Industry-scale CO_2 Flow Simulations with Model-Parallel Fourier Neural Operators

Philipp A. Witte
Microsoft
Redmond, WA
pwitte@microsoft.com

Russell J. Hewett
Microsoft
Redmond, WA
rhewett@microsoft.com

Ranveer Chandra
Microsoft
Redmond, WA
ranveer@microsoft.com

Abstract

Carbon capture and storage (CCS) is one of the most promising technologies for reducing greenhouse gas emissions and relies on numerical reservoir simulations for identifying and monitoring CO_2 storage sites. In many commercial settings however, numerical reservoir simulations are too computationally expensive for important downstream application such as optimization or uncertainty quantification. Deep learning-based surrogate models offer the possibility to solve PDEs many orders of magnitudes faster than conventional simulators, but they are difficult to scale to industrial-scale problem settings. Using model-parallel deep learning, we train the largest CO_2 surrogate model to date on a 3D simulation grid with two million grid points. To train the 3D simulator, we generate a new training dataset based on a real-world CCS simulation benchmark. Once trained, each simulation with the network is five orders of magnitude faster than a numerical reservoir simulator and 4,500 times cheaper. This paves the way to applications that require thousands of (sequential) simulations, such as optimizing the location of CO_2 injection wells to maximize storage capacity and minimize risk of leakage.

1 Introduction

Numerical reservoir simulations play a critical role in Carbon Capture and Storage (CCS) and are used to answer important key questions throughout the lifetime of a CCS project [1]:

- Planning: Where are potential CO_2 storage sites?
- Operation: How much CO_2 can safely be injected and stored in a given location?
- Monitoring: How does the CO_2 behave after injection and does it remain in the storage site?

Conventionally, numerical reservoir simulators such as GEOSX [2], Open Porous Media (OPM) [3] or Eclipse [4] are used for modeling subsurface CO_2 flow and for answering the above questions. Most reservoir simulators use the finite volume (FV) or finite element (FEM) method for solving the underlying multi-phase flow equations in porous media. Running numerical simulations for a production-scale storage site is computationally expensive, as it involves solving large-scale linear systems with iterative methods and it thus requires techniques from high-performance computing (HPC) for running simulators on distributed memory architectures. Simulations typically take multiple hours to run at the relevant scales, which is prohibitive for downstream applications that require thousands of simulations in sequence, such as inverse problems or uncertainty quantification (UQ).

Due to the high computational cost of conventional reservoir simulations, data-driven approaches based on deep learning (*Scientific AI/ML*) are increasingly being adopted for numerical CO_2 modeling [5–9]. The promise of deep-learning based reservoir simulations are speedups of multiple orders of magnitudes over conventional simulators, as well as the ability to compute sensitivities and

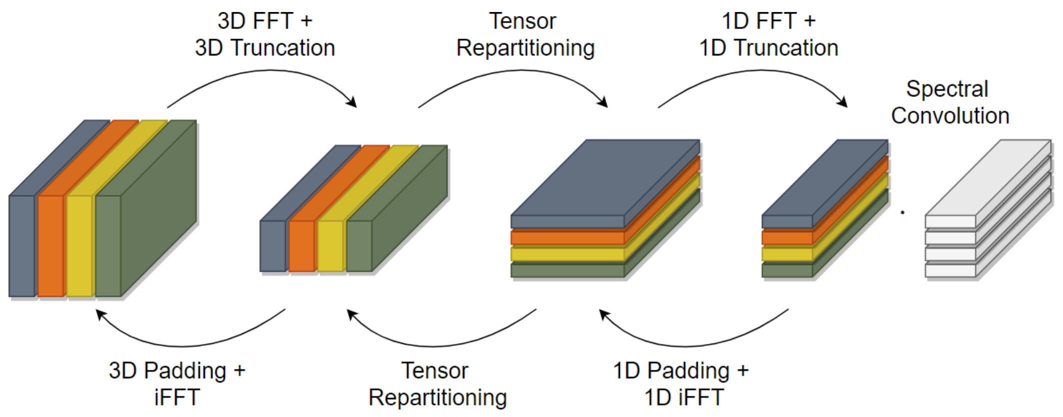

Figure 1: Parallel FNO block based on tensor/domain decomposition. The hidden states of the network are distributed across four GPUs (represented by different colors) along one of the spatial data dimensions. Each FNO block consists of a (distributed) forward FFT, multiplication with the complex network weights and and inverse FFT.

gradients with automatic differentiation (AD) - a feature that is currently not supported by most reservoir simulation packages [10]. However, one of the major road blocks in applying deep learning to industry-scale CO_2 projects is the size of the input and output data. Reservoir simulations in commercial settings involve geological models with millions (or billions) of grid points that describe physical properties of the subsurface (e.g., porosity, permeability) and they predict CO_2 saturation and/or pressure as a function of both space and time. Existing deep neural networks for CO_2 flow are either two-dimensional [7] or small-scale three-dimensional simulators [6, 5, 8] that operate on significantly smaller mesh sizes than required for commercial settings [2].

Existing deep learning-based simulators are not able to scale to industry-scale problem sizes, as they are implemented with data parallelism and are thus limited to network and data sizes that are supported by the available memory of a single GPU (i.e., up to 80 GB on the Nvidia A100). In data parallelism, samples of a (mini-)batch are distributed across multiple GPUs, but each GPU must at least fit a single data sample (including its hidden states) and the corresponding network weights into memory [11]. In this paper, we overcome the memory limitations of a single GPU through model-parallel deep learning [11], in which we distributed both the data (input-, output- and hidden states), as well as the network weights across multiple GPUs. Using a model-parallel version of the Fourier Neural Operator architecture [12, 13], we train the largest AI-driven simulator to date for simulating multi-phase CO_2 flow on a 3D simulation grid with two million grid points and we create a new training dataset based on the Sleipner CO_2 model, a real-world reservoir simulation benchmark from the world's first off-shore commercial CCS project [14].

2 Model-parallel Fourier Neural Operators

We base our network architecture for simulating 3D CO_2 flow on Fourier Neural Operators (FNOs) [12], which have shown very promising performance on a variety of challenging PDEs, including the Navier-Stokes equation and 2D multi-phase flow equations [7]. As discussed in the introduction, we use the model-parallel implementation of FNOs introduced in [13]. The implementation is based on domain (or tensor) decomposition, in which all tensors of the network (including weights and hidden states) are distributed across multiple GPUs. The input to our FNO is a three-dimensional binary map that indicates the location of the CO_2 injection wells. Even though we only vary the number and locations of CO_2 wells, we also supply the geological permeability and topography as additional input channels, as we find it improves convergence during training. The network output is a four-dimensional tensor of the predicted CO_2 saturation (three spatial dimensions and time). As FNOs require that the input and output have the same number of dimensions, we repeat the input data along the time axis.

The network architecture closely follows the original serial FNO implementation [12] and uses an encoder-decoder structure with multiple FNO blocks. Each FNO block performs a spectral convolution by computing a 4D Fourier transform (FFT) of the hidden state variable, followed by an element-wise multiplication with a set of complex-valued learnable weights and an inverse FFT (iFFT). As in the original paper, we only keep approximately 20% of the lowest frequencies, while truncating high frequencies to reduce the number of network weights.

For the parallel implementation, we partition (i.e., distribute) data along one the the three spatial data dimensions [13]. The encoder and decoder only act along the (non-partitioned) channel dimensions, so no communication, other than broadcasting the weights, is required in the forward pass. Inside each parallel FNO block (Figure 1), we first perform a 3D FFT and frequency truncation along the non-partitioned data dimensions and then re-partition the data along one of the other spatial dimensions. The re-partition and broadcasting operations are implemented with DistDL [15], a Python package that provides parallel communication primitives for Pytorch based on message passing and implemented through MPI [16] and NCCL [17]. After re-partitioning the data (i.e., changing the dimension along which data are distributed), we compute a 1D FFT and frequency truncation along the final dimension and compute the spectral convolution. As convolutions in Fourier space are element-wise multiplications, we initialize and update network weights of each FNO block on their respective worker, so no communication is required for the spectral convolution itself. To return to the original data dimension in the spatial-temporal domain, we repeat the same steps from the forward FFT in reverse order, using zero padding instead of truncation and inverse FFTs.

3 Sleipner dataset

As the model-parallel FNO enables us to scale to larger grid sizes for simulating CO_2 flow than previously possible, we generate a new training dataset for supervised learning based on the Sleipner 2019 benchmark model [18]. The Sleipner field is the world's first commercial off-shore CCS project, where approximately 16 mega tonnes of CO_2 were injected into a geological formation in the North Sea for permanent geological storage [14]. The 2019 Sleipner benchmark is a reservoir simulation model (i.e., not an AI benchmark dataset) for simulating the CO_2 plume behavior as observed during the project with a conventional reservoir simulator such as OPM. As in the original project, the benchmark simulates CO_2 flow for a single CO_2 injection well.

To train our FNO-based numerical simulator, we generate a training dataset in which we vary the number and locations of the CO_2 injection wells, while keeping the geological model (permeability, porosity) fixed. This experimental configuration is relevant for tasks such as well location optimization, in which we want to identify the number and locations of injection wells that maximize the amount of CO_2 that can be stored without exceeding pressure limits or causing CO_2 to leak from the storage site [19]. Such optimizations are typically not possible in practice, as algorithms for derivative-free global optimization (e.g., simulated annealing or genetic optimization) require hundreds of sequential objective function evaluations [20], each of which involve running multiple hours-long simulations.

We simulate CO_2 flow for 1,600 different well locations with OPM, an open-source reservoir simulator written in C++ and based on the finite volume method. We use the permeability model and OPM configuration from the original Sleipner benchmark, including the 3D simulation grid of size $262 \times 118 \times 64$ grid points (close to two million grid points). We simulate the CO_2 saturation history for 15 years and store the results at 84 equally spaced time intervals, so the total data size per input/output sample is $262 \times 118 \times 64 \times 86$ (a total of 167 million input and output variables per training sample). Each training sample has a size of 648 MB and the total dataset is 1.04 TB. (For future use and because we want to make the dataset publicly available, we also store the pressure history, even though it is not used in this paper.)

4 Training and results

We train our model-parallel FNO on a single node with eight NVidia A100 GPUs with 80 GB memory each. The total of 640 GB of GPU memory allows us to train the FNO on the full spatial-temporal grid of size $262 \times 118 \times 64 \times 86$. We use a batch size of two during training (across eight GPUs), which is the largest possible batch size before running out of memory. Note that we are not able to use data parallelism for this configuration, which requires a minimum batch size of eight (on eight GPUs) and exceeds the available memory by a factor of four.

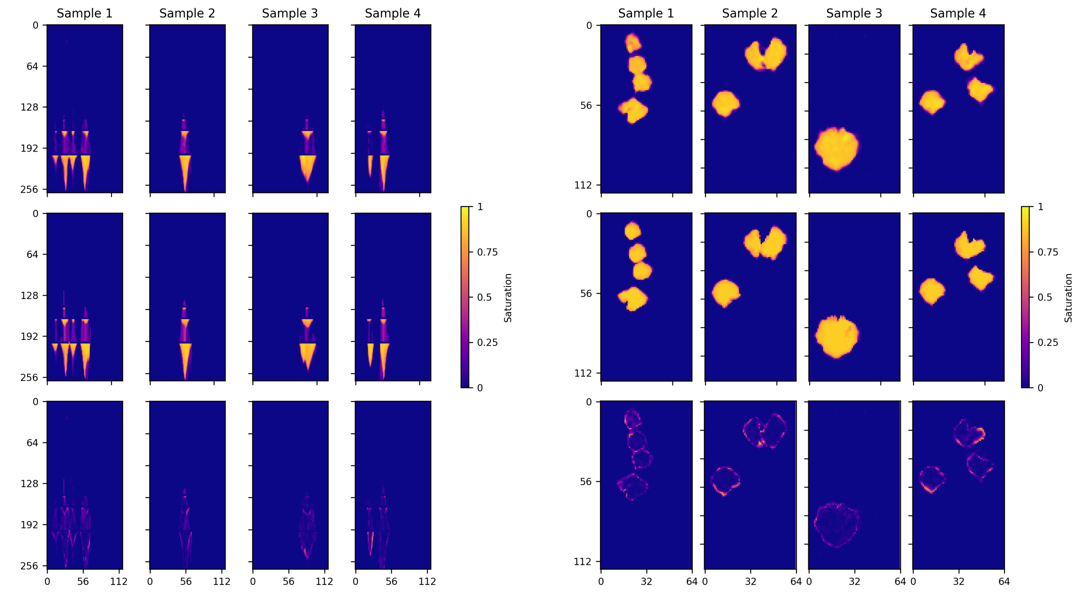

Figure 2: Two-dimensional vertical and horizontal slices through the three-dimensional CO_2 saturation at the final time step. The top row is the FNO result, the center row is the ground truth simulated with OPM and the bottom row shows the (absolute) difference. All four well location combinations are drawn from the test dataset. Each OPM simulation takes around 6.5 hours, while each FNO simulation takes 0.12 seconds.

We use 1,400 of our data samples for training, 176 for validation and 24 for testing and we train our distributed model for 50 epochs. Each training sample has three input channels (well location, topography and permeability), one output channel (CO_2 saturation) and 10 hidden channels. We minimize the relative ℓ_2 loss [12] between the predicted and true spatial-temporal saturation using the Adam optimizer with a learning rate of 10^{-3}. Additional training and network parameters are listed in the Appendix. Each training epoch takes around 20 minutes and total training time is 17 hours.

After training, our network generalizes to new locations and combinations of CO_2 injection wells (Figure 2). The average time to simulate a training example with OPM on 8 CPU cores is 6.5 hours, whereas prediction time with the FNO on eight A100s is 0.12 seconds. Taking into account the price differences of the respective hardware on Azure, we arrive at a price of \$3.4 per simulation with OPM and 0.08 cents per simulation with the FNO – a factor of 4,500. If we take the cost for generating the 1,600 data samples and training into account (Appendix A), the FNO amortizes the cost after 1,855 simulations. As all possible combinations to place up to four injection wells in our model is much higher than the number of simulated training samples (over 12 billion combinations), the FNO provides a cheap, fast and accurate (Table 1) surrogate model that can be used in downstream applications that require thousands of simulations such as well location optimization or uncertainty quantification. Our example also shows that, using model parallelism for deep learing, we are able to scale scientific AI to industry-scale problem settings, even with a moderate amount of eight GPUs for training.

Table 1: Network performance on the validation and test dataset.

	MSE	MAPE	R2
Validation	$1.1104 \cdot 10^{-4}$	1.0866	0.9453
Test	$1.1603 \cdot 10^{-4}$	1.0952	0.9487

Acknowledgments and Disclosure of Funding

We thank Erik Skjetne and his colleagues from the Northern Lights project for sharing their expertise on reservoir simulations and CCS with us and for providing valuable feedback. We also thank Thomas Grady from the Georgia Institute of Technology for his contributions to DistDL and the initial development of the model-parallel FNO with his advisor Felix J. Herrmann. Many thanks also to John Godlewski from SLB for the discussions and collaboration around AI for CO2 flow simulations.

References

[1] P. Ringrose, *How to Store CO2 underground: Insights from early-mover CCS Projects*. Springer, 2020.

[2] H. Gross and A. Mazuyer, "GEOSX: A multiphysics, multilevel simulator designed for exascale computing," in *SPE Reservoir Simulation Conference*. OnePetro, 2021.

[3] A. F. Rasmussen, T. H. Sandve, K. Bao, A. Lauser, J. Hove, B. Skaflestad, R. Klöfkorn, M. Blatt, A. B. Rustad, O. Sævareid, K.-A. Lie, and A. Thune, "The open porous media flow reservoir simulator," *Computers & Mathematics with Applications*, vol. 81, pp. 159–185, 2021.

[4] B. J. Graupner, D. Li, and S. Bauer, "The coupled simulator Eclipse–OpenGeoSys for the simulation of CO2 storage in saline formations," *Energy Procedia*, vol. 4, pp. 3794–3800, 2011.

[5] Z. Jiang, P. Tahmasebi, and Z. Mao, "Deep residual U-net convolution neural networks with autoregressive strategy for fluid flow predictions in large-scale geosystems," *Advances in Water Resources*, vol. 150, p. 103878, 2021.

[6] H. Tang, P. Fu, C. S. Sherman, J. Zhang, X. Ju, F. Hamon, N. A. Azzolina, M. Burton-Kelly, and J. P. Morris, "A deep learning-accelerated data assimilation and forecasting workflow for commercial-scale geologic carbon storage," *arXiv preprint arXiv:2105.09468*, 2021.

[7] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, "U-FNO — An enhanced Fourier Neural Operator-based deep-learning model for multiphase flow," *Advances in Water Resources*, vol. 163, p. 104180, 2022.

[8] B. Yan, B. Chen, D. R. Harp, W. Jia, and R. J. Pawar, "A robust deep learning workflow to predict multiphase flow behavior during geological CO2 sequestration injection and post-injection periods," *Journal of Hydrology*, vol. 607, p. 127542, 2022.

[9] M. Tang, X. Ju, and L. J. Durlofsky, "Deep-learning-based coupled flow-geomechanics surrogate model for CO2 sequestration," *International Journal of Greenhouse Gas Control*, vol. 118, p. 103692, 2022.

[10] A. Lavin, H. Zenil, B. Paige, D. Krakauer, J. Gottschlich, T. Mattson, A. Anandkumar, S. Choudry, K. Rocki, A. G. Baydin *et al.*, "Simulation intelligence: Towards a new generation of scientific methods," *arXiv preprint arXiv:2112.03235*, 2021.

[11] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[12] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier Neural Operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

[13] T. J. Grady II, R. Khan, M. Louboutin, Z. Yin, P. A. Witte, R. Chandra, R. J. Hewett, and F. J. Herrmann, "Towards large-scale learned solvers for parametric PDEs with model-parallel Fourier neural operators," *arXiv preprint arXiv:2204.01205*, 2022.

[14] A.-K. Furre, O. Eiken, H. Alnes, J. N. Vevatne, and A. F. Kiær, "20 years of monitoring CO2-injection at Sleipner," *Energy procedia*, vol. 114, pp. 3916–3926, 2017.

[15] R. J. Hewett and T. J. Grady II, “A linear algebraic approach to model parallelism in deep learning,” *arXiv preprint arXiv:2006.03108*, 2020.

[16] W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum, and A. D. F. E. E. Lusk, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[17] S. Jeaugey, “NCCL 2.0,” in *GPU Technology Conference (GTC)*, vol. 2, 2017.

[18] “Sleipner 2019 benchmark model,” https://co2datashare.org/dataset/sleipner-2019-benchmark-model, 2019, accessed: 2021-12-20.

[19] D. A. Cameron and L. J. Durlofsky, “Optimization of well placement, CO2 injection rates, and brine cycling for geological carbon sequestration,” *International Journal of Greenhouse Gas Control*, vol. 10, pp. 100–112, 2012.

[20] K. A. Dowsland and J. Thompson, “Simulated annealing,” *Handbook of natural computing*, pp. 1623–1655, 2012.

A Appendix

Figure 3 shows the training and validation loss history for the Sleipner example. The FNO architecture and dimensions of the hidden data states for each layer are listed in Table 2. The input/output/hidden data are 6-dimensional tensors with dimensions batch size, channel, spatial x, y, z dimensions and time. The network weights are 6-dimensional as well (input channels, output channels, spatial dimensions x, y, z and time). The weights of the FNO blocks have an additional block dimension (which is 8 for the 4D FNO). Training parameters are listed in Table 3 and the cost for training data simulation, training and cost per simulation are provided in Table 4.

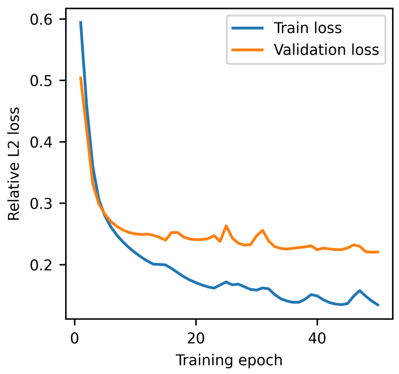

Figure 3: Training and validation relative ℓ_2 loss.

Table 2: Dimensions of the hidden network states and learnable FNO parameters. The dimension along which each tensor is distributed is underlined. Encoder and decoder weights are not partitioned as they have no spatial-temporal dimension and are broadcasted during the forward pass.

Layer	Data shape [N C X Y Z T]	Learnable weights
Input	$2 \times 2 \times 262 \times \underline{118} \times 64 \times 86$	-
Encoder 1	$2 \times 5 \times 262 \times \underline{118} \times 64 \times 86$	$2 \times 5 \times 1 \times 1 \times 1 \times 1 \in \mathbb{R}$
Encoder 2	$2 \times 10 \times 262 \times \underline{118} \times 64 \times 86$	$5 \times 10 \times 1 \times 1 \times 1 \times 1 \in \mathbb{R}$
FNO Layer 1	$2 \times 10 \times 262 \times \underline{118} \times 64 \times 86$	$10 \times 10 \times \underline{16} \times 16 \times 12 \times 10 \times 8 \in \mathbb{C}$
FNO Layer 2	$2 \times 10 \times 262 \times \underline{118} \times 64 \times 86$	$10 \times 10 \times \underline{16} \times 16 \times 12 \times 10 \times 8 \in \mathbb{C}$
FNO Layer 3	$2 \times 10 \times 262 \times \underline{118} \times 64 \times 86$	$10 \times 10 \times \underline{16} \times 16 \times 12 \times 10 \times 8 \in \mathbb{C}$
FNO Layer 4	$2 \times 10 \times 262 \times \underline{118} \times 64 \times 86$	$10 \times 10 \times \underline{16} \times 16 \times 12 \times 10 \times 8 \in \mathbb{C}$
Decoder 1	$2 \times 32 \times 262 \times \underline{118} \times 64 \times 86$	$10 \times 32 \times 1 \times 1 \times 1 \times 1 \in \mathbb{R}$
Decoder 2	$2 \times 1 \times 262 \times \underline{118} \times 64 \times 86$	$32 \times 1 \times 1 \times 1 \times 1 \times 1 \in \mathbb{R}$
Total	$15.31 \cdot 10^9 \in \mathbb{R}$	$196.49 \cdot 10^6 \in \mathbb{R}$

Table 3: Data and training parameters.

No. of training samples	$1,400$
No. of validation samples	276
No. of test samples	24
Batch size	2
No. of training epochs	50
Optimizer	Adam
Learning rate (LR)	10^{-3}
LR scheduler type	Reduce on plateau
LR min, patience, cool-down, factor	10^{-9}, 5, 0, 0.25
Azure VMs for data simulation	E8s v3 (8 $\times$ vCPU cores)
Azure VM for training	ND96asr (8 $\times$ Nvidia A100 80 GB)

Table 4: Cost for training data simulation, model training and running a single simulation with OPM and the trained FNO. All prices are based on hourly prices of Linux virtual machines on the Azure cloud.

Training data simulation (1,600 samples)	On demand: $\$5,487$; Spot: $\$2,194$
Training (50 epochs)	On demand: \$462; Spot: \$254
OPM simulator	\$3.4 per simulation
Trained FNO	\$0.0008 per simulation