Application and effects of physics-based and non-physics-based regularizations in artificial intelligence-based surrogate modelling for highly compressible subsurface flow

Citation for published version:

Digital Object Identifier (DOI):
10.1016/j.geoen.2023.212474

Link:
Link to publication record in Heriot-Watt Research Portal

Document Version:
Publisher's PDF, also known as Version of record

Published in:
Geoenergy Science and Engineering

Publisher Rights Statement:
© 2023 The Author(s).

General rights
Copyright for the publications made accessible via Heriot-Watt Research Portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
Heriot-Watt University has made every reasonable effort to ensure that the content in Heriot-Watt Research Portal complies with UK legislation. If you believe that the public display of this file breaches copyright please contact open.access@hw.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 22. Mar. 2024
Application and effects of physics-based and non-physics-based regularizations in artificial intelligence-based surrogate modelling for highly compressible subsurface flow

Victor C. Molokwu\textsuperscript{a,}\textsuperscript{*}, Bonaventure C. Molokwu\textsuperscript{b}, Mahmoud Jamiolahmady\textsuperscript{a}

\textsuperscript{a} Heriot-Watt University, Institute of Geoenergy Engineering, Edinburgh, United Kingdom
\textsuperscript{b} Concordia University, Department of Computer Science and Software Engineering, Montreal, Canada

\textbf{A R T I C L E I N F O}

Dataset link: https://github.com/molokwuvictor/3D_feature_generator

\textbf{Keywords:}
artificial intelligence
Deep learning
Physics-based learning
Surrogate reservoir modelling
Reservoir simulation

\textbf{A B S T R A C T}

Artificial intelligence (AI)-based surrogate reservoir models (SRMs) can provide computationally feasible and accurate approximations to numerical simulations. An AI-based SRM is trained to a set of parameters that significantly reduces its variance. This can be done by either supervised or semi-supervised learning. The latter involves regularization of the model’s parameters using non-physics-based, physics-based or a combination of both regularization terms.

Effective enforcement of the physics-based and non-physics-based regularizations can significantly reduce the variance of AI-based SRMs. Little study has been reported on the application and effects of regularization terms. Also, for highly compressible subsurface flow where strong nonlinearities exist, well-constructed composite AI-based architectures and regularizations are necessary for learning.

This paper applies and studies the effects of various regularization terms for highly compressible subsurface flow; it proposes unique and effective techniques in AI-based surrogate development and training. The learning utilizes the discretized domain and boundary physics with derivatives obtained from both finite difference methods (FDM) and algorithmic differentiation (AD). The regularizations are partly enforced as a hard constraint in the network architecture using a trainable layer and as soft constraints using a multi-cost function. The soft constraints exploit a tank material balance and time-discretization numerical errors, in addition to the domain, boundary and non-physics-based $L_2$ regularization terms. The timely-trained AI-based surrogate predictions agree with those obtained from a numerical simulator.

The regularization terms separately contribute to improved learning. The non-physics-based $L_2$ norm if used in the right order of magnitude, improves grid block predictions. The tank material balance regularization term constrains the AI-based surrogate parameters to net domain accumulation, ensuring its reliability. The trainable hard enforcement layer enforces the initial condition and improves the predictions compared to other hard enforcement techniques. The discretized domain equation and time-discretization numerical errors allow for learning of variable timesteps, which give the best rounding-truncation error trade-off and improve the predictions compared to those of fixed timesteps. The AI-based surrogate, effectively trained by semi-supervision, can be reliably used as a state-dependent model in domain analysis like sensitivity and data assimilation.

1. Introduction

Highly compressible fluids are encountered in many subsurface systems and accurately simulating their flow behaviour using new frontiers like AI is essential for exploitation from or storage in these systems. Fluid flow in porous media is governed by the partial differential equation (PDE) and for highly compressible fluids, the PDE has state-dependent fluid properties, resulting in a nonlinear equation. This can be solved using numerical techniques. A numerical simulator solves the governing PDE using computational fluid dynamics (CFD) approaches like FDM and finite element methods. In practical scenarios, obtaining solutions that capture uncertainty in inputs like permeability requires multiple numerical simulations, up to hundreds or thousands

\textsuperscript{*} Corresponding author.

\textit{E-mail addresses:} vcm1@hw.ac.uk (V.C. Molokwu), bonaventure.molokwu@concordia.ca (B.C. Molokwu), m.jamiolahmady@hw.ac.uk (M. Jamiolahmady).


https://doi.org/10.1016/j.geoen.2023.212474

Received 25 August 2023; Received in revised form 7 October 2023; Accepted 7 November 2023
Available online 10 November 2023

2949-8910/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).
of runs. These PDE solutions can be computationally expensive for the numerical simulator and can delay decision-making.

A surrogate model can provide a computationally feasible and reliable approximation to the numerical simulations; it gives a fit-for-purpose model for quickly obtaining the multiple flow realizations of the subsurface domain without much sacrifice to accuracy or details (Memon et al., 2013; Mohaghegh et al., 1996). A surrogate’s only significant time expense is during the training phase, after which predictions are made in seconds. Surrogates developed and used in subsurface modelling can be categorized into the projection-based reduced order model (ROM) (Cardoso and Durlofsky, 2010; Markovic and Jansen, 2006; Xiao et al., 2019, 2022) and data-driven models.

Data-driven surrogate modelling involves empirical approximation by either a response surface or an ANN. The data-driven models do not require intrusive access to high-fidelity numerical simulator codes; their estimators are calibrated with datasets containing sufficient training examples. A training example consists of a feature vector and a label. The feature vector is an instance in the domain, e.g., the space and time characteristics, and the label is a collection of true responses at the instance. The calibration, called machine learning (ML), is performed by either supervised, unsupervised or weakly-supervised methods. Supervised learning involves learning on a labelled dataset (Hastie et al., 2009; Dridi, 2021). Unsupervised learning involves identifying patterns in datasets that are neither labelled nor classified (Ghahramani, 2003). Weakly-supervised learning involves learning by either incomplete, inexact or inaccurate supervision (Zhou, 2018). Semi-supervised, a kind of incomplete supervision is of interest in this paper. This involves using an incomplete dataset, in which a small amount is labelled while the remaining are unlabelled. However, the unlabelled training examples can utilize the labelled subdataset for learning (Chapelle et al., 2009; Zhou, 2018; Van Engelen and Hoos, 2020). Fig. 1 is a description of the learning process.

Data-driven surrogate models constructed using response surfaces, e.g., polynomial basis function (Hussain et al., 2002), radial basis function (Regis and Shoemaker, 2007; Regis, 2011), Gaussian process (Kennedy and O’Hagan, 2001; Yang et al., 2018), stochastic polynomial chaos expansion (Laloy et al., 2013) can be challenging in training examples with strong nonlinearities and high dimensionality. Another approach to data-driven surrogate modelling involves using ANNs. This is called the artificial intelligence AI-based surrogate model (Amini and Mohaghegh, 2019; Mohaghegh et al., 2009; Pan et al., 2014; Vida et al., 2019). AI-based surrogate models provide rapid space–time solutions of the domain by learning on training examples from sampled locations in the domain. Recent applications (2010 and beyond) of AI involve deep learning, which seeks to discover representations (or patterns) in a dataset using architectures with multiple levels of abstraction rather than a direct input to output (Bertinetto et al., 2016). Deep learning architectures and their applications include: fully connected deep neural networks (FCDNNs)—(Meng and Kariardakias, 2020; Lu et al., 2021; Molokwu and Jamilolahmady, 2022); convolutional neural networks (CNNs)—(Zhang et al., 2018; Molokwu, 2019; Lee et al., 2020; Reginald et al., 2021; Mohd Razak and Jafarpour, 2020) and CEDNNs—(Wang et al., 2021; Xiao et al., 2022; Zhang et al., 2021).

Deep learning architectures can handle high-dimensional parameters and can overcome the curse of dimensionality, i.e., the volume of data needed to estimate an arbitrary function increases exponentially with respect to the input parameters. The exact method in which deep learning (DL) algorithms break this curse remains a fundamental question. The manifold hypothesis (Fefferman et al., 2013) is one theory that justifies how DL can handle this curse; it implies that in a real-world high-dimensional data, there lies a low-dimensional manifold embedded in the high-dimensional space. DL techniques can discover and utilize the low-dimensional manifold. The hierarchically local compositional functions in deep convolutional networks, where all the constituent functions are local, and have bounded, small dimensionality without no weight sharing can approximate a high dimensional data without the curse of dimensionality (Poggio and Liao, 2018). In AI-based surrogate modelling, learning follows an inductive approach to reduce its bias and variance. The bias error is due to erroneous assumptions in the AI-based surrogate structure and the variance is the sensitivity of the surrogate to changes in input. An optimum surrogate model is one that yields low bias and low variance errors. The bias can be reduced by changing the ANN architecture, e.g., increasing the number of trainable parameters by adding more transformation layers. The variance is reduced by either the supervised learning or regularization

Regularization involves using non-physics-based or physics-based penalties to constrain the surrogate’s parameters when learning on a sparse dataset; this is a semi-supervised method. Sparse dataset can occur when the training examples are few; the additional simulation and processing time associated with generating more training examples can defeat the timelines of the surrogate. Sparse dataset can also occur when there are sufficient inexpensive training examples but incompletely labelled, i.e., a small amount is labelled and the remaining, unlabelled. The regularized parameters prevents overfitting and also reduce the likelihood of exploding gradients during training. Non-physics-based regularization has been applied for improved
AI-based surrogate modelling of gas and gas-condensate subsurface flow (Molokwu and Jamilahmady, 2022).

Physics-based regularization can significantly reduce the surrogate model’s variance by utilizing the domain and boundary physics, called the physics-informed neural network (PINN) (Rasisi and Karniadakis, 2018; Raisisi, 2018; Sun et al., 2020; Tartakovsky et al., 2020) or theory-guided convolutional neural network (TgCNN) (Wang et al., 2021). Effective implementation of physics-based regularization can compensate for dataset incompleteness. Physics-based regularization of neural networks has been used to learn linear and nonlinear PDEs and hidden fluid relationships in fluid flow problems with limited amounts of labelled datasets (Kadeethum et al., 2020; Raisisi, 2018; Meng and Karniadakis, 2020); and without labelled datasets (Sun et al., 2020).

Physics-based methods have also been applied in surrogate reservoir modelling to learn the theories of subsurface hydrocarbon flow in linear and nonlinear PDEs, solved using FDM for single and multiphase flow of slightly compressible fluids (oil and water) (Wang et al., 2021, 2022).

Physics-based regularization requires the evaluation of derivatives and their higher orders. These derivatives can be evaluated algorithmically—AD or numerically—FD. AD allows for exact evaluation using software computing libraries; however, this can introduce significant overhead when computing higher-order derivatives as the gradients obtained by chain rule are stored in memory. AD can adversely affect computation time if the network graph is deep. FDM use finite differences for faster evaluation of the space–time derivatives; it requires that the space–time domain is discretized, resulting in round-off (floating point precision) and truncation errors. ANNs can be used to learn the numerical errors associated with the discretization, then reduced as part of the physics-based losses. Learning the associated errors yields an optimal discretization step at each time point, which is used during training.

Highly compressible fluids, due to their nonlinear PDE, require deeper network architectures for evaluation of pressure-dependent fluid properties. The FDM can be used as timely solutions for evaluation of the derivatives; however, this could result in non-convergence if the AI-based surrogate is not properly enforced and the discretization steps not carefully selected. This paper investigates the application and effects of applying various physics-based regularization terms including the non-physics-based \( L_2 \) norm regularization term in AI-based surrogate modelling for highly compressible subsurface flow. The training is done by semi-supervision—i.e., only the initial and boundary subdatasets are labelled with initial pressure and boundary (well) production constraints respectively while the remaining, larger training examples are unlabelled. This paper uniquely proposes the following for effective learning:

1. A custom three-module network architecture, comprised of pressure, fluid property and timestep modules. The pressure module utilizes FDM while the fluid property module uses AD in computing its derivatives. The timestep module allows for variable timesteps by learning the combined discretized domain PDE and time-discretization numerical errors.
2. A tank material balance regularization term, in addition to the domain, boundary and \( L_2 \) norm regularization terms.
3. A trainable layer for hard enforcement of the initial condition and output refinement.
4. Time-discretization numerical errors derived from two Euler differences.

The paper is organized as follows: The physics-based and non-physics-based regularizations are described in the methodology. In the next section, the custom network architecture is explained. In the case study section, the implementation of the AI-based surrogate on a dry gas, multi-well reservoir with varying operating conditions, including a shut-in is presented. In the results section, the performance of the AI-based surrogate is compared with results obtained from a numerical reservoir simulator for a permeability-time dataset of 146 × 48 realizations, and the effects of various regularization terms are presented. The paper concludes with a summary of the main findings.

2. Methodology

2.1. Physics-based equations for single-phase compressible fluid flow in porous media

The general equation describing the flow of a single-phase compressible fluid (i.e., gas) in a porous media is obtained by combining the continuity equation and Darcy’s transport equation (Aziz and Settari, 1979):

\[
V \cdot \left( \frac{\rho}{\mu} (V \cdot \nabla) z \right) = \frac{\partial}{\partial t} \left( \rho \phi \right) + \frac{m}{\nabla} \delta V
\]

\( m \) is the rate of mass accumulation in a sink or source (negative if source) and \( \delta V \) is the infinitesimal control bulk volume, both located in the fluid flow domain. Dividing through by the fluid density at standard conditions \( \rho_\text{s} \), then expanding the right-hand-side (RHS) of Eq. (1) yields:

\[
\begin{align*}
V \cdot \left( \frac{k}{\mu B} (V \cdot \nabla)^2 z \right) &= \left( \phi \frac{d}{d\rho} \left( \frac{1}{\rho B} \right) + \phi \frac{c_B}{B} \right) \frac{\partial \rho}{\partial t} + \frac{q}{\rho B} \\
B &= \frac{\rho c}{\rho} \\
m &= \rho q B \\
c &= \left( \phi \frac{d}{d\rho} \left( \frac{1}{\rho B} \right) + \phi \frac{c_B}{B} \right)
\end{align*}
\]

Eq. (2) is the domain solution, \( q \) is the sink/source rate at standard conditions. \( k \) is the domain permeability, a constitutive relation between the pressure gradient and the flow velocity distribution in the domain. \( k \) can be represented by a tensor or assumed isotropic. The PDE is linear in Eq. (2) if the fluid’s viscosity \( \mu \) and formation volume factor \( B \) are assumed constant, and the derivative term \( \frac{d}{d\rho} \left( \frac{1}{\rho B} \right) \) is represented by a constant fluid compressibility; this is the assumption for a slightly compressible subsurface flow, e.g., oil flow. However, for a highly compressible subsurface flow, the fluid properties and their derivatives are not constant but functions of pressure. The PDE for highly compressible subsurface flow has variables and derivatives, which are strongly pressure-dependent, resulting in a nonlinear PDE and composite (deeper) neural network architecture. Solving Eq. (2) requires specifying the boundary conditions—Dirichlet and Neumann boundary conditions (Abou-kassem et al., 2006). Combining the domain and boundary equations yield the physics-based losses. In this work, the physics-based losses are evaluated by computing the space- and time-dependent first-order and second-order partial derivatives in Eq. (2) using FDM. AD, a superior technique in shallow computational graph architectures is used in computing the pressure-dependent (fluid) derivatives. The time-discretization errors are reduced by the timestep ANN. Table 1 shows the different differentiation schemes used in evaluation of the derivatives.

2.2. Domain equation

A 2D reservoir with uniform thickness, anisotropic rock properties, zero dip and a highly compressible fluid is considered. The reservoir is penetrated by a group of wells.

Evaluating the domain PDEs by FDM involve the specification of fixed finite dimensions—\( \Delta x \), \( \Delta y \), \( \Delta z \) and a timestep \( \Delta t \). For the 2D reservoir model, \( \Delta x \) is the reservoir thickness \( h \). A schematic of the discretized reservoir is shown in Fig. 2. The 2D domain loss \( \text{MSE}_2 \) for a given timestep is as follows:

\[
\text{MSE}_2 = \lambda \Omega \frac{1}{N_x N_y} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \left( e_{ij} \right)^2
\]

\( \lambda \) is a hyperparameter, trade-off between the domain and boundary losses. The domain loss is a measure of how well the domain values are predicted at each timestep. The boundary loss is a measure of how well the boundary values are predicted at each timestep.
\[ \varepsilon_{\omega,j} = \Delta V_{i,j} \left[ \left( \frac{1}{\Delta x} \right)_{i,j} \left( \frac{\partial k_s}{\partial B \Delta x} \right)_{i,j} + \left( \frac{1}{\Delta x} \right)_{i,j} \left( \frac{\partial k_s}{\partial B \Delta x} \right)_{i+1,j} \right] - \left( \frac{1}{\Delta y} \right)_{i,j} \left( \frac{\partial k_s}{\partial B \Delta y} \right)_{i,j+1} - \left( \frac{1}{\Delta y} \right)_{i,j} \left( \frac{\partial k_s}{\partial B \Delta y} \right)_{i+1,j+1} - \left( \frac{1}{\Delta y} \right)_{i,j} \left( \frac{\partial k_s}{\partial B \Delta y} \right)_{i,j+1} + \left( \frac{1}{\Delta y} \right)_{i,j} \left( \frac{\partial k_s}{\partial B \Delta y} \right)_{i+1,j+1} + \varepsilon_{\omega,j} \right] \] (7)

\[ \Delta V_{i,j} = \Delta \chi_{i,j} \times \Delta x_{i,j} \times \Delta z_{i,j} \] (8)

\[ \varepsilon_{n_{i,j}} = \Delta V_{i,j} \left( \frac{\partial \phi_{n_{i,j}}}{\partial \phi_{n_{i,j}}} \right) \left[ \frac{2 \chi_{i,j} + \Delta t_{i,j} \rho_{n_{i,j}} + \Delta t_{i,j} \rho_{n_{i,j}+12} - (\Delta t_{i,j} + \Delta t_{i,j+1}) \rho_{n_{i,j}}}{(\Delta t_{i,j} + \Delta t_{i,j+1})} \right] \] (9)

\[ \lambda_{ij} \] is the regularization weight for the domain loss, \( \Delta V_{ij} \) is the grid block bulk volume, \( N_x \times N_y \) is the number of collocation points or grid blocks in the flow domain and \( \varepsilon_{n_{i,j}} \) is the time-discretization round-off and truncation errors. \( \sigma \) and \( \delta \) are field unit conversion constants with values of 0.001127 and 5.615, respectively.

Evaluating Eq. (7) at an unknown time level \( \zeta = n + 1 \) is the centroid-in-distance, backward-in-time formulation. This is a stable finite difference scheme as the associated numerical errors do not grow when the computations are carried forward (Peaceman, 2000). \( \zeta = n + 1 \) is used this paper. Eq. (7) evaluation requires specifying a small advancement (timestep) and recomputing the AI-based graph. The evaluation, a forward pass, is performed twice to obtain the training losses. This is a less expensive process when the network computational graph is static—i.e., the flow of data, its size and function evaluations type are known before the start of the training. The nonlinearities introduced by the compressibility, viscosity and formation volume factor terms are locally linearized in space. The local linearization approximates the properties at the grid block edges and time level. Different linearization schemes have been used to evaluate the pressure-dependent terms, e.g., single-point upstream weighting, average function value weighting, and average pressure value weighting (Abou-kassem et al., 2006). In this work, the average function value weighting is used for local linearization. The average function value weighting \( f_{p_{i+1/2}} \) linearized implicitly is given by:

\[ f_{p_{i+1/2}}^{n+1} = \frac{1}{2} \left( f_{p_{i+1}}^{n+1} + f_{p_{i}}^{n+1} \right) \] (10)

The pressure-dependent term linearized in space is given by:

\[ \left( \frac{1}{\mu B} \right)_{i+1/2,j} = \left( \frac{1}{\mu} \right)_{i+1/2,j} \left( \frac{1}{B} \right)_{i+1/2,j} \] (11)

\[ \left( \frac{1}{\mu} \right)_{i+1/2,j} \left( \frac{1}{B} \right)_{i+1/2,j} = \left[ \left( \frac{1}{\mu} \right)_{i,j} + \left( \frac{1}{\mu} \right)_{i+1,j} \right] \left[ \left( \frac{1}{B} \right)_{i,j} + \left( \frac{1}{B} \right)_{i+1,j} \right] \] (12)

The directional x-transmissibility across a grid face is given by:

\[ \left( \frac{\partial \phi_{n_{i+1/2,j}}}{\partial \phi_{n_{i+1/2,j}}} \right) = \frac{\partial \phi_{n_{i+1/2,j}}}{\partial \phi_{n_{i+1/2,j}}} \left( \frac{1}{\mu B} \right)_{i+1/2,j} \] (13)

\[ \left( \frac{\partial \phi_{n_{i+1/2,j}}}{\partial \phi_{n_{i+1/2,j}}} \right) = \frac{2(k_s)_{i+1/2,j}}{(k_s)_{i+1/2,j} + (k_s)_{i+1/2,j}} \] (14)

Eqs. (11)–(14) can be applied to compute the directional y-terms by changing the parameter axis. The compressibility term is evaluated as:

\[ c(p_{n_{i,j}}) = (1 - S_w) \left[ \phi_{n_{i,j}} \frac{d}{dp} \left( \frac{1}{B} \right)_{i,j} \left| p_{n_{i,j}} \right| + \phi_{n_{i,j}} C_R \left( \frac{1}{B} \right)_{i,j} \right] \] (15)

\[ \frac{d}{dp} \left( \frac{1}{B} \right)_{i,j} \left| p_{n_{i,j}} \right| \] is defined as the chord slope; it affects the convergence of numerical simulations and can also affect physics-based gradients propagated by a machine learning model. The chord slope is computed by polyharmonic spline layers in the fluid property module. These layers have radial basis functions as their neurons, rapidly pre-trained to observed fluid properties and are highly memory-efficient.
The pretraining maps smooth functional models to discrete pressure, volume, and temperature (PVT) tables. These tables are noisy in most practical cases, and if obtained from equation of state (EOS) models, in some cases, these models are not simple and can diverge or give erroneous results. The memory efficiency of this layer allows for AD, thereby obtaining accurate representations of the chord slope.

2.3. Neumann and Dirichlet boundary conditions

The Neumann boundary condition (NBC) specifies the normal derivative or flux of the potential function \( f_p \), i.e., the PDE on the boundary \( \partial \Omega \), to be a constant or zero (Mazumder, 2016; Venkateshan and Swaminathan, 2014). NBC can be specified either at the inner boundaries or at the outer edges. In a Neumann inner boundary condition, the well (or group of wells) is under a constant pressure at the sand face, and Darcy law (steady-state) can be applied. Similarly, in a Neumann outer boundary condition specification, the outer boundary is under a certain or zero flux at any time. The Dirichlet boundary condition (DBC) specifies the value that the solution function \( f \) to the PDE must have on the boundary \( \partial \Omega \). DBC can be at the inner boundaries or at the outer boundaries. In a Dirichlet inner boundary condition, the well (or group of wells) is under constant pressure at the sand face at any time. Similarly, in a Dirichlet outer boundary condition, the outer boundary is under constant pressure at any instance of time, e.g., an aquifer or adjoining compartment (Idorenyin, 2016). Under steady-state, the net flow across the inner boundary is zero. The finite difference for the well grid block in steady-state is as follows:

\[
(q_{i-1/2,j} - q_{i+1/2,j}) + (q_{j-1/2,j} - q_{j+1/2,j}) = 0
\]  

Eq. (16) can be evaluated at the grid block as:

\[
\epsilon_{IBC_{i,j}} = \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta y}{\Delta x} p_{i,j}^{\text{IBC}} + \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta x}{\Delta y} p_{i,j}^{\text{IBC}} - \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta x}{\Delta y} p_{i,j+1}^{\text{IBC}}
\]

\[
+ \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta y}{\Delta x} p_{i,j}^{\text{IBC}} - \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta y}{\Delta x} p_{i+1,j}^{\text{IBC}} + \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta y}{\Delta x} p_{i,j-1}^{\text{IBC}} - \left( \frac{k_j h}{\mu B} \right)^{\zeta} \frac{\Delta y}{\Delta x} p_{i,j}^{\text{IBC}}
\]

Eq. (17), expressed as the inner boundary condition loss MSE_{IBC}, is as follows:

\[
\text{MSE}_{IBC} = I \cdot \lambda_{IBC} \sum_{j \in N_j} \sum_{i \in N_i} \left| \epsilon_{IBC_{i,j}} \right|^2
\]  

\[
I \text{ is a binary matrix having same dimension as the domain; it specifies the grid block locations penetrated by group of wells. The binary matrix allows for multiple well connections existing in the grid domain. The rate } q_{i,j}, \text{ is a dataset label, which does not require an expensive process to generate. For a constant rate well, } q_{i,j}, \text{ is the initial rate of the well. Eq. (17) can be solved to relate the rate, as the difference between the well grid block (centroid) pressure and a flowing bottom hole pressure (Peaceman, 1978).}
\]

\[
q_{i,j+1}^{\text{eff}} = 2 \pi k_j h \phi_j \sigma \left[ \ln \left( \frac{r_o}{r_w} \right) \right]^{-1} \left( \frac{1}{\mu B} \right)^{\zeta} \left( p_{i,j}^{\text{eff}} - p_{i,j}^{\text{eff}} \right)
\]  

Eq. (19) can be rearranged to evaluate the flowing bottom hole pressure for a constant rate system as:

\[
p_{i,j}^{\text{eff}} = p_{i,j}^{\text{eff}} + \frac{q_{i,j+1}^{\text{eff}}}{2 \pi k_j h \phi_j \sigma} \ln \left( \frac{r_o}{r_w} \right) \left[ \ln \left( \frac{1}{\mu B} \right) \right]^{-1} \zeta
\]  

\[
r_o \text{ is the equivalent radius of the well block—the radius at which the steady-state flowing pressure for the actual well is equal to the numerically calculated pressure for the well block. The equivalent radius for an anisotropic system is given by Peaceman (1983)}
\]

\[
r_o = 0.28 \left( \frac{k_v}{k_h} \right)^{1/4} \left( \frac{k_h}{k_v} \right)^{1/4} \left( \Delta x^2 + \left( \frac{k_h}{k_v} \right) \Delta y^2 \right)^{1/2}
\]  

The outer boundary residual error \( \epsilon_{IBC} \), if any can be computed by expressing the flow across the outer boundaries using auxiliary grid blocks as shown in Fig. 3. For a no-flow outer boundary, the auxiliary (or image) grid blocks have centred pressures equal to that of the real grid blocks located around the boundary. The outer boundary residual error using the auxiliary grid blocks is a subset of the domain loss.

2.4. Initial condition

This is a special case of the Dirichlet boundary condition, specifying the solution at a time boundary of zero and for all spatial locations. The AI-based surrogate requires supervision for learning the initial condition (IC) loss. The dataset label for this loss is the initial pressure \( p_{i,j}^{\text{IC}} \), which can be obtained without an expensive process. AI-based learning on this loss conditions the network’s trainable weights to recognize the initial state of the system. The IC residual error is given by:

\[
\epsilon_{IBC} = p_{i,j}^{\text{IC}} - f_g \sum_{i,j} C_{i,j}
\]  

\[
p_{i,j}^{\text{IC}} = p_i - \delta h f_g (x, y, k, t) \implies p_i + 70.69 B_M k \phi \left( \frac{948 \mu c r_i^2}{k h} \right) \text{ Hard enforcement layer}
\]

\[
f_g (x, y, k, t) \text{ is the output from the AI-Based SRM, } \delta \text{ is a scaling function based on a constantly changing input variable. A permeability-time dependent scaling function is used and is given by:}
\]

\[
\delta = \left( \frac{t - t_{\text{min}}}{t_{\text{max}} - t_{\text{min}}} \right) \theta_t
\]  

\[
t_{\text{min}} \text{ and } t_{\text{max}} \text{ are the input normalization limits, which are set to } -1 \text{ and } 1, \text{ respectively. The updated pressure output in Eq. (23), ensures the initial condition loss } \epsilon_{IBC} \text{ is always zero at } t = 0 \text{. } \theta_t \text{ can be set as } 0.5 \text{ or made a trainable parameter in the layer. The parameter } \theta_t \text{ when made trainable is estimated by the CEDDM optimizer updates, subject to constraint: } \{ \theta_t \in \mathbb{R} | 0 < \theta_t \leq 1 \}. \text{ This constraint is necessary to prevent exploding gradients associated with power functions in deep learning. The effects of the trainable and non trainable states of } \theta_t \text{ are shown in the discussion section.}
\]

2.5. Tank material balance equation

A material balance equation is adapted as a physics-based loss to accelerate the learning of the reservoir physics by the AI-based surrogate. Setting the divergence term in Eq. (2), \( \nabla \cdot \left[ \frac{k}{\mu B} \left( \nabla p - \gamma \nabla z \right) \right] = 0 \), results in a tank model. Material balance checks are used in numerical simulators to ensure the net tank accumulation due to a proposed rate
at the sink (or source), for a given timestep, is conserved. The rate of accu-

Fig. 3. Domain and solution grid for 2D model.

The space-discretization numerical effects cannot be eliminated, but they can be reduced by proper grid selection. A cut-off resolution can be established where successive increments have little effect on the physical (real) domain. The numerical errors, as artefacts, diffuse similarly as the pressure transient. The magnitude of the numerical dispersion is affected by the finite difference scheme, pressure diffusivity (mobility- storativity ratio), space and time steps (Peaceman, 2000; Zhu et al., 2019). When the physical diffusivity is small, e.g., in low permeability systems, the finite difference solutions can contain significant numerical errors due to numerical diffusion having a greater effect compared to the physical diffusivity. The AI-based surrogate will have numerical errors inherent in its learning when FDM are used to solve the PDE. These numerical errors cannot be eliminated but can be reduced using smaller discretization steps, a stable finite difference scheme in which the numerical errors do not grow as the computations are carried forward in time or mobility weighting in two-phase flow.

Grid sensitivity can be performed as a pretraining operation, using a numerical simulator, to select an optimum grid size with reduced numerical effects. Fig. 4 shows the numerical effects on the cumulative gas production and average pressure for a 2900 ft \( \times \) 2900 ft, dry gas subsurface domain, which is simulated for different grid resolutions and different uniform permeability values. Each grid resolution corresponds to the number of grid blocks in the domain: 9, 19, 29, 39, 49, and 59. The highest and lowest resolutions correspond to grid block sizes 49.15 ft and 322.22 ft respectively.

Successive increase of the grid resolutions cause deviations in their simulated responses; these deviations are more pronounced in the low permeability systems (Fig. 4A and B) than in the high one, where it is almost negligible (Fig. 4C). In Fig. 4, considering the date 01-June and incremental recoveries between the lowest and highest resolutions, the high permeability case (\( k = 300 \) mD) yields incremental gas production and recovery of 1700 Mscf and 0.0623% respectively. As \( k \) is decreased by a factor of 100, the incremental values become 135 000 Mscf and 4.95% respectively, which is greater than those of the high permeability system. Further reduction by a factor of 100 reduces the incremental values to 11 680 Mscf and 0.428% respectively. However, it is still greater than those of the highest permeability system (Fig. 5). The reduced incremental quantities are due to very low diffusivity, which has been affected by the low value of the permeability, i.e., \( k = 0.03 \) mD.

The space-discretization numerical effects cannot be eliminated, but they can be reduced by proper grid selection. A cut-off resolution can be selected for the low permeability cases; this can be lowered for the high permeability case. However, the cut-off affects the training time. High cut-off resolutions take longer training times than low ones; this is due to increased DL operations, such as movements of convolution filters, in the high cut-off resolutions compared to the low ones. The optimal grid size will be a trade-off between the space-discretization numerical errors and the cost of training the AI-based surrogate.

The numerical errors due to time discretization can be minimized by learning an average advancement (timestep) at each training time.
that the set of round-off errors \( \epsilon \) follows:

\[
\text{The round-off and truncation errors due to time discretization are as error using the backward Euler in time is a first-order error} \quad \Delta t^2
\]

\[
\text{puting)} \quad \Delta t^2
\]

\[
\text{of accumulation of mass} \quad m
\]

\[
\text{The backward Euler approximation for the rate} \quad \text{point. The learned timestep can effectively control the rate of accumulation in the domain. The backward Euler approximation for the rate of accumulation of mass} \quad m \text{ yields:}
\]

\[
\left( \frac{dm}{dt} \right)_{i,j}^{n+1} = \frac{m_{i,j}^{n+1} - m_{i,j}^n}{\Delta t} + O(\Delta t)
\]

\[
(27)
\]

The total error in Eq. (27) consists of the round-off errors (computing) \( \epsilon_r \) and the truncation error (discretization) \( \epsilon_t \). The truncation error using the backward Euler in time is a first-order error \( O(\Delta t) \). The round-off and truncation errors due to time discretization are as follows:

\[
\epsilon_{r,i,j} = \frac{e_{r,i,j}^{n+1} - e_{r,i,j}^n}{\Delta t}
\]

\[
\epsilon_{t,i,j} = \frac{\Delta t}{2} \left( \frac{d^2m}{dt^2} \right)_i^j \epsilon
\]

\[
(28)
\]

\[
(29)
\]

\( \epsilon \) is the truncation error bound in the time domain. If it is assumed that the set of round-off errors \( \{e_{r,i,j}^{n+1}, e_{r,i,j}^n\} \) is of an absolute magnitude \( \epsilon \), then the total error term \( \epsilon_{r,t} \) is given by:

\[
\epsilon_{r,t} = \frac{2\epsilon}{\Delta t} + \frac{\Delta t}{2} \left( \frac{d^2m}{dt^2} \right)_i^j \epsilon
\]

\[
(30)
\]

The non-physics-based loss is computed as the Euclidean norm \( ||\theta||_2^2 \) using parameters \( \theta \) in the model (Loshchilov and Hutter, 2019a). The
square of the Euclidean norm is as follows:

$$\frac{\lambda'}{2} \| \theta \|^2 = \frac{\lambda'}{2} \sum_{i=1}^{N_\theta} \theta_i^2$$  \hspace{1cm} (36)

\( \theta \) is a set of model parameters (i.e., weights and biases) of the AI-based SRM. The non-physics-based loss, known as the ridge penalty or \( L_2 \) regularization reduces the effect of multicollinearity that could occur between model parameters in regression models (Golam Kibria and Lukman, 2020; Hoerl and Kennard, 1970). In highly correlated model parameters, the variance is high and can yield unsatisfactory predictions. The variance determines the model’s performance when a dataset outside the training dataset is used. The ridge penalty is a non-zero value, which shrinks the neural network trainable parameters and also prevents exploding gradients that can occur during training. In linear regression problems, the ridge hyperparameter \( \lambda' \) must be tuned as it can adversely affect the optimization process if its magnitude becomes too large; usually, a small or decaying value of this weight is applied. The decaying weight, \( \lambda' = \lambda' \alpha \), inline a first-order optimizer can be used in place of the ridge hyperparameter. Fig. 6 shows the effect of using the non-physics-based regularization.

2.8. Total loss

The total loss is a weighted average of the physics-based and non-physics-based losses. The weighted physics-based loss \( \text{MSE}_{\text{PHY}} \) is given by:

$$\text{MSE}_{\text{PHY}} = \text{MSE}_{\text{ID}} + \text{MSE}_{\text{vl}} + \text{MSE}_{\text{vl,BBC}} + \text{MSE}_{\text{MB}}$$  \hspace{1cm} (37)

Eq. (37) is the summation of all soft constraints to be minimized by the optimizer. The initial condition loss \( \text{MSE}_{\text{IC}} \) is not included in Eq. (37) as it is fully enforced in the neural network architecture by the hard enforcement layer. The weighted non-physics-based loss Eq. (36) is added to the weighted physics-based loss to give the total loss \( \text{MSE}_T \):

$$\text{MSE}_T = \text{MSE}_{\text{PHY}} + \text{MSE}_n$$  \hspace{1cm} (38)

2.9. Artificial neural network development and training

The proposed architecture is a custom three-module ANN with each module trained independently. The custom modules are defined as follows:

1. A CEDNN, with a trainable hard-enforced output layer and encoder–decoder skip connections. This module determines the pressure solution at a given time.
2. A fully connected deep neural network with residual connections (ResNet) to determine the advancement (timestep) at a given time.
3. A pretrained fluid-property layer for prediction of pressure-dependent fluid properties.

In ML, \( \lambda' \) must be tuned as it can adversely affect the optimization process if its magnitude becomes too large; usually, a small or decaying value of this weight is applied. The decaying weight, \( \lambda' = \lambda' \alpha \), inline a first-order optimizer can be used in place of the ridge hyperparameter. The CEDNN is preferred as the main network for learning the pressure solution, as its convolutional layers can capture the local dependencies between neighbouring grid blocks during feature extraction. The CEDNN module can be compared to the standard U-Net architecture as it encodes latent variables by convolution and decodes outputs by transposed convolution. In contrast, the internal architecture of the CEDNN module permits real-squared dimensions, e.g., a discretized domain with a centrally located sink or source requires odd grid dimensions and padding operations at convolution operations; this can be achieved, without any further transformation, using the custom-built CEDNN module. A comparison of the standard U-Net and the CEDNN module is shown in Table 2.

The CEDNN has some features similar to the encoder–decoder structure in Wang et al. (2021); however, it uses a set of skip connections for transfer learning, a trainable hard enforcement technique and is coupled with other trainable network modules for effective learning. A schematic of the custom three-module AI-based architecture is shown in Fig. 7; the details of the network are given in Table 3. The described network module in Table 3 has a 2D input dimension of 39 x 39.
A schematic of the custom three-module neural network is shown in Fig. 7.

In Table 2, the CEDNN uses a real square domain shape as against the power-of-two used in the standard U-Net architectures. The CEDNN module uses strided convolution during encoding. Strided convolution operation may perform very well or slightly outperforms the non-strided convolution and subsequent downsampling (averaging or maxpooling) operations (Springenberg et al., 2015). Also, the absence of the downsampling operation in a strided convolution operation can improve its memory efficiency. The skip connections, which allow for transfer learning between an encoder–decoder pair, are added rather than concatenated. The addition is a layer-layer arithmetic operation, which does not change the decoder’s inner dimension whereas the concatenation operation causes an increase in the decoder’s inner dimension, which has to be resized back by linear projection. The concatenation and then linear projection can be computationally expensive for the physics-based training. The addition of skip connection layers has yielded excellent results in the standard ResNet (He et al., 2016).

In Table 3, the encoder with layers: 1, 2, 3 and 4 performs convolution operations to encode latent features from the main input. Encoding is done progressively by the strided convolution operation; it involves moving a feature detector (filter or kernel) across the domain with strides greater than one. The 2D input tensor of the convolution operation follows the shape definition: [batch size, height, width, depth]. For the 2D reservoir, the second and third indexes (i.e., height and width) of the main input tensor represent the number of grid blocks in the $y$ and $x$ axes respectively. The depth describes the number of filters required for each convolution operation. This convention is followed throughout this paper. The expression between the input and output spatial dimensions for each convolutional layer is given by:

$$L_{e,o} = \left( L_{e,i} - f + 2P \right) \div s + 1$$

(39)

$L_{e,i}$ is the input length (height or width) of the convolutional layer; $f$ is a kernel filter used to extract the features from the input; $s$ is the...
stride, indicating how many steps the filter is moved during convolution; \( P_d \) is the input padding, indicating the number of grid blocks to be added at the boundary or edges during the convolution operation. Each convolution operation reduces the output spatial dimension (height and width) and increases the number of output filters (depth). The depth of each convolution operation \( d_i \) is increased progressively as follows:

\[
d_i = 1.5^i d_0
\]

\( d_0 \) is the depth of the first convolution operation and \( d \) is the encoding depth sequence. The CEDNN module has a depth sequence: \((d_i)_{i=0}^{5} = (0, 1, 2, 3, 3)\). The output from the encoder has a smaller spatial dimension and larger depth representation (latent) of the main input. The latent representation is passed on to the bottle neck layer 5 for each convolution operation \( \sigma \) and concatenation (axis = -1); output = (39, 39, 2)

<table>
<thead>
<tr>
<th>Module</th>
<th>Layers</th>
<th>Inner architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Input (features); permeability, time; concatenation (axis = -1); output = (39, 39, 2)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Convolution 1: Kernel filters = 32; kernel size = (3, 3); stride = 1; skip connection 1; activation function = Swish; output shape = (37, 37, 32); zero padding; output shape = (39, 39, 32)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Convolution 2: kernel filters = 48; kernel size = (5, 5); stride = 2; skip connection 2; activation function = Swish; output shape = (18, 18, 48); zero padding; output shape = (20, 20, 48)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Convolution 3: kernel filters = 72; kernel size = (5, 5); stride = 2, skip connection 3; activation function = Swish; output shape = (8, 8, 72); zero padding; output shape = (10, 10, 72)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Convolution 4: kernel filters = 108; kernel size = (3, 3); stride = 1; activation function = Swish; output shape = (4, 4, 108)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Dense latent layer: neuron size = 128; activation function = none/Mish; output shape = (4, 4, 128)</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Transposed convolution 1: kernel filters = 72; kernel size = (3, 3); stride = 2; add skip connection 3; activation function = Swish; output shape = (9, 9, 72)</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Transposed convolution 2: kernel filters = 48; kernel size = (3, 3); stride = 2; add skip connection 2; activation function = Swish; output shape = (19, 19, 48)</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Transposed convolution 3: kernel filters = 32; kernel size = (3, 3); stride = 2; add skip connection 1; activation function = Swish; output shape = (39, 39, 32)</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Constant padding by reflection; output shape = (41, 41, 1); post convolution 1: kernel filters = 32; kernel size = (3, 3); stride = 1; activation function = Swish; output shape = (39, 39, 32)</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Dense layer: neuron size = 32; main input projection; activation function = Swish; dense layer and projected main input addition; output shape = (39, 39, 1)</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Convolution 2: kernel filter = 1; kernel size = (1, 1); stride = 1; activation = none; output shape = (39, 39, 1)</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Hard enforcement: inputs = (permeability, time, pressure); enforcing function = power function; exponent initial value = 0.5; output shape = (39, 39, 1)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Fluid property module: inputs = pressure; inputs reshape; polyharmonic spline radial basis = 1 or 2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Timestep module: inputs = (permeability, time); layer architecture = ResNet; hidden layer activation function = Swish; number of residual blocks = 4; number of dense layers per residual block = 1; final output activation = non-parametric linearly scaled hyperbolic tangent (LISH)</td>
</tr>
</tbody>
</table>

The network’s initial weights are obtained using the Xavier (Glorot) normal, while its biases are initialized to zero. Xavier normal draws an initial weight set from a truncated normal distribution with a mean of 0 and a standard deviation, \( \sigma = \sqrt{2/(N_{in} + N_{out})} \), where \( N_{in} \) is the number of trainable hidden layers and \( N_{out} \) is the number of output units. This can be useful during the weighting of the timestep, ensuring that gradients associated with negative values are propagated.
of input nodes to a given layer and \( N_{x_{in}} \) is the number of output nodes of the same layer. To allow for standard comparison of regularization effects, the Xavier normal initialization is obtained from a stateless (deterministic) pseudo-random algorithm, such that for a given seed, and for multiple initializations, the exact sequence of weight sets are generated. One hundred weight initializations are generated within five seconds, and at each instance, the global norm of the weight set is estimated. The accepted weight initialization is the weight set with the median norm.

The pressure output and its numerical derivatives from module 1, the fluid properties and their algorithmic derivatives from module 2, and the average timestep from module 3 are used to compute the physics-based losses. The AI-based model is tuned by equally weighting the non-physics-based losses. The AI-based model is tuned by equally weighting the physics-based losses using a weighting value of \( 1/N_{phys} \); setting a small weight for the non-physically-based loss, obtained either from a hyperparameter search, published literatures or experience of the analyst; using a trainable parameter in the hard enforcement layer; using a trainable parameter in the hard enforcement layer and a trainable timestep module. A first-order optimizer is used for training the network; the adaptive estimate of moments with weight decay (AdamW), wrapped with a multi-optimizer (Jin et al., 2016), is used for updating the model parameters during each step. One step corresponds to the forward and backward passes of a batched dataset. The weight decay of the optimizer is only applied to the non-physically-based loss. The first-order optimizer iteration update with a decaying non-physically-based loss \( \lambda \) is given by:

\[
\theta_{i+1} = \left( 1 - \lambda \eta \right) \theta_i - \alpha \nabla f_i(\theta_i)
\]

The learning rate relationship is given by:

\[
\alpha = \alpha_0 \left( 1 - \lambda \eta \right)^{\frac{i}{n_T n_R}}
\]

The non-physically-based loss weight decays similarly as the learning rate:

\[
\lambda_\eta = \lambda_\eta_0 \left( 1 - \lambda \eta \right)^{\frac{i}{n_T n_R}}
\]

Using batched training, the number of iterations per epoch \( n_T \) is given by:

\[
n_T = \frac{n_P}{n_B}
\]

Number of iterations for the training dataset \( n_T \) is given by:

\[
n_T = n_{f} \times n_T
\]

\( \theta \) = model parameters (weights and biases); \( \alpha \) = learning rate; \( \lambda_\eta \) = weight decay hyperparameter; \( \lambda_\eta \) = decay rate constant; \( i \) = iteration counter \((0 \leq i \leq n_T); n_B = \) epoch decay step; \( n_f = \) number of epochs; \( n_P = \) number of rows in training data; \( n_B = \) number of rows per training batch.

As deep neural network architectures are well-known for their high non-convexity (Chen and Wang, 2014), multiple minima and multiple maxima could exist in the loss manifold. Therefore, using model parameters just at the end of training might not be an optimal solution. The optimized model parameters are selected from a collection of model parameters watched during the last 25% of the total epochs. The validation loss is taken as the weighted domain and boundary losses, including the material balance loss.

2.10. Stochastic generation of the permeability field

The log-normal distribution is assumed to be the basic type of distribution of geological variables like permeability in a naturally occurring permeable medium. The log-normal distribution is the probability distribution of a random variable \( x \), whose logarithm \( \ln x \), follows a normal distribution. The standard normal variate, \( Z \), is given by:

\[
Z = \frac{x - \mu\_x}{\sigma\_x}
\]

The standard normal can be expressed for the log-normal distribution as:

\[
Z = \frac{\ln x - \mu\_lnx}{\sigma\_lnx}
\]

The mean and standard deviation of the logarithm, \( \mu\_lnx \) and \( \sigma\_lnx \), is related to the mean and standard deviation, \( \mu\_x \) and \( \sigma\_x \), by the expression:

\[
\mu\_lnx = \ln \mu\_x - \frac{1}{2} \sigma\_x^2
\]

\[
\sigma\_lnx^2 = \ln \left( \frac{\sigma\_x^2}{\mu\_x^2} \right)^2 + 1
\]

\[
\frac{\sigma\_x}{\mu\_x} = \text{Coefficient of variation}
\]

One of the methods of generating stochastic realization of a property field is using the Karhunen-Loeve expansion (KLE). KLE represents a stochastic process as an infinite linear combination of orthogonal functions. In the KLE expansion, a stochastic process \( \{X(t), t \in T \} \) is represented via a sequence of independent simple random variables \( \{ \xi_i, i \in N \} \). The corresponding kernel is defined as: \( K(s, t) = \text{cov}(X(t), X(s)) \) in the reproducing Hilbert kernel space. Assuming \( \{ \phi(t) \} \) is a series of basis (orthogonal) functions derived from certain integral equations, the stochastic process is given by:

\[
X(t) = \sum_{i=1}^{\infty} \sqrt{\lambda_i} \xi_i \phi(t)
\]

The independent random variables are given by:

\[
\xi_i = \frac{1}{\sqrt{\lambda_i}} \int X(t) \phi(t) \delta t
\]

If \( X(t) = \sum_{i=1}^{\infty} \sqrt{\lambda_i} \xi_i \phi(t) \), where \( \{ \xi_i \} \) is identically independent distributed with a zero mean and unit variance, then the integral of the kernel is given by:

\[
\int K(s, t) \phi(t) \delta t = \lambda_s \phi(s)
\]

Where \( \{ \phi(s), i \in N \} \) is the set of eigenfunction and \( \{ \lambda_i, i \in N \} \) is the set of eigenvalues.

Further reading on the KLE can be found in the following literatures (Lu and Zhang, 2004; Wang, 2008; Wang et al., 2021). The KLE is used in this work to generate stochastic processes from a prescribed covariance function. The log-transformed stochastic field has the following statistical information:

\[
K(s, t) = C\_lnk(\alpha\_1, \alpha\_2) = \sigma\_lnk^2 \exp \left[ -\frac{(|x\_1 - x\_2|}{\ell\_x} + |y\_1 - y\_2|}{\ell\_y} \right]
\]

\( C\_lnk = \) covariance function

\( \mu\_lnx = \) normally distributed mean of the log-normal permeability, \( k \)

\( \alpha = \) flattened spatial locations in the x and y directions; \( \alpha = [(x\_1, y\_1), (x\_2, y\_2)] \)

\( \ell = \) correlation length in the x or y axis. In this work, \( \ell\_x = \ell\_y \)

\( \sigma\_lnx^2 = \) normally distributed standard deviation of the log-normal permeability, \( k \)

3. Case study

A dry gas reservoir with a dimension of 2900 ft × 2900 ft × 80 ft is considered. The model has a no-flow outer boundary with four wells operating under different production constraints of limiting rates and minimum bottomhole pressure, including a shut in as given in
3. Artificial intelligence-based surrogate reservoir model results and discussion

The accuracy of the predictions are evaluated using the percent error \( \epsilon_p \) for each grid block, and the relative \( L_1 \) and \( L_2 \) errors for the domain. The percent error \( \epsilon_p \) is given by:

\[
\epsilon_p = \frac{|p_{ij,t} - \hat{p}_{ij,t}|}{p_{ij,t}} \times 100
\]  

(55)

The relative \( L_1 \) and \( L_2 \) errors for the domain \( \Omega \) at a given time \( t \) are as follows:

\[
L_1 = \frac{\int_{\Omega} |p_{ij,t} - \hat{p}_{ij,t}|}{\int_{\Omega} p_{ij,t}}
\]  

(56)

\[
L_2 = \frac{\int_{\Omega} (p_{ij,t} - \hat{p}_{ij,t})^2}{\int_{\Omega} \hat{p}_{ij,t}^2}
\]  

(57)

Fig. 10A and B shows the comparison between two wells grid block predicted pressures and those obtained from the numerical simulator. Fig. 11 shows the 2D image (snapshots) of pressures using a test permeability realization \( (k = 2.39 \text{ mD}) \) at four prediction times \([260, 290, 310, 340] \) days. In Fig. 11, the third column shows the percent errors for the grid blocks.

In Fig. 10A and B and Fig. 11, the predictions of the tuned AI-based surrogate compared to those of the numerical simulator are in good agreement; the pressure distributions around the wells are predicted by the surrogate. The regularization weights of the tuned AI-based surrogate and relative error statistics using the \( 146 \times 44 \) test realizations are given in Table 5. Fig. 12 shows the relative \( L_1 \) and \( L_2 \) errors of the tuned AI-based surrogate using the test realizations.

In Table 5, the non-physics-based regularization weight is set as \( 0.0005 \alpha_0 \) after training on other limits: \( 0.0 \alpha_0 \) and \( 0.05 \alpha_0 \). The physics-based regularization terms are equally weighted, and the hard layer parameter and timesteps are made trainable. In Fig. 12A and B, the relative errors follow a lognormal distribution, i.e., the low-error class intervals are more dense than the high-error ones and the modal relative error tends towards zero. This is a good indication that the tuned model’s variance has been significantly reduced. A model with good generalization will have its error histogram positively skewed on a linear bin scale and bell-shaped on a logarithmic bin scale.

The effects of changing the non-physics-based regularization weight, material balance regularization weight, hard enforcement layer variable and timestep variability are studied. These are done by performing one-variable-at-a-time analysis. In studying these effects, the relative \( L_1 \) and \( L_2 \) errors are visualized using histograms with logarithmic binning, which allow for comparison over a wide range of relative errors.

4.1. Effect of changing the non-physics-based regularization weight

The non-physics-based regularization term can further constrain the grid block pressures to those of the numerical reservoir simulator. Fig. 13 shows the grid-predicted pressures and percent errors for different non-physics-based regularization weights using the test permeability realization \( (k = 2.39 \text{ mD}) \) at four prediction times \([260, 290, 310, 340] \). Three different non-physics-based regularization weights are considered: \( 0 \times 10^{-4} \alpha_0 \) (Fig. 13A), \( 5 \times 10^{-4} \alpha_0 \) (Fig. 13B) and \( 500 \times 10^{-4} \alpha_0 \) (Fig. 13C). In each subplot, the first and second columns represent the predicted and observed pressure responses and the third column shows the percent error of the predictions. Fig. 14 shows the relative \( L_1 \) and \( L_2 \) error histograms for the different non-physics-based regularization weights, and Table 6 shows their relative error statistics.

As shown in Figs. 13 and 14, increasing or decreasing the weight affects the predictions; the colour of third column in Fig. 13C darkens compared to those of Fig. 13A and B. The darkening effect is due to increased residual errors. The ranges of the relative error histograms first decrease, then increase as the non-physics-based regularization weight is increased from zero (Fig. 14). This effect is clearly visualized in the plot of the mean relative \( L_1 \) and \( L_2 \) errors for the different non-physics-based regularization weights (Fig. 15 and Table 6). There exists
Fig. 9. Matrix plot of some permeability realizations.

Fig. 10. Well grid blocks predicted and observed pressures using the 146 test permeability realizations including (A) [9,29,0] and (B) [29,9,0] (selected realizations are in bold).
Table 4
Input parameters used in training the AI-based surrogate reservoir model.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output hard enforcement</td>
<td>Power function with trainable exponent</td>
</tr>
<tr>
<td>Grid</td>
<td>39 × 39 × 1</td>
</tr>
<tr>
<td>Number of wells</td>
<td>4</td>
</tr>
<tr>
<td>Control mode</td>
<td>Rate with min bottomhole pressure</td>
</tr>
<tr>
<td>Maximum rate [0000, 10000, 5000, 5000] Mscf/D</td>
<td></td>
</tr>
<tr>
<td>Minimum bottomhole pressure [4000, 3000, 2000, 1000] psia</td>
<td></td>
</tr>
<tr>
<td>Shut-in</td>
<td>90 days—[152-242] days</td>
</tr>
<tr>
<td>Bottomhole pressure approximation</td>
<td>Dry gas pseudopressure integral</td>
</tr>
<tr>
<td>Initial pressure, $p_i$</td>
<td>5000 psia</td>
</tr>
<tr>
<td>Initial water saturation, $S_{wi}$</td>
<td>0.2</td>
</tr>
<tr>
<td>Permeability, $\mu$</td>
<td>$\mu = 3.0 \quad \sigma = 1.0$</td>
</tr>
<tr>
<td>Porosity, $\phi$</td>
<td>0.2</td>
</tr>
<tr>
<td>Total production time</td>
<td>365</td>
</tr>
<tr>
<td>Timestep, $\Delta t$</td>
<td>Determined by the ResNet</td>
</tr>
<tr>
<td>Number of time points for training and testing</td>
<td>48 and 48</td>
</tr>
<tr>
<td>Number of permeability samples for training and testing</td>
<td>54 and 146</td>
</tr>
<tr>
<td>Optimizer type</td>
<td>AdamW (decoupled weight decay regularization)</td>
</tr>
<tr>
<td>Training batch size, $n_b$</td>
<td>144</td>
</tr>
<tr>
<td>Learning rate schedule</td>
<td>Stepwise exponential decay</td>
</tr>
<tr>
<td>Initial learning rate—[pressure, timestep] modules</td>
<td>$[0.005, 0.0005]$</td>
</tr>
<tr>
<td>Physics-based weights—[(\lambda_d, \lambda_{\text{IBC}}, \lambda_{\text{MBC}})]</td>
<td>$[0.333, 0.333, 0.333]$</td>
</tr>
<tr>
<td>Initial non-physics-based weights—[pressure, timestep]</td>
<td>$[0.0005\eta_{\text{w}}, 0.]$</td>
</tr>
<tr>
<td>Decay constant</td>
<td>10%</td>
</tr>
<tr>
<td>Decay step</td>
<td>25 epochs</td>
</tr>
<tr>
<td>Epochs</td>
<td>600</td>
</tr>
<tr>
<td>Training time</td>
<td>~42 min (NVIDIA\textsuperscript{TM} RTX3060)–graph mode with accelerated linear algebra (XLA)</td>
</tr>
<tr>
<td>Backend computations datatype</td>
<td>Float 32</td>
</tr>
<tr>
<td>Training dataset dimensions</td>
<td>$54 \times 48 \rightarrow 18 \times 144$</td>
</tr>
<tr>
<td>Testing dataset dimensions</td>
<td>$146 \times 48$</td>
</tr>
</tbody>
</table>

Table 5
Regularization weights of the tuned AI-based surrogate and relative error statistics using the test realizations.

<table>
<thead>
<tr>
<th>Physics-based regularization weights</th>
<th>Non-physics-based regularization weight</th>
<th>Hard enforcement layer</th>
<th>Time</th>
<th>Relative $L_1$-error</th>
<th>Relative $L_2$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_d$</td>
<td>$\lambda_{\text{IBC}}$</td>
<td>$\lambda_{\text{MBC}}$</td>
<td>$\lambda_{\text{\phi}}$</td>
<td>$\Delta t$</td>
<td>$\mu_{L_1}$ x10^{-3}</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>5</td>
<td>Trainable</td>
<td>3.31</td>
</tr>
</tbody>
</table>

Table 6
Non-physics-based regularization weights and their relative error statistics using the test realizations.

<table>
<thead>
<tr>
<th>Physics-based regularization weights</th>
<th>Non-physics-based regularization weight</th>
<th>Hard enforcement layer</th>
<th>Time</th>
<th>Relative $L_1$-error</th>
<th>Relative $L_2$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_d$</td>
<td>$\lambda_{\text{IBC}}$</td>
<td>$\lambda_{\text{MBC}}$</td>
<td>$\lambda_{\text{\phi}}$</td>
<td>$\Delta t$</td>
<td>$\mu_{L_1}$ x10^{-3}</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0</td>
<td>Trainable</td>
<td>3.978</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>5</td>
<td>Trainable</td>
<td>3.310</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>25</td>
<td>Trainable</td>
<td>3.525</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>50</td>
<td>Trainable</td>
<td>4.868</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>500</td>
<td>Trainable</td>
<td>5.232</td>
</tr>
</tbody>
</table>
Table 7
Material balance regularization weights and their relative error statistics using the test realizations.

<table>
<thead>
<tr>
<th>Physics-based regularization weights</th>
<th>Non-physics-based regularization weight</th>
<th>Hard enforcement layer</th>
<th>Time</th>
<th>Relative $L_1$-error</th>
<th>Relative $L_2$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_d$</td>
<td>$\lambda_{\text{W}}$</td>
<td>$\lambda_{\text{MBC}}$</td>
<td>$\delta_0$</td>
<td>$\delta_t$</td>
<td>$\mu_{L_1}$ $\times 10^{-3}$</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.</td>
<td>5</td>
<td>Trainable</td>
<td>146.248</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>5</td>
<td>Trainable</td>
<td>3.310</td>
</tr>
</tbody>
</table>

Table 8
Hard enforcement techniques and their relative error statistics using the test realizations.

<table>
<thead>
<tr>
<th>Physics-based regularization weights</th>
<th>Non-physics-based regularization weight</th>
<th>Hard enforcement layer</th>
<th>Time</th>
<th>Relative $L_1$-error</th>
<th>Relative $L_2$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_d$</td>
<td>$\lambda_{\text{W}}$</td>
<td>$\lambda_{\text{MBC}}$</td>
<td>$\delta_0$</td>
<td>$\delta_t$</td>
<td>$\mu_{L_1}$ $\times 10^{-3}$</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>5</td>
<td>0.5</td>
<td>572.103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trainable</td>
<td>5</td>
<td>Trainable</td>
<td>3.310</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trainable</td>
<td>24.727</td>
<td>13.241</td>
<td>113.798</td>
</tr>
</tbody>
</table>

Fig. 11. Grid predicted and observed pressures, percent errors using the test permeability realization ($\bar{k} = 2.39$ mD) at four prediction times [260, 290, 310, 340] days.

Fig. 18 shows the total loss trends for the different material balance regularization weights. The learning progresses, as observed by the decreasing total loss, when the material balance regularization weight is set to zero; however, the updated model parameters are poorly constrained, yielding a high model variance at the end of training. This high model variance is reflected in the mismatch between the predicted well grid block pressures and those obtained from the numerical simulator (Fig. 19A). The addition of the material balance

an optimum for which the AI-based surrogate gives the lowest relative errors. This lies in the search space: $\lambda_0 < 0.01 \delta_0$.

In general cases, starting values of this weight can be obtained from published literatures, e.g., Loshchilov and Hutter (2019b) and

experience of the data analyst. The search can be performed between the extremes: $\lambda_0 \in \mathbb{R} | 0 < \lambda_0 \leq 0.1 \delta_0$.

In Fig. 10A and B, the tuned AI-based SRM is very reliable in predicting the flow realizations, i.e., grid block pressures for the unseen permeability realizations at the given training times. This is because the unseen permeability realizations follow a Gaussian distribution whose characteristic features can be discovered by the DL convolutional layers. The AI-based SRM can also be used in predicting the flow realizations using the given permeability realizations at unseen times, i.e., outside the training data time points; However, it can become less reliable when the prediction times are 25% more than the maximum training time. The optimum non-physics-based regularization weight improves the predictions in the unseen time region, up to 50% more than the maximum training time ($t_{\text{max}} = 360$ days), without relying on additional training time points in this region (Fig. 16B).

4.2. Effect of changing the material balance regularization weight

Two cases are considered: no material balance weight and when it is equally weighted with other physics-based losses. Fig. 17 shows the relative $L_1$ and $L_2$ error histograms for the different material balance regularization weights using the test realizations, and Table 7 shows their relative errors statistics.

As shown in Fig. 17, for the case of no material balance regularization weight ($\lambda_{\text{MBC}} = 0$), the relative $L_1$ and $L_2$ error histograms are negatively skewed having very high mean and standard deviation. The utilization of the material balance regularization weight yields bell-shaped (i.e., logarithmic binning) relative error histograms, their mean and standard deviation are significantly reduced compared to those of no regularization (Table 7). This weight accelerates the learning by updating the trainable parameters with additional gradients obtained from tank material balance.
Fig. 12. Relative (A) $L_1$ and (B) $L_2$ errors of the tuned AI-based surrogate using the test realizations.

Fig. 13. Grid predicted pressures and percent errors for different non-physics-based regularization weights: (A) $0 \times 10^{-4} \sigma_0$, (B) $5 \times 10^{-4} \sigma_0$ and (C) $500 \times 10^{-4} \sigma_0$, using the test permeability realization ($\bar{k} = 2.39$ mD) at four prediction times [260, 290, 310, 340].
V.C. Molokwu et al.

Fig. 14. Relative (A) $L_1$ and (B) $L_2$ error histograms for the different non-physics-based regularization weights using the test permeability realizations.

Table 9
Relative errors statistics for the test realization using the different timesteps.

<table>
<thead>
<tr>
<th>Physics-based regularization weights</th>
<th>Non-physics-based regularization weight</th>
<th>Hard enforcement layer</th>
<th>Time</th>
<th>Relative $L_1$-error</th>
<th>Relative $L_2$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda _\Omega =$</td>
<td>$\lambda _\Omega$</td>
<td>$\lambda _MBC$</td>
<td>$\delta$</td>
<td>$\Delta t$</td>
<td>$\mu_{L_1}$</td>
</tr>
<tr>
<td>$0.33$</td>
<td>$0.33$</td>
<td>$0.33$</td>
<td>$5$</td>
<td>Trainable</td>
<td>$18.921$</td>
</tr>
<tr>
<td>$1.0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$4.413$</td>
</tr>
<tr>
<td>$10.0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$4.665$</td>
</tr>
<tr>
<td>Trainable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$3.310$</td>
</tr>
</tbody>
</table>

Fig. 15. Mean relative $L_1$ and $L_2$ errors for the different non-physics-based regularization weights.

regularization weight reduces the model’s variance, yielding an optimal solution by the end of training (Fig. 19B). For an AI-based surrogate employing FDM, the material balance regularization term is important for validating the predictability of the surrogate.

The material balance regularization weight is a zero-dimension regularizer; it averages the spatial variation of grid block fluxes. This term cannot be used alone or jointly weighted with the inner boundary conditions as it results in a high model’s variance (Fig. 20A-C).

4.3. Effect of using the trainable hard enforcement layer

The hard enforcement layer, which has a trainable exponent $\theta_h$, is compared to cases of a nontrainable parameter and the nontrainable hard enforcement (Wang et al., 2021). Fig. 21 shows the relative $L_1$ and $L_2$ error histograms for the different hard enforcement techniques using the test realizations, and Table 8 shows their relative error statistics.

As shown in Fig. 21, the trainable hard enforcement layer results in bell-shaped histograms having error limits smaller than those of the nontrainable hard enforcement techniques. In Table 8, the trainable hard enforcement significantly reduces the mean and standard deviation of the relative $L_1$ and $L_2$ errors compared to those obtained by other nontrainable hard enforcement techniques. The trainable layer enforces
Fig. 16. Well grid block predicted and observed pressures for different non-physics-based regularization weights: (A) $0 \times 10^{-4} \alpha_0$, (B) $5 \times 10^{-4} \alpha_0$, and (C) $50 \times 10^{-4} \alpha_0$, using the 146 test permeability realizations at prediction times up to 1.5 times the maximum training time.

Fig. 17. Relative (A) $L_1$ and (B) $L_2$ error histograms for the different material balance regularization weights using the test realizations.

Fig. 18. Total loss trends for the different material balance regularization weights.

the initial condition whilst improving the predicted pressures. The trainable exponent initially set as 0.5778 is optimized to 0.159 at the end of the training.

4.4. Effect of using the trainable timestep module

The trainable timestep module allows for varying timesteps in the FDM. Three fixed timesteps: 0.1 day, 1 day, 10 days, and a trainable timestep are considered. The fixed timesteps are constant values for all sampled space–time points, and the trainable timestep is an averaged value that varies for all the sampled space–time points. Fig. 22 shows the relative $L_1$ and $L_2$ error histograms for the fixed and variable timesteps using the test realizations, and Table 9 shows their relative error statistics.

As shown in Fig. 22A and B, using the 0.1-day timestep and float32 computation causes the relative errors to be negatively skewed; the mean and standard deviation of the relative errors are higher compared to those of other timesteps (Table 9). As seen in Eq. (30), for small timesteps, the round-off errors due to machine precision becomes significant, hindering the learning process. The 10-day timestep can be suitable for learning, reducing the round-off errors but with likely growth of truncation errors. This is observed by the low relative error statistics but still greater than those of the 1-day and variable timesteps (Table 9). The 1-day timestep can reduce the truncation errors and improve the learning as shown by the lower mean relative errors compared to those of the 0.1-day and 10-day timesteps (Table 9);
Fig. 19. Well grid block [29,9,0] predicted and observed pressures for the different material balance regularization weights including (A) 0. and (B) 0.33.

Fig. 20. Well grid block [29,9,0] predicted and observed pressures for different combinations of the domain, inner boundary and material balance regularization weights including (A) $\lambda_\Omega = 0.0005; \lambda_{\partial \OmegaIBC} = 0.0005; \lambda_{MBC} = 0.0005; \lambda_\eta = 0.0005$; $a_0 = 0.005$;
(B) $\lambda_\Omega = 0.0005; \lambda_{\partial \OmegaIBC} = 0.0005; \lambda_{MBC} = 0.0005; \lambda_\eta = 0.0005$; $a_0 = 0.005$;
(C) $\lambda_\Omega = 0.5; \lambda_{\partial \OmegaIBC} = 0.5; \lambda_{MBC} = 0.5; \lambda_\eta = 0.0005$; $a_0 = 0.005$;
(D) $\lambda_\Omega = 0.0005; \lambda_{\partial \OmegaIBC} = 0.0005; \lambda_{MBC} = 0.0005; \lambda_\eta = 0.0005$; $a_0 = 0.005$. 
Fig. 21. Relative (A) $L_1$ and (B) $L_2$ error histograms for the different hard enforcement techniques using the test realizations.

Fig. 22. Relative (A) $L_1$ and (B) $L_2$ error histograms for the fixed and variable timesteps using the test realizations.
V.C. Molokwu et al.

Fig. 23. Minimum and maximum grid block timesteps using three test permeability realizations, and average timesteps using the 146 test permeability realizations.

Fig. 24. Runtimes of the numerical simulator and AI-based surrogate using the training and test realizations including (A) 54 training and 146 test permeability realizations and (B) 54 training and 540 projected test permeability realizations.

however, the round-off errors are still significant at early time points. The variable (trainable) timesteps, obtained from the LiSHT output activation limits $0.1 \leq \Delta t < 10$, gives the best rounding-truncation error trade-off for all space–time points; that is, the mean relative $L_1$ and $L_2$ errors are reduced by 25% and 49% respectively compared to those of the 1-day mean errors.

The minimum and maximum grid block timesteps using three test permeability realizations, and average timesteps using the 146 test permeability realizations are shown in Fig. 23. These are the optimized (i.e., after training using the training dataset) timestep statistics for each time point if the domain PDE were to be solved by the AI-based SRM using the test data and FDM. However, the AI-based SRM during testing only performs a forward pass without evaluation of the domain and boundary equations. In Fig. 23, it is observed that the timestep statistics do not show significant deviations when the test permeability realizations are varied; considering the 146 test dataset, the maximum deviation in the average timesteps is about 0.5 day (pale blue region in Fig. 23). However, the average timestep values change with time; these values are higher at early time periods and decrease over time to an average timestep value of 2 days. The higher timesteps, as determined by the timestep module at early periods, are required to offset the ML round-off errors and produce the required offtake rates.

4.5. Comparison of the artificial intelligence-based surrogate and numerical simulator performance

The comparison is based on training and predictions performed on the NVIDIA RTX 3060 graphics processing unit (GPU), and numerical simulations performed on the Core i7 11 700 K central processing unit (CPU). The total training time for the 54 by 48 training dataset, trained for 600 epochs using a batch size of 144 data rows is 42 min. The AI-based surrogate prediction runtime for the 146 by 48 test dataset
is 292 s. In contrast, it takes 2929 s (48.7 mins) to run the numerical simulator using the same test dataset. The runtime ratio of the AI-based surrogate predictions to that of the numerical simulator for the test dataset is 0.1. Fig. 24A shows the different runtimes of the numerical simulator and the AI-based surrogate during prediction and training. In situations, where hundreds or even thousands of permeability-time realizations are required, using similar machine and ANN architecture, a runtime ratio ≤ 0.1 will favour the AI-based surrogate over the numerical simulator (Fig. 24A). The only significant time expense of the AI-based surrogate is during the training; however, this is a fixed cost that becomes discounted for large test datasets (Fig. 24B). The training time can be reduced using advanced computing architectures, carefully increasing the batch size and more efficient graph parallelization, e.g., using distributed training across multiple GPUs or cloud nodes, swapping GPU to CPU usage during peak memory in the GPU, further simplification of some arithmetic operations to prevent recopying data, and large-batch optimization algorithms.

5. Conclusions

This paper investigates the application and effects of physics-based and non-physics-based regularizations in AI-based surrogate modelling for highly compressible subsurface flow. The AI-based surrogate is constructed as a custom three-module neural network with discriminative training; it is trained by exploiting the theoretical directions of fluid flow using a combination of FDM and AD on an incompletely-labelled dataset, i.e., only the initial condition and wells' subdatasets need labels of initial pressure and well constraints respectively.

The advantages of this workflow are the effective utilization of modular AI-based network architectures for simulation of flow behaviour, including the fluid property learning and time stepping; the grid sensitivity analysis performed as a pretraining operation ensures proper selection of an optimum AI-based grid size with reduced space-discretization numerical effects and training cost. Furthermore, the learning utilizes additional regularizations for improved predictions, including the \( \mathcal{L}_2 \) norm, tank material balance, trainable hard enforcement layer and time-discretization numerical errors. The predictions of the AI-based surrogate for unseen permeability-time realizations in a dry gas reservoir with multi wells operating under different constraints, including a shut in, agree with those obtained from a numerical simulator. The AI-based predictions are timely with an AI-based model to numerical simulator computational time ratio of 0.1, based on the test realizations and computing infrastructure.

The study shows that the non-physics-based regularization term, if used in the right order of magnitude improves grid block predictions; the material balance regularization term is important for constraining AI-based model trainable parameters towards optimal solutions in shorter epochs. A nonoptimal solution can be obtained, even as the total loss is minimized, when the material balance regularization term is not used; also, using it without the domain loss yields nonoptimal solutions. The trainable exponent used in the hard enforcement layer fully satisfies the initial condition and refines the grid block pressures; this is more effective than other hard enforcement techniques. The timestep module, which learns variable timesteps for solving the discretized domain PDE during training, improves the predictions compared to those of fixed timesteps. The trainable timestep module gives the best rounding-truncation error trade-off by effectively minimizing the combined discretized domain PDE and time-discretization numerical errors.

The limitation of this workflow is the time expense during pretraining and training. The pretraining costs can be during the grid sensitivity analysis or stochastic data generation. The training cost, which is the most significant cost, is due to the deep learning convolution operations, physics-based computations and their derivatives, and backpropagation within the network architecture. These costs could be a challenge in multiphase, 3D flow problems and finer grids as the trainable parameters are increased, in orders, according to the quantities of interest (e.g., multiphase flow) or grid dimension (e.g., 3D and finer grids). Nevertheless, with advanced computing and memory architectures, proper batch size selection and large-batch optimization, graph parallelization, effective model hard-enforcements, these costs can be reduced. The training cost is also dependent on the experience of the data analyst as the network hyperparameters, e.g., learning rates and regularization weights, and the hard enforcement techniques can require experience.

The effectively regularized and timely AI-based surrogate can be used as a state-dependent function in techniques like sensitivity analysis and data assimilation.

Nomenclature

\[ B = \text{formation factor of the fluid, rb/Mscf} \]
\[ C_{\text{fit}} = \text{covariance function} \]
\[ c_R = \text{formation compressibility, psia}^{-1} \]
\[ d = \text{encoding or decoding depth sequence} \]
\[ d_f = \text{depth of each convolution operation} \]
\[ f = \text{convolution filter dimension} \]
\[ h = \text{domain thickness, ft} \]
\[ h_c = \text{completion ratio} \]
\[ i = \text{iteration counter} \]
\[ K = \text{covariance kernel} \]
\[ k_x = \text{x-directional permeability, mD} \]
\[ k_y = \text{y-directional permeability, mD} \]
\[ L_{d, in} = \text{input length (height or width) of each transposed convolution layer} \]
\[ L_{d, out} = \text{output length (height or width) of each transposed convolution layer} \]
\[ L_{d,in} = \text{input length (height or width) of each convolution layer} \]
\[ L_{d,out} = \text{output length (height or width) of each convolution layer} \]
\[ m = \text{mass accumulation, lbm} \]
\[ m = \text{rate of mass accumulation, lbm/D} \]
\[ MSE_{\text{MBC}} = \text{weighted tank material balance loss} \]
\[ MSE_{\text{PHY}} = \text{weighted physics-based loss} \]
\[ MSE_T = \text{total loss} \]
\[ MSE_p = \text{weighted non-physics-based loss} \]
\[ MSE_{\text{p}} = \text{weighted domain loss} \]
\[ MSE_{\text{b}} = \text{weighted inner boundary condition loss} \]
\[ MSE_{\text{bc}} = \text{weighted initial condition loss} \]
\[ MSE_{\text{ob}} = \text{weighted outer boundary condition loss} \]
\[ N_{\text{in}} = \text{number of input nodes to a layer} \]
\[ N_{\text{out}} = \text{number of output nodes from a layer} \]
\[ N_{\text{PHV}} = \text{number of output nodes from a layer} \]
\[ N_x = \text{number of grid blocks in the x-direction} \]
\[ N_y = \text{number of grid blocks in the y-direction} \]
\[ N_b = \text{number of trainable parameters in the neural network} \]
\[ n_b = \text{number of rows per batch} \]
\[ n_p = \text{number of rows in training data} \]
\[ n_e = \text{number of epochs} \]
\[ n_I = \text{number of iterations during training} \]
\[ n_{\text{it}} = \text{number of iterations per epoch} \]
\[ P_d = \text{input padding of the convolution/deconvolution} \]
\[ p = \text{pressure, psia} \]
\[ p_0 = \text{initial pressure, psia} \]
\[ \beta = \text{predicted pressure, psia} \]
\[ q = \text{initial source or sink rate at standard conditions, Mscf/D} \]
\[ r_o = \text{Peaceman radius, ft} \]
\( r_w \) = source or sink radius, ft
\( S_{wi} \) = initial water saturation in the domain/reservoir
\( s \) = stride of the convolution/deconvolution operation
\( t \) = time, days
\( x_d \) = spatial locations in the \( x \) direction
\( a \) = flattened spatial locations in the \( x \) and \( y \) directions, ft
\( y_d \) = spatial locations in the \( y \) direction
\( Z \) = standard normal variate
\( a \) = learning rate
\( \gamma \) = density in terms of pressure/distance, psi/ft
\( \delta \) = fixed or average timestep, days
\( \Delta V \) = grid block bulk volume ft\(^3\)
\( \Delta x \) = grid block \( x \)-dimension, ft
\( \Delta y \) = grid block \( y \)-dimension, ft
\( \Delta z \) = grid block \( z \)-dimension, ft
\( \delta V \) = infinitesimal control bulk volume ft\(^3\)
\( \delta \) = scaling function
\( \epsilon_{IBC} \) = inner boundary condition residual error
\( \epsilon_{MBBC} \) = material balance residual error
\( \epsilon_r \) = round-off error
\( \epsilon_t \) = truncation error
\( \epsilon_{rt} \) = time-discretization round-off and truncation errors
\( \epsilon_p \) = percent error
\( \epsilon_d \) = domain residual error
\( \epsilon \) = machine learning fuzz factor (1 \times 10^{-7})
\( \theta \) = trainable parameters (weights and biases) in the neural network
\( \theta_h \) = trainable parameter in the hard enforcement layer
\( \lambda_{MBBC} \) = inner boundary condition regularization weight (hyperparameter)
\( \lambda_{MBBC} \) = material balance regularization weight (hyperparameter)
\( \lambda_n \) = non-physics-based regularization weight (hyperparameter)
\( \lambda' \) = ridge hyperparameter
\( \lambda_{DL} \) = domain loss regularization (hyperparameter)
\( \mu \) = fluid viscosity, cp
\( \xi \) = independent simple random variables
\( \varphi \) = porosity
\( \varphi_p \) = porosity at reference conditions
\( \ell \) = correlation length in the \( x \) or \( y \) axis

Subscripts
\( g \) = gaseous phase
\( i, j \) = grid index in the 2D domain
\( \text{max} \) = maximum value of a variable or tensor
\( \text{min} \) = minimum value of a variable or tensor
\( \text{opt} \) = optimizer iteration counter
\( \text{start} \) = start of simulation or training

Superscripts
\( n \) = current time level
\( n+1 \) = next time level
\( n+2 \) = next two time levels
\( \text{time level} \)

CRediT authorship contribution statement

**Victor C. Molokwu**: Conceptualization, Methodology, Python programming, Draft preparation. **Bonaventure C. Molokwu**: Software methodology, Debugging, Testing. **Mahmoud Jamilahmadry**: Theory, Methodology, Review.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Victor Chinemelu Molokwu reports financial support was provided by Petroleum Technology Development Fund.

Availability of data and materials

The AI-based SRM datasets trained during the current study can be obtained using a python module in the repository: https://github.com/molokwuvictor/3D_feature_generator. The numerical simulation results can be obtained using the Schlumberger Eclipse\textsuperscript{TM} reservoir simulator.

Acknowledgements

This case and study has been funded by the Petroleum Technology Development Fund (PTDF). Additionally, the authors would like to acknowledge the support provided by the Heriot-Watt University Gas Condensate Research Group.

References


