J., 2020. Multilevel strategies and geological parameterizations for history matching complex reservoir models. SPE J. 25 (01), 81–104.

Liu, Y., Sun, W., Durlofsky, L.J., 2019. A deep-learning-based geological parameterization for history matching complex models. Math. Geosci. 51, 725–766.

Mo, S., Zhu, Y., Zabaras, N., Shi, X., Wu, J., 2019. Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media. Water Resour. Res. 55 (1), 703–728.

Peaceman, D.W., 1978. Interpretation of well-block pressures in numerical reservoir simulation. SPE J. 18 (03), 183–194.

Raissi, M., Perdikaris, P., Karniadakis, G., 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.

Rewienski, M., White, J., 2003. A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 22 (2), 155–170.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany.

Tan, X., Gildin, E., Florez, H., Trehan, S., Yang, Y., Hoda, N., 2019. Trajectory-based DEIM (TDEIM) model reduction applied to reservoir simulation. Comput. Geosci. 23 (1), 35–53.

Tang, M., Liu, Y., Durlofsky, L.J., 2019. A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. arXiv preprint arXiv:1908.05823.

Temirchev, P., Gubanova, A., Kostoev, R., Gryzlov, A., Voloskov, D., Koroteev, D., Simonov, M., Akhmetov, A., Margarit, A., Ershov, A., 2019. Reduced order reservoir simulation with neural-network based hybrid model (SPE paper 196864). In: SPE Russian Petroleum Technology Conference, Moscow, Russia.

Temirchev, P., Simonov, M., Kostoev, R., Burnaev, E., Oseledets, I., Akhmetov, A., Margarit, A., Sitnikov, A., Koroteev, D., 2020. Deep neural networks predicting oil movement in a development unit. J. Pet. Sci. Eng. 184, 106513.

Trehan, S., Durlofsky, L.J., 2016. Trajectory piecewise quadratic reduced-order model for subsurface flow, with application to PDE-constrained optimization. J. Comput. Phys. 326, 446–473.

Wang, Z., Xiao, D., Fang, F., Govindan, R., Pain, C.C., Guo, Y., 2018. Model identification of reduced order fluid dynamics systems using deep learning. Internat. J. Numer. Methods Fluids 86 (4), 255–268.

Watter, M., Springenberg, J., Boedecker, J., Riedmiller, M., 2015. Embed to control: A locally linear latent dynamics model for control from raw images. In: Advances in Neural Information Processing Systems, Montreal, Canada.

Xiao, D., Fang, F., Pain, C., Hu, G., 2015. Non-intrusive reduced-order modelling of the Navier–Stokes equations based on RBF interpolation. Internat. J. Numer. Methods Fluids 79 (11), 580–595.

Yang, Y., Ghasemi, M., Gildin, E., Efendiev, Y., Calo, V., 2016. Fast multiscale reservoir simulations with POD-DEIM model reduction. SPE J. 21 (06), 2141–2154.

Yoon, S., Alghareeb, Z.M., Williams, J.R., 2016. Hyper-reduced-order models for subsurface flow simulation. SPE J. 21 (06), 2128–2140.

Zahr, M.J., Avery, P., Farhat, C., 2017. A multilevel projection-based model order reduction framework for nonlinear dynamic multiscale problems in structural and solid mechanics. Internat. J. Numer. Methods Engrg. 112 (8), 855–881.

Zhang, J., Cheung, S.W., Efendiev, Y., Gildin, E., Chung, E.T., 2019. Deep model reduction-model learning for reservoir simulation (SPE paper 193912). In: SPE Reservoir Simulation Conference, Galveston, Texas, USA.

Zhou, Y., 2012. Parallel General-Purpose Reservoir Simulation with Coupled Reservoir Models and Multisegment Wells (Ph.D. thesis). Stanford University.

Zhu, Y., Zabaras, N., 2018. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. J. Comput. Phys. 366, 415–447.

Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., Perdikaris, P., 2019. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J. Comput. Phys. 394, 56–81.