Accelerating Physics-Based Simulations Using End-to-End Neural Network Proxies: An Application in Oil Reservoir Modeling

Jiří Navrátil[1]*, Alan King[1], Jesus Rios[1], Georgios Kollias[1], Ruben Torrado[2] and Andrés Codas[3]

[1] IBM Research, Yorktown Heights, NY, United States, [2] Repsol S.A., Móstoles, Spain, [3] IBM Research, Rio de Janeiro, Brazil

We develop a proxy model based on deep learning methods to accelerate the simulations of oil reservoirs–by three orders of magnitude–compared to industry-strength physics-based PDE solvers. This paper describes a new architectural approach to this task modeling a simulator as an end-to-end black box, accompanied by a thorough experimental evaluation on a publicly available reservoir model. We demonstrate that in a practical setting a speedup of more than 2000X can be achieved with an average sequence error of about 10% relative to the simulator. The task involves varying well locations and varying geological realizations. The end-to-end proxy model is contrasted with several baselines, including upscaling, and is shown to outperform these by two orders of magnitude. We believe the outcomes presented here are extremely promising and offer a valuable benchmark for continuing research in oil field development optimization. Due to its domain-agnostic architecture, the presented approach can be extended to many applications beyond the field of oil and gas exploration.

Keywords: reservoir model, surrogate model, physics-based simulation, deep neural network, sequence-to-sequence model, long short-term memory cell, reservoir simulation

1. INTRODUCTION

Reservoir modeling plays an essential role in modern oil and gas exploration. After an acquisition of a reservoir field, energy companies plan the field development based on a capital expense of billions of dollars. A placement of even a single well in a bad spot can mean a significant economical loss for the developer. A reservoir model (RM) is a computerized representation of the field drawing from various data sources, such as geological expert analyses, seismic measurements, well logs, etc., with added properties determining the dynamic reservoir behavior. Its primary purpose is to allow optimization and better prediction of the field's future output using mathematical simulators. Given a set of input actions (well drilling), a simulator operates on an RM by solving large systems of non-linear PDEs to predict future outcomes over long time horizons (up to tens of years).

RM simulators may require a considerable amount of computation (time) to produce predictions (minutes, hours, or days, depending on the RM size). Current optimization techniques in reservoir engineering are therefore able to simulate only a small number of cases. In our context, the problem of field development is formulated as an optimization over a very high number (millions) of candidate sequential well placements. As such, the solving time of a simulation quickly becomes the bottleneck rendering methods such as Monte Carlo Planning and Reinforcement

Learning impractical. While the idea of creating an approximating surrogate, or proxy, is not new, previous approaches targeted accelerating steps inside the physics-based simulator (see section 2). Instead, we pursue a deep learning approach to learn approximating essential output variables of a PDE-solver in an *end-to-end*[1] fashion using training data generated by that solver, and demonstrate its performance and accuracy at predicting production rates on a benchmark RM with complex geological properties, namely the SPE9 (SPE9, 2018).

2. PREVIOUS WORK

Accelerating reservoir simulations has a wide literature. State of the art techniques can be classified in two categories: (1) reducing the complexity of the PDEs while providing an acceptable loss of prediction accuracy [e.g., reduced order modeling (ROM) (He and Durlofsky, 2015) can accelerate the simulations by factors of 10^2 and has been found useful in optimization and control (Jansen and Durlofsky, 2016), and upscaling which achieves acceleration with coarser reservoir models (Durlofsky, 2005)], and (2) simple polynomial interpolation techniques computing the objective function (e.g., Net Present Value, or NPV) or to characterize the uncertainty (Valladao et al., 2013). It is important to note that ROM techniques typically target optimization of certain variables, such as well controls, for fixed well locations (in contrast, our task is to accelerate across different sequences of well locations). Furthermore, it is noted that while techniques listed under (2) are fast their accuracy tends to be poor.

Artificial Neural Networks (ANN) have been proposed to accelerate oil reservoir simulations and aim to achieve both objectives: accuracy and acceleration. A special issue of *Journal of Petroleum Science and Engineering* in 2014 was devoted to this topic, and a good summary can be found in the editorial (Aïfa, 2014). As outlined in this summary, ANN approaches can be categorized as (1) physics-based models, and (2) data-driven models. Physics-based models use the PDE structures as features. Recent example from this category is application of Convolutional Neural Networks (CNN) to modeling flow around complex boundaries for the purpose of accelerating animation (Tompson et al., 2016). A sequential neural network approach using recurrent units, applied to modeling output of oil fields was described in Sagheer and Kotb (2018). This work is relevant in its application of the Long Short Term Memory (LSTM) cell to modeling well outputs, however, the task tackled in Sagheer and Kotb (2018) is considerably different: the model consumes previous observations (or optionally own predictions) to predict one time step of the same variable, i.e., it is a sequence task. In contrast, our task is to generate a series of rates predictions with the input being the well placement schedule, i.e., a sequence-to-sequence task, with input and output sequences being diametrically different. The latter allows us to

completely replace the physics-based simulator: Given input and output data, our proxy method is agnostic with respect to the choice of the particular simulator at training time, and, at test (run) time, the proxy model operates fully independently of the simulator. The challenge to purely data-driven approaches is the extreme nonlinearity of the reservoir dynamics. It seems important to incorporate features of the reservoir. Examples of basic, fully connected ANNs to predict reservoir production are Wei et al. (2017) and Wang et al. (2018). In the context of pre-existing literature, we believe our contribution is twofold: (1) our architectural approach is unique as it deals with sequential action input (i.e., varying timing and locations of wells) and output of varying span, reservoir uncertainty, and optimized well control, while modeling the simulator in an *end-to-end* manner, and (2) we present a thorough experimental analysis on a publicly available reservoir model thus creating a reference for future comparison by the community.

3. RESERVOIR MODEL SIMULATION

The Reservoir Model (RM) simulation considered in this work is based on a so-called Black Oil model (Trangenstein and Bell, 1989) that describes the flow of reservoir three fluids through porous rock: oil, water, and gas.

3.1. Physics
The Black Oil model equates the time change of mass in a region with the mass flux across the region boundary. The flux is driven by pressure differences caused by well operations. The system of equations is highly nonlinear due to the non-stationary interactions between rock types and fluids. Rock compressibility and relative permeability are altered by changes in fluid pressure and saturation levels. In addition, petroleum fluids will undergo phase changes, between gas and liquid form, as they move through pressure and temperature gradients. An accessible introduction into the model and inner-workings of Black Oil Simulators (BOS) can be found in Lie (2014).

A BOS is a basic discrete-time finite-volume simulation. The reservoir is partitioned into cells and the values of the primary variables: oil pressure, water saturation, and gas saturation, which are evaluated over a sequence of time steps out to a fixed time horizon.

At each time step, the BOS solves the mass balance equations:

$$\frac{M_{c,f}(x_t) - M_{c,f}(x_{t-1})}{\Delta t} = F_{c,f}(x_t) + Q_{c,f}(x_t, u_t) \qquad (1)$$

for each cell c and each fluid f, where

x_t	– Field properties $[p_o, s_w, s_g]$
$M(x_t)$	– Fluid mass in cell
$F(x_t)$	– Mass flow/to from neighbor cells
u_t	– Well controls
$Q(x_t, u_t)$	– Well flows

(2)

In a real reservoir model there may be up to 10^6 cells and 10^3 time steps, so the number of *nonlinear* equations to be

---

[1]The term "end-to-end" refers to a modeling approach in which a proxy completely replaces the original system, i.e., only accesses its inputs and outputs. This term has been used previously with neural approaches in the context of speech recognition, machine translation, and other applications.

solved simultaneously can be on the order of 10^9, although some practical problems are smaller. Depending on the degree of the nonlinearity and the length of the time step, the set of equations to be solved for each time step can require many iterations of a nonlinear equation solver. A high-fidelity reservoir simulation for industrial applications can take hours or days to complete one simulation.

3.2. Reservoir Uncertainty

Each cell has a rock type, for example sandstone or shale, which defines important properties, such as compressibility and permeability. However, due to the depth of reservoirs and the complex geology surrounding their formation, the attribution of a rock type to any given cell is somewhat uncertain.

An important notion is one of a *realization*, representing a particular spatial distribution of rock type across the grid cells of the RM—a distribution that comes with a natural uncertainty (Lie, 2014). This can be modeled by three-dimensional probability density functions, called variograms, that are calibrated from geological knowledge, seismic data, and cores from test wells. In a typical application, analysts will work with a few hundred samples (referred to as *Realizations*). To take this uncertainty into account in our experimental setup, we have used 500 realizations generated from the original RM according to Caers and Zhang (2004) with subsequent porosity and permeability calculation described in Mariethoz and Caers (2014).

3.3. Wells

Another important element in BOS are *wells*. They have two functions: (1) the production of commercially valuable fluids (referred to as *Producer* wells), and (2) the management of pressure differentials and fluid saturations in the reservoir. Some wells are drilled specifically to inject water or gas (referred to as *Injectors*).

The simulator predicts the impact of well operations which consist of two types of decisions: (1) the location and time sequence of well completions, and (2) the control of the production or injection rate. The ultimate goal is to maximize the expected NPV.

3.4. BOS Implementations

Our work relies on the following BOS packages: (1) *Open Porous Media* (OPM)—an open source project supported by a consortium of companies and academic institutions (OPM, 2018), and (2) *Eclipse* [Eclipse (Schlumberger Company), 2018]—a commercial software.

4. SPE BENCHMARKS

Our study draws data from one (of several) RM considered reference within the oil and gas industry: the SPE9 model (Killough, 1995; SPE9, 2018). The model consists of a $24 \times 25 \times 15$ grid and we used a varying number of injector and producer wells as will be described later. One particular realization of the SPE9 RM is shown in **Figure 1** illustrating the shape (and incline),
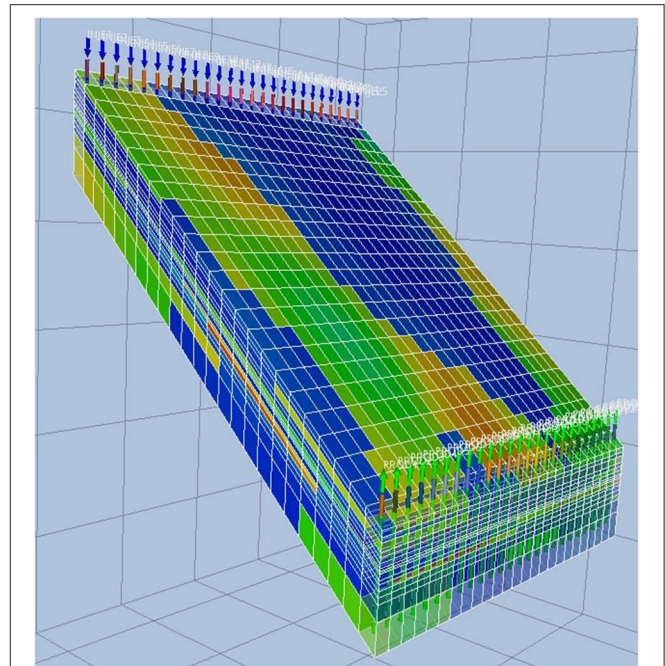

FIGURE 1 | An example realization of the SPE9 RM (water pressure distribution).

grid structure and property (water pressure in this example) distribution within the RM.

5. LEARNING TO PREDICT PRODUCTION RATES—THE PROXY APPROACH

5.1. Task

The main targets for the proxy model to predict are a set of variables referred to as *production rates*, taking into account interactions between wells, fluid, and rock. More specifically, given a particular RM realization and a series of *drilling actions*, a simulator predicts rates at which the wells produce or inject fluids at future times (referred to as time steps). A major application of such simulations is in Field Development Planning (Jahn et al., 2008), where the total production rates constitute economic metrics, such as the Net Present Value (NPV). Individual rates of each well drilled may also be of importance and we will include these in our investigations as well.

Considering the above, we define the task of proxy modeling as follows:

Given:

1. Sequence of actions. An action encapsulates a drilling decision, in our case one of {"Drill a Producer Well (P)," "Drill an Injector Well (I)," "Do Nothing (X)"}, accompanied by applicable coordinates on the reservoir surface.
2. Realization ID. This ID maps to a known distribution of rock properties in the reservoir.

Predict:

1. Field rates, i.e., aggregate output of the entire field, typically including oil, and water production as well as water injection over a desired future horizon in (equidistant) time steps.
2. (Optional) Individual well rates, i.e., individual output of each producer well placed so far, typically oil production is of interest.

An example of an action sequence suitably encoded could be as follows:

$$x - P(0, 15) - I(2, 5) - x - x - P(23, 19) - \ldots - x$$

along with an integer identifying the realization. In this example, P(x,y) and I(x,y) refer to a Producer and Injector wells being placed at surface grid coordinates (x,y), and "x" represents a "Do Nothing" action at the particular time. Thus, `P(0, 15)` refers to drilling a Producer at surface location (0, 15), etc. Continuing on the above example, an expected output could be a series of rates predicted at increasing time steps, each representing a 30-day interval. In our experiments we used action sequences of length 20, and varying prediction spans (horizon) of 20–40 months.

5.2. Model
5.2.1. Encoder-Decoder Architecture
Given the above task definition, our method adopts the sequence-to-sequence approach based on recurrent neural networks (RNNs), specifically a version of a gated RNN known as Long-Short Term Memory (LTSM) cell (Hochreiter and Schmidhuber, 1997), arranged in a encoder-decoder architecture. This approach has gained popularity across various application fields (Graves and Jaitly, 2014; Sutskever et al., 2014; Vinyals et al., 2017) and excels at modeling temporal sequences of (typically discrete) variables. In our setting, however, the model performs a continuous-variable regression, as will be explained more in detail below. **Figure 2** illustrates our solution. In this view, the series of drilling actions, along with additional geological information is the input sequence, $X = \{x_i\}_{0 \leq i < K}$, while the series of production rates is the output sequence, $Y = \{y_t\}_{0 \leq t < T}, y_t \in \mathbb{R}^D$, where D is the number of fluid phases. Here, x_t represents a union of potentially heterogeneous features, such as discrete and multivariate continuous variables. Later we will discuss how X is transformed into a common space via embeddings.

While typical encoder-decoder applications deal with a discrete output defined over a closed set (of classes), i.e., modeling class probabilities with a training objective being a likelihood function (cross-entropy), in our setting the model performs a

FIGURE 2 | An architecture overview of the model.
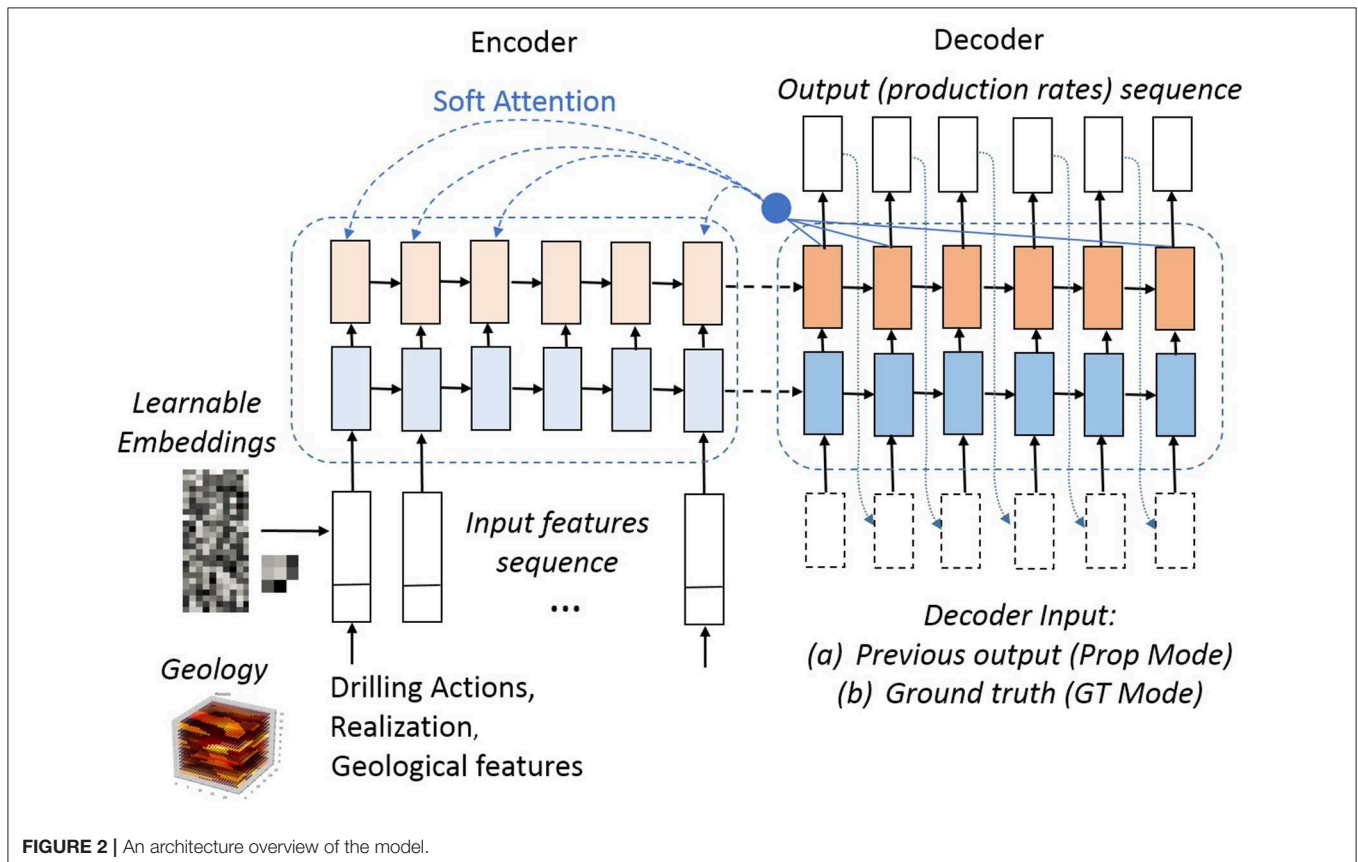
regression in $\mathbb{R}^D$ and is trained with the objective of minimizing the mean squared error (MSE).

The role of the Encoder is to capture the information about the input X and pass it to the Decoder to generate an accurate prediction $\hat{Y}$ about Y. Both the encoder and the decoder involve multiple layers of LSTMs in our architecture.

5.2.2. Neural Attention Mechanism

Our model adopts the attention mechanism introduced in Bahdanau et al. (2014) to aid modeling the causal nature between the actions and the output. In order to generate a prediction $\hat{y}_t$, the attention mechanism applies a probability distribution (a "mask"), $\{\alpha_{t1}, ..., \alpha_{tK}\}$, over the individual input actions to emphasize actions particularly relevant to produce y_t. As a result, a new context variable, c_t, is synthesized within the decoder at time t as follows:

$$c_t = \sum_{i=1}^{K} \alpha_{ti} h_i \qquad (3)$$

where $\{h_1, ..., h_K\}$ are the internal states of the encoder corresponding to the input X. The context c_t is then appended to the internal decoder state. The mask, α, itself is a softmax function of a link (or "alignment") function, l_{ti}:

$$\alpha_{ti} = \frac{e^{l_{ti}}}{\sum_{j=1}^{K} e^{l_{tj}}} \quad . \qquad (4)$$

l_{ti} models a relationship strength between the previous, $(t-1)$-st, decoder (output) state and the i-th encoder (input) state (Bahdanau et al., 2014). The link function is modeled as a feed-forward neural network and is trained jointly with the rest of the encoder-decoder parameters. In contrast to a typical setup where the attention is applied on the upper-most encoder layer, we have found that concatenating encoder states from all M layers, i.e., for the i-th position in the input X, $h_i := [h_i^{(1)}, ..., h_i^{(M)}]$, outperformed the former.

5.2.3. Decoding Variants

We investigate the following modes of operation with respect to the decoder:

- Ground Truth (GT) Training, in which the decoder is provided the GT, i.e., $\{y_t\}_{0 \leq t < T}$ as input at each time step.
- Prediction Propagation Training (or *Prop Training*), where in order to generate $\hat{y}_t$, the decoder uses its own prediction $\hat{y}_{t-1}$ as input (dotted decoder connections in **Figure 2**)
- *GT Pre-Training*, in which the model is first trained in GT Training mode, followed by Prop Training. This relates to an idea proposed in Bengio et al. (2015).
- Hybrid Propagation (or *HybridProp*), in which first k time steps are performed in the GT mode, followed by decoding the rest in Prop mode. Note that we utilize this mode in both training as well as inference (test). This mode is a novel variant with benefits demonstrated in section 5.6.6.

TABLE 1 | Datasets and their partitioning.

Name	Partition size (simulations)		
	TRAIN	**VALID**	**TEST**
OPM-22k	17,600	2,200	2,200
ECL-32k	28,682	3,585	3,585
OPM-163k	130,778	16,347	16,347

5.3. Experimental Evaluation

5.3.1. Datasets

In order to develop various aspects of the neural network architecture, we have used the two BOS mentioned in section 3, namely, OPM and Eclipse. While the OPM was used in majority of the tests, Eclipse was used to generate simulations with well control optimization—a feature not yet available in OPM.

Two main datasets were created: (a) 22k simulations[2] via OPM (fixed well control), and (b) 32k simulations via Eclipse (well control optimized). Each such set was partitioned into training (TRAIN), validation (VALID), and test (TEST) sets, as shown in **Table 1**, maintaining a proportion 80, 10, and 10%, respectively. Additionally, a large dataset involving 163k OPM simulations (OPM-163k) was collected to examine effects of varying training set size on the resulting error metrics.

We want to emphasize that the OPM-163k only serves exploratory purposes of this paper and, in its size, would unlikely be a practical choice due to the considerable computational burden required to generate the corresponding simulations. In selected experiments (section 5.6.2), the OPM-22k was further downsampled to investigate dependency on the training size.

Each simulation input consists of a uniform random choice over 500 realizations (see section 3) and an action sequence of length 20. For each of the 20 actions, a decision to drill was made with 99% probability, with a ratio 5 : 1 in favor of drilling a producer. The well location follows a uniform random distribution with the constraint of not drilling within a 2-cell neighborhood of pre-existing wells.

5.4. Features and Preprocessing

5.4.1. Actions and Realization

The primary information to be encoded in the input $X = \{x_k\}_{0 \leq k < K}$ are the drilling decisions (see section 5.1). Since the SPE9 reservoir has dimensions $24 \times 25 \times 15$ grid cells, initially, a joint action-location encoding was defined on a discrete set of 1201 (2 "drill" actions $\times$ 24 x-coordinate values $\times$ 25 y-coordinate values + 1 "no drill" action) symbols. An alternative separate encoding for the action (3 symbols), the x-coordinate (24 symbols), and the y-coordinate (25 symbols), however, outperformed the joint encoding in our experiments, as will be shown below.

A realization ID associated with an input sequence is attached to each x_k as a discrete variable.

---

[2]The unit of "simulation" encompasses the series of outputs (time steps), given a well schedule.

In order to provide the neural network with the above input, the discrete variables are "embedded" in a continuous space of certain dimensionality (a hyperparameter), similar to a method of embedding words in Natural Language Processing (Mikolov et al., 2013). The embedding vectors for each discrete variable are initialized to random values and are trained jointly with the rest of the network. Negligible sensitivity was observed due to the dimensionality hyperparameter. Values we used are listed in **Table 2**.

5.4.2. Geological Features

Hypothesizing that local geological properties influence the flow through the wells the most, we added features related to the neighborhood of the well locations. More specifically, the following local (per-cell) features were considered: (a) rock type (shale, sandstone), (b) porosity, (c) permeability in horizontal and vertical direction. In our case, 5 cells at the bottom (for injectors) and 5 at the top (for producers) are affected by drilling, resulting in 20 real-valued features concatenated to form the final feature vector (see **Table 2**).

5.4.3. Standardization

Standardization was applied to the output variables before modeling, via a linear transform $y'_{(k)} = (y_{(k)} - \mu_k)/\sigma_k$ with $y_{(k)}$ being the k-th dimension of the output vector and μ_k, σ_k denoting the mean and standard deviation estimates obtained from the corresponding dimension over the TRAIN partition. The variables y' are transformed back to their physical range using the inverse transform. All experimental evaluation is then performed on unnormed output.

5.4.4. Error Metric

The core metric in our experimentation is the prediction error relative to target (GT) generated by the simulator. We base this metric on an L2 norm. Let $Y_k = \{y_{kt}\}_{0 \leq t < T}$ and $\hat{Y}_{kt} = \{\hat{y}_{kt}\}_{0 \leq t < T}$ denote target and predicted values for a simulation k of length T. Let K denote the total number of simulations in the test set. While the reservoir simulator produces rates at each time step, for practical use we are interested in the *cumulative* output. Hence, the cumulative value $z_{kt} = \sum_{\tau=0}^{t} y_{k\tau}$ is calculated forming a vector $\mathbf{z}_k = [z_{k0}, ..., z_{kT-1}]$ (and similarly for $\hat{z}_{kt}$). The (relative) error of a simulation k is then defined as follows

$$e_k = \frac{||\hat{\mathbf{z}}_k - \mathbf{z}_k||_2}{||\mathbf{z}||_2} \tag{5}$$

TABLE 2 | Input features with their type and dimensionality.

Input	Type (cardinality)	Dim
Well type	Discrete (3)	3
Location X	Discrete (24)	10
Location Y	Discrete (25)	10
Realization	Discrete (500)	20
Geology	Continuous	20

Note that the denominator, **z**, is calculated as an average over the entire test set. This mitigates issues with near-zero targets that occur in some valid cases. The final error rate used for reporting in this paper is the average of e from Equation (5) over the entire test set.

5.4.5. Training Procedure

The system was implemented in Tensorflow (Abadi et al., 2016). Each model was trained using the TRAIN partition (see **Table 1**) to minimize the sequence MSE via the Adam optimizer (Kingma and Ba, 2014), performed in batches of 100 simulations and an initial learning rate of 0.001. Parameters with the best loss on the VALID partition were then chosen. In case of GT Pre-Training (see section 5.2.3), the resulting parameters served as starting point for the next training round with a slower learning rate of 0.0002. Trained models were evaluated using the TEST partition.

We experimented with varying model size in terms of the number of LSTM layers (ranging between 1 and 5), and hidden units (ranging between 32 and 2,048). In the experimental sections below we report results on three representative configurations by memory footprint ("#units × #layers"): 1024×2 (large), 128×5 (medium), and 64×1 (small).

5.5. Baselines

We consider several traditional techniques as references for comparison, e.g., upscaling (Durlofsky, 2005) offers itself as a suitable baseline to achieve acceleration due to reduced refinement and complexity (see below). We also describe two additional simple machine learning baselines as references.

5.5.1. Upscaling

While oil reservoir fluids flow through microscopic pores, practical reservoir models, such as SPE9 in **Figure 1**, have homogeneous properties over tens of meters. These coarser models are constructed attempting to approximate and analyze reservoir flow performance of multiscale nature with the available computational power. To this end, upscaling consists of a set of procedures to obtain coarser reservoir models for flow performance prediction from geological characterizations which typically contain $10^7 - 10^8$ cells (Durlofsky, 2005). Extrapolating from this idea, a reasonable question to ask is how the proxy models based on Neural Networks perform compared to even coarser versions of the base case reservoir model obtained with simple upscaling procedures.

We apply single-phase upscaling based on averaging reservoir properties aiming at a comparison of accuracy and performance with the proxies. Given the base grid of $24 \times 25 \times 15$ grid-blocks we constructed two coarser grids of $12 \times 13 \times 15$ and $8 \times 9 \times 15$ grid-blocks, referred to as "UP2" and "UP3," respectively. The coarser grids average four ($2 \times 2 \times 1$) and nine ($3 \times 3 \times 1$) horizontal neighbor blocks in the "UP2" and "UP3" case, respectively. All the relevant spatial reservoir properties, namely pressure, saturation, absolute permeability and porosity, are averaged on the pore-volume of the union of cells. These averaging computations are performed in negligible

computational time when compared to the solution of non-linear equations for simulation. Finally, the coarser reservoir description is simulated with the BOS as usual.

While an increased error relative to the refined base model is anticipated, it should be pointed out that the upscaling procedure does not require any data to train or tune its parameters. In this respect, upscaling offers an advantage over learning proxies in use cases where there is an absolute lack of such training data.

5.5.2. Fixed Predictors
Two simple baselines using fixed predictors are created as follows:

- A predictor that at all times outputs the mean value for each variable as seen in the training data (referred to as "Mean Baseline");
- A predictor outputting the mean vector of all observations at the corresponding time step in the training data (referred to as "Time-Step Mean," or "TSM"). In other words, this predictor generates average flow curves for each component, as observed in the corresponding TRAIN partition.

The two baselines above use the same training data as the proxy model and thus offer helpful calibration points. Obviously, the two fixed predictors above incur only negligible latency.

5.6. Experimental Results
5.6.1. Results on OPM-22k
Calculated according to Equation (5), **Table 3** summarizes error rates for the essential techniques described in previous sections, using the $1,024 \times 2$ model. For comparison, we also compute the simple baselines proposed in section 5.5. "Mean Baseline" and "TSM Baseline" have an error of 43.1% and 39.3%, respectively, whereas the upscaling cases "UP2" and "UP3" have an error of 19.1% and 32.3%, respectively. Then, starting with the basic encoder-decoder at 21.5%, pre-training a model in GT mode, followed by Prop training at a slower learning rate results in a significant decrease of error rate—by 6.2% absolute to 15.3%. Replacing jointly encoded action-location input (see section 5.4.1) by factored action-location information and by adding geological features (see section 5.4.2) decreases the error rate further to 12.2%. Finally, the attention mechanism attending to all encoder layers achieves an error rate of 10.3%. For comparison we also give results for the more standard attention setup using the top layer which is inferior at 11.2%. All differences are statistically significant with $p < 0.001$ on a Wilcoxon signed rank test. Later on, we refer to the multi-layer attentive, GT-pretrained models using a prefix M, e.g., as M1024×2.

In order to gain insight into how the attentive decoder focuses with respect to the input, **Figure 3** shows two example attention masks, i.e., the weighting distributions $\{\alpha_{kt}\}_{0 \le k < K}$ at each decoder time step t over the input action space. In each panel, the horizontal axis corresponds to decoder time and the vertical axis to well drilling sequence (abbreviated as $\{I, P\}\#(x, y)$). It is interesting to note that while there is a fairly linear alignment between the decoder focus (light areas) and the input actions in the Prop mode (a), the bulk of attention is dispersed over an initial third of the input sequence

TABLE 3 | Sequence error rates of various configurations of the 1024x2 model on the OPM-22k TEST partition.

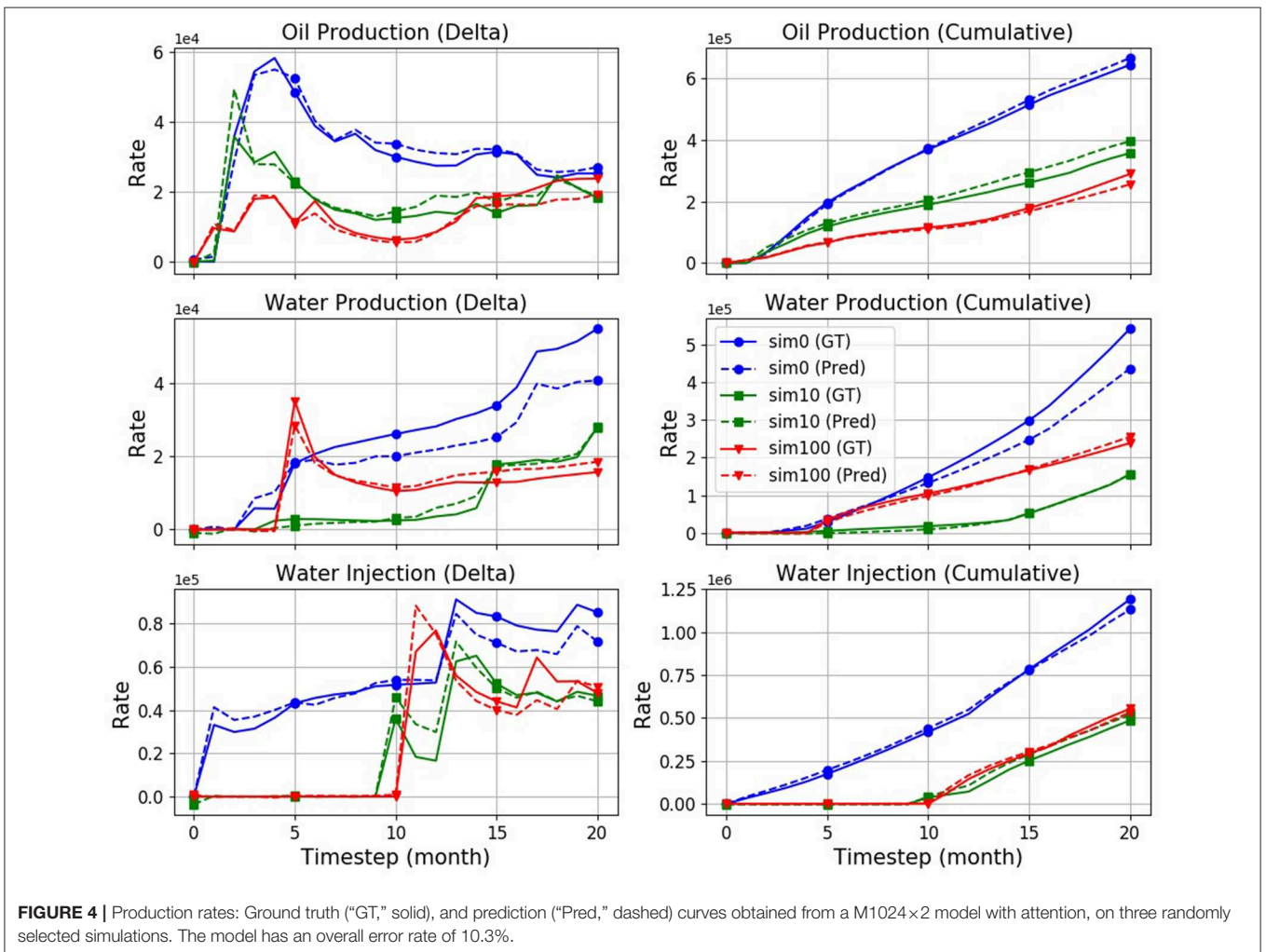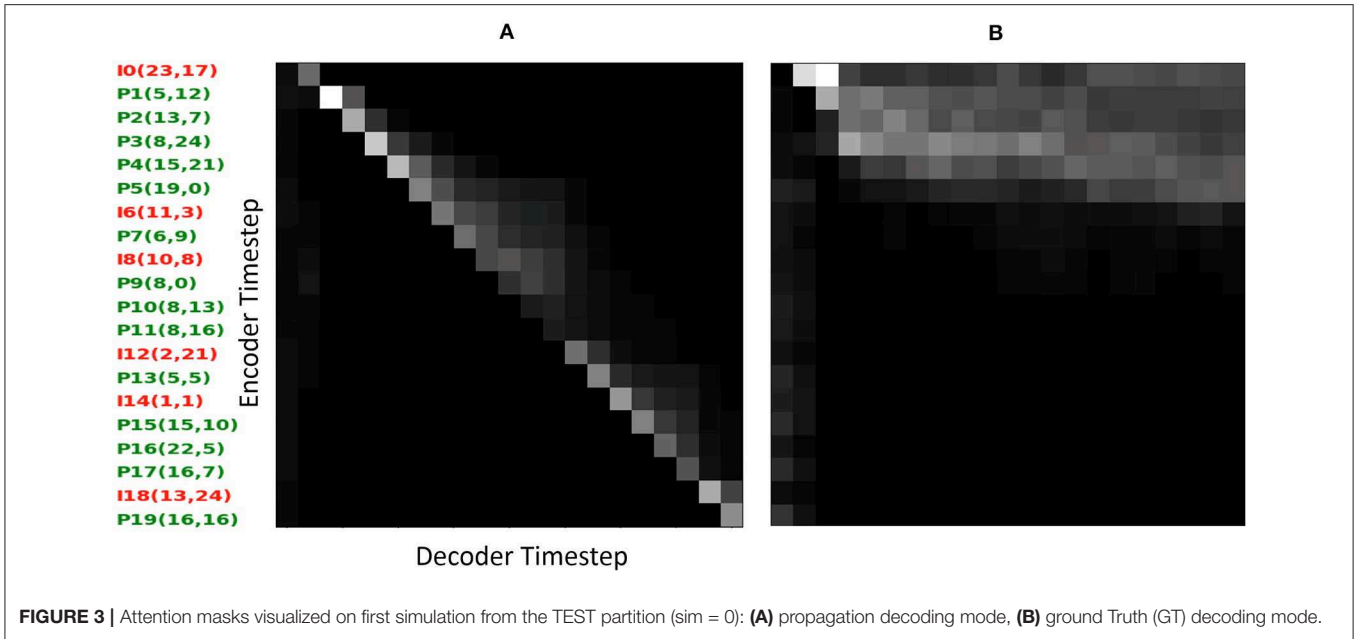| Configuration	Sequence error (%)
Mean baseline	43.1
TSM baseline	39.3
Upscaling UP3	32.3
Upscaling UP2	19.1
Encoder-decoder (1,024 $\times$ 2)	21.5
+ GT Pre-training	15.3
+ Factored geol. features	12.2
(+ Single-layer attention)	11.2
+ Multi-layer attention	10.3

when the decoder is given the GT at every time step (b). We hypothesize that the sharper action alignment emerges as the decoder obtains strong signal from the action sequence, while consuming its own, error-prone prediction as input. In contrast, the GT decoding mode supplies the decoder with accurate input making the decoder mostly rely on that, rendering the attention less important.

Figure 4 compares target (GT) and predicted production rates in three randomly selected simulations drawn from the TEST partition (sim: 0, 10, and 100). The first column in **Figure 4** shows direct output of the model, i.e., rates (in barrels) for a given time period (30 days), while the second column compares the corresponding cumulative curves. The latter eventually serves the NPV calculation of the field as mentioned in section 5.1.

Figure 5 compares predicted oil production rates curves across the various baselines on an example of the first ("sim0") simulation, namely the "Mean," "Time-Slot-Mean (TSM)," upscaled "UP2," and "UP3" baselines as well as the proxy. As can be seen, the proxy tends to approximate the ground-truth (GT) flow most accurately, followed by the UP2 baseline. The UP3, Mean, and TSM baselines tend to be considerably further off the target.

5.6.2. Varying Training Size
To assess the dependency of the prediction error on training data amount we conducted an experiment with subsampling the TRAIN partition to smaller amounts. **Figure 6** shows the results: starting from the main OPM-22k result (see above) with 17.6k simulations, the training amount was halved repeatedly to 1.1k simulations—an amount extremely small considering that there are 500 different reservoir realizations to be covered. On the other extreme, the training set was augmented by the remaining simulations from OPM-163k yielding a TRAIN partition with about 130k simulations. In **Figure 6**, two trends are evident: (1) models with higher parameter complexity tend to outperform lower-complexity ones for larger training amounts, with trend reversal at the lower end of the x-axis, and (2) there is a roughly linear relationship between the error rate and the log-scaled training amount. The accuracy of the rel. small 64×1 model is remarkably competitive across the conditions.
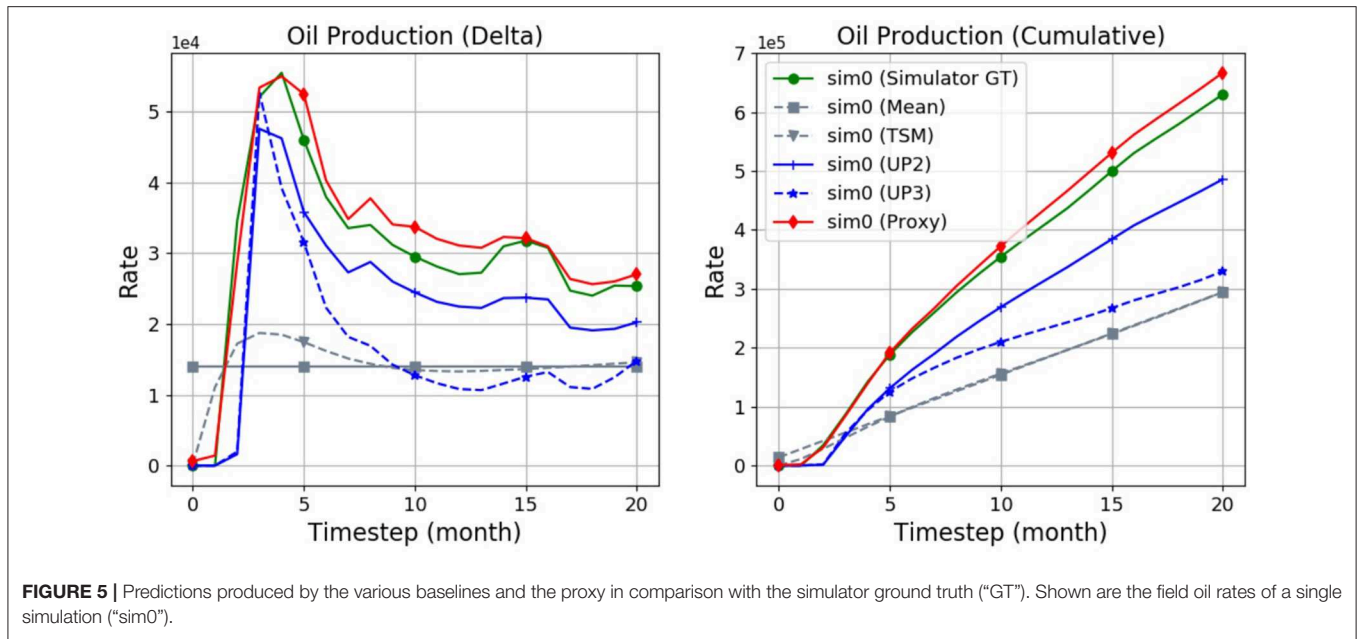
FIGURE 3 | Attention masks visualized on first simulation from the TEST partition (sim = 0): **(A)** propagation decoding mode, **(B)** ground Truth (GT) decoding mode.

FIGURE 4 | Production rates: Ground truth ("GT," solid), and prediction ("Pred," dashed) curves obtained from a M1024×2 model with attention, on three randomly selected simulations. The model has an overall error rate of 10.3%.

FIGURE 5 | Predictions produced by the various baselines and the proxy in comparison with the simulator ground truth ("GT"). Shown are the field oil rates of a single simulation ("sim0").
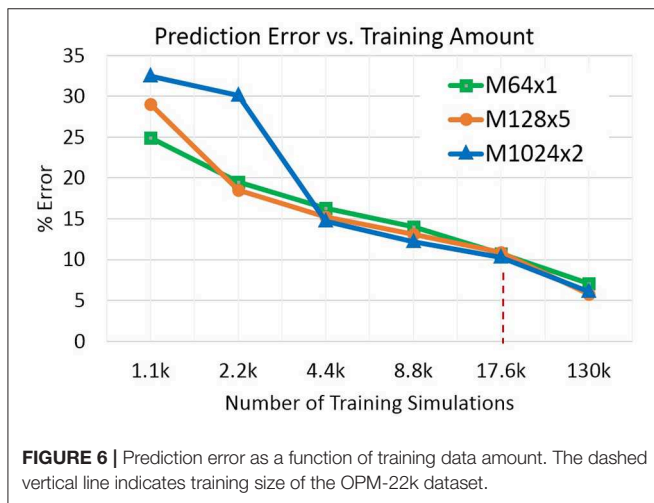

FIGURE 6 | Prediction error as a function of training data amount. The dashed vertical line indicates training size of the OPM-22k dataset.

5.6.3. Timing Comparison

The main goal of the proxy is achieving significant acceleration compared to the physics-based simulation. Having established accuracy figures in the experiments above, we now turn to the question of performance (timing). We want to compare the average time needed to generate a simulation with 20 actions and 20 outputs, as above, between the OPM simulator and a selected set of proxy models. For the timing experiments we use OPM (2107.10 release) `ebos_2p` which we found to be the fastest implementation with an executable optimized for speed. Benchmarking was conducted on a server equipped with 4 × E7-4850 v2 CPUs @ 2.30 GHz (48/96 physical/virtual cores), 256 GB RAM, Open MPI implementation of MPI (for `mpirun`; version: 1.2.10) and standard `multiprocessing.Pool` objects (from Python, version: 2.7.14). A comparison is somewhat complicated

by the fact that (1) the OPM simulator is used as a black box and timing includes its initialization, and (2) the OPM code does not support the use of a GPU thus leaving only CPU comparison. (1) has a relatively minor effect as, judging from logging, the initialization stage of OPM takes a negligible fraction of the total time.

Table 4 summarizes the benchmarking results. We distinguish three devices of practical interest: (a) CPU 1-core, (b) CPU 8-core, and (c) single GPU (NVIDIA Tesla K80). All values in **Table 4** are durations (in milliseconds) of a simulation averaged over 100 measurements using different realizations. The first row shows the OPM time on a single core as well as an 8-core CPU (employing MPI), resulting in an average runtime of about 10 and 4.3 s/sim, respectively. The next three rows give the timing for the three model sizes benchmarked across the three devices. A clear advantage of the small-footprint `M64×1` model emerges, in particular in the 1-Core case achieving a speedup factor of 2343X over the OPM simulator. The GPU seems to provide an advantage only in case of the largest model.

The last block of three rows shows simulation time in a case where 100 input sequences (simulation requests) can be batched up at once, thus allowing for the respective device to better utilize matrix operations. This sort of batching is of practical use in certain applications (e.g., Field Development Optimization using Monte Carlo methods). A significant speedup (0.1 ms/sim) can be seen now on the GPU side taking full advantage of its internal memory and architecture.

In a separate experiment, performance gains due to upscaling are assessed on a single core of an Intel Xeon CPU E5-2680 v2 @ 2.80GHz contrasted with Eclipse 2011.1 [Eclipse (Schlumberger Company), 2018] (we experienced difficulties getting the desired upscaling setup working using OPM). In this setting, the average running times of the OPM-22k TEST partition are 5,125 (ms) when performing no upscaling, 1,489 (ms) when performing the

TABLE 4 | Timing results.

Test	CPU device		GPU device
	1-Core	8-Core	1-GPU
	Simulation time (ms)		
OPM ebos_2p	10,310	4,340	n/a
M64×1	4.4	4.9	12.1
M128×5	34.1	18.9	27.4
M1024×2	299	330	31.6
M64×1 Batch 100	2.9	0.6	0.1
M128×5 Batch 100	32.2	5.1	0.6
M1024×2 Batch 100	170	25.2	1.8

All values are averages over 100 measurements using same random set of simulations.

TABLE 5 | Error rates of two models on the extended-time horizon (OPM-22k-E) and regular-horizon (OPM-22k) data with the corresponding models, M128×5-E and M128×5.

Testset	M128 × 5	M128 × 5-E
OPM-22k	10.8%	11.1%
OPM-22k-E	13.0%	10.6%

TABLE 6 | Error rates comparison between the wells model predicting 20 wells + 3 field rates and the best field-rates-only model on OPM-22k.

Model	Error rate (OPM-22k)		
	Total	Field rates	Well rates
TSM baseline	51.1%	39.3%	53.0%
UP3 baseline	24.9%	32.3%	23.8%
UP2 baseline	16.9%	19.1%	16.6%
M64×5+W	18.3%	14.0%	19.0%
M1024×2	10.3%	10.3%	n/a

UP2, and 1,155 (ms) the UP3 upscaling. Thus, the speed-up with upscaling ranges from 3X to 4X.

Another important (and advantageous) aspect of the proxy benchmarking results is their consistency. For instance, the M64×1 single-core result is 4.4 ± 0.1 ms/sim which is a range of ±3% relative to the mean. The corresponding OPM measurement of $10,310 \pm 4,030$ ms/sim exhibits a considerably larger range of ±39%, which is typically caused by convergence issues for certain action sequences and realizations. This large variability is also observed when running the upscaled cases.

Overall, considering the error patterns associated with each model size (see **Figure 6**), the small proxy model M64×1 offers an acceptable error rate while achieving the highest speedups.

5.6.4. Extrapolating to a Longer Simulation Horizon
An interesting question arises regarding the proxy model's extrapolation capability. Suppose we are given a model trained with 20 input actions and 20 months worth of output. We now want to run the model for an extended period, say, 40 months. How well does such a model generalize in comparison to a model trained on 40 months worth of ground truth? To investigate, we generated simulations identical to those in OPM-22k but with output extended to 40 time steps, i.e., 1,200 days, (labeled OPM-22k-E) allowing a direct comparison of such two models. **Table 5** summarizes the results in terms of average error rates, with M128×5-E denoting a model trained on OPM-22k-E and M128×5 one trained on OPM-22k, as before. The first row in **Table 5** shows that both models perform comparably on OPM-22k, with M128×5-E having no troubles to predict a shortened horizon of 20 time steps. The more interesting case of M128×5 extrapolating to 40 time steps is shown in the second row, where it achieves an error rate of 13.0%—a moderately elevated error over the matched M128×5-E at 10.6%.While this increase is statistically significant, it seems to be sufficiently limited for us to conclude that the model has a reasonable capability to extrapolate to longer horizons not seen during its training. Furthermore, it is reassuring to observe the error rates of both models tested on the shorter time horizon perform equally well. This suggests that training on data with longer horizon is beneficial, whenever possible.

5.6.5. Modeling Individual Wells
All experiments so far involved predicting *field* rates, i.e., total production and injection rates over all wells. In some applications, however, more detail may be needed. In one of our use cases (an optimization of a Field Development Plan) predictions of all producer wells were required. This corresponds to adding 20 new output variables each mapped to wells in the order of drilling (as there may be up to 20 producers). We then train the model using OPM-22k via same steps as before resulting in a model with 23-dimensional output.

A comparison between the best wells model (using suffix "W," i.e., M64x5+W) and the best field-rates-only counterpart (M1024×2) is given in **Table 6**. Clearly, the task has become considerably more challenging as the wells model must maintain accuracy across 23 variables. Compared on the field rates only, the wells model is at 14%, while the dedicated model is around 10%, and at this error rate it still outperforms all other baselines. The error over the 20 well rates is at 19% for the proxy compared to 16.6% and 23.8% for the UP2 and UP3 baselines, respectively. **Figure 7** shows a small sample of individual well predictions along with their ground truth. The output variables map to each producer in the order of drilling which corresponds to the timed onset of production visible in the cumulative curves. We observed that the model tends to overestimate wells that remain non-productive (zero output for entire simulation), albeit by relatively small amounts.

5.6.6. Well Control Optimization
Optimizing the well controls is an enhancement currently only available in the commercial Eclipse simulator (see section 3). The simulator runtime is typically increased by a factor of 5-10 due to the optimization step. We want to assess the ability of the proxy model to capture the optimization implicitly from control-optimized simulations. The dataset Ecl-32k (see **Table 1**) serves this purpose.
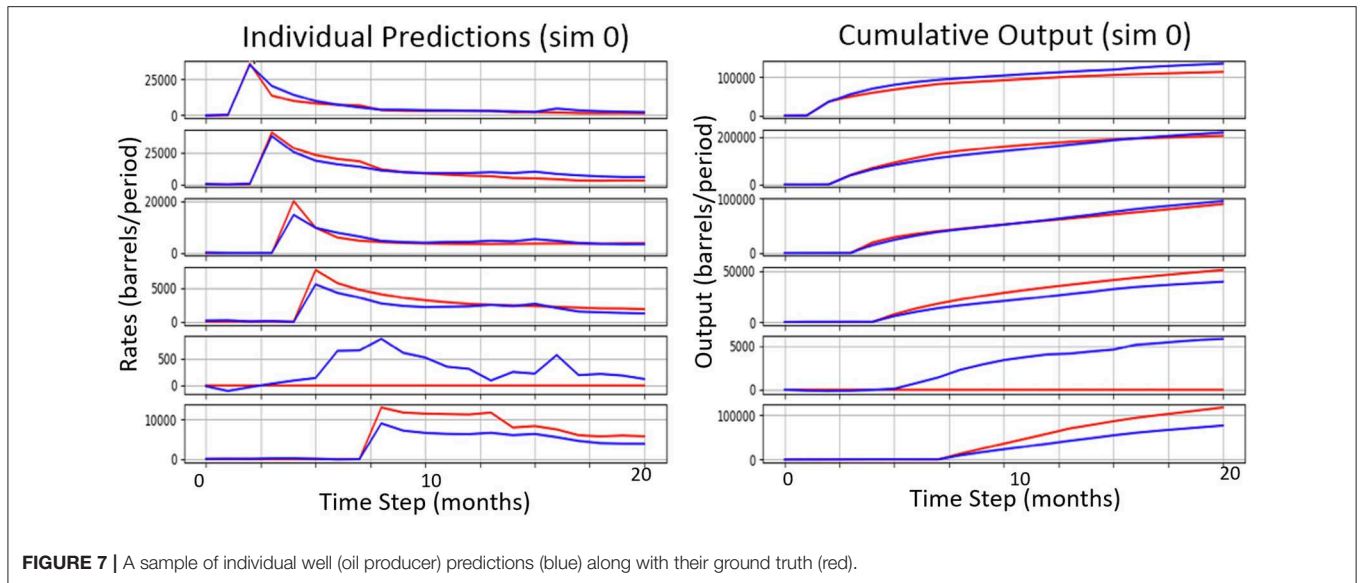
FIGURE 7 | A sample of individual well (oil producer) predictions (blue) along with their ground truth (red).

TABLE 7 | Error rates on Ecl-32k using a model without and with additional simulator input at prediction time.

Model	Error rate (Ecl-32k)		
	Total	Field rates	Well rates
TSM Baseline	51.0%	41.5%	52.4%
M128×5+W	27.1%	34.7%	29.9%
+ HybridDec-1	22.8%	27.5%	22.1%
+ HybridProp-1	15.8%	21.3%	14.9%
+ HybridProp-2	15.4%	21.0%	14.6%
+ HybridProp-4	15.1%	21.5%	14.2%

We start with a model M128×5+W trained on Ecl-32k. In comparison to levels seen with OPM-22k, the resulting error rates in the first row of **Table 7** lie considerably higher, at 27.1%. An inspection of the model output (see **Figure 8**, dashed curve) reveals a rel. high prediction error at time step 1 propagating further during decoding. An in-depth variance analysis, which is omitted here due to space constraints, also reveals a high variability of the GT at time step 1 when (and only when) the optimization is present. This first step appears to be challenging for the decoder to predict accurately. This observation motivated the idea behind HybridProp described in section 5.2.3. In HybridProp, the decoder is given 1 or several GT frames as input both at training and test time to generate the rest of the sequence using regular propagation. An experimental assessment of this method is shown in **Table 7**. With M128×5+W at 27.1% error, giving the same model a single frame of GT during decoding only (referred to here as HybridDec) improves the error rate by 4.3%. However, when the model is *retrained* with the same modification (HybridProp), the error declines dramatically, to 15.8%. Providing more than one GT frame seems to yield diminishing returns. Despite the added requirement of having a simulator available at test time, this a remarkable and a practical

result. Returning to **Figure 8**, after applying HybridProp1 the original (dashed red) curve moves down (solid red) to a trend comparable to one seen on non-optimized data (OPM-22k). The HybridProp method was also tested on models trained on OPM-22k and observed only negligible improvements further confirming our hypothesis regarding the high variance observed at first time step being unique to the well control optimization process. While the impact of the HybridProp1 method on error reduction is significant, it should be pointed out that the added requirement of generating one (or several) frames per simulation has a negative impact on the expected acceleration rate. Assuming the overall simulator runtime is proportional to the number of output frames, the need to generate 1 frame per simulation will reduce the original acceleration rate (α) to a value $\frac{\alpha}{1+\alpha/T}$, where T is the number of output frames. In our case, considering an acceleration rate of $\alpha = 20,000$ for simulations with $T = 20$ output frames with well-control optimization, the final acceleration will become 40X. Obviously, this figure will improve with longer horizons.

6. CONCLUSION

The series of experimental results presented in this paper demonstrates the effectiveness of the described end-to-end proxy approach at accelerating reservoir model simulations, including variability due to well locations and geological uncertainty. We have observed a significant acceleration capability of more than 2000X compared to an industry-strength physics-based simulator OPM. Furthermore, we demonstrated it is possible to approximate the simulations with well control optimization thus offering an additional relative speedup. For practical amounts of training data, the accuracy of the neural network proxies presented here ranges between 10% and 15% error over 20–40 months horizons, relative to the simulator. The model shows a good extrapolation capability. Our proxy model
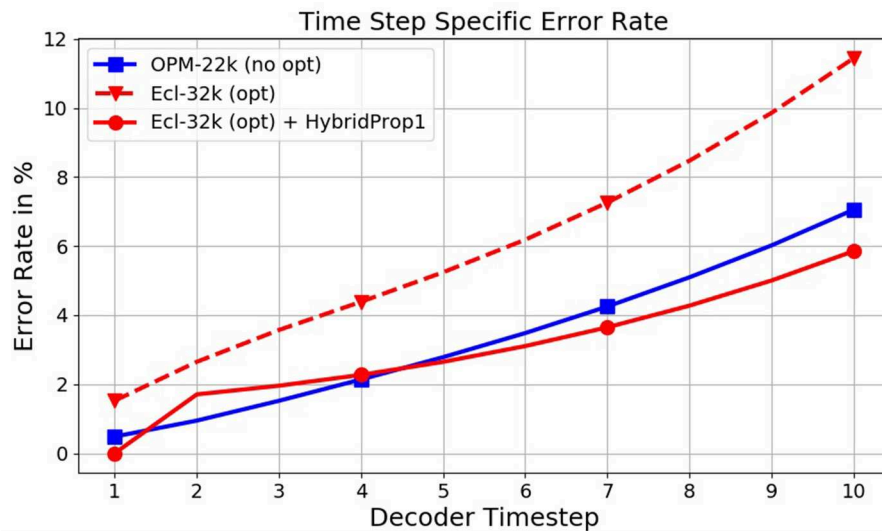
FIGURE 8 | Effect of the HybridProp method on time step specific error rates, with the first time step being crucial.

captures non-linear interactions between wells, fluid, and rock, giving it a great advantage over state-of-the-art commercial techniques. When compared to simple upscaling procedures, we observe that the proxy models are capable to run about 500X faster and provide higher accuracy. We believe these outcomes, generated on a publicly available reservoir, are extremely promising and represent a valuable benchmark for future research in oil field development optimization. Moreover, we anticipate that, due to its application-agnostic nature, the approach is suitable for solving tasks in related fields of energy and environment modeling.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

JN: lead researcher, explored DNN architectures, generated the proxy experimental results and analysis. AK: performed early-stage architecture exploration and analysis. JR: involved in design of the sequence-to-sequence architecture. GK: contributed to SPE9 data generation as well as performance measurement and analysis. RT: involved in early-stage planning of the project leading to the work and results being presented. AC: created the upscaled simulator setup and generated results serving as the main baseline in our analysis.

FUNDING

ACKNOWLEDGMENTS

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). "Tensorflow: a system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (Savannah, GA), 265–283.

Aïfa, T. (2014). Neural network applications to reservoirs: physics-based models and data models. *J. Petrol. Sci. Eng.* 123, 1–6. doi: 10.1016/j.petrol.2014.10.015

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR* abs/14 09.0473.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15,* eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Cambridge, MA: MIT Press), 1171–1179.

Caers, J., and Zhang, T. (2004). Multiple-point geostatistics: a quantitative vehicle for integrating geologic analogs into multiple reservoir models. *AAPG Memoir* 80, 383–394.

Durlofsky, L. J. (2005). "Upscaling and gridding of fine scale geological models for flow simulation," in *8th International Forum on Reservoir Simulation* (Stresa), 1–59.

Eclipse (Schlumberger Company) (2018). *ECLIPSE Reservoir Simulator.* Available online at: https://www.software.slb.com/products/eclipse (Accessed August 29, 2018).

Graves, A., and Jaitly, N. (2014). "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14,* II–1764–II–1772. Available online at: https://JMLR.org

He, J., and Durlofsky, L. J. (2015). Constraint reduction procedures for reduced-order subsurface flow models based on POD-TPWL. *Int. J. Numer. Methods Eng.* 103, 1–30. doi: 10.1002/nme.4874

Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.* 9, 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Jahn, F., Cook, M., and Graham, M. (2008). *Hydrocarbon Exploration and Production.* Developments in Petroleum Science. Amsterdam: Elsevier Science.

Jansen, J. D., and Durlofsky, L. J. (2016). Use of reduced-order models in well control optimization. *Optim. Eng.* 18, 105–132. doi: 10.1007/s11081-016-9313-6

Killough, J. (1995). "Ninth SPE comparative solution project: a reexamination of black-oil simulation," in *SPE Reservoir Simulation Symposium* (San Antonio, TX: Society of Petroleum Engineers).

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *CoRR* abs/1412.6980.

Lie, K.-A. (2014). *An Introduction to Reservoir Simulation Using MATLAB - User Guide for the Matlab Reservoir Simulation Toolbox (MRST).* Oslo: Sintef ICT, Department of Applied Mathematics.

Mariethoz, G., and Caers, J. (2014). *Multiple-Point Geostatistics: Stochastic Modeling With Training Images.* Chichester, UK: John Wiley & Sons, Ltd.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26,* eds C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Lake Tahoe, NV: Curran Associates, Inc.), 3111–3119.

OPM (Open Porous Media Project) (2018). Available online at: https://opm-project.org/ (Accessed August 20, 2018).

Sagheer, A., and Kotb, M. (2018). Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing* 323, 203–213. doi: 10.1016/j.neucom.2018.09.082

SPE9 (Open Porous Media Project) (2018). Available online at: https://github.com/OPM/opm-data/blob/master/spe9/SPE9.DATA (Accessed August 20, 2018).

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14,* eds Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Cambridge, MA: MIT Press), 3104–3112.

Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. (2016). Accelerating eulerian fluid simulation with convolutional networks. *CoRR* abs/1607.03597.

Trangenstein, J. A., and Bell, J. B. (1989). Mathematical structure of the black-oil model for petroleum reservoir simulation. *SIAM J. Appl. Math.* 49, 749–783. doi: 10.1137/0149044

Valladao, D. M., Torrado, R. R., Flach, B., and Embid, S. (2013). "On the stochastic response surface methodology for the determination of the development plan of an oil & gas field," in *SPE Middle East Intelligent Energy Conference and Exhibition* (Manama: Society of Petroleum Engineers).

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2017). Show and tell: lessons learned from the 2015 mscoco image captioning challenge. *IEEE Trans. Patt. Anal. Mach. Intell.* 39, 652–663. doi: 10.1109/TPAMI.2016.2587640

Wang, C., Ma, Z., Leung, J. Y., and Zanon, S. D. (2018). Correlating stochastically distributed reservoir heterogeneities with steam-assisted gravity drainage production. *Oil Gas Sci. Technol. Revue d'IFP Energies Nouvelles* 73:9. doi: 10.2516/ogst/2017042

Wei, C., Wang, Y., Ding, Y., Li, Y., Shi, J., Liu, S., et al. (2017). "Uncertainty assessment in production forecasting and optimization for a giant multi-layered sandstone reservoir using optimized artificial neural network technology," in *SPE Reservoir Characterisation and Simulation Conference and Exhibition* (Abu Dhabi: Society of Petroleum Engineers).