LEARNING LARGE-SCALE SUBSURFACE SIMULATIONS WITH A HYBRID GRAPH NETWORK SIMULATOR

Anonymous authors
Paper under double-blind review

ABSTRACT

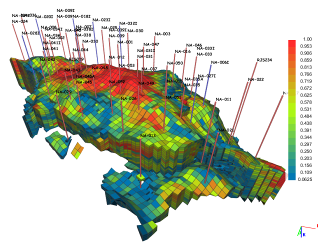
Subsurface simulations use computational models to predict the flow of fluids (e.g., oil, water, gas) through porous media. These simulations are pivotal in industrial applications such as petroleum production, where fast and accurate models are needed for high-stake decision making, for example, for well placement optimization and field development planning. Classical finite difference numerical simulators require massive computational resources to model large-scale real-world reservoirs. Alternatively, streamline simulators and data-driven surrogate models are computationally more efficient by relying on approximate physics models, however they are insufficient to model complex reservoir dynamics at scale. Here we introduce Hybrid Graph Network Simulator (HGNS), which is a data-driven surrogate model for learning reservoir simulations of 3D subsurface fluid flows. To model complex reservoir dynamics at both local and global scale, HGNS consists of a subsurface graph neural network (SGNN) to model the evolution of fluid flows, and a 3D-U-Net to model the evolution of pressure. HGNS is able to scale to grids with millions of cells per time step, two orders of magnitude higher than previous surrogate models, and can accurately predict the fluid flow for tens of time steps (years into the future). Using an industry-standard subsurface flow dataset (SPE-10) with 1.1 million cells, we demonstrate that HGNS is able to reduce the inference time up to 18 times compared to standard subsurface simulators, and that it outperforms other learning-based models by reducing long-term prediction errors by up to 21%.

1 INTRODUCTION

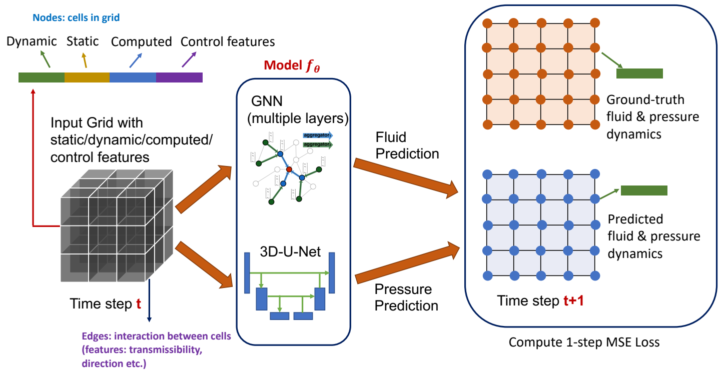
Subsurface simulation is a discipline that uses computational models to predict the flow of fluids (e.g., oil, water, gas) through porous media. It is pivotal for management of hydrocarbon and groundwater resources. The goal of simulations is to take as input static properties of the rock, initial states of quantities such as oil and water, and external control variables such as injection of water, then predict the evolution of pressure and saturation of fluids over time (see Fig. 1 and Appendix A).

Two key challenges need to be addressed for practical, large-scale applications. First, subsurface flow is a complicated multi-scale problem. On the smaller spatial scale, it needs to model multiphase fluids flow through the complex subsurface structures: between neighboring cells (a cell is a discretization of space containing porous media through which fluid can flow), with various well configurations (wells can inject or produce fluids externally), and in presence of flow barriers (e.g., fault planes that can divert or prevent fluid paths). On a larger spatial scale, the flow of fluids is driven by convection forces and pressure gradient, whereas the dynamics of reservoir pressure is governed by global pore fluid distribution and reaches the equilibrium much faster. Therefore, we need to model both small and large spatial scales. Second, the model often needs to scale to extremely large grids. For example, a standard industry problem typically consists of millions or tens of millions of cells. Field development applications, such as well configuration and completion for optimal dynamic performance and improved sweep, could benefit significantly from subsurface simulators that can perform fast inference and can scale to large grids.

Standard subsurface simulators employ domain-specific implicit partial differential equation (PDE) solvers. For large grids with tens of millions of cells, they still need to solve an equation involving the full grid, which may be computationally exhaustive and requires substantial inference time.



(a) Subsurface simulation model.


Nodes: cells in grid
Dynamic
Static
Computed
Control features
Model f_θ
Input Grid with static/dynamic/computed/ control features
GNN (multiple layers)
3D-U-Net
Time step t
Edges: interaction between cells (features: transmissibility, direction etc.)
Fluid Prediction
Pressure Prediction
Ground-truth fluid & pressure dynamics
Predicted fluid & pressure dynamics
Time step t+1
Compute 1-step MSE Loss

(b) Our HGNS architecture.

Figure 1: **Overview of our HGNS architecture.** (a) Water and oil exist in the porous rock which is discretized into a computational grid cells. Blue pins are water injectors, red pins are oil producers. (b) HGNS consists of a Subsurface Graph Neural Network (SGNN) to model the fluid dynamics, and a 3D-U-Net to model the pressure dynamics. The input grid on the left is treated as a graph by modeling each cell as a node, and connecting the adjacent cells via edges.

Recently, data-driven surrogate models provide a promising complementary approach (Sanchez-Gonzalez et al. (2020); mro (2018)). They learn directly from data and may alleviate years of engineering development. Moreover, prior works in other domains show that the models can learn forward evolution (Sanchez-Gonzalez et al. (2020)) and can have larger spatial and temporal intervals (Kochkov et al. (2021); Um et al. (2020)). However, these models are insufficient to address the two challenges of subsurface simulation. They are not able to model the multi-scale dynamics, because they utilize Gaussian process (Hamdi et al. (2017)), polynomial chaos (Bazargan et al. (2015)), feed-forward neural network (Costa et al. (2014)), convolutional neural networks (Zhu & Zabaras (2018)), or recurrent R-U-Net (Tang et al. (2020)), which cannot simultaneously model lower-level interactions between neighboring cells as well as global dynamics such as pressure. Additionally, they do not scale to large-scale simulations as they have only been applied to 2D grids with up to 10k vertices (while a practical simulation requires 3D grids), and up to 20k nodes for machine-learning-based surrogate models developed in other domains, e.g. GNS (Sanchez-Gonzalez et al. (2020)), which is two to three orders of magnitude smaller than needed for standard industry applications.

Present work. We introduce a Hybrid Graph Network Simulator (HGNS) for subsurface simulation, which addresses the above two challenges (see Fig. 1b). HGNS consists of a Subsurface Graph Neural Network (SGNN) designed to model the fluid flow through the complicated subsurface structure, and a 3D-U-Net (Çiçek et al. (2016)) to model the more global dynamics of pressure. SGNN and 3D-U-Net cooperatively learn to evolve the subsurface dynamics. For large grids, which make training especially hard (exceeding GPU memory, training may take weeks), we developed a sector-based training. It allows training on grids of up to 1.1 million of cells, two orders of magnitude larger than previous machine-learning-based surrogate models. The ability to deal with models of this size makes HGNS the first machine-learning-based subsurface model applied to realistic 3D scenarios.

We use an industry-standard subsurface flow dataset to evaluate our model's generalization capabilities in a challenging setting where the initial conditions, static properties and well locations are different than that of training, and compare its performance with strong baselines. We show that HGNS produces more accurate long-term evolution, and outperforms other learning-based models by reducing long-term prediction errors by up to 21%.

2 Method

Our aim is use machine learning to perform subsurface simulation of oil-water flow, which models how the pressure and saturation of fluids evolve over time, given initial states, static properties of the rock, and external control variables such as injection of water. We want our approach to handle complex well configurations (vertical wells, slanted wells, horizontal wells), faults planes in rock that reduce, divert or prevent fluid flow.

Formally, we want to learn a simulator f_θ, which takes as input: (a) X^t representing dynamic variables such as water saturation, oil saturation and pressure at time t, (b) Q representing constant values such as cell porosity, absolute permeability, cell depth, and pore volume, and (d) U^t which may include the injection of water, and predicts X^{t+1}.

Our Hybrid Graph Network Simulator (HGNS) architecture (Fig. 1b) consists of a Subsurface Graph Neural Network (SGNN) to model the dynamics of fluids (water, oil) on a finer scale, and a 3D-U-Net (Çiçek et al. (2016)) to model the more global dynamics of pressure. Concretely, our HGNS f_θ can be written as:

$$\begin{cases} \hat{S}^{t+1} = g_\theta(X^t, Q, U^t) + S^t \\ \hat{P}^{t+1} = h_\theta(X^t, Q, U^t) + P^t \end{cases} \tag{1}$$

Here S^t is the saturation for water and oil, P^t is pressure, g_θ is the SGNN model and h_θ is a 3D-U-Net (Çiçek et al. (2016)).

Hybrid architecture. Our SGNN uses the encoder-processor-decoder architecture, similar to the work by Sanchez-Gonzalez et al. (2020). The encoder embeds the input (X^t, Q, U^t) into a latent graph $\mathcal{G}^{(0)} = (\mathcal{V}^{(0)}, \mathcal{E}^{(0)})$, where $\mathcal{V}^{(0)} = \{v_i^{(0)}\}$ is the collection of input node features and $\mathcal{E} = \{e_{ij}^{(0)}\}$ is the collection of edge features defined on the edges. After obtaining $\mathcal{G}^{(0)}$, the encoder has a Multilayer Perceptron (MLP) that encodes the node features $\mathcal{V}$ into some latent embedding. The processor is a stack of M graph neural network layers, each one performing one round of message passing that mimics the flow of fluid between neighboring cells. The decoder is a simple MLP that maps the output of the processor back to the predicted dynamic variables at the next time step. Overall, our SGNN models the complex subsurface flow by encoding the properties and dynamics of each cell and cell-cell relation into node and edge features, then uses a very expressive edge-level MLP_e to compute the interaction between neighboring cells, and a node-level MLP_v to update the state of the cells. In this way, it is able to model subsurface flow in through complex subsurface structures. Additionally, we use 3D-U-Net (Çiçek et al. (2016)) to model the more global dynamics of pressure, since its hierarchical structure is better suited for this phenomena.

Sector-based training. Training of the model presents a special challenge, since realistic data is too large to fit in a single GPU. To address this problem, we developed sector-based training, which partitions data into many sectors and which can be then processed individually. In this way, we can train with a full grid of arbitrary size, because we can always partition the full grid into sectors with constant size to fit into GPU's memory. Our sector-based training accounts for the boundary conditions, makes sure that all cells contribute to the loss computation in at least one sector, and takes advantage of randomly mixing sectors in minibatches, which can reduce the correlation between examples and help the model learn more general dynamics. In addition, sectors enable multi-GPU training, which leads to significant speeding up of the training step. In addition, we develop *multi-step rollout in training* (Appendix B) which significantly improves long-term prediction.

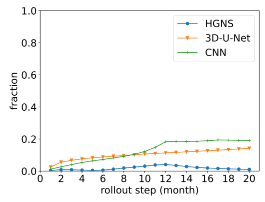
3 EXPERIMENTS

To evaluate our HGNS model we compare the long-term prediction performance of our HGNS model against strong baselines and investigate how different components of HGNS contribute to predictive performance. We evaluate the models in a challenging setting in which the static properties, initial conditions and the well locations in testing are all different from that during training. In this way, we measure how the models are able to generalize to novel scenarios.

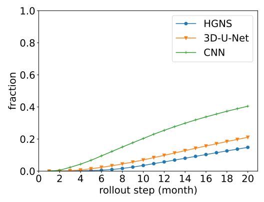
Dataset and training regime. We evaluate our model on an industry-standard subsurface simulation dataset, a Single Porosity Single Permeability (SPSP) model, which does not have fractures. Our evaluation set consists of 20 trajectories, each trajectory has 61 time steps (the time interval between neighboring steps is 1-month), and the full grid has 1.1 million cells (thus each epoch needs to go through 1.37 billion cells). These trajectories differ in their static properties, initial states and well locations. We randomly choose 16 trajectories for training, and the other 4 trajectories for testing. Since different trajectories have drastically different static properties, initial states and well locations, the performance at the test set evaluates how the models are able to generalize to novel subsurface structures. In the testing, we provide the models the initial state at $t = 3$ (allowing for 3 steps of initial stabilization), and autoregressively roll out the models for 10 and 20 steps, and compute the Mean Absolute Error (MAE) between the model's prediction and the ground-truth.

Table 1: Mean Absolute Error (MAE) of our HGNS and other baseline models on the test set for pressure, water and oil prediction, after 10-step (10 months) and 20-step rollout. HGNS outperforms other strong baselines by an pressure error reduction of 5.5% and 6.7% for 10-step and 20-step, water error reduction of 21% and 8.4% for 10-step and 20-step, and oil error reduction of 21% and 8.3% for 10-step and 20-step, compared with the best performing scenario in other models.

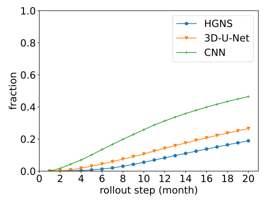
Model	10-step prediction MAE			20-step prediction MAE		
	Pressure (psi)	Water (barrel)	Oil (barrel)	Pressure (psi)	Water (barrel)	Oil (barrel)
Predict no change	210.8	0.941	0.941	296.1	1.541	1.541
CNN	77.9	0.628	0.608	104.2	1.157	1.090
3D-U-Net	94.6	0.361	0.361	142.6	0.725	0.724
HGNS (ours)	**73.6**	**0.286**	**0.286**	**97.2**	**0.664**	**0.664**



(a) Fraction of cells whose absolute error of pressure is greater than 100 psi.



(b) Fraction of cells whose absolute error of water volume is greater than 1 barrel.



(c) Fraction of cells whose absolute error of oil volume is greater than 1 barrel.

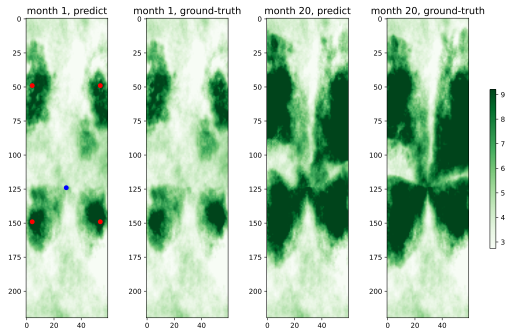
Figure 2: Comparison of models during rollout on the fraction of cells whose error of (a) pressure (b) water volume (c) oil volume is above a given threshold. The lower the fraction, the better the prediction. HGNS outperforms 3D-U-Net and CNN in both scenarios, achieving a reduction of fraction of 71% for pressure, 29.7% for water volume, and 28.7% for oil volume, compared with the best performing model.

Baselines. We compare our HGNS model, trained with 4-step rollout, with two other strong baselines, 3D-U-Net (Çiçek et al. (2016)) and standard CNN, each trained with 4-step rollout. We also use a “predict no change” baseline, in which the model simply copies the previous time step as its prediction, to estimate the error scale. Table 1 shows the results. The ground-truth pressure is typically on the level of 6000-10000 psi, and the water and oil volume is typically on the level of 10-20 barrels. Typically, a 100 psi error on the pressure and 1 barrel error on the water/oil on a cell in a long-term prediction is deemed as acceptable.

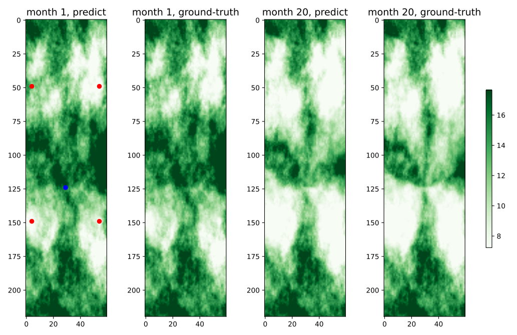
Accuracy of 10- and 20-month predictions. Table 1 shows that our HGNS outperforms other baselines by a wide margin, achieving 5.5% and 6.7% reduction of pressure error at 10-step and 20-step rollout, 21% and 8.4% reduction of water error at 10-step and 20-step, and 21% and 8.3% reduction of oil error at 10-step and 20-step than the best-performing model. We also see that the error of HGNS is significantly less than the “predict no change” baseline, showing that it has learned non-trivial dynamics of the system.

Another important angle to evaluate the prediction is to compute the *fraction* of cells whose prediction error is less than 100 psi for pressure and 1 barrel for water/oil volume. Fig. 2 shows the above fractions vs. rollout steps for HGNS and the two other compared models. We see that HGNS outperforms the other models consistently across all rollout steps, in fraction for pressure, water and oil. Moreover, even after 20 steps (months) rollout of HGNS, the fraction of cells whose pressure error is greater than 100 psi remains below 4.1%, 71% lower than the best performing model (3D-U-Net) whose maximum fraction is 14.2%. Similarly, HGNS’s fraction of cells whose water and oil volume error is below than 1 barrel remains below 14.8% and 18.9%, respectively, 29.7% and 28.7% lower than the best performing model. This shows that the predictions of HGNS are above standard for a majority of cells, and that they are a significant improvement over strong baselines.

Fig. 3 shows our HGNS’s 20-step water and oil predictions in a typical test dataset and compares them with ground-truth. Fig. 3 shows that our model captures reliably the water flow from the injector (located at the corners and waist) to the middle producer and that our model captures the oil flow and drainage due to the producer.



(a) Water volume (barrel) prediction vs. ground-truth



(b) Oil volume (barrel) prediction vs. ground-truth

Figure 3: Comparison of HGNS 1-step (1-month) and 20-step rollout vs. ground-truth, on (a) water volume and (b) oil volume, on one of the trajectories, at a cross section of depth 20. For water and oil volume, images are from left to right: month 1 prediction, month 1 ground-truth, month 20 prediction, month 20 ground-truth. Notice that HGNS reliably captures water flow from 4 injectors (red dots) to the producer (blue dot at the middle), and the oil flow and drainage due to the producer.

Table 2: Mean Absolute Error (MAE) of our HGNS and its ablations on the test set for pressure, water and oil prediction, after 10-step (10 months) and 20-step rollout. Hybrid design and multi-step training of HGNS improve performance by an average error reduction of 15.7% and 22.4%, respectively.

Model	10-step prediction MAE			20-step prediction MAE		
	Pressure (psi)	Water (barrel)	Oil (barrel)	Pressure (psi)	Water (barrel)	Oil (barrel)
HGNS (ours)	73.6	**0.286**	**0.286**	**97.2**	**0.664**	**0.664**
HGNS without 3D-U-Net (only SGNN)	74.8	0.307	0.307	110.5	0.829	0.829
HGNS without SGNN (only 3D-U-Net)	94.6	0.361	0.361	142.6	0.725	0.724
HGNS with 1-step	**47.3**	0.500	0.500	122.8	1.144	1.144

Ablations. To evaluate how the hybrid design and multi-step training contribute to the improved performance, we compare HGNS with its ablations (see Table 2): one without 3D-U-Net (using SGNN to predict both the pressure and fluid), one without SGNN (using 3D-U-Net to predict pressure and fluid, same as in Table 1). Results show that using only 3D-U-Net and SGNN results in worse performance than the hybrid design, and multi-step training improves long-term prediction.

Runtime comparison. On the dataset above with 1.1 million cells per trajectory, our HGNS took 20.7s to roll out 20-steps with an NVIDIA Quadro RTX 8000 GPU, compared to approximately 46s-370s (varying depending on the number of wells) required by the standard PDE solver using 4 compute nodes, each with 2 CPUs Intel(R) Xeon(R) E5-2680 v3 2.50GHz, a 2 to 18-fold reduction in execution time. We expect that HGNS gains will be even larger with grid sizes over 10 million.

Industry deployment. We are finishing up deploying and integrating our HGNS into an industry pipeline for subsurface simulations and field development planning. The HGNS in deployment shares the same interface as the standard solver in the pipeline. During field development planning, HGNS will be used for fast rollout and finding candidate solutions to accelerate high-level well placement and operational decisions.

4 Conclusion

We presented a Hybrid Graph Network Simulator (HGNS) for learning subsurface simulations, which employs a hybrid Subsurface Graph Neural Network (SGNN) to model the fluid flow through the complicated subsurface structures, and a 3D-U-Net to model the more global dynamics of pressure, addressing the challenge of multi-scale dynamics. HGNS is able to perform training and inference on grid sizes of at least millions, two orders of magnitude larger than previous learning-based subsurface surrogate models. Future work includes extending our HGNS to more complicated Dual Porosity Dual Permeability (DPDP) subsurface models, where fractures act as conduits for fast fluid flow, and accelerated history matching, where static properties can be inferred and updated by solving the inverse problem, conditioned to observed dynamic data.

A PRELIMINARIES

Here we provide background for the subsurface simulation. We consider the problem of subsurface simulation of oil-water flow, which models how the pressure and saturation of fluids evolve over time, given initial states, static properties of the rock, and external control variables such as injection of water. Here we present a simplified Partial Differential Equation (PDE) for the system:

$$\frac{\partial(\phi\rho_j S_j)}{\partial t} = \nabla \cdot (\frac{\rho_j}{\mu_j} k_{rj}(S_j)\mathbf{k}\nabla P) + q_j \tag{2}$$

Here $j = w, o$ denotes different components/phases, with w for water and o for oil. $S_j \in [0, 1]$ is the saturation for phase j (saturation is the ratio between the present volume of component j and the pore volume V the rock can hold at a location), and P is the pressure. ρ_j is phase density, μ_j is the phase viscosity, and ϕ is the rock porosity. $k_{rj}(S_j)$ is the relative permeability that is a function of the saturation S_j, usually obtained via laboratory measurements. $\mathbf{k}$ is the absolute permeability tensor. q_j is the source/sink term, which corresponds to the injecting or producing of component j at the well location. In this equation, the saturation S_j and pressure P are dynamic variables that vary with time and space. ϕ, ρ_j, μ_j, $\mathbf{k}$ are static variables that are constant in time but typically vary in space. The injection of water is externally controlled, and the production of water/oil q_j at producer wells is also a dynamic variable. We can intuitively understand this equation as follows: the change of water/oil saturation S_j is due to two terms: the divergence of the flux $\Phi_j = -\frac{\rho_j}{\mu_j} k_{rj}(S_j)\mathbf{k}\nabla P$ and a source/sink q_j term. The flux Φ_j is driven by the pressure gradient ∇P, where fluids flow from places with higher pressure to those with lower pressure. Note that the coefficient $k_{rj}(S_j)$ depends on the dynamic variable S_j, making the dynamics nonlinear.

Eq. (2) is a simplified model. The problem we consider here is much more complicated and high-dimensional. Current (non-machine learning) approaches use implicit methods to evolve the system, which involves solving a system of equations containing up to tens of millions of equations (each cell of the discretized grid contributes one equation). Solving such a system can be slow even after linearization. Moreover, in reality there are complex well configurations (vertical wells, slanted wells, horizontal wells), faults planes in rock that reduce, divert or prevent fluid flow. This in addition adds significant complexity to modeling and evolving the system in an accurate way. In contrast, machine learning represents an attractive approach to alleviate these issues and develop faster, more scalable and accurate simulators of such complex phenomena.

B MULTI-STEP ROLLOUT DURING TRAINING

A standard learning objective for learned simulation is to minimize the following Mean Squared Error (MSE) on the 1-step prediction $L = \mathbb{E}_t\left[\ell(f_\theta(X^t, Q, R(X), U), X^{t+1})\right]$ where $\ell(\hat{X}^{t+1}, X^{t+1}) = (\hat{X}^{t+1} - X^{t+1})^2$. However, we find that even with a very small 1-step training loss, the multi-step rollout

$$\hat{X}^{t+k+1} = f_\theta(\hat{X}^{t+k}, Q, R(\hat{X}^{t+k}), U^{t+k}), k = 0, 1, ...K \tag{3}$$

can have a large error due to error accumulation. To improve long-term prediction, we develop multi-step rollout during training. It performs multiple steps of rollout, and the loss is given by:

$$\begin{cases} L = \mathbb{E}_t\left[\sum_{k=1}^K \lambda_k \cdot \ell\left(f_\theta(\hat{X}^{t+k}, Q, R(\hat{X}^{t+k}), U^{t+k}), X^{t+k+1}\right)\right] \\ \hat{X}^{t+k} = f_\theta(\hat{X}^{t+k-1}, Q, R(\hat{X}^{t+k-1}), U^{t+k-1}), k = 1, 2, ...K \end{cases} \tag{4}$$

Here $\hat{X}^t := X^t$ is the initial state for the rollout. The model f_θ performs rollout autoregressively, using the predicted $\hat{X}^{t+k}$ as the next state's input, and the loss is a weighted sum of loss on the autoregressive predictions $\hat{X}^{t+1}, ..., \hat{X}^{t+K+1}$ for all rollout steps, weighted by weights λ_k. During training, the backpropagation can pass through the full rollout steps, so that the model is *directly* trained to minimize long-term prediction error. In practice, there is a tradeoff between computation and accuracy. The larger the total rollout steps K in training, the better potential accuracy it can achieve, but the more compute and memory it requires (scales linearly with K). We find that using $K = 4$ strikes a good balance, which uses reasonable compute, achieves far better accuracy than 1-step loss, and there is minimal gain to further increase K. In our experiments, we use $K = 4$ and $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (1, 0.1, 0.1, 0.1)$. The weights for the steps > 1 are smaller, since at the beginning of training when the model is inaccurate, the multi-step loss is much worse. Having a larger weight on those steps would make the model not able to learn anything. Having 1-step loss dominates as we are using helps the model to find a good minimum of the loss landscape first, and can further improve by the multi-step part.

C FEATURES ENCODED

Table 3 shows all features we used for the experiments, consisting of dynamic, static, computed and control features.

Table 3: Encoded dynamic, static, computed and control features for our HGNS model and compared models. Here the node type one-hot encoding denotes whether a cell is a normal cell, injector, producer. The boundary encoding is a 3-vector encoding if a cell is near the boundary of the full grid, and has value ramping from 0 to 1 if it is from 5 to 1 cell distance from the boundary.

Dynamic features X^t	Static features Q	Computed features $R(X^t)$	Control features U^t
Pressure P	Depth of cell	Water relative permeability $k_{rw}(S_w)$	Water injection rate $q^t_{w,inj}$
Water volume V_w	Porosity ϕ	Oil relative permeability $k_{ro}(S_o)$	Pressure at injector location
Oil volume V_o	Pore volume V	Spatial gradient of dynamic features ∇X^t	
	Connate water volume V_{wc}		
	Permeability in x direction k_x		
	Permeability in y direction k_y		
	Permeability in z direction k_z		
	Transmissibility in x direction T_x		
	Transmissibility in y direction T_y		
	Transmissibility in z direction T_z		
	Node type one-hot encoding		
	Boundary encoding		

D HYPERPARAMETERS FOR HGNS

Table 4 shows the hyperparameter values used for HGNS.

Table 4: Hyperparameters used for HGNS

Parameter name	Value
Number of GNLayers for the processor	2
Latent size for the processor	16
Activation	elu
Type of nomalization	Group normalization
Number of layers for each MLP	2
Convoluion type	GNLayer
Number of neurons for each layer of MLP in the processor	128
Number of neurons for encoder MLP	128
Number of neurons for decoder MLP	128
Number of layers for encoder MLP	2
Number of layers for decoder MLP	2
Number of layers for the pooling and unpooling models	1
Number of groups for GroupNorm	2
Residual connection to use in GNN model	None
Fluid decoder model	MLP
Number of feature maps for first conv layer of U-Net encoder	32
Number of levels in the U-Net encoder/decoder path	3
Number of groups in U-Net group norm	2

The pressure model for HGNS uses 3D-U-Net (Çiçek et al. (2016)). We modify 3D-U-Net's order of operation such that for each convolution layer, it first performs 3D-convolution, then applies activation (ReLU) followed by group normalization. We observe that this improves performance compared to the default order of Conv $\rightarrow$ BN $\rightarrow$ ReLu.

Table 5 shows the general structure of the HGNS fluid model structure, as well as the detailed structure for the node level MLP (MLP$_v$) and edge level MLP (MLP$_e$). The structure of MLP$_v$ and MLP$_e$ in the two GNLayers are the same.

Table 5: HGNS fluid model structure

General structure	Node MLP (MLP$_v$)	Edge MLP (MLP$_e$)
Input: x	Input: x	Input: x
$x = \text{ELU}(x)$	$x = \text{Linear}(144, 128)(x)$	$x = \text{Linear}(36, 128)(x)$
$x = \text{GNLayer}_0(x)$	$x = \text{ELU}(x)$	$x = \text{ELU}(x)$
$x = \text{GroupNorm(2, 16)}(x)$	$x = \text{Linear}(128, 16)(x)$	$x = \text{Linear}(128, 128)(x)$
$x = \text{GNLayer}_1(x)$		

E Details for training

Table 6 shows the values of all hyperparameters used for training. The noise added refers to the random walk noise added during training. As explained by Sanchez-Gonzalez et al. (2020), adding random walk noise brings the training distribution closer to the distribution during prediction rollout. The weight of cell refers to a weight we assign to different cells depending on their distance to a well location. In cases where we want more accurate prediction results near well locations, we can assign a higher weight to the cell near the well, and a gradually decreasing weight for cells further away following some density function (e.g. Gaussian function).

F Details for dataset and pre-processing

The datasets we used for training consist of 16 different trajectories, and the test set consists of 4 different trajectories. Each trajectory is the evolution of a $85 \times 220 \times 60$ grid, in the depth (vertical), length and width direction, respectively, and spans over 61 times steps. Such grid size amounts to a total of 1122000 cells.

We create an edge in the graph between edge valid cell and its valid neighbor. If either the cell or its neighbor is an invalid cell, no edge will be created between this cell pair.

There can be a varying number of wells (producer or injector) in each trajectory, all between 5 and 10 in our dataset.

Table 6: Hyperparameters used for training

Parameter name	value
Loss function	MSE
Number of epochs	40
Batch size	1
Learning rate	0.001
Weight decay	0
Otimizer	Adam
Optimizer scheduler	cos
Noise added	3e-5
Weight of cell	Gaussian decay with std.=10, minimum weight=0.2 and weight=50 at well centers

REFERENCES

Flexible neural representation for physics prediction, 2018.

Hamid Bazargan, Mike Christie, Ahmed H Elsheikh, and Mohammad Ahmadi. Surrogate accelerated sampling of reservoir models with complex structures using sparse polynomial chaos expansion. *Advances in Water Resources*, 86:385–399, 2015.

Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pp. 424–432. Springer, 2016.

Luís Augusto Nagasaki Costa, Célio Maschio, and Denis José Schiozer. Application of artificial neural networks in a history matching process. *Journal of Petroleum Science and Engineering*, 123:30–45, 2014.

Hamidreza Hamdi, Ivo Couckuyt, Mario Costa Sousa, and Tom Dhaene. Gaussian processes for history-matching: application to an unconventional gas reservoir. *Computational Geosciences*, 21(2):267–287, 2017.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Meng Tang, Yimin Liu, and Louis J Durlofsky. A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. *Journal of Computational Physics*, 413:109456, 2020.

Kiwon Um, Robert Brand, Philipp Holl, Nils Thuerey, et al. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *arXiv preprint arXiv:2007.00016*, 2020.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.