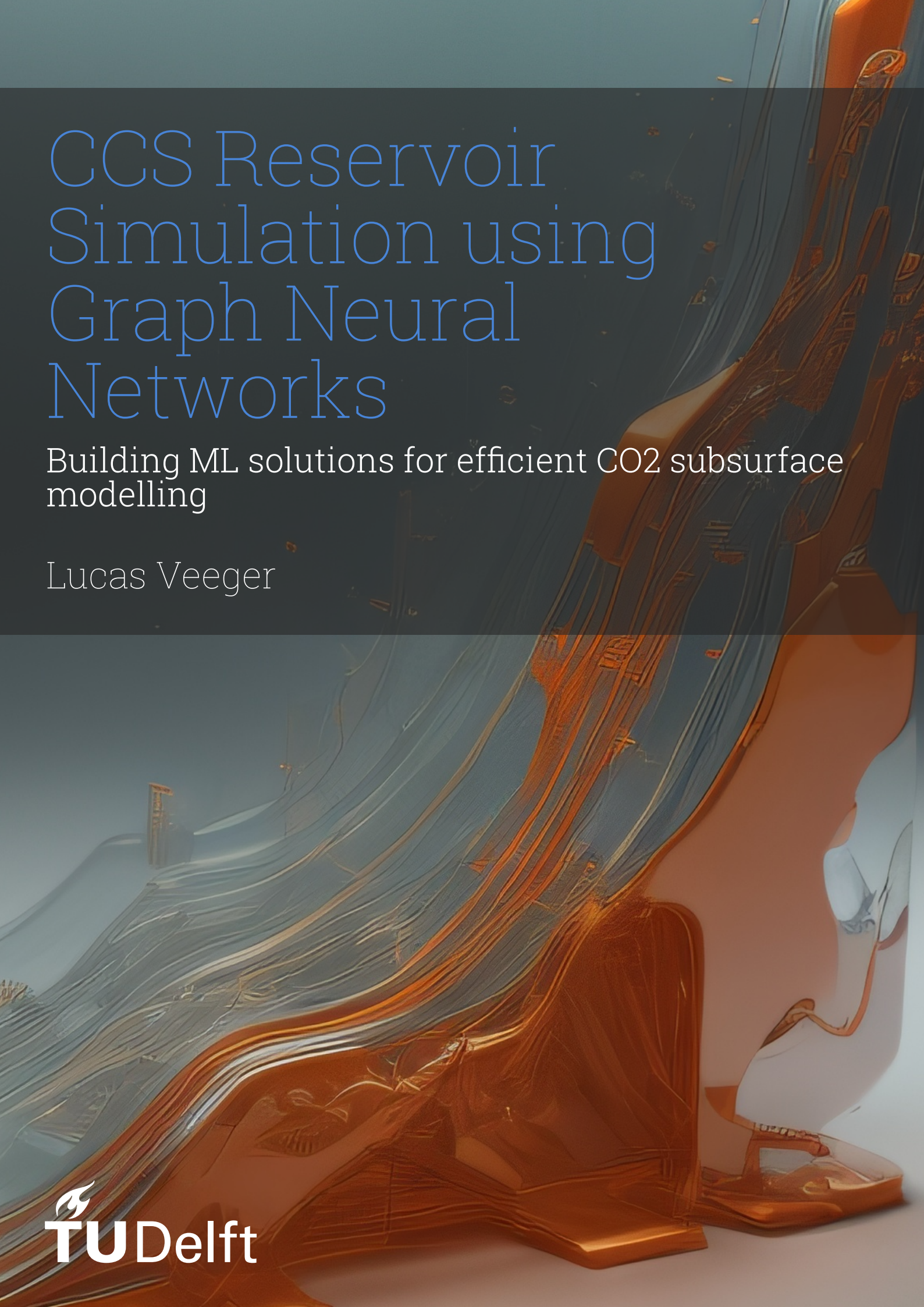


# CCS Reservoir Simulation using Graph Neural Networks

## Building ML solutions for efficient CO2 subsurface modelling

Lucas Veeger

TU Delft

# CCS Reservoir Simulation using Graph Neural Networks

## Building ML solutions for efficient CO2 subsurface modelling

by

Lucas Veeger

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on December 6th, 2023 at 11:00 AM.


| | | |
|---|---|---|
| Student number: | 4459628 | |
| Project duration: | April 17, 2023 – December 8, 2023 | |
| Thesis committee: | Neil Yorke-Smith, | TU Delft, supervisor |
| | Kees Vuik, | TU Delft, 2nd committee member |
| Company supervisors: | Tom Jönsthövel, | SLB |
| | Soham Sheth, | SLB |



| | |
|---|---|
| Cover: | AI generated image (www.imagine.art) promted with 'Subsurface Modelling' |
| Template: | TU Delft Report Style, with modifications by Daan Zwaneveld |

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.


TU Delft

# Preface

*This thesis project came with the beautiful opportunity to experience a new location. For the past 6 months I have been enjoying my time in Norway to great extents. Both the experiences in and out of the office were very appreciated components of my life this year, centered around the greater goal of graduating. I got to meet some great people in Oslo which I look forward to meeting again in the years to come. I want to thank Diederik for unknowingly setting up this opportunity by introducing me to Tom at their SLB networking event in Delft. A lot of thanks to Tom for putting in effort before and during the project. He managed to find the right person within SLB with a lot of interesting data science ideas in the new energy domain. I want to thank this person, Soham, for providing some key ideas during the development of this research. Together with Tom's inspiring translations to the mathematical foundations I developed a profound interest in the computational technologies that can and should aid CCS technology. I am sure CCS will be an essential technology transitioning our energy ecosystem, and I hope this work will be a step in a longer path of creating machine learning technologies for a sustainable future. For me this project has been a great means of growing as a data scientist, and I look forward to the years ahead of me, hopefully filled with similarly inspiring challenges.*

*Lucas Veeger*
*Delft, November 2023*

# Abstract

Reducing cost and improving computability of reservoir simulation is an important goal in the process of enabling CCS (Carbon Capture & Storage) as a large-scale technology for mitigating CO2 emissions. In terms of computation time data-driven approaches have potential to outweigh the performance of numerical reservoir simulators, learning instead of solving an approximation of the subsurface physics. In this research a Graph Neural Network (GNN) is trained on 2D reservoir model samples to construct a ML autoregressive reservoir simulator. As such the aim of the model is to integrate a spatiotemporal learning task in a graph representation learning problem. Where GNNs have shown their potential in various graph representation and node classification tasks, their use in a reservoir simulation setting is practically unexplored. Graph methods are studied in this setting because of their potential to handle data in non-Euclidean space, which is desired in reservoir simulation to efficiently handle unstructured grids. A simple GNN autoencoder is introduced, implementing the common GCN (Graph Convolutional Network) layer as a locally-operating message-passing operator. The model does not work with spatial graph information, accepting arbitrary graph sizes and structures in 2D or 3D. The model is trained, validated and tested against 2D simulated datasets, generated with the INTERSECT numerical reservoir simulator. The trained model remains numerically stable and makes correct predictions for >140 autoregressive steps. It also shows capability to translate point-source information (the injection rate at the injection well) correctly into a CO2 saturation propagation pattern and corresponding pressure field across the reservoir grid. The model exhibits the correct behaviour around geological layers with variable permeability/porosity, indicating the model successfully adapts to geological variability in the subsurface. Finally the model shows it can infer on totally new reservoir structures. The latter showcases the local method's potential to train a ML reservoir simulator for arbitrary reservoir grids by training it with small generalized subsurface samples, dismissing the need to fully (re)train on usual reservoir grid sizes. Predicting using the GNN methods gives a 60x speedup compared to the numerical simulator on the small training grids, which likely improves for larger grids given the complexity scaling of numerical simulators. As expected, capturing the elliptic behaviour of the PDE governing pressure is challenging for the local GNN method. While the resulting expressiveness of pressure is passable, ideas for improvement are suggested which can be incorporated using the same proposed GNN architecture.

*Visit* `https://lucasveeger.github.io/` *for animated experimental results. This work has been accepted for presentation at the Geopublishing 2024 NEXT conference (23-25th January 2024, Bergen,* `nextenergy.no`*)*

# Contents

# List of Figures

# 1

# Introduction

Energy is abundantly required in society, and demand will continue to rise as the world population grows, societies develop and economies grow. On the other hand, systematic boundaries are reached by the accumulating atmospheric concentration of a residual product of fossil energy production: carbon dioxide or CO2. The heat-trapping greenhouse effect caused by CO2 is vital for life on earth, but has become problematic as growing atmospheric CO2 concentrations increase global temperatures. Recent measurements (Aug 2023) put global atmospheric CO2 concentration at 420 ppm (parts per million) [4], which should be reduced to below 350 ppm for a normal climatological equilibrium [39].

Carbon Capture and Storage (CCS), a carbon dioxide removal (CDR) technology, is expected to play a vital role in mitigating CO2 emissions. The Intergovernmental Panel on Climate Change (IPCC) has set out strategies in the IPCC 1.5° Special Report to achieve net emission reductions [6]. CCS plays an important role in these. Without CCS technology fossil emissions are required to reduce to near zero by 2050, an increasingly challenging task given the estimated energy demand. The importance of Carbon Capture and Storage (CCS) is emphasized by the knowledge that certain hard-to-abate industries are not expected to vanish or easily transition to low-carbon alternatives (e.g. cement and steel production). CCS can mitigate inevitable residual emissions required during and after the energy transition. Figure 1.1 from the Global Status of CCS report [16] illustrates the up-scaling of CCS as projected in the International Energy Agency's Sustainable Development Scenario.

## CCS

Carbon Capture & Storage is a carbon dioxide removal technology. Initially, CO2 is (chemically) captured as the residual product of a (large-scale) production process, e.g. a coal power plant or cement production factory. Instead of atmospherically emitting the CO2 it is filtered, contained and transported to a permanent storage location for injection. After injection the CO2 should remain contained indefinitely to ensure an effective emission reduction. Logical containers for CO2 are previously explored subsurface porous rock formations (i.e. depleted oil/gas fields), which have successfully contained hydrocarbons for millions of years, and they generally are already mapped and studied [16]. Saline aquifers (subsurface salt water volumes) are also capable of storing large amounts of CO2 and are generally located closer to CO2 capturing sources. However, injection and CO2 storage characteristics have been studied less in such reservoirs [16].

For testing and for specific EOR (enhanced oil recovery) purposes, the injection of CO2 has been implemented successfully over a surprisingly large time span (1972 - now) [32, 16], indicating technical potential and feasibility of the injection component of CCS. Furthermore, a significant amount of domain knowledge and technology in the current energy field will help to facilitate large scale deployment of CCS installations. An important dynamic modelling technology is *reservoir simulation*.

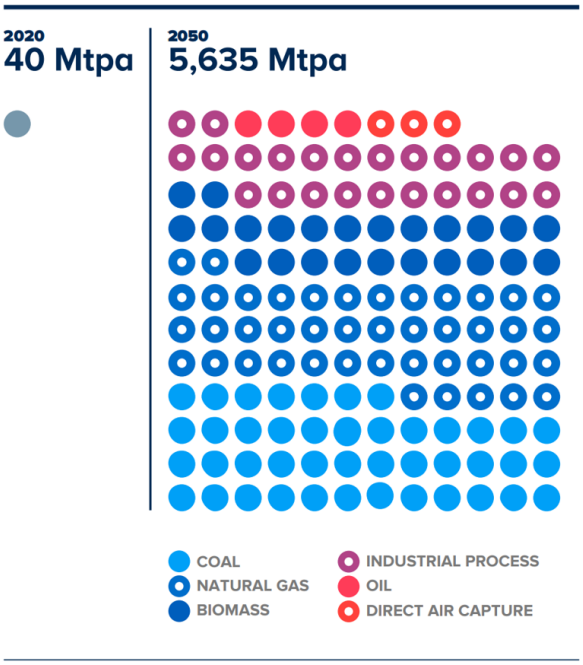



**Figure 1.1:** Current (2020) vs prospected (2050) yearly CCS storage capacity, with specification of capture sources, based on technological advances and IPCC targets set out in 1.5 Special Report. Visual from CCS Global Report 2020 [16], data from International Energy Agency [1].

## Reservoir Simulation

Reservoir simulation is a computational technology to model behaviour of components in subsurface 'reservoirs', i.e. subsurface porous rock formations that allow fluid (i.e. liquids and/or gasses) flow. Subsurface reservoirs have been the subject of extensive research for decades because of their hydrocarbon contents. With the rise of computing power, the capabilities of reservoir simulation have progressively shaped engineering decisions in this field. Current reservoir simulators are numerical mathematical models that numerically solve the underlying physical systems. In the case of CCS reservoir simulators will be used to model flow, migration and saturation of injected CO2 and determine the assosciated pressure across the reservoir. Such simulations assist in determining reservoir capacity, optimal well placement(s) and safe injection rate(s). Despite their great informative potential, numerical reservoir simulators are costly in time and computational resources. This results in a trade-off between more simulations for optimization or less for time and cost efficiency.

## Problem definition

The constrained simulation speed of existing reservoir simulators imposes limitations on the variability during optimization. To maximize the efficiency of CCS installations, exploration of various injection scenarios is crucial. Additionally, to ensure long-term containment CCS simulations should cover extended time frames up to 1000 years, well beyond the typical time span of a hydrocarbon extraction reservoir simulation (1-10 years). Lastly, CCS projects will likely be commercially driven and prioritize cost-effectiveness. These aspects encourage accelerated approaches for reservoir simulation. Machine learning methods are being studied in the reservoir simulation field and show growing accuracy and reliability, but the technology has no effective role yet in industry applications as challenges remain [43, 53].

## Scope: Graph methods for unstructured reservoir grids

One of the challenges for machine learning in the context of reservoir simulation lies in the data format popular in modern subsurface models: unstructured grids. Unstructured grids (or meshes) are useful in subsurface modelling because they are unordered, adaptable and allow for better abstractions of arbitrary complex subsurface geometries compared to structured (e.g. cartesian) grids. As such they represent a more logical geometrical framework for solving physical processes around complex

shapes. However, many ML methods require data to be in a structured format (e.g. Convolutional Neural Networks). Converting unstructured grids to structured grids using interpolations is theoretically challenging and computational complexity scales with grid size. In reservoir simulation, reservoir grids can become especially large at the relevant resolutions, containing millions of cells to simulate. An example of an unstructured reservoir grid is shown in figure 1.2.

A more efficient way of handling unstructured grids are methods that handle data as graphs, where grid cells form the nodes/vertices, and neighbouring grid cells are connected using edges. Such methods generally operate on the node level and process information in the node's neighbourhood, accessible through the node's edges. In the context of reservoir simulation such methods could be considered as a network flow model, predicting the flow between discretized cells of the porous media. Different node and edge types, edge weights, aggregation functions and physics-inspired boundaries (more details about these methods in chapter 3) can be incorporated to integrate domain-specific knowledge. This work will present a data-driven ML approach for predicting CO2 flow by converting reservoir data to graph data and using this to train a Graph Neural Network that can predict consecutive graph states.

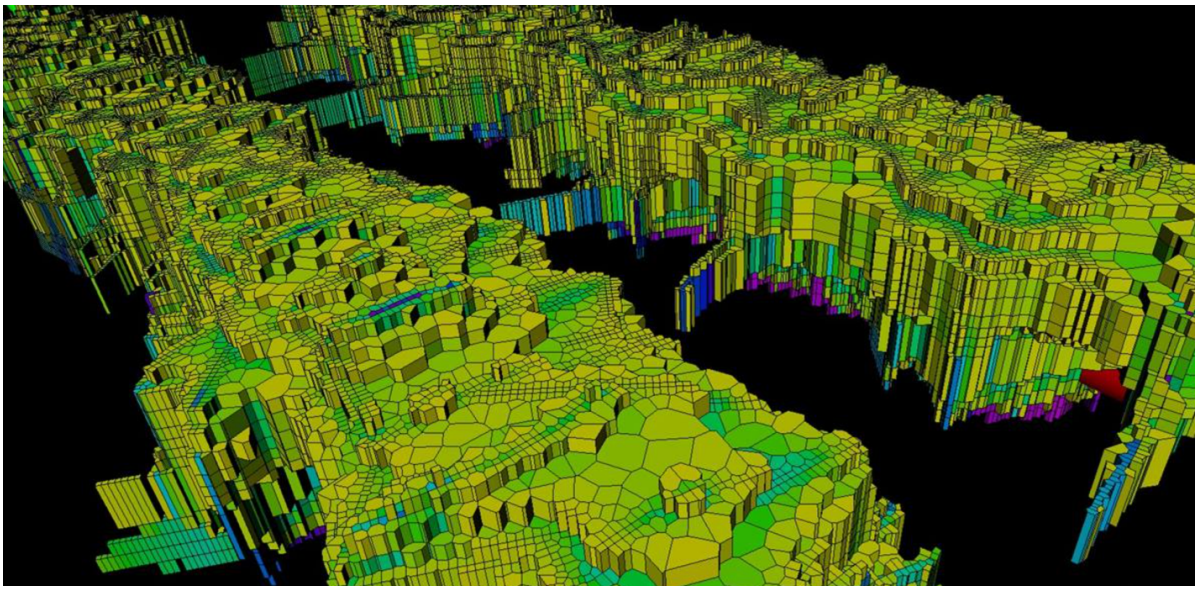


**Figure 1.2:** Snapshot of a subsurface reservoir model constructed using an unstructured grid. Unstructured grids are better in capturing the geological characteristics of subsurface rock formations. Source: SLB Petrel hydraulic fracture modeling

## 1.1. Reservoir Simulation using Graph Neural Networks

As described above Carbon Capture & Storage (CCS) technology will strive for technological advances that reduce cost and increase operational efficiency and safety. Accelerated surrogate ML models can enhance the potential of reservoir simulation for CCS target reservoir exploration. A key objective in achieving this goal is to identify a method capable of making inferences on large reservoir grids without the need for extensive training on each, as this is likely to be computationally expensive.

This work focuses on testing an increasingly recognized methodology, Graph Neural Networks (GNNs), to evaluate if this method has such potential. GNNs have not been extensively applied in such types of tasks yet, which can be described as a *spatiotemporal graph representation* learning task. The problem combines graph representation / reconstruction learning with learning temporal system dynamics. In this case, the system represents a subsurface physical system represented by different dynamic and static features, with underlying physical principles governing the development of the dynamic features. It is known from numerical methods that certain physical properties are better described using local methods, where others are derived better with more expensive global methods (this will be described in chapter 2). The proposed method operates locally, processing nodes using information in the nodes' neighbourhoods, often referred to as *message-passing*. This works test certain hypotheses on the use

of GNNs in the reservoir simulation domain, described in the following sections.

### 1.1.1. Expressivity

For the use in spatiotemporal prediction, GNNs should be capable of transforming an input reservoir at a certain state $t$ to the grid at state $t + x : x \in T$, where $T$ is the time span to simulate (e.g. 1000 years). Based on the specific objective or phase of the simulation the time intervals in $T$ can be days, months, years or any arbitrary combination. This work focuses on a GNN autoencoder architecture with limited local reach, forcing it to predict *autoregressively* (more about this in chapter 3, section 3.3).

RQ1.
**Can simple GNNs autoregressively capture the dynamic behaviour of CO2 saturation and pressure in a perturbed subsurface reservoir?**

This question addresses the hypothesis that a GNN has the ability to learn and express relationships between the static and dynamic features of a subsurface physical system, using a non-spatial representation (i.e. a graph representation without encoding of the nodes' coordinates). As the saturation and pressure features are linked in a physical sense (see chapter 2), a successful GNN model should learn and express the right correlations between them.

### 1.1.2. Injectivity

For successful use in CCS reservoir simulation, the learned model should have acquired the skill to differentiate on the amount of CO2 injected into the system. Increasing the injection rate is expected to accelerate both saturation propagation and pressure buildup. This relationship is likely nonlinear and influenced by a multitude of factors.

RQ2.
**Can locally operating GNNs correctly express nonlinear variations in injection rate across the reservoir grid?**

While it is theoretically known that nonlinearity can be captured in ML models through use of appropriate activation functions, it has to be tested whether a local method like GNN can effectively propagate the perturbation's scale (i.e. injection rate) away from the injection node (i.e. where the injection well is located) to the rest of the grid.

### 1.1.3. Transferability

To evaluate the greater potential of GNN models for reservoir simulation, the transferability or generalizability of a GNN model to unseen grids is desired. Most GNNs are local (convolutional) methods operating on every graph node, and the computational complexity of training GNNs on reservoir grids scales linearly with the amount of cells in the reservoir. As reservoir grids can contain millions of cells at the relevant scale, training on common reservoir grids is expected to be costly and counter-productive in achieving any computational speedups.

RQ3.
**Are GNNs capable of transferring their expressivity to unseen subsurface physical systems with different static composition?**

The hypothesized GNN method will be local, i.e. it will process every node with the same learned parameters, differentiating on the feature information gathered in the node's neighbourhood. Ideally the model is trained on a set of relatively small but 'sufficiently' informative grids so that its parameters are learned a proper generalization of the physics featured in geological systems. This way a model can be trained on a complete set of geological variations, without fitting to a single large reservoir's composition. If successful then ultimately the model should be applicable to any arbitrarily sized reservoir graph (keeping in mind the scale of the features in the inferred graph has to be similar to the training samples) and be capable of inferring the next dynamic feature states with decent prediction times.

## 1.2. Related work

Machine learning technologies have been tested in the reservoir simulation domain over the years. Most work aids reservoir simulation and engineering in a hybrid approach using ML models or ML metaheuristic algorithms on subtasks like well placement or certain enhanced oil recovery (EOR) methods, e.g. using CO2 injection [37, 59]. In CCS related context, ML algorithms have been explored to predict reservoir geological features (e.g. permeability, porosity) [61, 62], chemical characteristics of injected CO2 [36, 52, 50] and development of dynamic reservoir features (pressure, CO2 flow) [58, 55, 54]. As described in the previous sections, the focus of this work is also modelling the CO2 saturation and corresponding pressure.

Physics-Informed Neural Networks (PINN) and more generally Physics-Informed Machine Learning (PIML) have created an environment of physics inspired approaches around subsurface flow machine learning [10], some particularly applied in the CCS domain [47, 34]. These approaches leverage physical principles by integrating the governing PDEs in loss calculations and apply other domain-specific knowledge to design neural network reservoir simulators. Another approach using Fourier Neural Operators (FNOs) has also shown potential, using FNOs in a U-Net architecture [55, 19], resembling CNN's U-Net [40]. Notable for this approach is the resemblance of such methods with the numerical multigrid methods used in numerical simulators, which tackle the problem by solving the system on a coarsened representation first (like the bottleneck in U-Net) and pass on the solution to increasingly finer representations until the required output dimensionality and resolution is reached. A similar encoding-decoding approach using a embed-to-control framework is also seen in various subsurface flow studies [20, 2, 17]

In the comparable field of weather forecasting, which models systems using similar physical dogmas (e.g. the Navier-Stokes equations, a set of PDEs describing fluid and gas behaviour), a GNN autoencoder architecture managed to forecast geopotential height, temperature and humidity reasonably well for several days (weather systems are much larger and more parametrically complex to solve than subsurface systems) [21]. Another study on weather prediction shows unprecedented predictive performance using a GNN autoencoder multi-leveled in a multigrid approach, creating exceptionally accurate 10-day global weather predictions [24]. Other fields that have integrated GNNs in spatiotemporal settings include traffic forecasting [18], COVID forecasting and solar plant power prediction [42].

Graph Neural Networks have not yet been studied widely in the context of reservoir simulation. Recently, a hybrid computational framework combined the U-Net architecture to predict the pressure field with a GNN to infer the fluid flow [56]. In a very recent study a GNN-LSTM deep learning model is implemented to predict oil and water rates under variable well controls [15]. In another recent study, a 2D grid-based autogregressive GNN predicts oil and water saturation development in a constant pressure field, reliably predicting for several years [49]. This type of spatiotemporal modelling and problem setting resemble closely the problem setting and autoregressive prediction studied in this work. No work is published yet specifically on subsurface CO2 flow modelling using GNNs.

## 1.3. Outline

This work is divided into the following chapters:

**Chapter 2: Reservoir Simulation**
The current technology of reservoir simulation relies on mathematical and physical foundations that are relevant and analogous to elements in the GNN methods applied later in the work. The geophysical variables are discussed and a high-level description of the numerical methods is given.

**Chapter 3: Graph Neural Networks**
The principles of the proposed machine learning approach are discussed in this chapter. The chapter introduces Graph Neural Networks as message-passing algorithms. A distinction between spatial and spectral graph convolutions is provided, and the common GCN layer is formulated and demonstrated.

**Chapter 4: Methods & Experimental Design**
Using the introduced GCN layer from chapter 3 an autoencoder architecture is constructed for autoregressive prediction. Feature pre-processing, data generation and the model hyper-parameters are discussed. Lastly, the experiments are formulated based on the research questions formulated in the Introduction chapter.

**Chapter 5: Results**
The results of the experiments are presented and discussed. The expressiveness of the GNN model is assessed, and observed side-effects are highlighted and discussed. Animations of the results presented in the chapter are available at `https://lucasveeger.github.io/`.

**Chapter 6: Discussion**
The last chapter present an evaluating discussion of the created model and the problem it is solving. It compares characteristics of the machine learning model with the numerical reservoir simulation methods from chapter 2 and suggests improvements for the created GNN model. The work is finished with a brief conclusion.

# 2

# Reservoir Simulation

To get an intuition of what a data-driven reservoir simulation model has to learn, this chapter will briefly go into reservoir models and *reservoir simulation*. The latter technology forms the motivation for this research. Without going in depth, important physical aspects and equations are covered, as well as the underlying mathematical concepts of reservoir simulators. The numerical methods used in INTERSECT (IX) reservoir simulators will be taken as a base for this. Certain components are intentionally left out, refer to the IX Technical Documentation [46] for a detailed description.

Operating in the subsurface is a blind process. Without any visual reference and with limited physical access, analysis of the subsurface relies predominantly on intelligent processing of seismic/acoustic sensory inputs. Combined with geophysics, mathematics and computation a translation is made to a subsurface reservoir model. These models determine reservoir characteristics like volume, capacity and structure. Numerous reservoir modelling softwares have been developed through the years, e.g. Petrel subsurface software. Proper modelling of subsurface reservoirs has proven vital for modern-day oil/gas field exploration and operation. In the CCS context, such models will be equally relevant.

On top of static modelling of the reservoir, modelling a reservoir system dynamically has become paramount for operational planning and execution. For CCS operation, the temporal modelling of CO2 injected in a reservoir will be highly relevant. It helps to determine the CO2 storage capacity, a safe injection rate and the security of containment. Temporal reservoir modelling, commonly known as *reservoir simulation*, has become very effective in approximating the dynamics of perturbed reservoir systems. In a CCS related context, injection of CO2 has already been part of reservoir simulations for the sake of Enhanced Oil Recovery (EOR) technologies, where CO2 is injected in a reservoir to create an over-pressure that drives hydrocarbons to separate extraction wells. Certain reservoir simulation software is capable of simulating CO2 behaviour, e.g. ECLIPSE and INTERSECT (IX). In this thesis, the IX reservoir simulator is used to generate (i.e. simulate) training data for the introduced GNN model.

## 2.1. Subsurface physics

Reservoir simulation software implements mathematical methods to model the physical relations in a (perturbed) reservoir. For every reservoir cell it restricts the variables to a set of equations. The 3 important physical concepts constraining these variables are

- Conservation of mass
- Darcy's law
- Thermodynamic equilibrium of the fluid phases

Before going into these topics, it is good to be familiarized with some geological units and quantities.

### 2.1.1. Geological features

The give an introduction of geophysical entities used in the sections below and in the rest of this work, this section lists some important quantities and units encountered in reservoir simulation.

Porosity
quantifies the capacity of a rock or sediment to contain and transmit fluids. It is the ratio of a pore volume within a geological material to its total volume, and it is expressed using a fraction or percentage. As it defines a ratio, it is unitless.

Pore volume
is the value calculated by multiplying a total rock or sediment volume with its porosity. It defines the volume of fluid that can be present in the the larger volume it is part of. E.g. a $1$ m$^3$ rock with a porosity of $0.2$ has a pore volume of $200$ liters.

Permeability
is a measure of a material's ability to transmit fluids through its pore volumes. It quantifies the ease with which fluids can flow through a porous medium. It is not strictly dependent on pore volume. Some sedimentary rocks can have high pore volume due to many small pores, but the permeability can be low because these pores are small and interconnected in a way that restricts fluid flow. Permeability in geology is often denoted in millidarcy (mD), 1 mD = 9.869233×10−16 m$^2$.

Transmissibility
describes the ability for fluid flow within the plane of the material and is defined as the in-plane permeability multiplied by the material thickness [38]. Its unit can vary, a variant is m·mD/mPa·s [35].

Saturation
refers to the unitless fraction of a (pore) volume that is occupied by a particular fluid or a fluid phase. In the rest of this work the CO2 saturation is referred to when the term *saturation* is used, unless noted otherwise.

Pressure
The force exerted by fluids (such as oil, gas, or water) within a subsurface reservoir, acting on the rock/sediment. Typical units are Pa, psi and bar. This work uses bar.

Quantities not used in the rest of this work but mentioned in the next sections are

- Molar density $\rho$: number of moles per volume ($n/m^3$)
- Mole fraction: fraction of moles of component $c$ among total amount of moles present (unitless)
- Viscosity: a fluid's internal resistance to flow (Pa·s)

### 2.1.2. Conservation equations

Conservation of mass represents the principle that in any volume the change of mass equates the difference between the mass leaving and the mass entering the volume. This equality is applied in reservoir simulation to balance the flow of every component in every reservoir cell. In a reservoir with multiphase CO2 and water flow there will be 2 conservation equations applicable to each cell $j$:

$$\begin{cases} \frac{\phi V}{\Delta t}\delta\left(\sum_{p}^{N_{p,CO2}} \rho_p S_p x_{p,CO2}\right) + q_{CO2}^{inj} - \sum_{i}^{Faces} T_i \left(\sum_{p}^{Np,CO2} \rho_p \frac{k_{rp}(S_p)}{\mu_p} x_{p,CO2}\Delta p\right) = 0 \\ \frac{\phi V}{\Delta t}\delta\left(\sum_{p}^{Np,water} \rho_p S_p x_{p,water}\right) + q_{water}^{inj} - \sum_{i}^{Faces} T_i \left(\sum_{p}^{Np,water} \rho_p \frac{k_{rp}(S_p)}{\mu_p} x_{p,water}\Delta p\right) = 0 \end{cases} \tag{2.1}$$

where

$\phi V$ = porosity $\times$ cell volume = pore volume
$\Delta t$ = time interval
$N_{p,CO2}$ = number of different phases CO2 has in system
$N_{p,water}$ = number of different phases water has in system
$\rho_p$ = molar density of phase $p$

$S_p$ = saturation of phase $p$
$x_{p,CO2}/x_{p,water}$ = mole fraction of $CO2$ / $water$ in phase $p$
$q^{inj}_{CO2}$ = CO2 inflow rate (entering the system, i.e. through injection) at specific cell
$q^{inj}_{water}$ = Water inflow rate at specific cell
$T_i$ = transmissibility of $i^{th}$ cell face
$k_{rp}$ = relative permeability, dependent on $Sp$
$\mu_p$ = viscosity of phase $p$
$\Delta p$ = pressure potential between cell and neighbours

These balance equations are the discretized version (as used in IX) of the general formulation for modelling phase-component partitioning [3], ignoring gravitational and capillary terms present in the original formulation. To simplify the look of these equations, they can be reduced to the variables that will be used later in this work. In the first term, the *saturation* term, CO2 and water saturation can be formulated as

$$S_{CO2} = V \sum_{p}^{N_{p,CO2}} \rho_p S_p x_{p,CO2}, \quad S_{water} = V \sum_{p}^{N_{p,water}} \rho_p S_p x_{p,water}.$$

For the second term, the *injection* term, we can assume $q^{inj}_{water} = 0$ (i.e. no water is injected), and note that $q^{inj}_{CO2}$ will be nonzero only for cells connected to injection wells. In the third term, the *flow* term, the component mobility $\lambda_c$ is introduced as a function of the phase mobility $\lambda_p = \frac{k_{rp}(S_p)}{\mu_p}$, the cell's absolute permeability $k_j$ (derived from $T_i$),

$$\lambda_c(S_c) = k_j \sum_{p}^{Np,c} \rho_p \frac{k_{rp}(S_p)}{\mu_p} x_{p,c}.$$

The sum $\sum_i^{Faces} \Delta p$ can be written in the continous form as the divergence of the pressure differences with respect to all the cell neighbours, i.e. $div\ \vec{p}_{Faces} = \nabla \cdot \nabla p$. In continuous form this gives the conservation equations as

$$\begin{cases} \phi \dfrac{\partial S_{CO2}}{\partial t} + \nabla \cdot (\lambda_{CO2} \nabla p) + q^{inj}_{CO2} = 0 \\ \phi \dfrac{\partial S_{water}}{\partial t} + \nabla \cdot (\lambda_{water} \nabla p) = 0 \end{cases} \tag{2.2}$$

Given that the components' saturations have a constant unit sum $S_{CO2} + S_{water} = 1$, adding the conservation equations gives an elliptic partial differential equation for the pressure, coupled with saturation through the mobility term:

$$\nabla \cdot (\lambda_{CO2} + \lambda_{water}) \nabla p = -q^{inj}_{CO2} \tag{2.3}$$

Note that the operation $\nabla \cdot \nabla p$ resembles the Laplace operator. In the graph theory discussed in subsection 3.2.2 the *graph laplacian* will be used to exploit the gradient in feature values between different nodes. The graph laplacian can be seen as the discrete analogue of the continuous Laplacian operator, with the graph nodes as spatial discretization.

### 2.1.3. Darcy's law

Equation 2.3 can be solved for $p$, and using the new pressure the fluid flow can be computed according to the adapted version of Darcy's law applicable to multi-phase flow:

$$q_p = -k_j \cdot \lambda_p \cdot \nabla p, \quad q = \sum_{p}^{N_p} q_p. \tag{2.4}$$

This produces a flow term $q$ per phase. Note this equation also dismisses gravitational and capillary forces (to account for capillary forces $p$ is solved phase dependent as $p_a$ [27]). Using the solution for $p$ and $q$, the updated saturations can be derived reversely.

### 2.1.4. Thermodynamic equilibrium of fluid phases

The thermodynamic equilibrium or *phase equilibrium* dictates that any chemical component $i$ present in multiple phases is in some phase equilibrium determined by the component's *fugacity*. Fugacity $f$ describes a chemical potential of transitioning to another phase. $f$ is calculated using a component's fugacity coefficient $\phi_i$, the pressure $P$ and the component's mole fraction $x_i$ (the latter only when it is in the aqueous phase). Without going into this concept further, it is good to note that the phase equilibrium is relevant in the CCS context. In the first place because CO2 can dissolve in water, described as the aqueous phase. Secondly, across the typical CCS trajectory (transport, pipe flow, reservoir flow), CO2 phase transitions are likely triggered due to typical pressure and temperature changes in the processes.

## 2.2. Solving the nonlinear system

The system described in Equation 2.3 is continuous and nonlinear. To solve it numerically on a computer, it has to be made discrete in time and space. Furthermore, to make computational simulation efficient, the nonlinear system has to be described as a sum of linear systems using a *linearization* method. This section summarizes these steps as performed in IX (see the Formulation and solution method chapter in the IX Technical Documentation [46] for more details).

### 2.2.1. Discretization

First, the system is temporally discretized. For simplicity, assume the governing equations are discretized using a fully implicit time stepping scheme. This makes every next state $(n+1)$ dependent on its previous state(s) $(n, n-1, ...)$. In practice, IX applies an adaptive implicit method (AIM), which will not be covered here. Either method aims to select a timestep size that minimizes overall simulation time by balancing the amount of timesteps with the expected amount of Newton-Raphson iterations (discussed below) required per timestep. Spatial discretization is performed by applying the *finite volume method* on the conversation equations and reservoir grid. The finite volume method is typically used in CFD and also translates properly to unstructured grids. It is calculated using the two-point flux approximation (TPFA) method, which linearizes the conservation equations given in Equation 2.2.

### 2.2.2. Newton-Raphson iterative method

Given the discretized system of nonlinear equations $R$, the aim is to construct the solution vectors $X$ (i.e. the CO2 saturation and pressure) at the next state $n+1$ for which the system is solved as $R^{t+1}(X) \approx 0$. IX uses the Newton-Raphson iterative method to *linearize* the nonlinear system, solve it using a linear solver, update the solution vectors and test if the solution has converged.

#### Linearization

At the start of each Newton iteration the system is linearized using a first-order Taylor expansion the nonlinear system:

$$R_{t+1}(X) \approx R_{v+1}(X) = R_v(X) + \left(\frac{\partial R}{\partial X}\right)_v \cdot \Delta X = 0, \tag{2.5}$$

ignoring any higher order terms of the Taylor expansion. $v$ indicates the iteration number, giving $R_{t+1}(X) \approx R_{v+1}(X)$ when the solution vectors converge after $v$ iterations. $\partial R/\partial X$ is the first-order derivative of the system with respect to the solution vectors, i.e. the first-order *Jacobian* matrix of discretized $R(X)$. The Jacobian will not be further elaborated here, but it is good to note that the construction of this Jacobian constitutes a large part of the computational load during reservoir simulation. Since $X_{v+1}$ is updated after every Newton iteration, $(\partial R/\partial X)_v$ has to be recomputed as well.

$\Delta X$ resembles the *solution update vectors* and is the solution of the linear system given by Equation 2.5. At the end of the iteration the solution vectors are updated:

$$X_{v+1} = X_v + \Delta X. \tag{2.6}$$

At the start of the iterative method the solution vectors are approximated by the previous timestep solution vector, $X^{v=0} = X^t$.

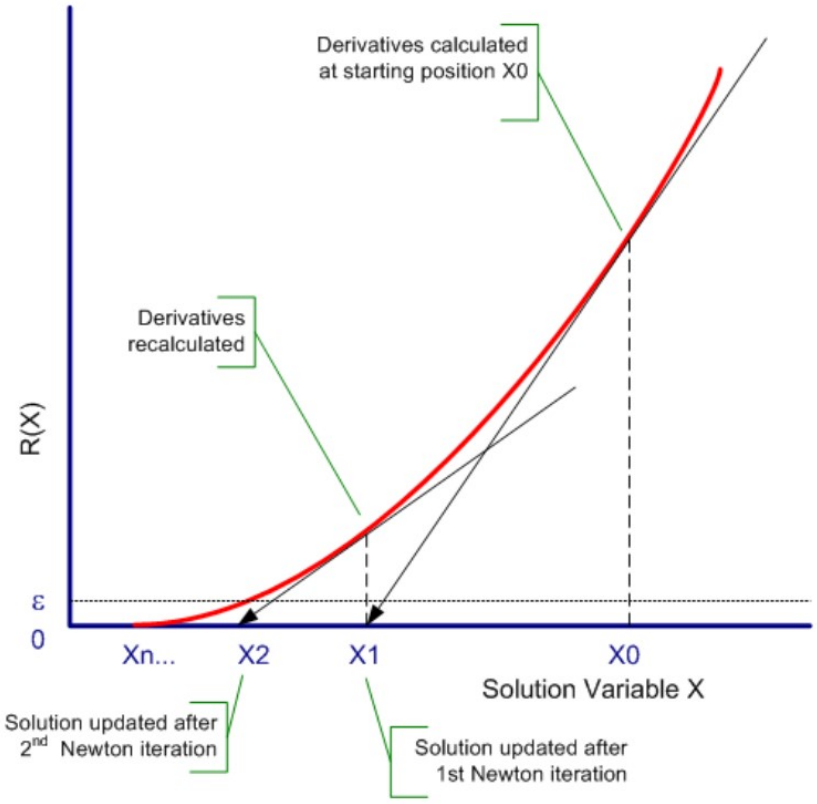



**Figure 2.1:** Solution approximation during the Newton-Raphson iterative method. Every iteration a $\Delta X$ is calculated to update the solution vectors (Equation 2.2.2) that are used in the next Newton iteration, until the solution $R_{t+1} \approx 0$ (Equation 2.5). Source: IX Technical Documentation [46]

#### linear solver

Solving the linear system forms another computationally intensive part of the simulation algorithm. To solve the linear system, the *flexible generalized minimum residual* (FGMRES) method, used with a multistage *constrained pressure residual* (CPR) preconditioner. These concepts will not be explained in this work, but it is good to mention that this preconditioner is used to decouple the pressure and saturation solution vectors in order to reserve computational resources. The CPR preconditioner includes the AMG preconditioner, which focusses on solving the global-type solution of the pressure solution vector. This will be elaborated in section 2.2.3.

#### Convergence

Every Newton iteration creates an improved linear approximation of the nonlinear system. To decide when the approximation is sufficiently good and can be stopped, the iteration checks for convergence. The convergence criteria include

- Solution deltas: for each solution vector, the values in the solution update vectors must be lower than a user-defined $\epsilon_{\Delta}$.
- $R_v(X)$ in Equation 2.5 is called the *residual*, and it should tend to go to zero as $v$ increases. It is used to determine a mass conservation error which must be below a user-defined $\epsilon_{mass}$.

### 2.2.3. Solving the pressure system using Algebraic Multigrid

In reservoir simulation, the system's behaviour is a mix of different feature characteristics. Typically, the pressure gradient is long-range, which can be mathematically described using a PDE with *elliptic* behaviour. The PDEs for saturations (i.e. all possible fluids in the system) have *hyperbolic* traits with sharp local changes. Explained simply, a change in pressure at any given point propagates much further into a domain than changes in saturation. The elliptic type of PDE has a strong global coupling and is more complex to solve accurately due to its long-range character. Computational methods

for global-type solutions are expensive and not relevant to use on the hyperbolic parts of a system. As mentioned in the previous section, IX uses the CPR preconditioner to decouple the pressure and saturation system. The decoupled pressure linear system is then solved using the *algebraic multigrid* preconditioner. The concept of algebraic multigrid is relevant as it can recognized in certain ML auto-encoder methods, and it will linked to certain concepts in this work. It is also part of the discussion at the end of this work (see subsection 6.4.1).

### Geometric multigrid

To intuitively understand the idea of *algebraic multigrid*, explaining the original method *geometric multigrid* is helpful. Geometric multigrid is a numerical technique developed to solve partial differential equations (PDEs), particularly in the context of simulating physical phenomena like fluid flow or heat transfer. Without going into the mathematical formulation, consider the following explanation. First, a hierarchy of grids is created, where each grid represents the same system (i.e. reservoir grid) at a decreasing level of detail, i.e. increasing level of coarseness. The values per coarsened grid cell are derived from its finer grid equivalents using a *restriction* function. Then an iterative method (e.g. Jacobi or Gauss-Seidel method) is used to solve the linear system (to the next state $t + 1$) on the coarsest grid, where the problem is simpler, and that solution is used to inform the same problem on the finer grids, ultimately reaching the original level of detail. The upsampling is often referred to as *prolongation*. This method has shown to accelerate solving the eventual fine grid solution from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. The framework is visually shown in Figure 2.2.

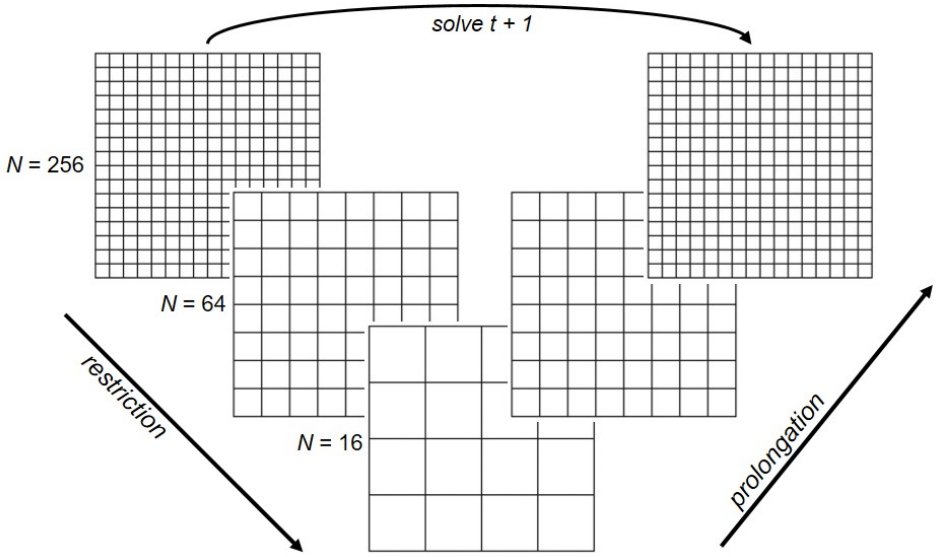



**Figure 2.2:** The 'V-cycle' multigrid approach.

### Algebraic multigrid

In contrast to geometric multigrid, which is based on the structure of the underlying grid, *algebraic multigrid* (AMG) operates directly on the linear system of equations without explicit knowledge of the grid. AMG is a more general multigrid approach for a wider range of problems, e.g. when dealing with complex geometry (i.e. unstructured grids). AMG is used to solve a linear system of equations like the one from Equation 2.5:

$$Au = f \tag{2.7}$$

like the rewritten from Equation 2.5:

$$\left(\frac{\partial R}{\partial X}\right)_v \cdot \Delta X = -R_v(X). \tag{2.8}$$

Here $A$ equals the Jacobian matrix in $\mathbb{R}^{N\times N}$. In reality, the CPR preconditioner decomposes the pressure components in the Jacobian and inserts that for $A$. $u$ remains to be the update vectors $\Delta X$ to solve for, and $f$ is the residual $R_v$ of the nonlinear system at the $v^{th}$ Newton iteration.

To solve $u$, AMG implements a coarsening hierarchy by using restriction and prolongation operators $R_i$ and $P_i$, where $R_i = P_i^\mathsf{T}$ for each hierarchal layer $i$. Then every next coarsening is given as $A_{i+1} = R_i A_i P_i$ according to the *Galerkin* principle. Restriction of $u_i$ and $f_i$ is derived as respectively $u_{i+1} = R_i u_i$ and $f_{i+1} = R_i f_i$. Similarly, prolongations bring back $u_i = P u_{i+1}$ and $fi = P f_{i+1}$. Derivations of the prolongation operator $P$ can vary, but it is generally composed using a heuristic method bases on the values in $A_i$. It summarizes to coarsening in directions where the algebraically smooth error changes slowly [48]. This will not be further discussed here. Using an iterative method the solution is found on the coarsest level and interpolated to the original grid. By default, smoothing (relaxation) iterations like Jacobi / Gauss-Seidel are also applied on every restriction/prolongation level.

The overall idea of solving the problem has remained the same compared to *geometric multigrid*, i.e. decreasing the size of the problem and gradually solving a more detailed version of the system. The iterative methods, usually the Jacobi or the Gauss-Seidel method, act as smoothers that reduce high-frequency errors in the linear system, which take a long time to smooth (i.e. solve) on the 'finer' levels. This way, the solution of the linear system as part of the Newton iteration is computed much faster (and thus allows for lower error thresholds) than without the AMG preconditioner.

# 3

# Graph Neural Networks

Graphs are a versatile tool for representing data. Certain types of data naturally focus on connections and as such intuitively map to graph representations. Examples are molecular structures [8], social networks [29] and traffic systems [18]. Less apparent examples of data that can be effectively represented as graphs include images [13], text [57] and interacting particles in physical systems [44]. Graph-based representations are useful as they allow data to be unstructured. This flexibility means graphs can be represent data in a non-Euclidean space, which can be useful when using graph representations for unstructured subsurface 3D grids, as will be discussed later.

Graph Neural Network (GNNs) principles were introduced in 2005 [11] and 2009 [45], based of Recurrent Neural Networks (RNNs). Most research interest has come in recent years after the widespread success of Deep Learning (DL) [26] and Convolutional Neural Networks (CNNs) [25]. Kipf & Welling (2017) [23] introduced the Graph Convolutional Network (GCN), which has become widely recognized and applied. Other important GNN extensions introduced are the Gated Graph Sequential Neural Network [30], a sequential output GNN using Gated Recurrent Units (GRUs), and the Graph Attention Network [51].

Most of the GNN methods follow a message passing paradigm, aggregating information from neighbouring nodes to spread information between graph nodes, creating an updated graph state, and repeating such aggregation several times. Various aggregation functions have been proposed in conjunction with different GNN architectures to suit specific graph representation learning problems like node classification, node/edge prediction and clustering. The following sections describe basic concepts in Graph Neural Network algorithms, which are then used to formulate the GCN layer [23].

## 3.1. Message Passing Algorithms

GNNs rely on a principle called *message-passing* to exploit relations between nodes in the network. A message-passing algorithm is a type of propagation algorithm that passes on the state of a node to another connected node. In the simplest form, a node receives information from every node it is connected with, and based on the combined information a node has received it creates a new node state. For that reason, message-passing is also termed neighbourhood aggregation. In a structured domain, e.g. an image, such an operation resembles a convolution. As such message-passing algorithms can be seen as convolution operators generalized to the irregular domain. For this reason, some message-passing graph operations have been named *graph convolutions*. To get an intuition of what happens during a message-passing operation, Figure 3.1a shows the information propagation from and to a single node during a series of graph convolutions, and figure Figure 3.1b shows the aggregated buildup of passed node states from 3 different starting nodes.

### Adjacency Matrix

Graph connections or edges can be described in an adjacency matrix $A \in \mathbb{R}^{N \times N}$ for a graph of $N$ nodes, using a binary encoding for an edge in a given direction. Graphs can be directional / directed,

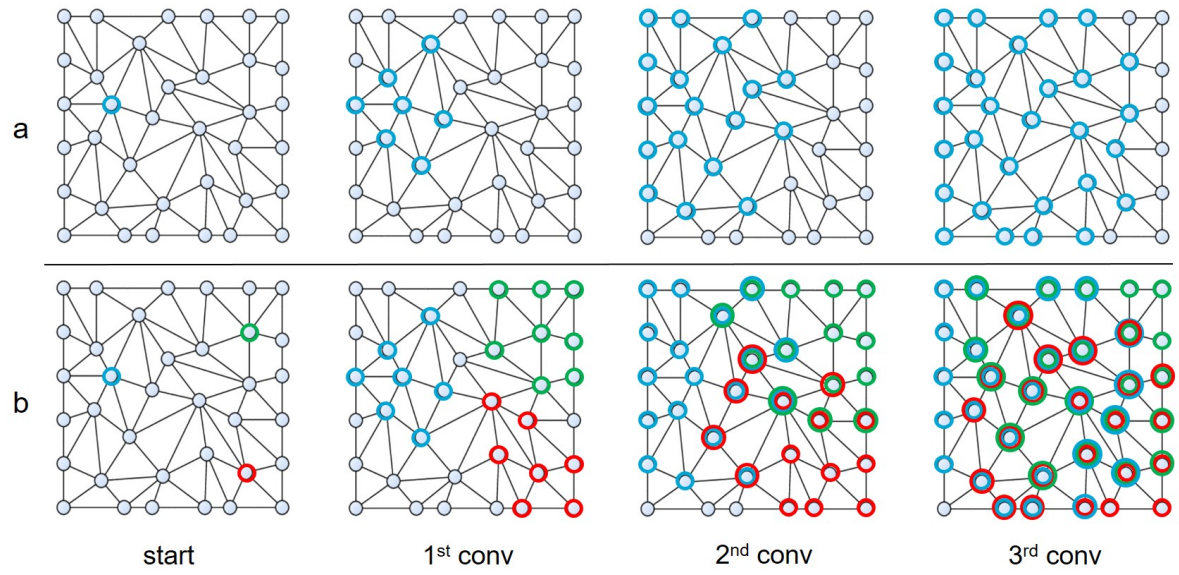



**Figure 3.1:** Information spread of a) a single node b) three nodes during over multiple graph convolutions. Message. a) After 3 convolutions, almost all the nodes have passed information to the starting node and vice versa. b) Information mix increases as more graph convolutions are performed, in practice happening for every node in the graph (only 3 nodes shown for simplification).

meaning the edges are one-way. This determines the direction in which a node can propagate information. Graphs can also be bidirectional / undirected, i.e. edges are both ways and information can propagate in either direction. Essentially bidirectional graphs are directional graphs where every edge has a duplicate in the opposite direction. This can be clearly seen using the adjacency matrix, which treats every graph as a directed one. Given the example graphs in figure Figure 3.2, examine the corresponding adjacency matrices. Observe the symmetry in the adjacency matrix for the undirected graph. Any undirected graph will have a symmetrical adjacency matrix, mirroring each edge.

$$A_a = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.1}$$

$$A_b = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.2}$$

Degree

In graph theory, the degree of a node defines the number of edges connecting to a node. In a directed graph every node will have a separate in- and outgoing degree. This explanation considers the situation of an undirected graph, which is the type of graph used in the rest of this work. A degree matrix $D \in \mathbb{R}^{NxN}$ denotes each node's degree in its diagonal. For graph $b$ in figure Figure 3.2, observe the degree matrix.

$$D_b = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Note that this matrix can be constructed by taking the sum of either the corresponding row or column in $A$ and filling its value on the diagonal. The sum of the diagonal is twice the amount of undirected edges in the graph, as the formalization still considers that every undirected edge is made up from two mirrored directed edges.

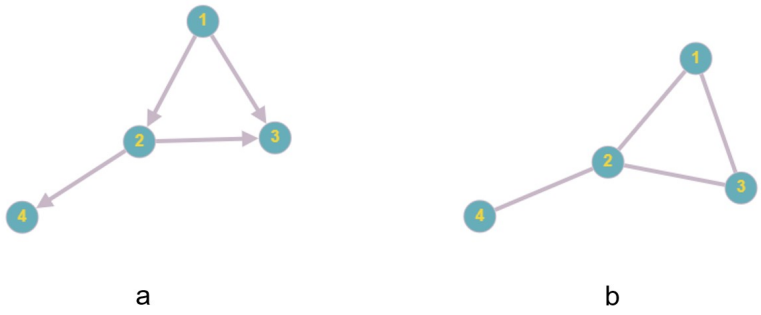



**Figure 3.2:** Example of a) a directed graph with directed edges going in a defined direction and b) an undirected graph where edges have no direction / all edges are bidirectional.

#### Self-connections

In the context of the reservoir simulation problem in this study, a node's saturation and pressure state of the next time step will depend on the state of its neighbouring grid cells. But the node's own saturation and pressure state will also be relevant for this. Like many graph problems, the message passing procedure should process the node's own value as well. To include this information in a message passing algorithm, self-connections (or self-loops) are introduced that represent an edge for every node with itself. It can be formalized in the adjacency matrix by adding an identity matrix [23]:

$$\tilde{A} = A + I_N, \tag{3.4}$$

In the adjacency matrix, this comes down to replacing the main diagonal with all ones. A self-connection has itself as source node and target, and thus does not have to be defined two-way for undirected graphs. In fact, this is not even possible looking at the adjacency matrix, as the edge is defined on the main diagonal.

## 3.2. Spatial vs spectral graph convolutions

There are two types of graph convolutions: spatial and spectral graph convolutions. In practice it can be difficult to define whether a specific type of graph convolution operates fully spatial or spectral, as will be made clarified below. The GCN layer formalized later in the chapter combines spatial and spectral characteristics.

### 3.2.1. Spatial graph convolutions

In spatial graph convolutions, feature information from a node's neighbours is aggregated to produce a new feature representation for that node. This can be done using a variety of aggregation operators, such as mean, sum, max, or a specific aggregation function. Classic CNNs for image classification can be seen as spatial convolution methods as the convolution kernels (i.e. the aggregation function) operate on a spatially encoded grid in the neighbourhood of the kernel center [60]. As such most spatial graph methods operate locally by spreading information in their local node neighbourhood. Complexity for spatial methods tends to scale linearly with the amount of nodes and edges. Despite their origin in spatial graph theory, where not only the graph's topology is relevant (i.e. the adjacency matrix $A$) but also the spatial ordering, spatial graph convolutions are used for many non-spatial problems. An example of this is GraphSage, a successful spatial graph convolutional method applied to citation and Reddit data [12].

### 3.2.2. Spectral graph convolutions

Spectral graph theory approaches graphs as matrices (i.e. the adjacency matrix $A$), and leverages $A$, the *graph Laplacian* $L_G$ and its eigendecompositions to perform analysis and permutations in the

spectral domain (i.e. the Fourier domain). The graph Laplacian of a graph $G$ is given as

$$L_G = D - A, \tag{3.5}$$

where $D$ and $A$ are the degree and adjacency matrix. In most literature, it is simply written as $L$.

To formalize a spectral convolution, we first find the eigendecomposition of $L$, which is necessary for the (reverse) translation of the node features to the Fourier domain. Since $D$ and $A$ are symmetrical matrices, so is $L$. In this symmetrical case the eigendecomposition can be written as

$$L = U\Lambda U^{-1} = U\Lambda U^{\top}. \tag{3.6}$$

Using the laplacian's eigenvalues $\Lambda = diag([\lambda_0, ..., \lambda_{n-1}]) \in \mathbb{R}^{N\times N}$ and eigenvectors $U \in \mathbb{R}^{N\times N}$ as the Fourier basis [7], $U^{\top}x$ translates every node's features $x \in \mathbb{R}^N$ to the Fourier domain. Since this convolution is spectral, the convolution filter (analogous to a spatial kernel in CNNs) $g_\theta = diag(\theta)$ is parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain by definition. The convolution theorem states that a convolution in the real domain can be described as the point-wise multiplication in the Fourier domain, so the spectral convolution becomes

$$g_\theta * x = U g_\theta U^{\top} x, \tag{3.7}$$

which translates back to the real domain using $U$. As $\Lambda = diag([\lambda_0, ..., \lambda_{n-1}]) \in \mathbb{R}^{N\times N}$ and $g_\theta = diag(\theta) \in \mathbb{R}^{N\times N}$, $g_\theta$ can then be seen as a function of the of the eigenvalues of $L$, i.e. $g_\theta(\Lambda)$.

The complexity of matrix multiplications with the eigenvector matrix $U$ is $\mathcal{O}(N^2)$, and eigendecomposition can be expensive for large graphs. Spectral convolutions are powerful in their ability to aggregate information across the entire graph, but costly as graph size increases. The next section discusses a computationally cheaper variation of the spectral graph convolution introduced by Kipf & Welling [23].

### 3.2.3. GCN

To create a computationally cheaper version of the spectral graph convolution, the eigendecomposition and the matrix multiplications have to be taken out of the computation. To do this, Kipf & Welling [23] proposed to use a *1$^{st}$*-order Chebyshev polynomial approximation of the spectral convolution:

$$\theta_0' x - \theta_1'(L - I_N)x, \tag{3.8}$$

for every node state $x \in \mathbb{R}^N$. Using the *1$^{st}$*-order approximation provides a localized graph convolution, reaching only the direct neighbours of the convoluted node. Its complexity is $\mathcal{O}(|\mathcal{E}|)$, where $|\mathcal{E}|$ is the number of edges in the graph. Like in spatial approaches, a higher-order convolutional filter (analogous to kernel size in CNNs) can be regained by stacking these convolutions in series.

#### Renormalization trick

In order for neural networks to be effective, normalization is usually essential to ensure numerical stability and prevent exploding or vanishing gradients. A normalizing operator could be created by multiplying the inverse of the degree matrix with the adjacency matrix $D^{-1}A$, averaging the aggregated node state based on the outgoing degree. Another way of normalizing would be to use the *normalized graph laplacian*

$$L = I_N - D^{-1/2}AD^{-1/2}, \tag{3.9}$$

in combination with the self-connected adjacency matrix $\tilde{A}$ from Equation 3.4, which Kipf & Welling [23] proposed as the *renormalization trick*:

$$I_N + D^{-1/2}AD^{-1/2} \longrightarrow \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}. \tag{3.10}$$

This squeezes the self-connections between the square roots of the inverse of the degree matrix $\tilde{D}$ (calculated from the self-connected $\tilde{A}$). This way of normalizing takes into account the degree of the source node degree as well as the degree of the destination node, whereas $\tilde{D}^{-1}\tilde{A}$ would only utilize the degree of the source node. Using this method tends to give better numerical stability [23].

Finally, by constraining $\theta = \theta_1^{'} = -\theta_0^{'}$ and using the *renormalization trick* the convolution becomes

$$\theta \left( I_N + D^{-1/2} A D^{-1/2} \right) x = \theta \left( \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} \right) x. \tag{3.11}$$

where $N$ is the number of nodes in the graph.

Given the proposed graph convolution in Equation 3.11, the formalization of the Graph Convolutional Network (GCN) layer becomes

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} W^{(l)}) \text{ [23]}, \tag{3.12}$$

where the network input $H^{(0)} = X$ and $H^{(l)}$ is the result of the convolution at the $l^{th}$ network layer after processing by an activation function $\sigma$. The learned parameters are stored and updated in the weight matrix $W^{(l)}$ of each layer.

The convolutional layer proposed in Equation 3.12 has gained a lot of reputation in the field of GNNs. Often when mentioned in online sources 'GCN' refers to this type of convolutional layer, and it also is the convolutional layer integrated in the network architecture proposed in this work. However, it is good to note the GCN layer formulated here is not necessarily a generalization of graph convolutions. There are various other types of graph convolutional methods in use for different types of problems. In section 3.5 an online PyTorch graph library is covered that has out-of-the-box implementations for many graph convolutional methods.

## 3.3. Network depth

A message passing algorithm passes node information to connected nodes. A message passing model needs to perform minimally **L** message passes to propagate information **L** node connections away, taking the shortest path from the information's source node. For a GCN this corresponds to the amount of graph convolutional layers it implements, defining its network depth **L** (refer to Figure 4.2 for a visual example of a GCN network architecture). Do not confuse **L** with the *graph laplacian* $L$ mentioned in the previous sections. The limit on the information propagation across the graph is given by this network depth [8]. This is demonstrated in an experiment shown in figure Figure 3.3, where it can be clearly seen that the $\mathbf{L} = 4$ is sufficient to propagate the starting node state to all the nodes of a $3 \times 3$ target square shape, but not a $5 \times 5$ target square shape. For the $3 \times 3$ grid, the network requires 4 node hops to reach the furthest node. For the $5 \times 5$ grid it requires 8 edge hops. In other words, the amount of connections it has to pass a message through is larger than the network depth, making it impossible to reach and construct the complete target shape.

This effect should be kept in mind when predicting spatiotemporal changes on an input grid given the timestep size $dt$. If the change happening during $dt$ is greater than the reach of the network, then the network is not be able to express it. How well the network is trained does has no impact on this.

In the context of the graph reconstruction problem in this work, the long-term samples with large $dt$ (post-injection, >10 years) and the early samples (start injection) with heavy saturation changes are at risk of this network depth clipping. The used time intervals $dt$ have to be chosen accordingly. Similarly, the pressure modelling is inherently limited by this effect. As put forward in subsection 2.2.3, pressure is described as an elliptic long-range effect. The network depth imposes a limit on this range in the GNN model.

## 3.4. Over-smoothing

Another effect that has to be accounted for when using GNNs is over-smoothing. It can be described as the exponential convergence of all node features to a constant value as network depth increases, and as such it limits the expressiveness of deep-layered GNN architectures [41]. In graph reconstruction problems, the over-smoothing effect can easily be observed. In the experiment shown in figure

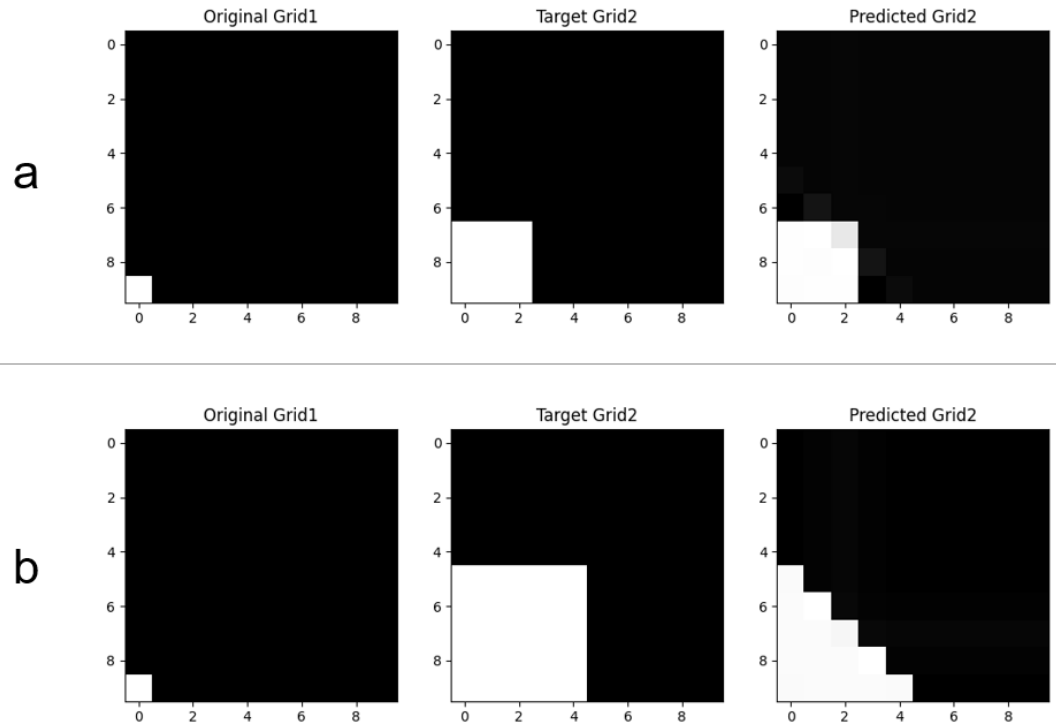



**Figure 3.3:** Simple experiment with **L** $= 4$, showing how network depth limits the flow of information in a message-passing GNN. The construction of the target shape in a) requires 4 edge hops between nodes to reach the furthest node in the target (right top), the exact range over which the model can propagate information. The model constructs the target shape, with a slight over-smoothing discrepancy (see section 3.4). The target shape in b) requires 8 hops, more than the network depth, making it impossible to spread the information far enough. The diagonal edge of the constructed shape appears as those cells are all maximally 4 edge hops away from the input grid's node.

Figure 3.4 the network depth **L** $= 10$ should is sufficient to reach every node of the target shape, but the features average as they propagate further from the input shape's source node, creating a smoothed edge around the pattern. The larger the network depth **L**, the stronger the over-smoothing effect will be.

### 3.4.1. Residual connection

To mitigate the effect of over-smoothing, residual connections are often added to the network layers. Residual connections feed part of the layer input straight to the output, skipping the message-passing in that layer. As such the layer output will be composed of a portion of the input feature state and the aggregated feature state before the learned weight vector $W^l$ is applied to it. A famous architecture built using residual principles is ResNet [14], designed to train very deep network architectures used for e.g. image recognition. While using residual connections in GNNs is useful for different type of graph problems that require deep network architectures [5], for reconstruction problems it also makes intuitive sense as we expect our output to be a changed version of the input. Feeding part of the input to the output helps to reconstruct basic feature characteristics that will likely re-appear in the next state. Residual connections usually skip the $l^{th}$ input to its output, but a better option for graph reconstruction might be to skip the initial input $H^0$ to every layer, as proposed by Chen et al. (2020) [5]. Figure Figure 3.5 shows how well such residual connections help to construct the target shape. Using this type of residual connection the formalization of the GCN layer becomes

$$H^{(l+1)} = \sigma\left((1-\alpha_l)\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)} + \alpha_l H^{(0)})W^{(l)}\right)\ [23], \tag{3.13}$$

Note the weight vector is multiplied only after adding the input residual to the aggregated feature state. This means the learnable weights of the layer are still trained to combine the input residual with the

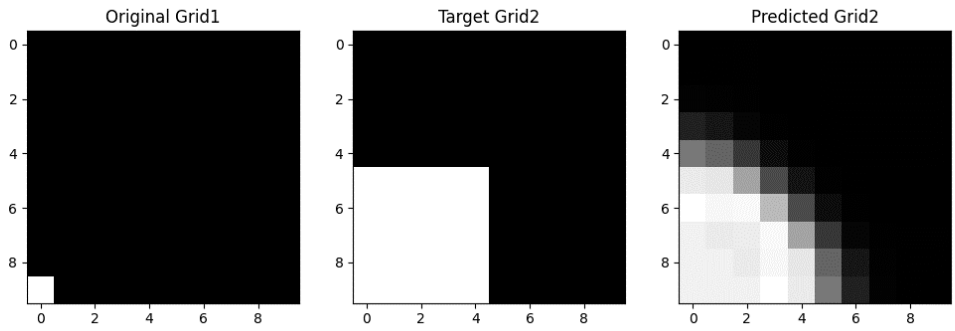



**Figure 3.4:** Over-smoothing effect visible on the same construction task as shown in Figure 3.3, but now with sufficient network depth to reach every node in the target. Still the shape is not perfectly generated, which is due the dilution of information that gets worse with more consecutive graph convolutions.

aggregated feature state. Another option would be to add the the input residual (or the entire input) after the layers' weight multiplication (or all of them), as shown by Keisler [21]. The model then learns to predict the change mask or feature changes compared to the previous state, and add that on top of the previous state.

To show the effectiveness of residual connections, observe the results in figure Figure 3.5. Not only are edges very hard to capture without the residual connections, the surface consistency of the input also really suffers. Besides the blurred edge, the over-smoothing effect creates artificial geometrical variations induced by information propagated from the input shape's nodes. Adding residual connections helps the model to reconstruct the original consistency and edge gradient perfectly in this simple case.

## 3.5. PyG

PyTorch Geometric (PyG) is a Python library for machine learning and deep learning on graphs [9]. It is built on top of PyTorch and provides a variety of features for working with graph data out-of-the-box. It provides an extensive set of tools and functionalities that make it a good choice for researchers intending to work with graph neural networks (GNNs). It offers a rich set of data transformation and pre-processing functions, and allows users to efficiently load, manipulate, and preprocess graph data. It also allows for efficient mini-batching, combining multiple data samples together in a larger disconnected graph. The library supports various types of graphs, i.e. homogeneous graphs (where all nodes and edges share the same attributes) and heterogeneous graphs (where nodes and edges can have distinct types and attributes). It facilitates (multi-)GPU training with NVIDIA RAPIDS™ support (a suite of GPU-accelerated Python libraries). Custom convolution operators, aggregation functions and network architectures can be constructed, but it offers many implemented graph convolutions, aggregation funtions and network models out-of-the-box with good documentation, referencing the original research papers. Visit https://pyg.org/ for more information.

## 3.6. GNN demo: structural learning

To demonstrate the (re)constructive power of a simple GNN in a 3D context, a simple model (similar to the one used in this work, see section 4.2) was trained to learn the unstructured shape shown in Figure 3.6. It is trained on a portion ($< 0.5$) of the nodes from the target shape, and has to infer the rest of the structure, as if constructing a reservoir grid from a set of seismically gathered data points. It shows the GNN is capable of approximating the general shape of the target to a good extent. It has to be noted that this experiment concerns constructing the reservoir grid, whereas the main research in this work assumes a known reservoir grid and tries to solve a *reservoir simulation* problem (revisit the introduction in chapter 2 for a clear distinction on this). Also, if such a reservoir construction model would be applied in practice it would assume there are structural characteristics in reservoir grids that generalize to other

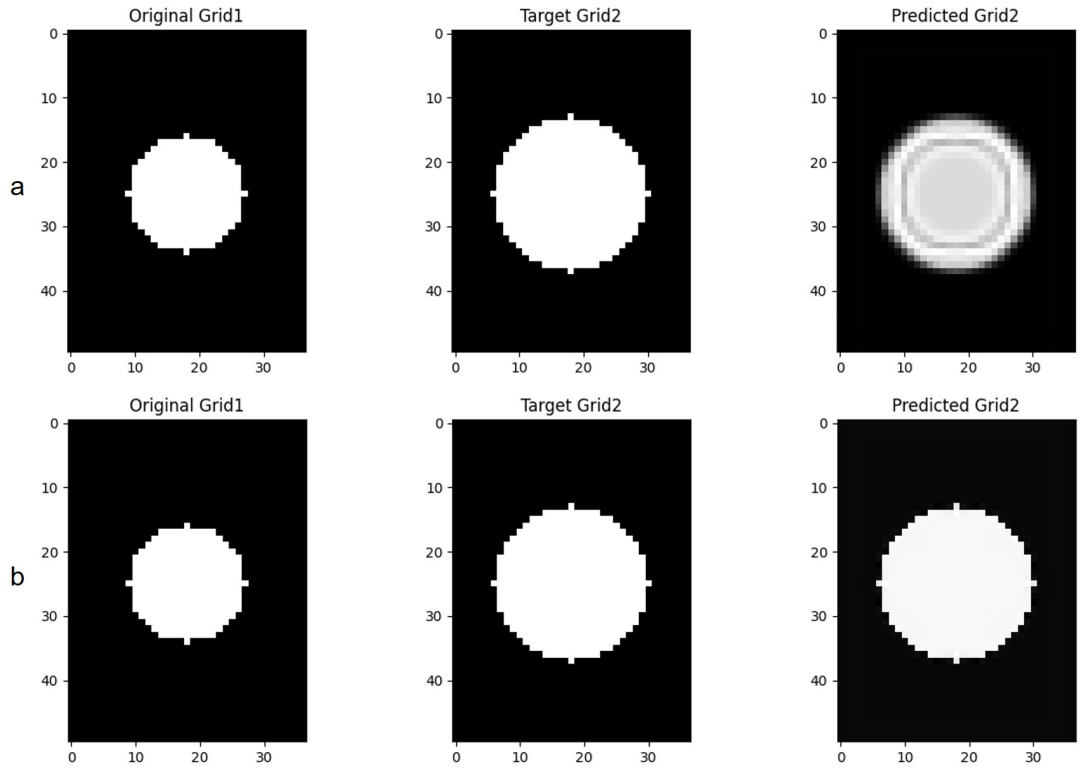



**Figure 3.5:** Experiment showing a) the over-smoothing effect on a circle-growing model, b) which is perfectly achieved by adding a residual connection to the graph convolutions.

reservoir grids. While this may be true, this experiment does not claim this and the rest of this work does not cover this hypothesis. A related hypothesis is covered, testing the generalization of CO2 flow to differently structured grids (subsection 1.1.3, subsection 4.5.4).

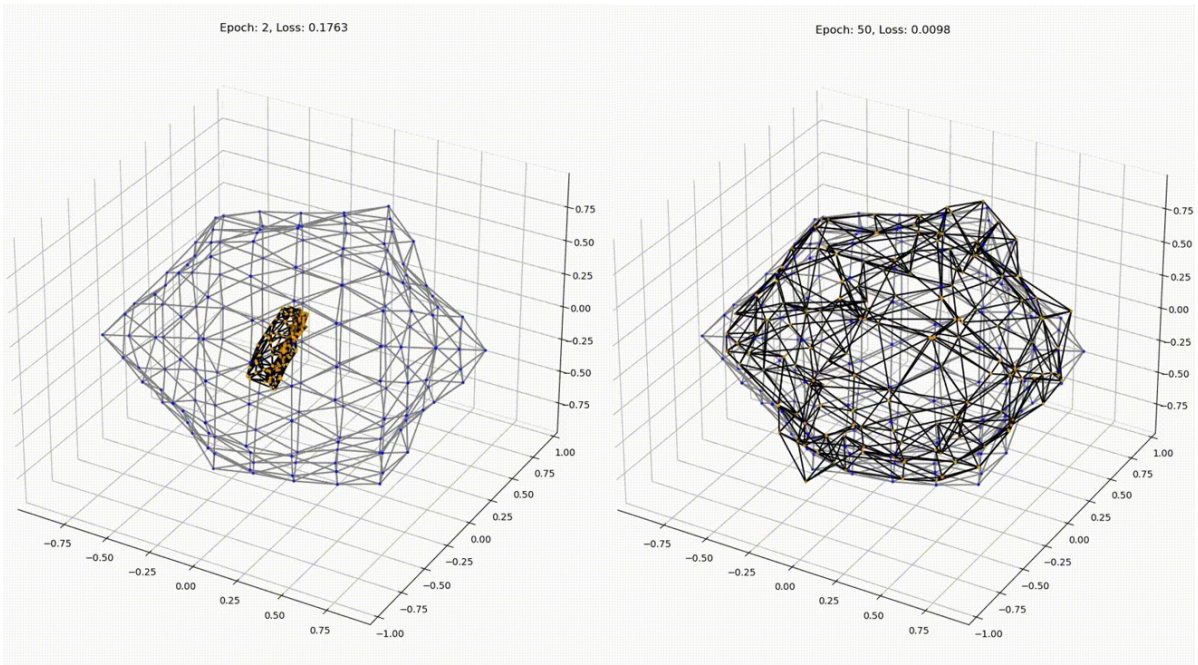



**Figure 3.6:** Simple autoencoder GNN that reconstructs a target graph structure. In this experiment, the model is not used as a simulator but as a grid approximation method.

# 4

# Methods & Experimental Design

This chapter conceptualizes how GNNs are applied to reservoir simulation. It first covers the proposed methods, and then lays out the experimental design used to answer the research questions formulated in section 1.1. It covers the network architecture using the GCN operator introduced in chapter 3, its *autoregressive* usage and feature pre-processing. Then it discusses the generated datasets used in this work and the experimental design with corresponding results following in chapter 5.

## 4.1. Autoregression

In the introduced method, the goal is to produce the sequential feature states $Y_T = [Y_{t+1}, Y_{t+2}, ..., Y_{t+x}]$ for a time range $T = [t, t + x]$. Generally, a simulation will start from the equilibrium feature state $X_{t=0}$, which represents the reservoir before CO2 injection. Given the limited spatial reach of the graph convolutional network introduced in the previous chapter, the proposed GNN method can not be used to produce a highly variated state $Y_{t+x}$ instantly (see section 3.3 for explanation). The network is used *autoregressively*, i.e. iteratively feeding the predicted state as input for the next prediciton. This approach resembles numerical reservoir simulators in the sense that the GNN solves a non-linear system in a discretized timestep. Discretization is essential in numerical methods to approximate solutions to the governing PDEs in the continuous time domain. Similar to the numerical approach, time discretization simplifies the nonlinear system to a smaller domain and allows the neural model to learn a generalization of the physics instead of solving the entire continuous time range at once. Besides the numerical similarity, a series of predicted feature states is analytically also more relevant than producing just a final state. On the downside, long-term reservoir simulation predictions involve many iterations to ensure a sufficient timestep resolution. The model requires a sufficiently small error on single iterations in order to provide meaningful predictions. Figure 4.1 shows the flow of information in the autoregressive loop. The features in the state vectors $X_t$ and $Y_t$ will be discussed further in this chapter.

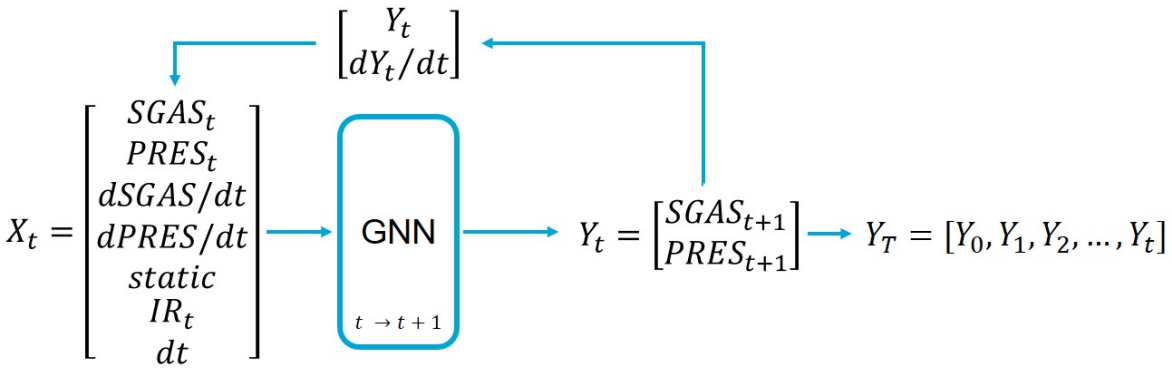



**Figure 4.1:** autoregressive flow of information during multistep training and long-term prediction. The model outputs and their temporal derivatives are fed back into the model as input for the next timestep, together with the *static* features, the injection rate (IR) at timestep $t$ and the timestep size. For multistep training of timestep $t$, $T = [t, t + 1, ..., t + x]$.

## 4.2. Architecture

The network architecture used in this work is based of the paper: ”Simple and Deep Graph Convolutional Networks” by Chen et al. (2020) [5], implementing their version of the GCN operator with included residual connection. The architecture can be seen as a simple autoencoder with an encoder, processer and decoder structure. As encoder It uses a regular fully-connected linear layer (i.e. dense layer) to map the input features to a larger latent dimension. As processor it applies **L** GCN layers (see subsection 3.2.3), performing the the message-passing. The used convolution includes a *residual connection* (explained in subsection 3.4.1 in chapter 3) with strength $\alpha$. The resulting feature states in the latent dimension are mapped back to the output dimension using another fully-connected layer as decoder. See figure Figure 4.2 for a visualization of the architecture. The output dimension represents $Y_t$, the 2 predicted output features *CO2 gas saturation* (SGAS) and *pressure* (PRES). To introduce non-linearity, after every layer (except the last linear layer) a (leaky) ReLu activation function is applied. More about the optional leaky ReLu coefficient in subsection 4.3.5.

This simple architecture operates node-wise and does not include any pooling or upsampling on the nodes $|\mathcal{V}|$ in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ respectively define the vertices and edges. This means the architecture relies on the success of message-passing in the **L**-local node neighborhood, making the architecture spatial in nature. The complexity of the graph convolution strictly does not rely on the amount of nodes $|\mathcal{V}|$, but on the amount of edges $|\mathcal{E}|$ that have to be passed. However, in the reservoir grid context we can expect $|\mathcal{V}|$ and $|\mathcal{E}|$ to be linearly related (every grid cell will only have so many neighboring cells, even in an unstructured grid), so network complexity can be described as either $\mathcal{O}(|\mathcal{E}| \cdot \mathbf{L})$ or $\mathcal{O}(|\mathcal{V}| \cdot \mathbf{L})$. Simply put, the network complexity scales with the input size of the graph and the amount of convolutional layers.

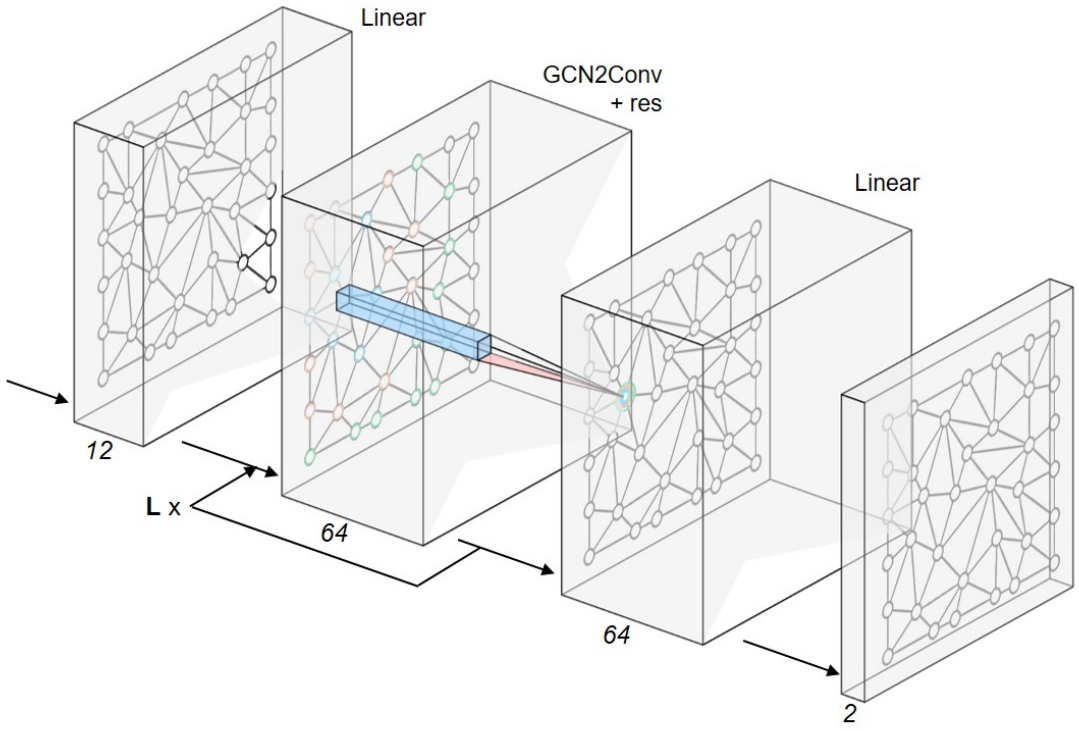



**Figure 4.2:** Network architecture used in this work. The model maps 12 input features to a 64-dimensional latent space using a fully connected layer, where it applies **L** GCN2Conv operators to perform message-passing and apply model weights. It then maps the final feature state to the 2 output features *saturation* and *pressure*. The architecture accepts any arbitrary graph size or structure.

## 4.3. Features

A simulated CCS reservoir model contains various geological (i.e. static) variables, as well as dynamic parameters (e.g. CO2 injection rate, saturation and pressure), which are referred to as *features* in the rest of this work. Static features encode the geological structure of the reservoir system. They are constant during the entire simulation, so they do not have to be updated or predicted. Dynamic features encode the changing variables of the reservoir system, influenced by their own state and CO2 entering the system through injection. The dynamic features *gas saturation* and *pressure* are the target features to predict. As their development is determined by the combined static and dynamic feature state, both static and dynamic parameters are used as input features in the GNN. The features are encoded per grid cell in the reservoir model, encoding them per graph node in the graph translation of the reservoir model. The dynamic features are predicted by the model on every network iteration and used to update the dynamic features' input of the next iteration, as depicted in Figure 4.1. Finally, externally controlled variable features (injection rate, timestep size) are known for every timestep and do not have to be updated based on the predicted states.

The static input features are

- *PORV:* pore volume, i.e. cell fluid capacity. Determined by the rock volume and associated porosity (*PORO*).
- *DEPTH:* cell depth, i.e. relative depth of cell. Relevant for upward buoyancy forces acting on injected CO2.
- *TRANX:* transmissibility in $x$-axis, determines the rate by which CO2 can propagate through the cell in the $x$ direction.
- *TRANY:* transmissibility in $y$-axis, not relevant for the used 2D case but relevant when transferring to a 3D model.
- *TRANZ:* transmissibility in $z$-axis

The updated dynamic input features are

- *SGAS:* CO2 gas saturation
- *PRES:* pressure
- *dSGAS/dt:* saturation accumulation speed
- *dPres/dt:* pressure accumulation speed

The non-updated variable input features are

- *IR:* injection rate, nonzero only for cells at which an injector well is located.
- *dt:* timestep size

### Permeability edge weights

Adding static features representing geological properties is essential, but some are more straightforward than others. Features like *depth* and *pore volume* are distinct grid cell characteristics, but directional features like *TRANX, TRANY, TRANZ* could be less informative in message-passing based on the direction they are passed in. They might be better represented in the edges between nodes. To test this, another directional feature, *permeability*, is used to encode edge weights on every edge in the grid. Permeability defines fluid flow potential through a plane of porous media. It relates to transmissibility, but is more suitable for use on the edges as it does not depend on the cell's own dimensions like transmissibility does. For a rectangular structured case, every edge weight becomes the harmonic mean between the relevant directional permeability: *PERMX, PERMZ, PERMY*. For unstructured grids this would be trickier, requiring a multi-directional transform for edges spanning multiple dimensions. For the 2D structured cases used in this work (no edges in the $y$-direction) the edge weights for the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ become:

$$w_{(i,j)} = \begin{cases} \dfrac{2}{\frac{1}{PERMX_i} + \frac{1}{PERMX_j}} & \textit{if } x\text{-directional} \\ \dfrac{2}{\frac{1}{PERMZ_i} + \frac{1}{PERMZ_j}} & \textit{if } z\text{-directional} \end{cases} \quad \text{for every edge } (i,j) \in \mathcal{E} \tag{4.1}$$

The implemented GCN convolution allows for a single weight. The *permeability edge weight* will likely help modelling the local effect of gas saturation, but the weighted information propagation will also affect the pressure components in the message passes. Despite the fact pressure is also affected by permeability, it is likely the weights will induce high-frequency errors (i.e. local errors) disturbing the elliptic long-range effect of pressure (see subsection 2.2.3 for more details). Having separate edge weights (i.e. multiple edge features) might solve this, but it adds extra model complexity. The network would have to learn a mapping in the latent space between node and edge features to utilize this extra information. A GNN architecture that allows this is a Graph Attention Network (GAT) [51], but these architectures are computationally considerably more expensive. This work focuses on using single edge weights and evaluates if the learned model is able to adapt its parameters for the pressure output feature.

The 'spatial' contradiction

It is good to note that even though the architecture and the used convolutions can be characterized as 'spatial' methods, the network does not take the actual spatial information from the reservoir grid cells (i.e. the cell center/edge/face or other geometric coordinates) as input features. From a macro perspective this may be logical, as the location of a group a grid cells should not differentiate the physical behaviour inside it. But from a local perspective any such geometric information could be relevant for the physical behaviour in and around the grid cell. However, note that the model does actually not omit all the spatial information, as spatial characteristics are still encoded deeper in the other static features. Pore volumes, permeabilities, and transmissibilities are outcomes derived from relationships that incorporate geometric data from grid cells and geophysical principles (see subsection 2.1.1 for more details). The potential benefit that follows from this approach is that the implemented model should be able to learn to handle reservoir grids at different scales, given that the geological features are properly coarsened / refined and the model is properly trained at different scales as well.

### 4.3.1. Accumulation speed features

A challenging aspect for a local message-passing algorithm is to propagate the scale of injection across the grid. As the injection rate feature defines the speed at which CO2 enters the system, this information is valuable for proper spatiotemporal development. However, the injection rate feature is only nonzero at the grid cell where the injector well is located, out of the message-passing reach for most of the grid. Given the network depth limitation (see section 3.3), this information will not spread further than the local injector node's neighbourhood. To solve this, temporal derivative features per cell are added to act as 'change rates' for every cell, adding extra information in a message pass about the state changes neighbouring cells are experiencing. In the GNN model, two separate features are added to facilitate this, *dSGAS/dt* and *dPres/dt*. They represent the features' previous timestep accumulation speed, calculating the saturation/pressure change with respect to the timestep's $dt$ value. Experimenting with and without these features learned they are beneficial for the expressiveness of different injection rates. Adding spatial gradients has also been explored, but the performance compared to the temporal derivatives was generally considered worse. It can be reasoned that using the features' temporal derivatives per cell in a message passing context is like using spatial gradients, but with added information about the rate of change of the gradient.

### 4.3.2. Normalization

In order for a numerically stable model, feature normalization is implemented to bring every feature in the range $[0, 1]$. As different features have different numeric ranges, specific normalization is applied per feature.

- *PORV:* natural logarithm + fixed max
- *DEPTH:* min-max scaling
- *TRANX/Y/Z:* natural logarithm + max value (max across all dimensions).
- *PERMX/Y/Z:* natural logarithm + max value (max across all dimensions).
- *SGAS:* normalized by default
- *Pres:* min-max scaling using fixed values
- *dSGAS/dt* min-max scaling using fixed min-max values

- *dPRES/dt:* min-max scaling using fixed min-max values
- *IR:* fixed max

| Feature | ln() | Min | Max |
|---|---|---|---|
| PORV | x | 0 | 10 |
| DEPTH | | min(DEPTH) | max(DEPTH) |
| TRANX/Y/Z | x | 1 | max(TRANX/Y/Z) |
| PERMX/Y/Z | x | 1 | max(PERMX/Y/Z) |
| SGAS | | 0 | 1 |
| Pres | | 90 | 350 |
| dSGAS/dt | | -1e3 | 3e3 |
| dPRES/dt | | -1e3 | 3e3 |
| IR | | | 2e4 |

**Table 4.1:** Used scaling factors per feature and whether the natural logarithm is applied before scaling. The factors are based on the limits of the features' numeric ranges across both ensembles, ensuring all features are numerically close to a $[0, 1]$ range.

Two notes on the normalization methods used for this data. First, for use on realistic datasets, the use of the natural logarithm should be re-evaluated since some artificially created static properties in the generated datasets are not entirely representative of numeric ranges in standard geological data. Second, if a model were to be applied to different reservoirs, fixed normalization values should be applied on all features to avoid different feature scaling on separate reservoir datasets.

### 4.3.3. Loss function

A loss metric is required to backpropagate and optimize the learnable parameters of the neural model. The loss function used in the final model is the $MSE$ (Mean Squared Error) loss on the accumulation speed features $\frac{dSGAS}{dt}$ and $\frac{dPres}{dt}$ of the predicted timestep:

$$\mathcal{L} = \frac{1}{|\mathcal{V}|}\left(\sum_{v\in\mathcal{V}}\left(\frac{dSGAS_{v,true}}{dt} - \frac{dSGAS_{v,pred}}{dt}\right) + \sum_{v\in\mathcal{V}}\left(\frac{dPres_{v,true}}{dt} - \frac{dPres_{v,pred}}{dt}\right)\right) \tag{4.2}$$

This loss penalizes the predicted difference of the output features' temporal derivatives, which arguably stimulates the model to optimize the rate of change (i.e. the system dynamics) more than the actual grid values itself. This loss has shown to give better expressivity than simply taking the $MSE(Y_t)$, especially when predicting variable injection rates (which will be discussed in subsection 4.4.1).

### 4.3.4. Training procedure

To train the model, a selection of cases in split into a training and validation set (ratio $0.8/0.2$). Every timestep of each training case is a trainable data sample (i.e. 167 data sample per training case). Mini-batching is used to accelerate the training. Using larger batches has shown to give slower convergence, so small batch sizes ($8$ or $16$) are used which also increases generalization [33]. Batches are shuffled to increase generalization, i.e. each batch contains a random selection of timesteps from randomly selected training cases. The mean loss across the batch is backpropagated though the model to calculate the gradients with respect to the changes in the model's parameters. The Adam optimizer [22] then uses these gradients to update the model's learnable parameters. A training epoch is finished after all training samples have passed trough the model.

#### Multistep training

To stimulate the model's autoregressive behaviour, *multistep training* is introduced. Instead of predicting only $t + 1$, the model is iterated multiple times to simulate multiple steps $[t + 1, t + 2, ..., t + k]$ for a chosen *multistep* hyper-parameter $k$.

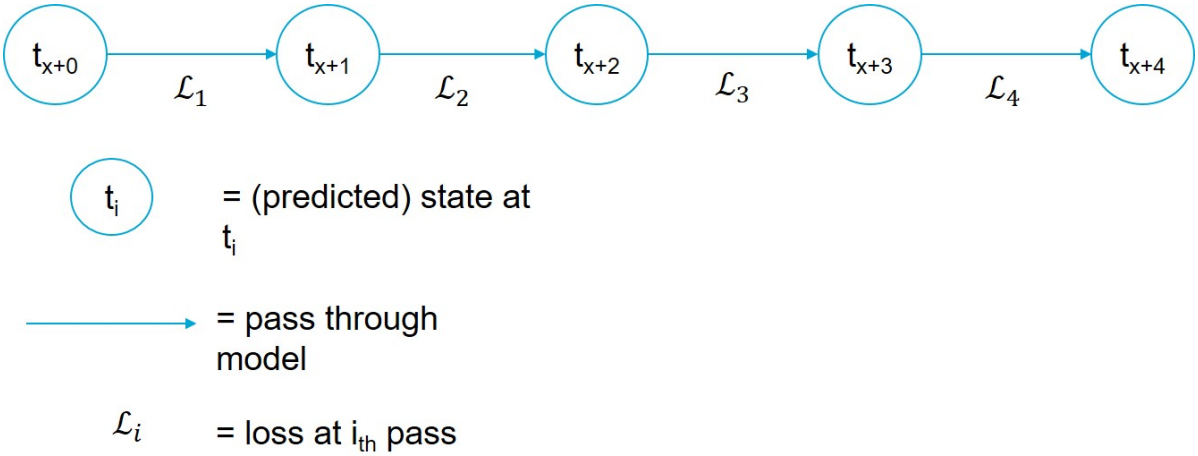



**Figure 4.3:** Multistep training procedure for $k = 4$. For every data sample (i.e. timestep starting at $t$) the model is autoregressively iterated for $k$ steps.

The resulting loss depends on the *multistep loss aggregation method* hyper-parameter, either '$sum$', '$mean$' or '$last$':

$$\mathcal{L} = \begin{cases} \sum^{k} \mathcal{L}_k & \text{if } 'sum' \\ \frac{1}{k} \sum^{k} \mathcal{L}_k & \text{if } 'mean' \\ \mathcal{L}_k & \text{if } 'last' \end{cases} \tag{4.3}$$

The idea of summarizing or averaging the losses is to learn the model to correct for mistakes it might make on single timesteps, whereas using only the last loss would exclude this learning behaviour by only focussing on the correct prediction of the $k^{th}$ multistep. This multistep loss is used as the training loss.

#### Model performance evaluation metric

The loss function is only used to calculate loss over the multistep prediction of each training sample. By minimizing the loss on the multistep intervals, the goal is learn the physical dynamics of the system to the model. However, to evaluate the performance of the model as a simulator, the autoregressive prediction over an entire timespan $T$ is imperative. To express the autoregressive performance, the same $\mathcal{L}_{dt}$ loss from Equation 4.3.3 can be used. The $MSE$ on the output features $Y_t$ is added for completeness:

$$\mathcal{L}_T = \frac{1}{|T|} \sum_{t=1}^{T} \mathcal{L}_t, \quad \mathcal{L}_t = \mathcal{L}_{dt,t} + MSE(Y_t) \tag{4.4}$$

### 4.3.5. Hyper-parameters

The model implements the following set of non-learnable hyper-parameters:

- learning rate
- batch size
- residual strength $\alpha$
- leaky ReLu slope coefficient $\theta$
- amount of GCN layers **L**, i.e. network depth
- multisteps $k$, i.e. how many timesteps the network predicts ahead for every trained timestep $t$, see subsection 4.3.4.
- multistep loss aggregation method, either 'sum', 'mean' or 'last'
- latent (hidden) dimensions, i.e. feature dimension in message-passing layers

Initially, these parameters have been chosen using common practices or theoretical motivations. At a later stage, they are empirically optimized using the hyper-parameter tuning as discussed in subsection 4.5.5.

## 4.4. Data

To study whether GNNs are suitable for reservoir simulation, which has not been extensively tested, a simple data model is a logical choice to start testing the proposed hypotheses about expressivity, injectivity and transferability (see subsection 1.1.1 - subsection 1.1.3). A 2D dummy reservoir is used in this work, referred to as the 'P7' reservoir. Using the INTERSECT numerical reservoir simulator and the P7 reservoir grid, two different ensembles are parameterized and simulated for 1000 years over 167 timesteps, i.e. $T = [0, 167]$.

#### P7 reservoir

The P7 reservoir is a 2D dummy reservoir containing 1850 rectangular grid cells of variable size. The 1850 cells are arranged in a rectangular $37 \times 50$ grid. Technically the grid is structured, having a fixed amount of neighbours (upper, lower, left, right). Despite the aim to test graph methods for the sake of processing unstructured grids, testing the validity on structured grids is a valid first step, also provided that the method does not rely on the structural (i.e. geometric) information. Furthermore, as unstructured grids are a generalization of structured grids, a graph method that works on unstructured grids should function on structured grids as well.

#### Shale layers

The P7 reservoir contains so called *shale layers*, layers with lower porosity and permeability than the average grid cells. These shale layers resemble possible variability in subsurface rock formation and influence the propagation pattern of the CO2 saturation. The reservoir has one injection well, located at cell $(31, 0)$. In Figure 4.5a, the original reservoir shape can be observed. As mentioned in Equation 4.3, the network does not consider the geometric grid shape, but rather a 'simplified' version like the grid shown in Figure 4.5c. This spatially-invariant abstraction is used in the rest of this work for visualizing predictions.

### 4.4.1. Ensemble 1: variable injection

An essential method in CCS reservoir simulation for making correct operational decisions is the ability to model different CO2 injection rates. Thus, a training set resembling different injection rate scenarios is required to create an effective predictive model. Using a uniform distribution, cases with variable constant injection rates are sampled and simulated using IX. All resulting samples have an identical injection phase (12 years, 143 monthly timesteps) and are sampled with identical static composition. See Figure 4.6 for the progression of true output features in the training data, summed over the entire grid.

### 4.4.2. Ensemble 2: variable shale composition

Any reservoir simulator model requires the ability to handle geological variations. To train this, an ensemble of the P7 reservoir is sampled with variable permeability and variable porosity in the shale layers. Porosity directly scales the pore volume (PORV) input feature, and permeability influences the edge weights as well as the transmissibility features. PERMX and PORO are independently uniformly sampled, then PERMZ is calculated by multiplication of PERMX with a decreasing coefficient (PERMY is irrelevant and thus zero in the 2D case). The surrounding non-shale reservoir properties are equal for every sample. The resulting gas propagation and pressure buildup across the shale layers varies per parametric combination, resembling different geological barriers present in reservoir data.

### 4.4.3. Post-injection timesteps

Both ensembles feature the same 12-year *injection* phase with constant timestep size of 1 month (28-31 days) for 143 timesteps. The *post-injection* phase timesteps increase progressively to 1, 10, 100 and eventually 500 years. As prolonged containment of CO2 is essential for the success of CCS, modelling the long-term behaviour is important. However, in the P7 simulations only the last 24 timesteps account for this variety of larger timesteps, forming only ~15% of the data points. Given the sparsity of these data points, the model is likely to struggle expressing them correctly. Another potential problem are the potentially larger state changes during larger timesteps, which could be out of the message-passing reach given the network depth (see section 3.3). For these reasons, performance of the model

in expressing this timestep variety is not a primary research objective, however it will be evaluated. Potential solutions for these problems are discussed in chapter 6.

### 4.4.4. Note on the training data

Readers without geological background might wonder to what extent the simulated data from INTERSECT resembles the subsurface reality. Given that we train a model on the simulated data, the ML surrogate model predictions will only get as good as the numerical simulations. For that reason. training a simulator on simulations might seem redundant. To understand why this is the objective, it is good to take into consideration that 1) the goal of building a ML surrogate model is not to exceed current reservoir simulation's performance but to accelerate it, 2) numerical reservoir simulation has shown validity through years of usage and reservoir models can be calibrated during operation through *history matching*[1], 3) the process of reservoir modelling (i.e. translating raw subsurface data into usable subsurface models) is a complex research field itself and beyond the research scope of most ML reservoir simulation research.

## 4.5. Experimental Design

To test the hypotheses set out in the research questions, an overview of the implemented experiments is listed below. Relevant details of the experiments and hyper-parameter choices are highlighted in the sections that follow.

RQ1: Expressivity
- Experiment 1: Single timestep prediction
- Experiment 2: Long-term autoregressive prediction

RQ2: Injectivity
- Experiment 3: Variable injection ensemble prediction

RQ3: Transferability
- Experiment 4: Variable shale ensemble prediction
- Experiment 5: Unseen grid prediction

### 4.5.1. Experiment 1: Single step prediction

To test basic expressive capability of the implemented GNN architecture, a single case is taken from the variable injection ensemble. A model is trained on this case to construct timestep $t+1\ \forall t \in T$. The model only trains on 167 data points (every available timestep), and evaluated is whether the architecture is capable of expressing the output feature behaviour at different timesteps correctly. One model is trained with multistep parameter set to $k=1$, to focus on single timesteps. Another model is trained with $k=6$, meaning the model already trains *autoregressively*. This way the effect of applying the *multistep training* can be evaluated.

### 4.5.2. Experiment 2: Long-term autoregressive prediction

For long-term autoregressive predictions, the model with multisteps $k=6$ is used. Instead of modelling every timestep separately, the model starts at $t=0$ and autoregressively predicts the full time range $T$. As shown in Figure 4.1, every iteration the previous output features $Y_t = [\text{SGAS}_{pred,t}, \text{PRES}_{pred,t}]$ are used to update the next iteration's dynamic feature inputs: *SGAS, PRES, dSGAS/dt, dPRES/dt*.

### 4.5.3. Experiment 3 & 4: Ensemble predictions

If the long-term expressiveness is confirmed, experimentation can be expanded to a more realistic setting by training on ensembles. For the variable injection ensemble, the goal is testing the model's capability to scale the progressive behaviour accordingly with the CO2 injection rate. Instead of using all the cases in the ensemble, the model is trained on 25 cases (4000 data points) with uniformly distributed injection rate. Early experimentation has shown that adding more cases does not improve expressiveness significantly, while it does linearly increase training time. For experimental efficiency

[1] Process of rebuilding a reservoir model with updated information, generally from wellbore measurements

the training set is kept limited. The variable shale ensemble is similarly treated, manually sampling 25 cases where porosity and permeability are distributed as uniform as possible. 6 cases with uniformly distributed features of interest are picked per ensemble as validation set.

### 4.5.4. Experiment 5: Unseen grid prediction

As the used network architecture implements a method operating in the local node neighbourhood, it is hypothesized (subsection 1.1.3) that a well-trained model can handle reservoir grid samples with untrained compositions, i.e. data samples within the trained parametric ranges but varying in grid size and geological structure. As the used GNN architecture can process any graph input size, a model trained on enough small but geologically characteristic samples should theoretically be able to accurately process any arbitrary reservoir grid. In the last experiment the model is tested on 3 grids where the geological structure of the grid has been modified (Figure 5.10). In structure A, gaps have been made in the shale layers by removing the low porosity and permeabilty, and some areas are made impermeable. In structures B and C, diagonal shale layers are constructed. The modified structures are not extensively geologically motivated, but the blocks in A and the (diagonal) layers in B and C could be interpreted as impermeable (cap)rock formations. Note that the model is not trained on these structures. The results are used in the final assessment of the model's transferability.

### 4.5.5. Ray tune hyper-parameter searching

To ensure the model's performance, a proper selection of hyper-parameters is required. Initially the hyper-parameters are selected and tuned during development using common practices and theoretical choices. A more structured approach is considered by implementing Ray Tune, an automated tool for distributed model training and hyper-parameter searching [31]. It initially samples hyper-parameters uniformly from pre-configured ranges, and uses the ASHA scheduler [28] to gradually further explore well performing parameter combinations. Using this framework a hyper-parameter selection is chosen for experimentation, given in Table 4.2.

| Parameter | Value |
|---|---|
| Learning rate | 0.0005 |
| Batch size | 8 |
| Res. $\alpha$ | 0.15 |
| LReLu $\theta$ | 0.1 |
| GCN **L** | 6 |
| Mult. $k$ | 3 |
| Mult. aggr. | '$sum$' |
| Hidden dim | 128 |

**Table 4.2:** Selected model hyper-parameters selected using Ray Tune hyper-parameter searching.

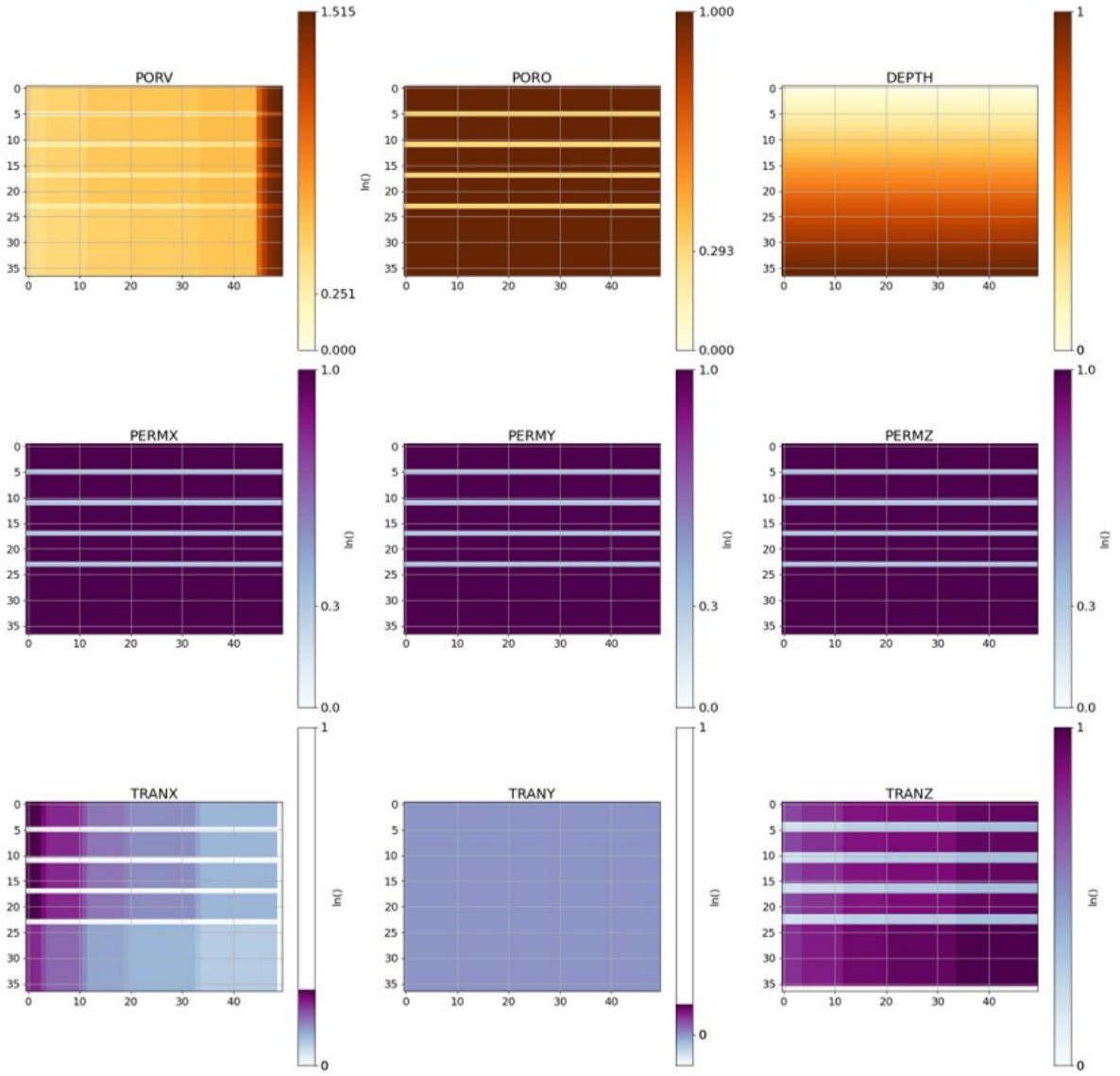
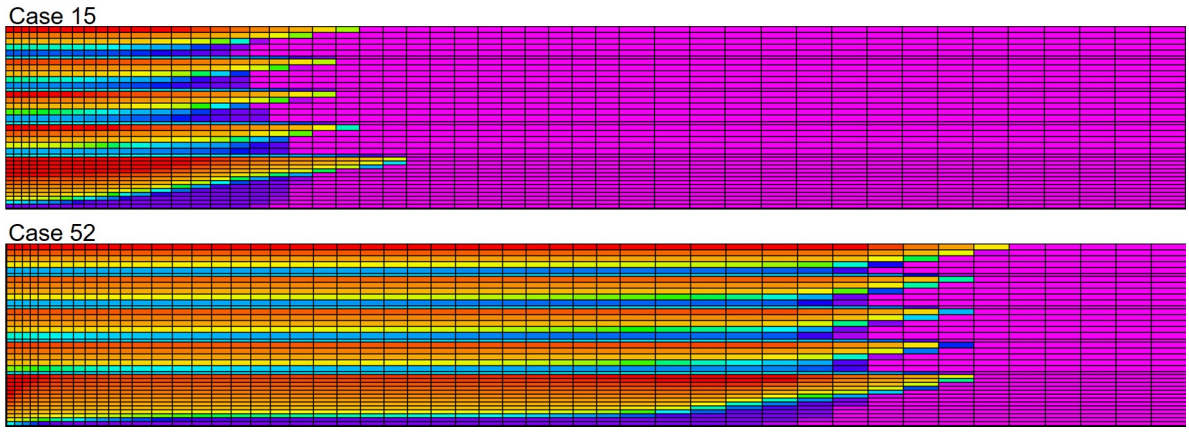


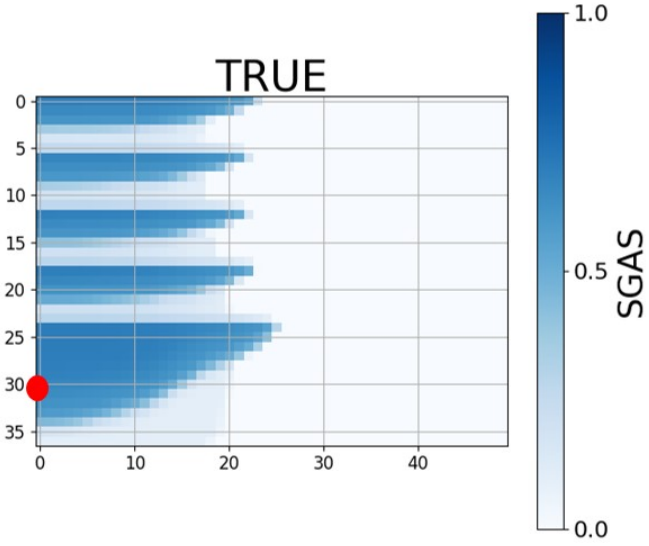

**Figure 4.4:** Static properties of a general P7 sample from the variable injection ensemble (constant across the ensemble). It shows the general structure with horizontal shale layers found in every P7 sample. PORV increases steadily in the $x$-direction. DEPTH increases in the increasing $z$-direction. PORO influences the input property PORV. PERMY is preconfigured nonzero, but is not encoded in any edge weight. Transmissibility depends on the cell dimension in its specific direction, making TRANY zero in the $xz$-dimensional grid. TRANX/TRANZ are influenced by PERMX/PERMZ, PORO and PORV. The variable shale ensemble is identical in the non-shale surroundings, but has varying PORO and randomly decaying PERMX/PERMZ in the shale layers.




**(a)** Original geometric shapes of the P7 reservoir, showing gas saturation for cases at $t = 155$

| Specification | Value |
|---|---|
| Size | 1850 |
| Shape | 37 x 50 |
| # Shale Layers | 4 |
| # Injector Wells | 1 |
| Injector well loc. | (31, 0) |

**(b)** P7 reservoir specifications




**(c)** Abstracted spatial-invariant reservoir view as used by the model at $t = 155$. The red dot indicates the injector well location. The shale layers are located horizontally in the areas with lower saturation.

**Figure 4.5:** Overview of the P7 reservoir grid: a) spatial and c) non-spatial view.

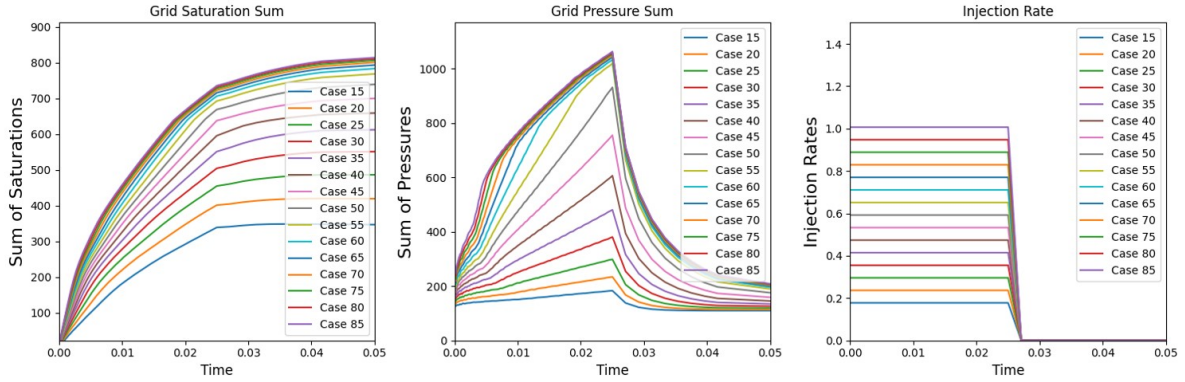



**Figure 4.6:** Accumulation of gas saturation (left) and pressure (middle) for a range of cases in the variable injection ensemble. Injection rate (right) shows that when CO2 injection stops (0.0275 ≈ 12 years), the dynamics of the system change. Pressure smoothes out gradually over the grid, decreasing across the grid overall back to an equilibrium. Even though no more CO2 is entering the system, gas saturation keeps increasing slightly. As pressure decreases the volume of the gas is likely to increase, increasing saturation across the grid cells overall. Note the y-axis does not represent the actual feature value, but the sum of the normalized features over the whole grid.

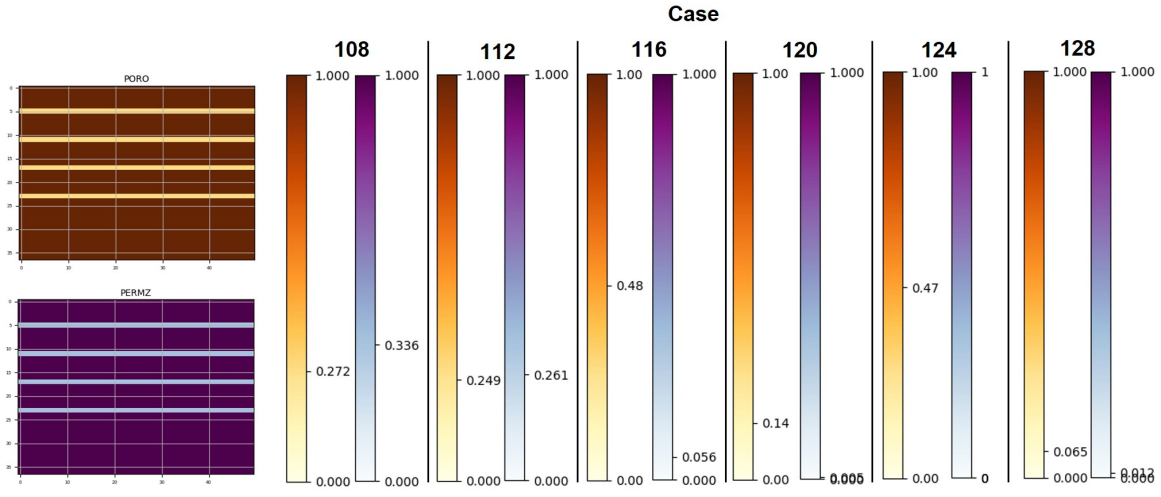



**Figure 4.7:** Different porosity (PORO) and z-directional permeability (PERMZ) values in the variable shale ensemble validation set.

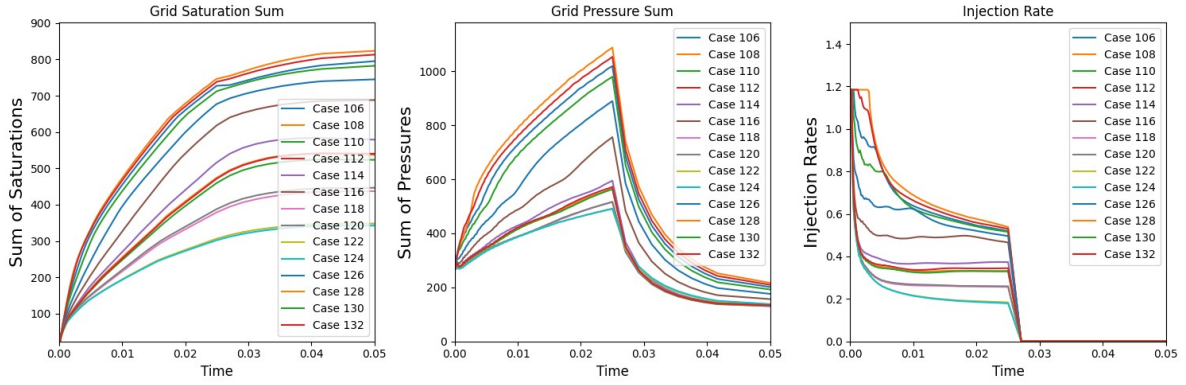



**Figure 4.8:** Accumulation of gas saturation (left) and pressure (middle) for a range of cases in the variable shale composition ensemble. Compared to the variable injection ensemble, the cases with decreased porosity and permeability show higher pressure. This is the result of more CO2 trapping under a shale layer, increasing pressure buildup. Injection rate in this ensemble is also variable, but is determined by a maximum pressure metric based on the bottomhole pressure (BHP, i.e. the pressure at the point of injection), simulated in IX.

# 5

# Results

In the previous chapter the experimental methods are introduced. Using these methods the research hypotheses on *expressivity*, *injectivity* and *transferability* (subsection 1.1.1 - subsection 1.1.3) are evaluated through experimental results presented in this chapter. The covered experiments evaluate (increasingly complex) objectives:

- Single timestep predictions
- Roll-out (multistep) predictions
- Ensemble training
- Hyper-parameter optimization
- Untrained data: variable static compositions

In the process of doing these experiments, the model's loss function, input features and graph edge weights have been adapted gradually to improve the model's performance. All the results shown in this chapter have been created using the optimized feature, hyper-parameter and loss selection. The motivation of these selections are covered in chapter 4:

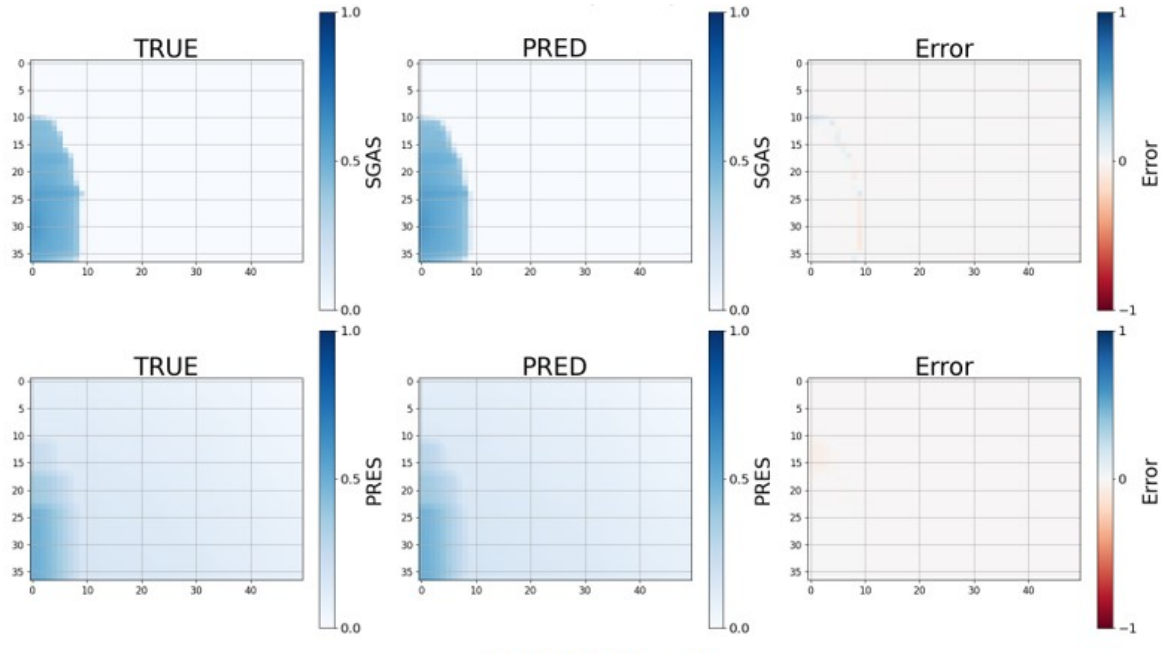
- Input features: section 4.3
- Edge weights: *permeability edge weights* subsection in section 4.3
- Loss function: subsection 4.3.3
- Hyper-parameters: subsection 4.5.5

The reader will often come across the term '*long-term*' predictions, which is defined in this work as autoregressive predictions covering more network iterations than the *multisteps* hyper-parameter $k$ (subsection 4.3.4). Another introduced term is '*roll-out*' prediction, generally indicating the prediction of a full timespan $T = [0, 167]$, or another specifically mentioned large interval like, e.g. the injection phase $T = [0, 143]$.

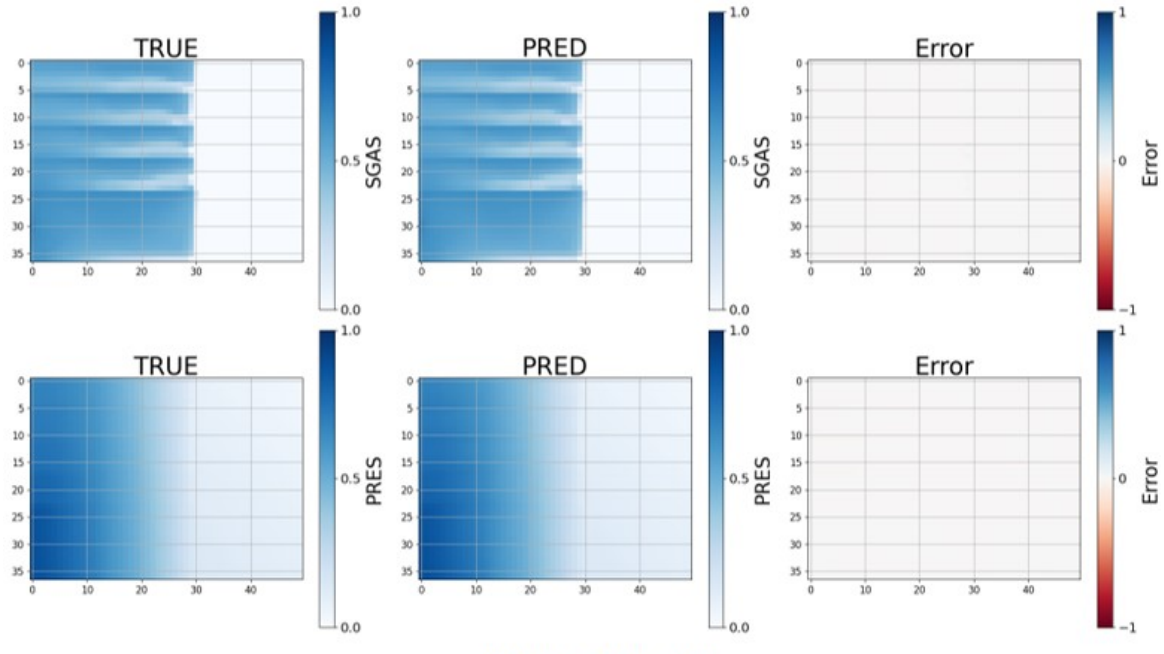
Visualizations like the one in Figure 5.1 are used several times in this chapter. To get a better intuition of this figure, take the following remarks in mind when observing it. The diagram generally contains 3 subplots of the reservoir grid per row, featuring an output feature for a given timestep:

- **Left**: ground truth output (INTERSECT reservoir simulation data)
- **Middle**: GNN (roll-out) prediction
- **Right**: error between ground truth and prediction (ground truth minus prediction)
- e.g. Figure 5.1a **Upper row**: CO2 saturation (SGAS) at $t = 10$
- e.g. Figure 5.1a **Lower row**: Pressure (PRES) at $t = 10$

Keep in mind both output features can only take on positive values and are normalized between 0 and 1 (saturation by default, for pressure see subsection 4.3.2 for the normalization procedures). Assuming the model remains numerically stable, the error should thus not exceed the range $[-1, 1]$, visualized using the colour gradient $[red, blue]$ in the error plot (i.e. $red$: over-prediction, $blue$: under-prediction).




**(a)** Case 45, $t = 10$




**(b)** Case 45, $t = 120$

**Figure 5.1:** Single timestep predictions on an ensemble trained model, case 45 (mid-range injection rate). a) Early injection phase timestep (8 months) b) Late injection phase timestep (10 years).

## 5.1. Animated experimental results available online

To complement the figures in this chapter, corresponding animations have been made available online at `https://lucasveeger.github.io/`. This webpage displays all shown roll-out experimental results in an animated format for the entire simulated time spans.

## 5.2. Experiment 1: Single predictions

To evaluate whether the GNN is capable of graph re(construction), the initial test is to construct the updated $Y_t$ state from an input state graph $X_t$ at any given *t* with a minimal error. A minimal error is desired for numerical stability in follow-up autoregressive experiments. Trained on the variable injection ensemble, Figure 5.1 shows that the model is capable of producing these target states with minimal visible error. Both the model trained with $k = 1$ and $k = 6$ produce equally well predicted states, indicating the multistep procedure does not negatively affect the model's single step predictive behaviour.

## 5.3. Experiment 2: Multistep autoregressive predictions

Given that the model can accurately construct every next target state in single step predictions, the next objective to confirm ***RQ1****: Expressivity* is to evaluate the autoregressive predictive capability. In Figure 5.2 and Figure 5.3 results are shown from a model trained on the variable injection ensemble.

Without going into the injection variations in this section, it can be seen the model shows good autoregressive behaviour. The roll-out states resemble visually realistic progressions for the gas saturation. Pressure predictions are numerically stable and decently low in error (Figure 5.7), but visually show the appearance of a slightly different field compared to the true pressure gradient that perfectly diverges from the injection well.

The true pressure gradient is largely unaffected by the shale layers in the injection ensemble, although some slight discontinuity is visible. Larger discontinuities can be seen in the variable shale ensemble (Figure 5.5). The elliptic nature of pressure, the network depth limitation (section 3.3) and the high-frequency effect of the permeability encoded edge weights (section 4.3) are likely to complicate the pressure prediction. Despite these inconsistencies, the model captures the global pressure gradient reasonably well. In chapter 6 model additions are discussed to possibly improve the pressure expressivity further.

## 5.4. Experiment 3: variable injection ensemble

Given the model's capability to express a time series realistically in a roll-out prediction, the model is tested for its ability to capture parametric variability of the CO2 injection rate. Results in Figure 5.2 and Figure 5.3 show the model captures the injection rate variability properly, supporting the hypothesis set out by ***RQ2****: Injectivity*. This injective expressivity is largely attributed to the introduction of the *accumulation speed* features (see subsection 4.3.1). Without them, models initially tended to converge to a more or less fixed propagation speed. In those models, a node likely struggled to spread its state rate of change to neighbouring nodes during message-passing. And given that the injection rate feature is only nonzero at the injector node, only the injector node's neighbourhood would aggregate relevant injection rate feature information. Message-passing of the accumulation features (subsection 4.3.1 enables the nodes to pass around their state's rate of change, providing all the grid nodes with a sense of how quickly CO2 is entering and saturating the system.

## 5.5. Experiment 4: variable shale composition ensemble

Besides good injective expressivity, a proper reservoir simulation model should be capable of handling geological variability. Trained on the variable shale composition ensemble, Figure 5.4 and Figure 5.5 show that the model can predict validation cases correctly. The model limits the CO2 flow through the shale layers for the cases with lower porosity/permeability, and builds up more pressure in the areas with high saturation under the shale layers. In some cases (e.g. Figure 5.7) the model tends to 'lag' at the start of the injection and builds up some underexpressive error. The model then corrects for this and produces a stable flow. The error generally increases gradually during the stable flow, as some parts of the saturation front become slightly over- or under-expressed.

The overal results in this experiment show the model responds correctly to parametric variability in the geological features. This conclusion is a first step in confirmation the hypothesis of ***RQ3****: Transferability*. The next experimental step (unseen structural compositions) is covered in section 5.7.

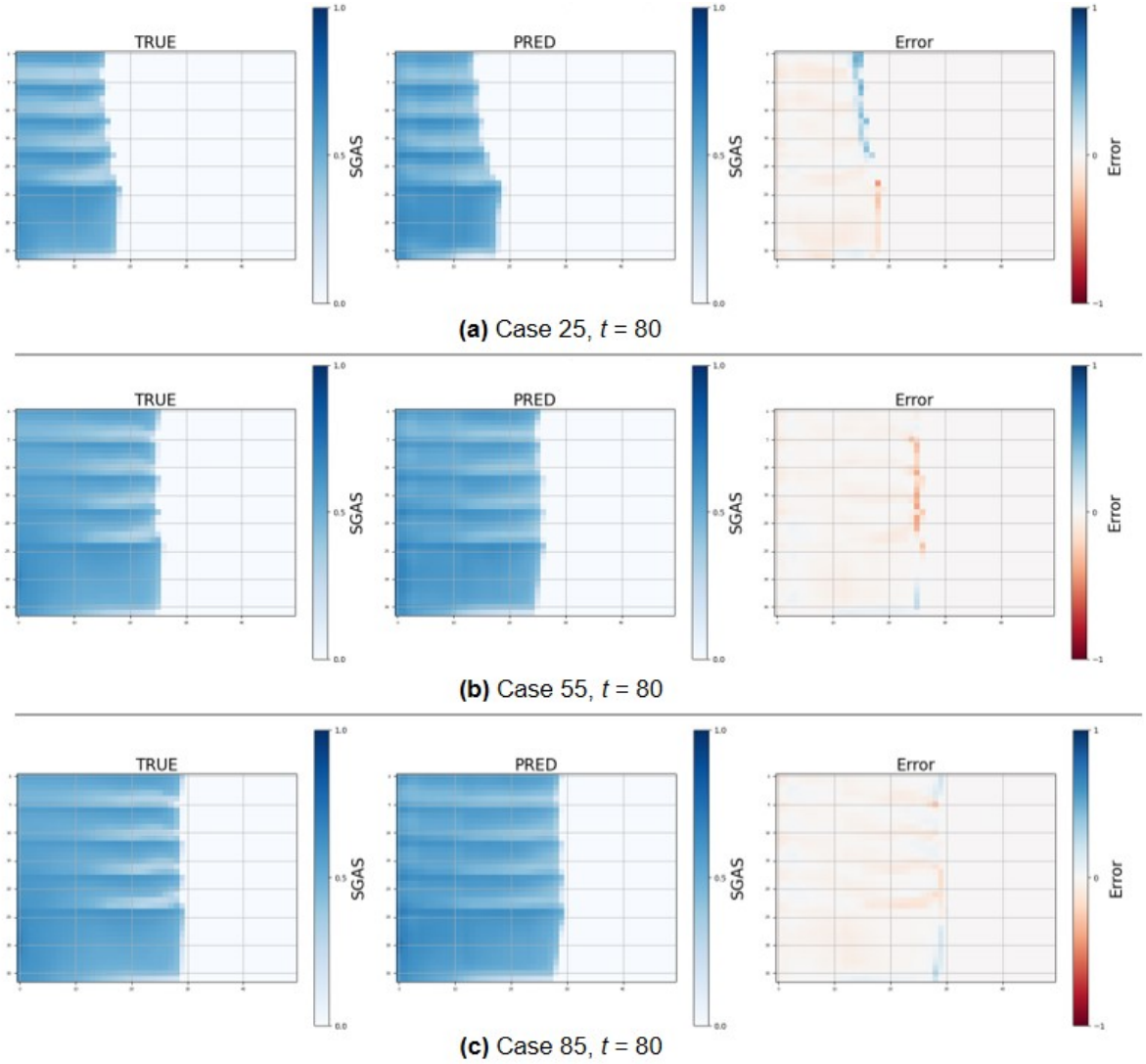



**Figure 5.2:** Gas saturation (SGAS) simulation, roll-out prediction and error at $t = 80$ for the variable injection ensemble. a) Low injection rate, b) medium injection rate, c) high injection rate

## 5.6. Training loss & roll-out evaluation

In the process of training and improving the model, it was noticed that identical hyper-parameter selections gives different performances per training. Model performance is measured using a validation loss and a roll-out evaluation metric. The validation loss is the roll-out version of the 'mean' multistep loss, calculated over the entire predicted timespan $T$. The evaluation metric is the validation loss plus the average squared reconstruction loss across the predicted roll-out states (Equation 4.3.4). The reconstruction term adds extra information and introduces slight variations with respect to the validation loss, which can be seen in Figure 5.6. Although the average validation loss / evaluation metric decreases as the training loss converges, they are very unstable. Small changes in the model's parameters can result in a large improvement or deterioration of the model's performance. The result is that only every so many epochs a new minimal validation loss or evaluation metric is reached. The model's parameters are saved on every new minimum in case the model does no longer improve (around 150-200 epochs in practice). Summarizing, training for the optimal model relies on marking the specific epochs that have the lowest roll-out evaluation metric loss. Training multiple identical models also showed different training runs have different success rates, possibly due to the model's parameters getting trapped in a local minimum.

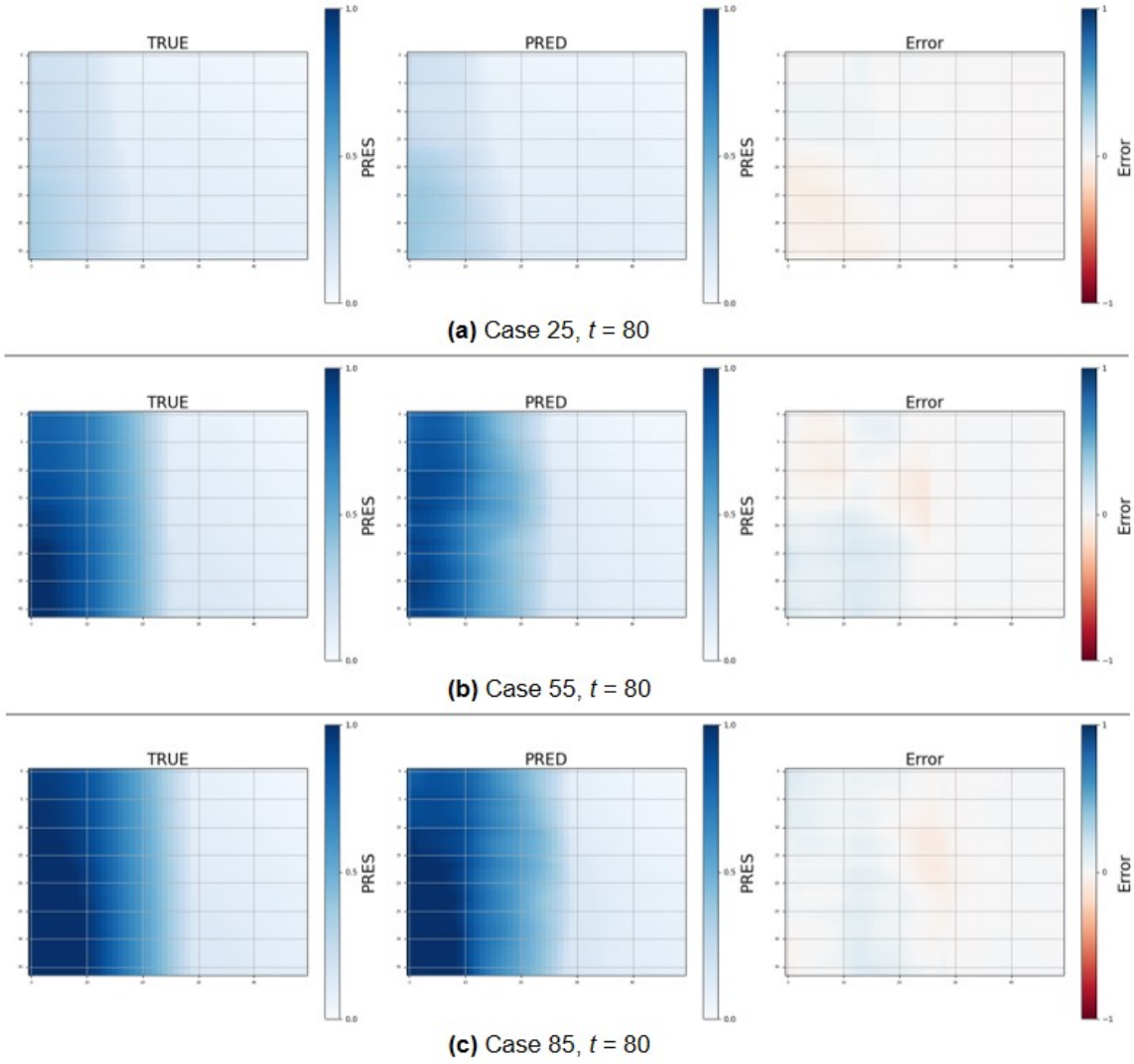



**Figure 5.3:** Pressure (PRES) simulation, roll-out prediction and error at $t = 80$ for the variable injection ensemble. a) Low injection rate, b) medium injection rate, c) high injection rate

### 5.6.1. Autoregressive value bouncing

The introduction of the accumulation speed components introduced an expressive anomaly. For a significant portion of the trained models one or more predicted features started to exhibit an increasingly strong oscillatory effect, like the model training depicted in Figure 5.8. This effect occurred for suboptimally trained models as well as optimally trained models (i.e. having the lowest training/validation losses). For suboptimal models, this effect generally causes an explosive gradient. However, for the optimal models, the model remains numerically stable within the predicted timespan and the resulting error can be significantly lower than optimal models with stable feature progressions. Comparing the error progressions in Figure 5.7 and Figure 5.8, the stable model error gradually diverges while the bouncing model actually converges.

Interestingly, any trained model exhibits small oscillatory errors between timesteps. Possibly the coupling in the model of the speed components with their antiderivative features is approximating some trigonomic function that approximates the continuous system. Despite the lower error of the value bouncing models, the stable models have been selected for experimentation. Given that the accumulation features are more stable, it is assumed they will generalize better to untrained samples.

### 5.6.2. Post-injection phase

A part of the simulation that has been neglected so far is the post-injection phase. As mentioned in subsection 4.4.3, the data availability is scarce, the timestep sizes increase by orders of magnitude and the system's dynamics are imperatively different compared to the injection phase (the injection term from Equation 2.2 goes to 0). In Figure 5.9, the model behaviour representative of every post-injection phase prediction is shown. The model learns to smooth out the pressure, but much slower than simulated (i.e. 1 or 2 timesteps in the IX simulation). While the model decouples the saturation properly from the pressure, the true saturation development becomes mainly dependent on buoyant forces. The model does not capture this, and instead it continues a steady flow similar to the injection phase. It raises the question whether the model has enough intuition of buoyancy effects relevant in CCS models. In section 6.2 this is further discussed.

### 5.6.3. Hyper-parameter tune

Using Ray tune, model training and evaluation was automated to study the effect of different hyper-parameters. Over 500 models were trained with different hyper-parameter sets. Where most training runs were stopped early due to poor performance, 80 well performing models were trained to the 200 epoch limit (where validation loss usually diverges). Interestingly, similarly good results were achieved on various hyper-parameter sets. More interestingly, a model retrained with a well performing hyper-parameter set would not necessarily perform as well again within the same epoch range. As mentioned in section 5.6, strong local minima might trap the model's learning progression during the gradient descent optimization.

## 5.7. Experiment 5: Transferability to unseen compositions

Using the variable shale composition (experiment 4) it is shown GNNs are expressive enough to handle parametric differences of the geological structure. The last experiment tests transferability of the GNN model, assessing its learned behaviour on unseen geological structures. The model from experiment 4 is used to predict the grids shown in Figure 5.10. The results are shown in Figure 5.11 & Figure 5.12.

For all structures, the saturation is well captured across all reservoirs A, B and C. Errors are generally higher than in the common shale layer cases, but this can be expected considering the added complexity. An effect the GNN model does not capture is clearly visible in the IX simulations. At the location of the original shale layers, the cells are less saturated. The $x$-directional transmissibility in these cells is lower due to their cell volume (which can be observed in Figure 4.5a), affecting saturation development in the IX simulation. However, the model has never trained on such transmissibility values in combination with the modified (i.e. increased) porosity and permeability values, making misinterpretation likely. It can be expected that a model can predict this effect without any issue if the parametric combination is present in the training data.

The accuracy of the predicted pressure fields are best discussed separately. In reservoir A, the pressure gradient is high in local errors. The CO2 flow is redirected multiple times, which likely complicates a smooth pressure field. The model also decouples pressure from the CO2 flow in the middle sections somewhat (rows 12-24), under-expressing pressure towards the right. Still, the overall presence and smoothness of the pressure field is reasonably captured.

The pressure fields in reservoirs B and C are reasonably smooth, and the reach of both pressure fields is accurate. However, the structural differences between reservoir B and C reveal a limitation of the model. In C, the flow is forced down upon hitting the second diagonal layer, increasing the pressure significantly compared to B. Where the model captures the pressure field of simulated reservoir B very accurately, it does not capture the increased pressure from the simulation of reservoir C. While the diagonal shale layers in C are similar to the horizontal shale layers in the training data, the orientation of the diagonal layer and the large (non-local) span of nodes in an open space possibly complicates the right expression of pressure.

Summarizing, the modified reservoirs demonstrate the model can utilize its local knowledge to predict saturation and pressure in *unseen* structural settings. With certain limitations present, the main hy-

pothesis on ***RQ3****: Transferability* can be demonstrated and confirmed in this experimental setup. In chapter 6 future steps will be discussed to test the hypothesis on larger domains.

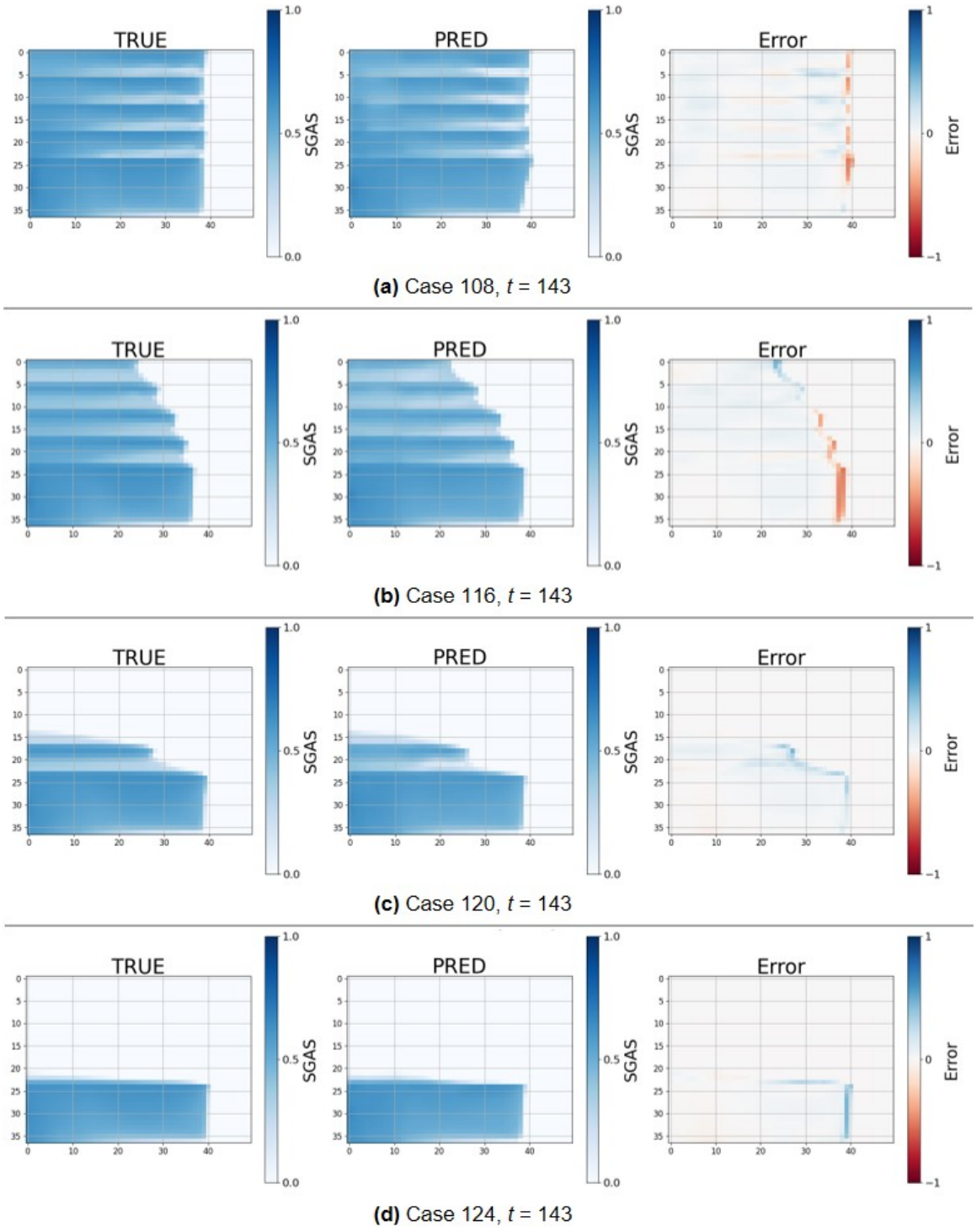



**Figure 5.4:** Gas saturation (SGAS) simulation, roll-out predictions and errors of 4 cases from the validation set of the variable shale composition ensemble, in order of decreasing permeability. a) Permeability shale ≈14mD, b) Permeability shale ≈0.7mD, c) Permeability shale ≈0.04mD, d Permeability shale ≈0.005mD)

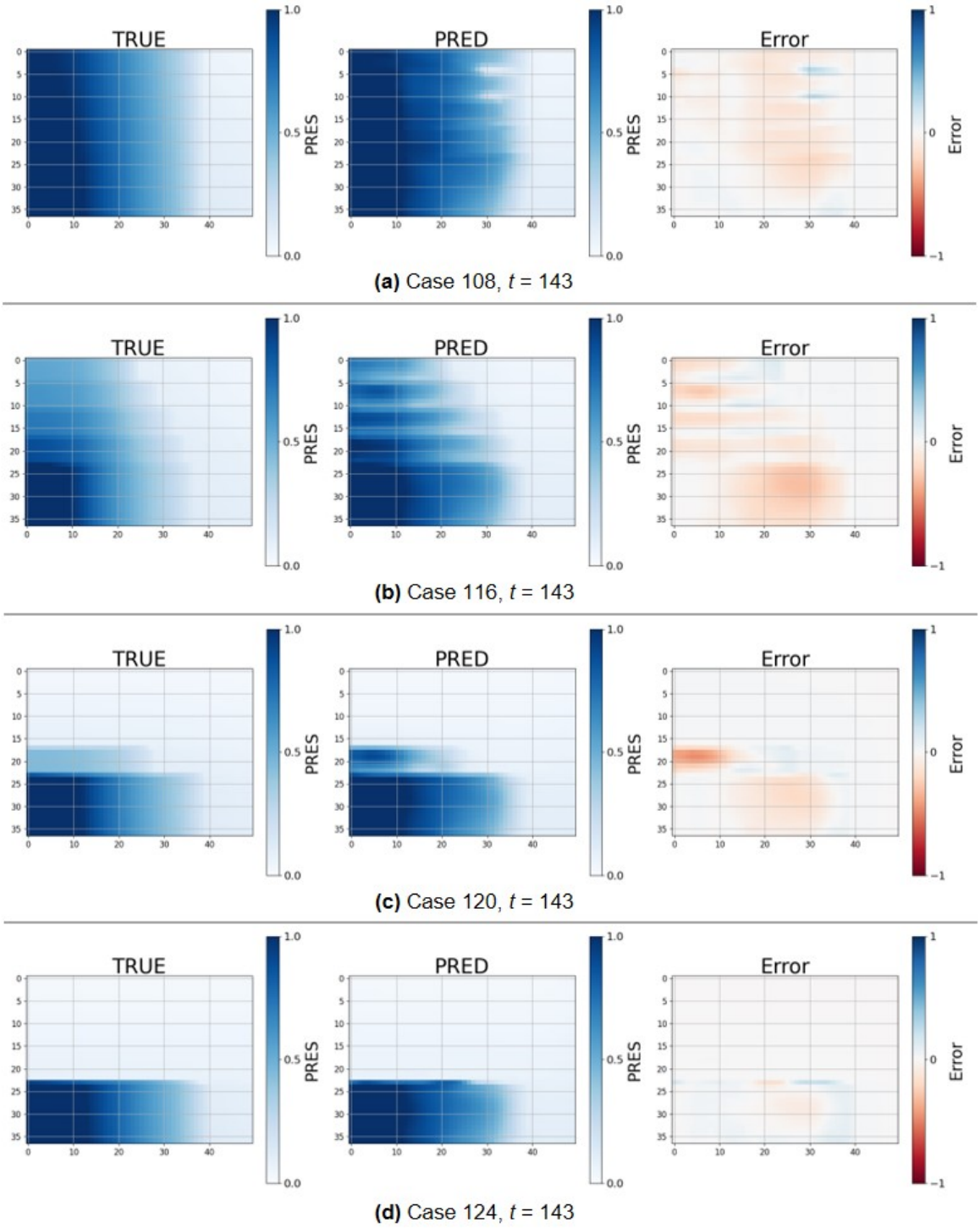



**Figure 5.5:** Pressure (PRES) simulation, roll-out predictions and errors of 4 cases from the validation set of the variable shale composition ensemble, in order of decreasing permeability. a) Permeability shale ≈14mD, b) Permeability shale ≈0.7mD, c) Permeability shale ≈0.04mD, d Permeability shale ≈0.005mD)

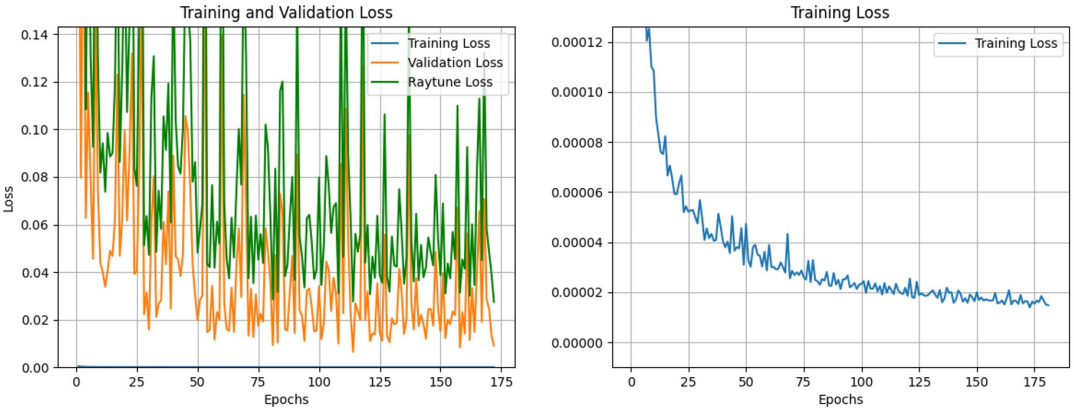



**Figure 5.6:** Training, validation and Raytune loss during model training on the variable shale ensemble. Raytune loss corresponds to the evaluation metric, and is used during the hyper-parameter searches as the optimizable metric. In the left plot, the training loss is a low value flat line, properly depicted on the right. The validation loss and the evaluation metric are calculated over the entire predicted timespan and generally a lot higher than the multistep loss, which is minimized during training and only covers loss of the multistep time range. The validation loss / evaluation metric show correlation with the training loss globally, but contain great variations on intermediate epochs. The final trained model is picked from the epoch with the minimal evaluation metric across the entire training.

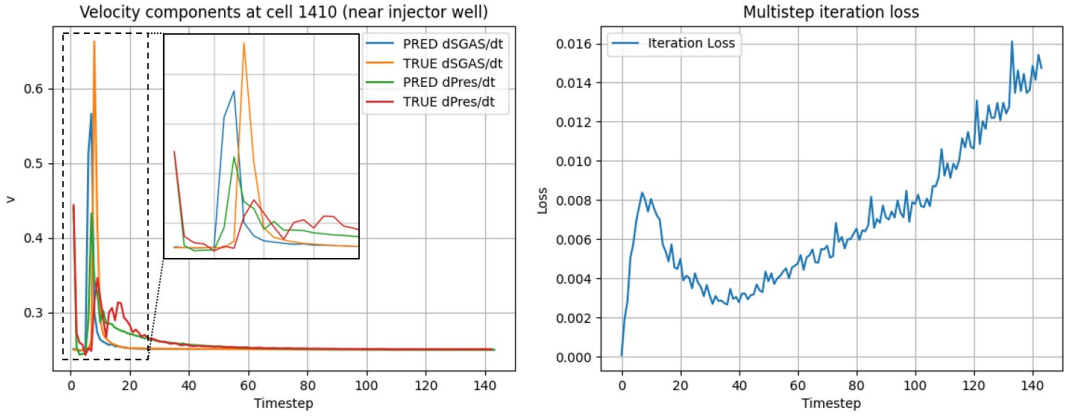



**Figure 5.7:** Dynamic feature progression during roll-out predictions at a particular grid cell near the injection well. Left: velocity components at cell 1410, (close to the injection well). Right: full grid average iteration loss, i.e. the unaveraged roll-out loss at every iterated timestep.

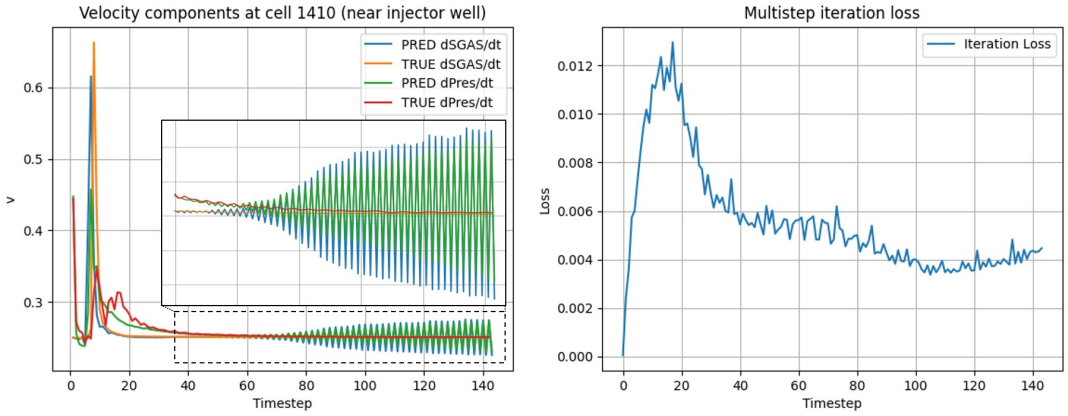



**Figure 5.8:** Dynamic feature progression at particular grid cell during roll-out predictions of a case expressing value bouncing of a dynamic feature, in this case the pressure accumulation speed. Left: velocity components at cell 1410 (close to the injection well). Right: full grid average iteration loss, i.e. the unaveraged roll-out loss at every iterated timestep.

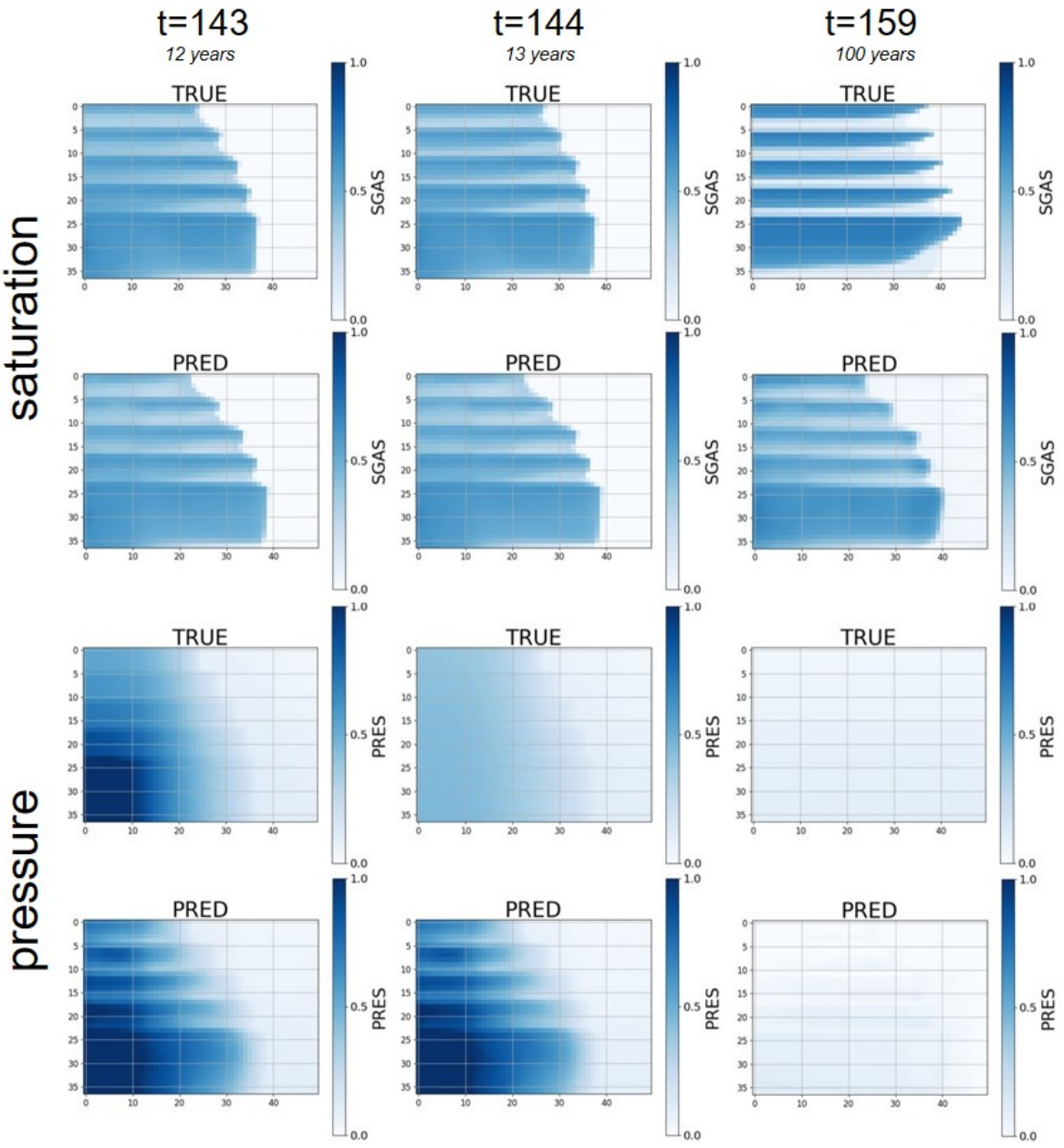



**Figure 5.9:** Post-injection phase typical development at 3 timepoints: 1) start of post-injection phase (left column, $t = 143$), 2) +1 year (middle column, $t = 144$) and 3) +88 years (right column, $t = 159$). Simulation (TRUE, uneven rows) and roll-out prediction (PRED, even rows) shown for saturation (SGAS) and pressure (PRES). The GNN model captures the pressure smooth-out, but only eventually. For the saturation, it fails in modelling enough buoyant force that concentrates the CO2 under the shale layers and instead seems to continue the steady flow from the injection phase.

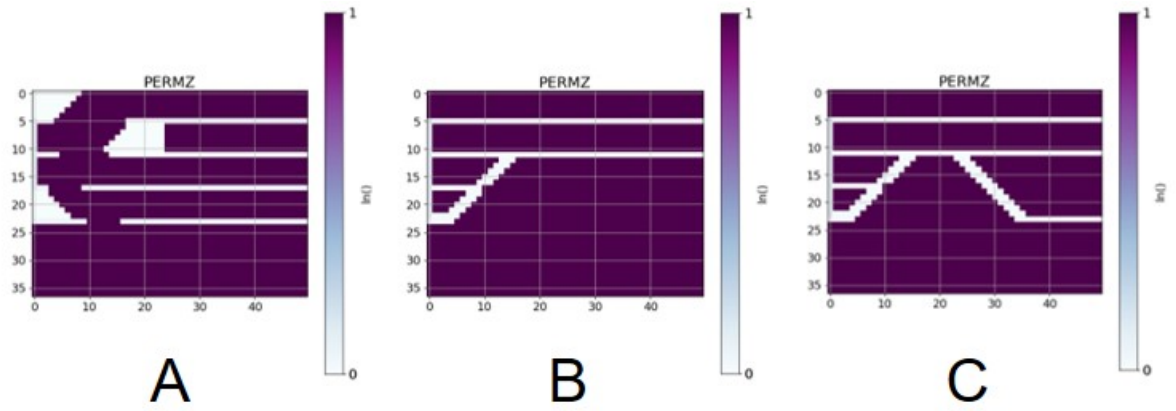



**Figure 5.10:** Permeability in z-direction (PERMZ) for structurally modified cases of the P7 dataset. Reservoir A includes holes in the shale layers allowing easy propagation of CO2, combined with zero permeability blocks resembling impermeable rock formations. Reservoir B contains a diagonal shale layer to evaluate the diagonal flow. Reservoir C extends B with a mirrored shale layer to evaluate the effect of the CO2 flow downward.

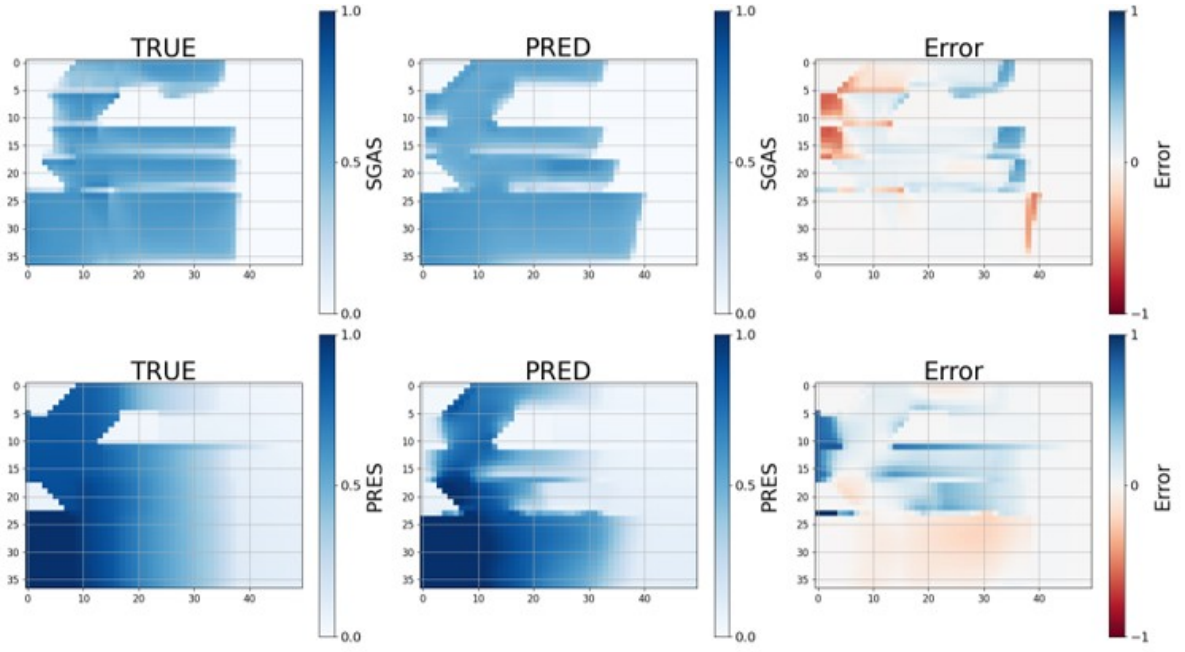
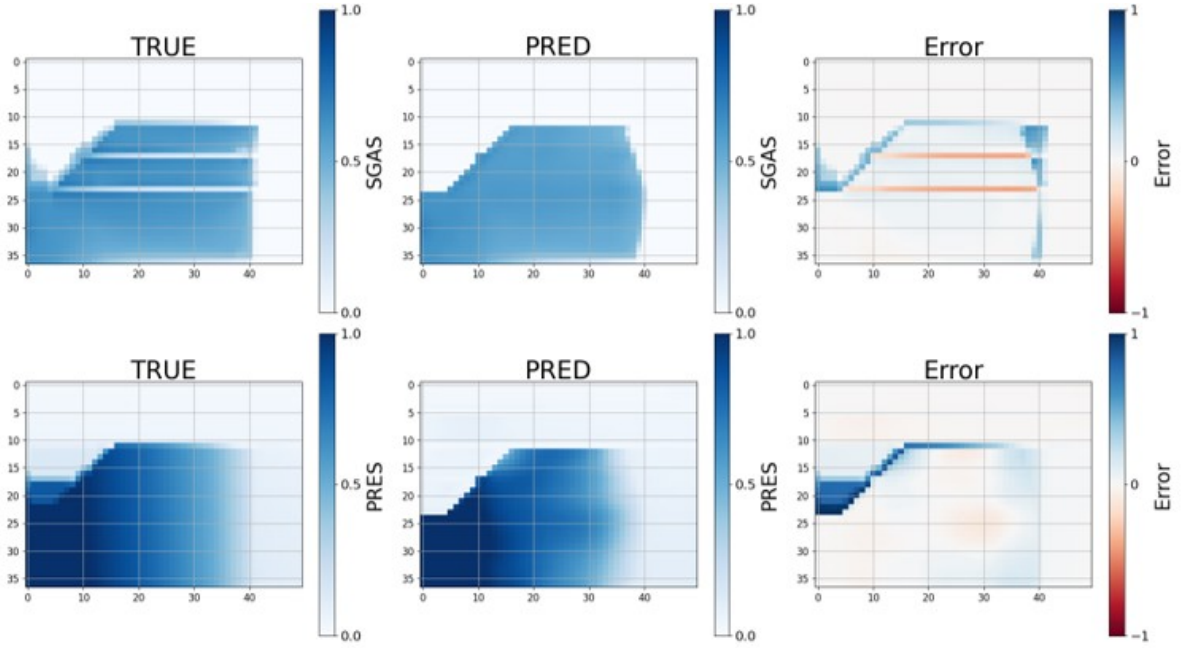
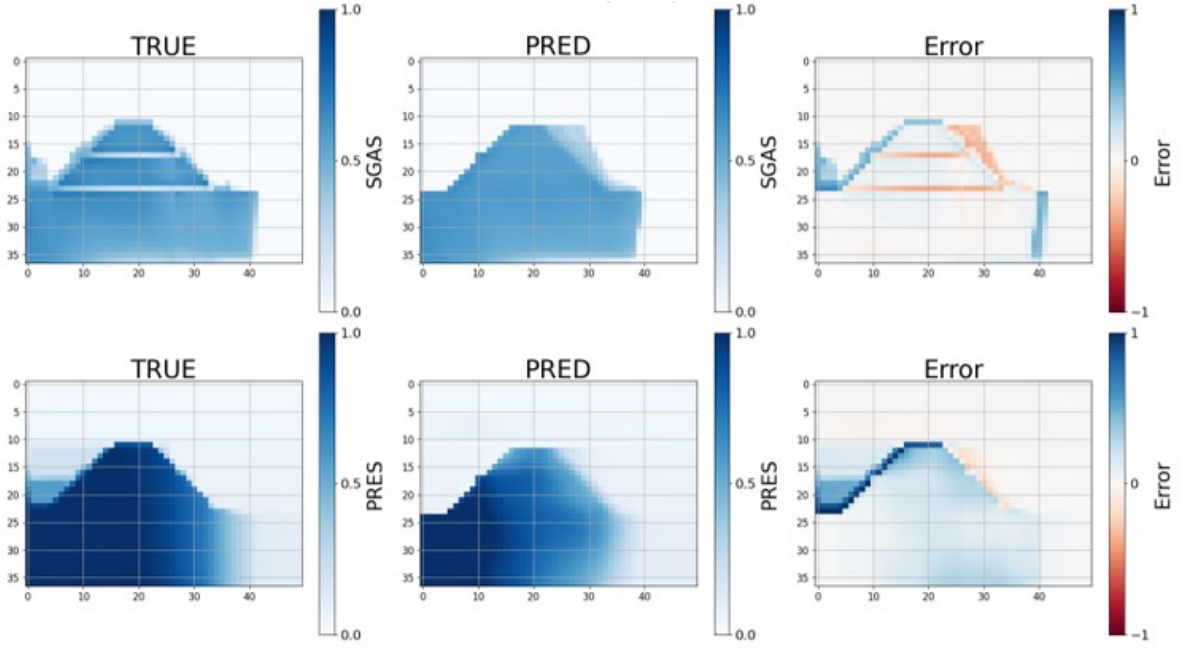



**Figure 5.11:** Simulation, roll-out predictions and errors of untrained reservoir case A at final injection phase state at $t = 143$ (reservoir structure provided in Figure 5.10).




**(a)** Reservoir B, $t = 143$ (reservoir structure provided in Figure 5.10).




**(b)** Reservoir C, $t = 143$ (reservoir structure provided in Figure 5.10).

**Figure 5.12:** Simulation, roll-out predictions and errors at final injection phase state at $t = 143$. a) Untrained reservoir case B, b) untrained reservoir case C. Reservoir structures are shown in Figure 5.10.

# 6 Discussion

In the last chapter it has been confirmed that a GNN can be expressive using an autoregressive method. The autoencoding model can reliably construct a series of output feature states, it is able to scale feature progression according to the injection rate and it can handle geological variability. This confirms all three research questions. However, the considered experimental domain does not yet resemble the domain of real reservoir models and it is inconsiderate of certain practical aspects. While reviewing experimental results, this chapter discusses such aspects, provides ideas for model improvements and makes suggestions for follow-up research.

## 6.1. Representation of the physical situation

Finding the right loss to capture the physical system correctly has been one of the challenging aspects in this study. Use of the MSE error of the output features did not express enough variability. Different loss terms were explored before implementing the accumulation speed loss. Using the *graph laplacian* a spatial gradient loss was constructed, as well as a more complex 'continuity' loss that connected output feature graphs sequentially to add the temporal dimension in the graph laplacian. However, these losses did not significantly improve performance. As their computation required matrix multiplications with $\mathcal{O}(|\mathcal{V}|^2)$, these losses were abandoned and replaced with the less complex accumulation speed loss with $\mathcal{O}(|\mathcal{V}|)$. This loss is only temporal, but by including the accumulation speed as input features their optimized feature state is also spreading spatially.

A linear constraint for saturation, calculated using the injection rate, has also been considered as extra loss term. The feature progression of saturation in Figure 4.6 can be linearly approximated during the injection phase. However, such a linear constraint is simplistic and does not account for other physical effects. To properly put constraints on saturation (or pressure) development, the model could be made physics-informed by factoring in (components of) the conservation equations (Equation 2.2, Equation 2.2). While this may be more costly in terms of computing the loss, studies in the field of physics-informed machine learning and CCS indicate models can be made more expressive this way (see Related Work in section 1.2 for some examples).

### 6.1.1. Pressure-saturation relation

In the P7 ensembles, all trained cases show a clear relation between the development of pressure and saturation, where pressure seems to act as a driving force of the saturation propagation. An added loss term initially increased performance by calculating for every node the Pythagorean distance between these features' true and predicted values:

$$\sqrt{(SGAS_{true} - SGAS_{pred})^2 + (PRES_{true} - PRES_{pred})^2}.$$

However, this relation is only partly relevant for the propagation of CO2. Under typical injection conditions, a portion of the CO2 will be in the gas or supercritical phase[1]. In these phases buoyant forces

[1] In the supercritical phase, CO2 is in a highly compressed state where it combines the properties of gases and liquids

act on the CO2, forcing it upwards. This makes the saturation-pressure relation less relevant in the $z$-dimension. In the final model, the saturation-pressure loss is abandoned entirely as the performance of the accumulation speed loss dismissed its effect all together.

## 6.2. Training the post-injection phase

The current model can not express the post-injection phase properly. This is due to the limited availability of data samples for variable timesteps, the size of the timesteps and the changed dynamics in the post-injection phase. A few simple solutions could improve the model.

#### Timestep interpolation & permutation

First, training on smaller timestep sizes would decrease the amount of change that has to be expressed. This might even be essential for the longest timesteps (500 years) to capture the largest state changes. Considering it is costly to use the reservoir simulator to generate more 'shorter' timesteps for such long timespans (e.g. 1 year steps for 1000 years), the given post-injection training samples could be interpolated as a data pre-processing step to generate more data samples in the post-injection phase. Alternatively, the given timesteps could be permuted to generate many different timestep sizes and corresponding training samples. This approach would also help the model to generalize, and could similarly be applied to the injection-phase timesteps. Similar to the numerical reservoir simulator, a trade-off could then be optimized between the maximal timestep size (i.e. amount of network iterations required) and the expressive performance.

#### Added post-injection cases

Secondly, during training extra cases could be added where the injection phase timesteps are excluded. This way, the amount of injection and non-injection timesteps fed to the model can be balanced, but still unique training samples are used.

#### Separate post-injection model

Lastly, an alternative is to train a separate model to capture the post-injection phase, and switch models when injection stops. Given the pressure field quickly smooths out (as seen in Figure 5.9), the physical system is changed to be governed by just capillary, gravitational and buoyant forces. A separate model might distinguish this situation better. Testing how effectively the buoyancy effects can be captured in the GNN model separately from the already learned physics is important in this consideration.

## 6.3. Application to larger domains

It is hypothesized in this research that the proposed GNN model is transferable to unseen grids. This hypothesis is supported by the experimental results, but further research is required to confirm its validity on arbitrary reservoir grids. The term 'arbitrary' used here might be too vague, as there are distinct target types of reservoirs for CCS: saline aquifers and depleted oil/gas reservoirs. In the IX reservoir simulator, different algorithmic methods are applied for these reservoir types. Different reservoir types might require different models in practice. A few more aspects of constructing an effective ML model using the implemented GNN methods are mentioned in the sections below.

### 6.3.1. Variety of Training data

The used training data is synthetic and unlikely to be representative of arbitrary geological formations. It is trained on 2D data, and when given a 3D input the model may become numerically unstable because of the increase in message-passed information. A properly generalized GNN model should be trained on an abundantly large set of unstructured 3D models with parametric and structural variety. Characteristics of geological structures in the reservoir type (e.g. caprock, shale layers, wellbores, fractures, faults) should be well represented in these. To optimize training time, the minimally required graph size required to train and learn the physical behaviour around all such structures should be determined.

### 6.3.2. Reservoir scale

A GNN model would become widely applicable if it has the ability to handle different reservoir scales. Reservoir grids come in different resolutions, meaning grids can be very coarse but also very fine

depending on how precisely it has been constructed. Since the proposed method already operates spatially-invariant, it should be able to process any scale (within reasonable ranges), provided that the model has been trained properly on the scale it is handling. A training suggestion is to train the model on different resolutions of the same grid to help it learn the scaling of the features involved. The IX reservoir simulator software has built-in coarsening operations for reservoir grids.

### 6.3.3. Reservoir complexity and well controls

Industry reservoirs cases are certain to contain more complexity than the P7 model used for training. An ignored detail from reservoir models is that they often include bended surfaces (i.e. non-flat surface horizons). Also, the entire reservoir model can be rotated under a 'dipping angle'. These aspects make the orientation of grid cells relevant, as it will influence the buoyancy effects of CO2, among other potential effects. Ideally, every node or edge (i.e. cell face) in the model would include its orientation. This addition will be highly relevant considering unstructured grids, where cells can be connected in any direction.

#### Well controls

In the current training method, the well controls (i.e. injection rate) are provided upfront. As seen in Figure 4.8, the injection rates are variable per case as a result of a maximum allowed pressure around the injector. In reservoir simulators, the injection rate of the next timestep is determined using the pressure data of the simulated system. A proper ML reservoir simulator used on untrained reservoirs should similarly adapt any dynamically changing well controls.

## 6.4. A local method on a global feature: pressure

The results show saturations are generally better captured than the pressure field. This is not surprising given the mathematical foundations provided in chapter 2, subsection 2.2.3. The saturations are described by hyperbolic PDEs with local effects, whereas the pressure is governed by an elliptic PDE with long-range effects. The performance on the experimented data is remarkable in the light of this theoretical deficit, but it is unsure if it will hold for much larger and more complex grids. To improve the pressure predictive quality in the implemented method, there are a few suggestions.

### 6.4.1. Multigrid approaches

In subsection 2.2.3 we discussed the use of AMG in IX. While the implemented GNN model shows some resemblance with multigrid in terms of iterative smoothing and feature aggregation, the multigrid concept of solving on coarsened grids can also be integrated. A weather forecasting study (see section 1.2) implementing this approach has shown to be very effective [24]. IX provides grid coarsening out-of -the-box, meaning derivation of the restriction/prolongation matrices can be omitted. Training the same model on different coarsening levels can improve the models generalization as well as improve the scale robustness.

### 6.4.2. Feature specific aggregation

The current model implements edge weights as a function of the permeability properties (section 4.3). While this is not necessarliy incorrect with respect to pressure (lower permeability gives less pressure propagation), it does create local error effects in the pressure field (high-frequency errors as discussed in subsection 2.2.3). An alternative would be a custom aggregation operator. Where the current method implements sum aggregation, summing all message-passed features, a custom aggregation function could impose a different function on every aggregated feature. This way a locally governed feature can be handled differently than a global feature or static feature. However, in latent space (where the aggregation takes place) the features are already in a mixed state. The aggregation operator would require learnable parameters that optimize the aggregation, effectively learning a custom aggregation function.

## 6.5. Conclusion

This work aims to test whether Graph Neural Network models can be applied to reservoir simulation. The motivation for testing GNNs is because of their potential to handle unstructured grids, a prevalent

data format in reservoir simulation. Despite this, the use of GNNs for grid feature prediction is novel in the ML reservoir simulation field. An autoregressive GNN autoencoder is introduced, implementing the common Graph Convolutional Network layer, a local message-passing method. The model shows good reconstructive capability, constructing a series of reservoir states in an autoregressive manner accurately over a complete time span relevant for CCS simulations. It also exhibits good injective expressibility, differentiating properly to the amount of perturbation introduced to the physical system. Lastly, the locally operating model shows it can adapt to geological variation, ranging from simple parametric variations to structurally different reservoirs. The next suggested research goal is training the model on diverse small unstructured reservoir models and testing these on larger industry reservoir models. Given the model operates spatially invariant, the model can be trained on different scales to increase generalization. Model enhancements derived from concepts used in numerical reservoir simulation can be used to further improve the GNN's expressiveness of elliptic features like pressure.

# References


[1] International Energy Agency. *Energy Technology Perspectives 2020*. 2020, p. 400. DOI: https://doi.org/https://doi.org/10.1787/d07136f0-en. URL: https://www.oecd-ilibrary.org/content/publication/d07136f0-en.

[2] A. Atadeger et al. "Deep Learning-Based Proxy Models to Simulate Subsurface Flow of Three-Dimensional Reservoir Systems". In: 2022.1 (2022), pp. 1–32. ISSN: 2214-4609. DOI: https://doi.org/10.3997/2214-4609.202244049. URL: https://www.earthdoc.org/content/papers/10.3997/2214-4609.202244049.

[3] *Efficient General Formulation Approach For Modeling Complex Physics*. Vol. All Days. SPE Reservoir Simulation Conference. Feb. 2009, SPE-119165–MS. DOI: 10.2118/119165-MS. eprint: https://onepetro.org/spersc/proceedings-pdf/09RSS/All-09RSS/SPE-119165-MS/2721565/spe-119165-ms.pdf. URL: https://doi.org/10.2118/119165-MS.

[4] NASA Global Climate Change. *Carbon Dioxide*. 2023. URL: https://climate.nasa.gov/vital-signs/carbon-dioxide/ (visited on 09/26/2023).

[5] Ming Chen et al. *Simple and Deep Graph Convolutional Networks*. 2020. arXiv: 2007.02133 [cs.LG].

[6] Intergovernmental Panel on Climate Change (IPCC). *Global Warming of 1.5°C: IPCC Special Report on Impacts of Global Warming of 1.5°C above Pre-industrial Levels in Context of Strengthening Response to Climate Change, Sustainable Development, and Efforts to Eradicate Poverty*. 2022. DOI: 10.1017/9781009157940.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. arXiv: 1606.09375 [cs.LG].

[8] David Duvenaud et al. *Convolutional Networks on Graphs for Learning Molecular Fingerprints*. 2015. arXiv: 1509.09292 [cs.LG].

[9] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG].

[10] *Physics Informed Deep Learning for Flow and Transport in Porous Media*. Vol. Day 1 Tue, October 26, 2021. SPE Reservoir Simulation Conference. Oct. 2021, D011S006R002. DOI: 10.2118/203934-MS. eprint: https://onepetro.org/spersc/proceedings-pdf/21RSC/1-21RSC/D011S006R002/2508046/spe-203934-ms.pdf. URL: https://doi.org/10.2118/203934-MS.

[11] M. Gori, G. Monfardini, and F. Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI].

[13] Kai Han et al. *Vision GNN: An Image is Worth Graph of Nodes*. 2022. arXiv: 2206.00272 [cs.CV].

[14] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[15] Hu Huang, Bin Gong, and Wenyue Sun. *A Deep-Learning-Based Graph Neural Network-Long-Short-Term Memory Model for Reservoir Simulation and Optimization With Varying Well Controls*. July 2023. DOI: 10.2118/215842-PA. eprint: https://onepetro.org/SJ/article-pdf/doi/10.2118/215842-PA/3151001/spe-215842-pa.pdf. URL: https://doi.org/10.2118/215842-PA.

[16] Global CCS Institute. *GLOBAL STATUS OF CCS 2020*. 2022. URL: https://www.globalccsinstitute.com/previous-reports/.

[17] *Physics-Aware Deep-Learning-Based Proxy Reservoir Simulation Model Equipped with State and Well Output Prediction*. Vol. Day 1 Tue, October 26, 2021. SPE Reservoir Simulation Conference. Oct. 2021, D011S008R002. DOI: 10.2118/203994-MS. eprint: https://onepetro.org/spersc/proceedings-pdf/21RSC/1-21RSC/D011S008R002/2686824/spe-203994-ms.pdf. URL: https://doi.org/10.2118/203994-MS.

[18] Weiwei Jiang and Jiayun Luo. "Graph neural network for traffic forecasting: A survey". In: *Expert Systems with Applications* 207 (Nov. 2022), p. 117921. DOI: 10.1016/j.eswa.2022.117921. URL: https://doi.org/10.1016%2Fj.eswa.2022.117921.

[19] Zhongyi Jiang et al. *Fourier-MIONet: Fourier-enhanced multiple-input neural operators for multiphase modeling of geological carbon sequestration*. 2023. arXiv: 2303.04778 [cs.LG].

[20] Zhaoyang Larry Jin, Yimin Liu, and Louis J. Durlofsky. "Deep-learning-based surrogate model for reservoir simulation with time-varying well controls". In: *Journal of Petroleum Science and Engineering* 192 (2020), p. 107273. ISSN: 0920-4105. DOI: https://doi.org/10.1016/j.petrol.2020.107273. URL: https://www.sciencedirect.com/science/article/pii/S0920410520303533.

[21] Ryan Keisler. *Forecasting Global Weather with Graph Neural Networks*. 2022. arXiv: 2202.07575 [physics.ao-ph].

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[23] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[24] Remi Lam et al. "Learning skillful medium-range global weather forecasting". In: *Science* 0.0 (), eadi2336. DOI: 10.1126/science.adi2336. eprint: https://www.science.org/doi/pdf/10.1126/science.adi2336. URL: https://www.science.org/doi/abs/10.1126/science.adi2336.

[25] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[26] Yann LeCun, Y. Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[27] M.C. Leverett. "Capillary Behavior in Porous Solids". In: *Transactions of the AIME* 142.01 (Dec. 1941), pp. 152–169. ISSN: 0081-1696. DOI: 10.2118/941152-G. eprint: https://onepetro.org/TRANS/article-pdf/142/01/152/2178004/spe-941152-g.pdf. URL: https://doi.org/10.2118/941152-G.

[28] Liam Li et al. *A System for Massively Parallel Hyperparameter Tuning*. 2020. arXiv: 1810.05934 [cs.LG].

[29] Xiao Li et al. "A survey of graph neural network based recommendation in social networks". In: *Neurocomputing* 549 (2023), p. 126441. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2023.126441. URL: https://www.sciencedirect.com/science/article/pii/S0925231223005647.

[30] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG].

[31] Richard Liaw et al. *Tune: A Research Platform for Distributed Model Selection and Training*. 2018. arXiv: 1807.05118 [cs.LG].

[32] Patricia Loria and Matthew B.H. Bright. "Lessons captured from 50 years of CCS projects". In: *The Electricity Journal* 34.7 (2021). Special Issue: Carbon Capture and Storage Today: Applications, Needs, Perceptions and Barriers, p. 106998. ISSN: 1040-6190. DOI: https://doi.org/10.1016/j.tej.2021.106998. URL: https://www.sciencedirect.com/science/article/pii/S1040619021000890.

[33] Dominic Masters and Carlo Luschi. *Revisiting Small Batch Training for Deep Neural Networks*. 2018. arXiv: 1804.07612 [cs.LG].

[34] Saro Meguerdijian et al. “Physics-informed machine learning for fault-leakage reduced-order modeling”. In: *International Journal of Greenhouse Gas Control* 125 (2023), p. 103873. ISSN: 1750-5836. DOI: https://doi.org/10.1016/j.ijggc.2023.103873. URL: https://www.sciencedirect.com/science/article/pii/S1750583623000439.

[35] Robert F. Meldau, Robert G. Shipley, and Keith H. Coats. “Cyclic Gas/Steam Stimulation of Heavy-Oil Wells”. In: *Journal of Petroleum Technology* 33.10 (Oct. 1981), pp. 1990–1998. ISSN: 0149-2136. DOI: 10.2118/8911-PA. eprint: https://onepetro.org/JPT/article-pdf/33/10/1990/2227565/spe-8911-pa.pdf. URL: https://doi.org/10.2118/8911-PA.

[36] Erfan Mohammadian et al. “Application of extreme learning machine for prediction of aqueous solubility of carbon dioxide”. In: *Environmental Earth Sciences* 75 (2016), pp. 1–11.

[37] Cuthbert Shang Wui Ng et al. “A Survey on the Application of Machine Learning and Metaheuristic Algorithms for Intelligent Proxy Modeling in Reservoir Simulation”. In: *Computers Chemical Engineering* 170 (2023), p. 108107. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2022.108107. URL: https://www.sciencedirect.com/science/article/pii/S0098135422004409.

[38] S.W. Perkins. *2 - The material properties of geosynthetics*. Ed. by R.W. Sarsby. 2007. DOI: https://doi.org/10.1533/9781845692490.1.19. URL: https://www.sciencedirect.com/science/article/pii/B9781855736078500023.

[39] Johan Rockström et al. “A safe operating space for humanity”. In: *Nature, v.461, 472-475 (2009)* 46 (Jan. 2013).

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[41] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. *A Survey on Oversmoothing in Graph Neural Networks*. 2023. arXiv: 2303.10993 [cs.LG].

[42] Zahraa Al Sahili and Mariette Awad. *Spatio-Temporal Graph Neural Networks: A Survey*. 2023. arXiv: 2301.10569 [cs.LG].

[43] Anna Samnioti and Vassilis Gaganis. “Applications of Machine Learning in Subsurface Reservoir Simulationmdash;A Reviewmdash;Part I”. In: *Energies* 16.16 (2023). ISSN: 1996-1073. DOI: 10.3390/en16166079. URL: https://www.mdpi.com/1996-1073/16/16/6079.

[44] Alvaro Sanchez-Gonzalez et al. *Learning to Simulate Complex Physics with Graph Networks*. 2020. arXiv: 2002.09405 [cs.LG].

[45] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[46] Chevron Schlumberger and Total. *Intersect Version 2013.1 Technical Description*. Tech. rep. Houston, TX: Schlumberger, 2013.

[47] Parisa Shokouhi et al. “Physics-informed deep learning for prediction of CO2 storage site response”. In: *Journal of Contaminant Hydrology* 241 (2021), p. 103835. ISSN: 0169-7722. DOI: https://doi.org/10.1016/j.jconhyd.2021.103835. URL: https://www.sciencedirect.com/science/article/pii/S0169772221000747.

[48] K. Stüben. *Algebraic Multigrid (AMG): An Introduction with Applications ; Updated Version of GMD Report No 53, March 1999*. 1999. URL: https://books.google.no/books?id=rzLiGwAACAAJ.

[49] Haoyu Tang and Wennan Long. *A Graph Neural Network Framework for Grid-Based Simulation*. 2022. arXiv: 2202.02652 [cs.LG].

[50] George Truc, Nejat Rahmanian, and Mahboubeh Pishnamazi. “Assessment of cubic equations of state: machine learning for rich carbon-dioxide systems”. In: *Sustainability* 13.5 (2021), p. 2527.

[51] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[52] Vishwesh Venkatraman and Bjørn Kåre Alsberg. “Predicting CO2 capture of ionic liquids using machine learning”. In: *Journal of CO2 Utilization* 21 (2017), pp. 162–168.

[53] Hai Wang and Shengnan Chen. “Insights into the Application of Machine Learning in Reservoir Engineering: Current Developments and Future Trends”. In: *Energies* 16.3 (2023). ISSN: 1996-1073. DOI: 10.3390/en16031392. URL: https://www.mdpi.com/1996-1073/16/3/1392.

[54] Gege Wen, Catherine Hay, and Sally M Benson. “CCSNet: a deep learning modeling suite for CO2 storage”. In: *Advances in Water Resources* 155 (2021), p. 104009.

[55] Gege Wen et al. “U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow”. In: *Advances in Water Resources* 163 (2022), p. 104180. ISSN: 0309-1708. DOI: https://doi.org/10.1016/j.advwatres.2022.104180. URL: https://www.sciencedirect.com/science/article/pii/S0309170822000562.

[56] Tailin Wu et al. “Learning Large-scale Subsurface Simulations with a Hybrid Graph Network Simulator”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2022. DOI: 10.1145/3534678.3539045. URL: https://doi.org/10.1145%2F3534678.3539045.

[57] Liang Yao, Chengsheng Mao, and Yuan Luo. *Graph Convolutional Networks for Text Classification*. 2018. arXiv: 1809.05679 [cs.CL].

[58] Peiyi Yao et al. “Application of machine learning in carbon capture and storage: An in-depth insight from the perspective of geoscience”. In: *Fuel* 333 (2023), p. 126296. ISSN: 0016-2361. DOI: https://doi.org/10.1016/j.fuel.2022.126296. URL: https://www.sciencedirect.com/science/article/pii/S0016236122031209.

[59] Junyu You et al. “Machine learning based co-optimization of carbon dioxide sequestration and oil recovery in CO2-EOR project”. In: *Journal of Cleaner Production* 260 (2020), p. 120866. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2020.120866.

[60] Si Zhang et al. “Graph convolutional networks: a comprehensive review”. In: *Computational Social Networks* 6 (Nov. 2019). DOI: 10.1186/s40649-019-0069-y.

[61] Zhi Zhong and Timothy R. Carr. “Geostatistical 3D geological model construction to estimate the capacity of commercial scale injection and storage of CO2 in Jacksonburg-Stringtown oil field, West Virginia, USA”. In: *International Journal of Greenhouse Gas Control* 80 (2019), pp. 61–75. ISSN: 1750-5836. DOI: https://doi.org/10.1016/j.ijggc.2018.10.011. URL: https://www.sciencedirect.com/science/article/pii/S1750583618301567.

[62] Zhi Zhong et al. “Application of a convolutional neural network in permeability prediction: A case study in the Jacksonburg-Stringtown oil field, West Virginia, USA”. In: *Geophysics* 84.6 (Oct. 2019), B363–B373. ISSN: 0016-8033. DOI: 10.1190/geo2018-0588.1. eprint: https://pubs.geoscienceworld.org/geophysics/article-pdf/84/6/B363/4933558/geo-2018-0588.1.pdf. URL: https://doi.org/10.1190/geo2018-0588.1.