Pacific Northwest
NATIONAL LABORATORY

PNNL-32590

NRAP-Open-IAM: Generic Aquifer Component

Development and Testing

February 2022

Diana H Bacon

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: mb-reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

PNNL-32590

NRAP-Open-IAM: Generic Aquifer Component

Development and Testing

February 2022

Diana H Bacon

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Abstract

The Generic Aquifer Model calculates the concentrations of dissolved salt and dissolved CO_2 surrounding a leaking legacy well. The Generic Aquifer model can also estimate the size of an "impact plume" where concentration changes exceed user-specified thresholds. The model is a component of NRAP-Open-IAM, an open-source Integrated Assessment Model (IAM) developed by the National Risk Assessment Partnership (NRAP) to perform risk assessment for geologic CO_2 storage. The input parameters were selected to cover a wide range of groundwater aquifers and leakage rates. The generic aquifer model was developed using a generative adversarial deep learning network, trained using a large synthetic dataset of STOMP multiphase flow simulations. The deep learning model predictions of dissolved salt and dissolved CO_2 in the aquifer compare well to the original STOMP simulation results. The extent of aquifer impacted by leaking CO_2 or brine is calculated using a user-defined mass fraction threshold. The aquifer impact volumes calculated based on STOMP simulation results compare well to those calculated based on the deep learning model. In a provided python script, gridded observation results from the generic aquifer component of NRAP-Open-IAM are converted to HDF5 format files for monitoring design with the DREAM code.

Acknowledgments

The research presented in this report was completed as part of the National Risk Assessment Partnership (NRAP). Funding for this project was provided to Pacific Northwest National Laboratory (PNNL) under DOE contract number DE-AC0576RL01830 by the U.S. Department of Energy's (DOE's) Office of Fossil Energy. A portion of the modeling research was performed using PNNL's Research Computing cluster.

Acronyms and Abbreviations

CCUS	Carbon capture, utilization, and storage
EDX	Energy Data Exchange
IAM	Integrated Assessment Model
L1 Loss	See MAE
MAE	The mean absolute error between STOMP and surrogate model predictions
NETL	National Energy Technology Laboratory
NRAP	National Risk Assessment Partnership
ROM	Reduced Order Model
SDWA	Safe Drinking Water Act
US DOE	United States Department of Energy
USEPA	United States Environmental Protection Agency

Contents

Figures

Tables

1.0 Introduction

Carbon capture, utilization, and storage (CCUS) technologies are being developed, both domestically and internationally, for their potential to mitigate environmental impacts associated with atmospheric release of carbon dioxide (CO_2) from anthropogenic sources, such as power production from fossil fuels and other large industrial sources. Over the last decade, the United States Department of Energy (US DOE) has invested millions of dollars developing carbon capture technologies and demonstrating safe and secure geologic carbon storage via a number of pilot-scale projects sited throughout the country (NETL 2015). To date, these projects have stored more than 16 million tonnes of CO_2 (NETL 2018).

Within the US, CO_2 injection activities are overseen by the US Environmental Protection Agency (EPA) following regulations (the Class VI Rule) promulgated under the Safe Drinking Water Act (SDWA) (USEPA 2010). The Class VI regulations are designed to protect underground sources of drinking water (USDWs), and include strict requirements for site characterization, CO_2 injection well construction, injection operations, site monitoring, financial liability, and record keeping/reporting. Key elements of the Class VI permitting process include delineating an Area of Review (AoR) and defining an appropriate Post-Injection Site Care (PISC) period for the project, both of which require simulated CO_2 saturations and pressure distributions from computational models. The models are based on site-specific data and are updated periodically during the lifetime of the project to evaluate reservoir performance and evolution of the storage system.

Despite the sophistication of today's multi-physics reactive transport codes, significant uncertainty exists in predicting the performance of geologic storage reservoirs. Challenges associated with developing greenfield sites include the inherit difficulty in scaling a few point source measurements of geological structure and reservoir permeability derived from characterization of borehole samples throughout the extensive area likely to be impacted by a commercial-scale CO_2 injection, a lack of site-specific data on the behavior of supercritical CO_2 in the reservoir being evaluated, and understanding changes in the transport behavior of carbon dioxide caused by changes in pressure and/or temperature and the buoyant nature of CO_2 over the long time scales required for geologic sequestration to have long-term benefit to atmospheric CO_2 levels. Additionally, the computational resources required to run high fidelity simulations limits their usefulness in performing sensitivity analysis for uncertainty reduction.

To help address this need, the US DOE established the National Risk Assessment Partnership (NRAP), an initiative across five US DOE national laboratories with the goal of developing defensible, science-based methodologies and platforms for quantifying risks amidst system uncertainty. In 2017, the NRAP team released a set of ten tools (i.e., the NRAP Toolset) that can be used to estimate risks associated with carbon sequestration (https://edx.netl.doe.gov/nrap/).

NRAP-Open-IAM is an open-source Integrated Assessment Model (IAM) developed by the National Risk Assessment Partnership (NRAP) to perform risk assessment for geologic CO2 storage (GCS) (Vasylkivska et al. 2021). The goal of NRAP-Open-IAM is to extend beyond risk assessment into risk management, containment assurance, and decision support. NRAP-Open-IAM builds on many years of NRAP tool development for risk assessment, including the NRAP-IAM-CS also developed by the NRAP project (Pawar et al. 2016). An open-source Python framework allows NRAP-Open-IAM to: 1) take advantage of standard Python libraries and other open source analytical libraries written in Python; 2) be applied on multiple platforms; 3) have

more flexible options of selecting modules for a specific study; and 4) give advanced users the option to modify the IAM to fit their need as well as enhancing the potential for community contributions to the software. The implementation of the reduced-order models and analytical tools within the NRAP-Open-IAM makes the risk assessment process computationally efficient enough to simulate an operational CO_2 storage site, potential events and various scenarios in a probabilistic/ensemble manner. The NRAP-Open-IAM is equipped with capabilities to: 1) inform monitoring design; 2) assess model concordance to measured field data; 3) evaluate mitigation alternatives; and 4) provide probabilistic risk assessment and update the risk as new data becomes available.

NRAP-Open-IAM models are created by linking reduced order representations of sophisticated component models together into a complete GCS system. Each component model describes the structure or flow behavior in a critical element of a GCS site. Component models are modular and are designed to be interchangeable. Users build NRAP-Open-IAM models by selecting component models and specifying inputs that represent the characteristics of their GCS site. Inputs to NRAP-Open-IAM component models can either be specified as a single value or a range of values. If a range of values is identified for some model inputs, these values will be randomly sampled when stochastic simulations are run. The component models of NRAP-Open-IAM fall are organized into four major categories:

- **Stratigraphy**. The stratigraphy component details the structure of the GCS system. Stratigraphy inputs include the number of shale and aquifer layers in the model, the thicknesses of these layers, and the thickness of the reservoir.

- **Reservoir**. The reservoir component describes the conditions in the reservoir during the simulation time period. NRAP-Open-IAM is not a reservoir simulator. However, users can simulate a simplified CO_2 injection using the simple and analytical reservoir components. Inputs for these models include reservoir characteristics (permeability, porosity, thickness, extent), CO_2 and brine characteristics (density, viscosity), and injection rate. More sophisticated reservoir behavior can be included in the NRAP-Open-IAM by including simulation results from a high-fidelity numerical simulator as a look up table.

- **Leakage pathway**. The leakage pathway component simulates the upward flow of CO_2 and brine out of the reservoir. NRAP-Open-IAM contains multiple interchangeable leakage pathway components that can simulate flow through cemented and uncemented wells, seals, and faults. Users must specify the properties of the leakage pathway, which vary depending on its type. For example, the inputs for the cemented wellbore component are the well radius, the permeability of the well cement, and the permeability of potential thief zones.

- **Receptor**. The receptor component simulates either the flow of CO_2 and brine in an aquifer (shallow or deep) or the atmosphere. Aquifer component models consider geochemical reactions and predict the size of CO_2 and brine impact plumes. Several aquifer components exist that represent different types of aquifers (e.g., carbonate, deep alluvium). Model inputs for each aquifer component typically include general characteristics of the formation, such as its thickness, depth, porosity, permeability, and anisotropy. The atmosphere component simulates CO_2 dispersion after leakage out of the ground. Inputs for the atmosphere component include ambient pressure and temperature, wind velocity, CO_2 source temperature, and coordinates of potential receptors.

The characteristics of the aquifer component models for NRAP-Open-IAM have evolved over time. The first two aquifer component models developed for NRAP-Open-IAM were based on using site-specific data from two aquifers, the Edwards Aquifer (Bacon et al. 2016) and the High Plains Aquifer (Carroll et al. 2016). However, the models accept aquifer characteristics as variable inputs and so they may have more broad applicability. (Keating et al. 2016) concluded that pH and TDS predictions are the most transferable to other aquifers based on the analysis of the nine water quality metrics (pH, TDS, 4 trace metals, 3 organic compounds) and presented guidelines for determining the aquifer types for which those two surrogate models should be applicable.

The FutureGen2 aquifer models were designed to be more general in order to encompass both carbonate and sandstone aquifers overlying the FutureGen 2.0 proposed carbon storage site (Bacon et al. 2019). Five monitoring metrics were used to indicate an aquifer impact: pressure, temperature, dissolved CO_2, pH and TDS. The precision thresholds of sensors from the proposed monitoring plan for the FutureGen 2.0 site (FutureGen Industrial Alliance 2013) were used to delineate an impact for each of these metrics.

The new Generic Aquifer Model described in this report calculates the concentrations of dissolved salt and dissolved CO_2 surrounding a legacy well. The Generic Aquifer model can also estimate the size of an "impact plume" where concentration changes exceed user-specified thresholds. Dissolved CO_2 was selected because Romanak et al. (2012) found that dissolved inorganic carbon (DIC) was a useful monitoring metric because changes in DIC with CO_2 leakage were consistent across geochemical environments, indicating that prior characterization of aquifer minerals may not be necessary if DIC is used as the primary monitoring parameter. Dissolved salt was selected because salinity often increases with depth (Bloomfield et al. 2020), resulting in a notable difference in salinity between the storage reservoir and an overlying aquifer.

This report describes the workflow for development and testing of the Generic Aquifer component for NRAP-Open-IAM, summarized in Figure 1.

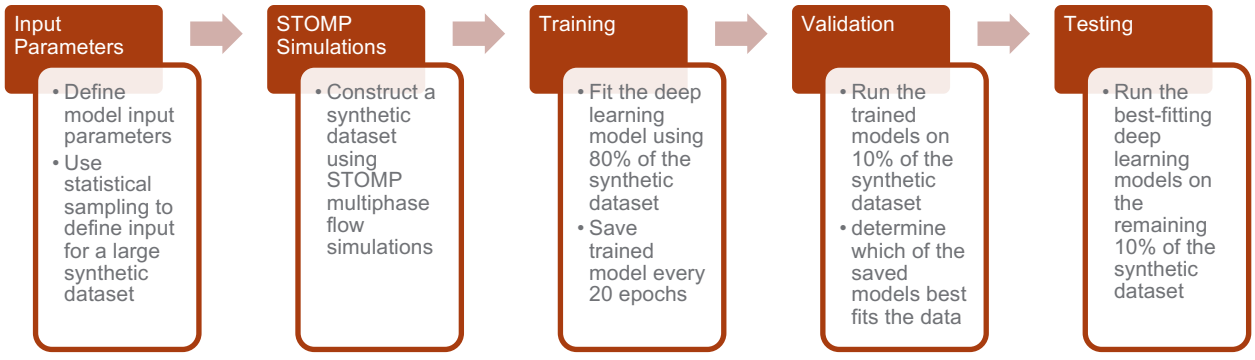

Figure 1. Workflow summary for development and testing of the Generic Aquifer component of NRAP-Open-IAM.

2.0 Input Parameters

The input parameters shown in Table 1 were selected to cover a wide range of groundwater aquifers. Thickness is the aquifer thickness from bottom to top. Depth is the depth below ground surface (bgs) to the top of the aquifer. Porosity is the fraction of void space in the aquifer rock. Horizontal permeability is a measure of the ability of the aquifer rock to transmit fluid in the horizontal direction. Anisotropy is the ratio of the vertical to horizontal permeability. The initial aquifer salinity is the mass fraction of salt in the aquifer before a leak occurs. The leak salinity is the mass fraction of salt in fluid leaking into the aquifer. The CO2 and brine leakage rates are given in mass units of kilograms per second. Because permeability, anisotropy and leakage rates vary over several to many orders of magnitude, the log base 10 of their values are used as input.

Table 1. Ranges of input parameters for the Generic Aquifer component.

Parameter	min	max
Thickness (m)	25	250
Depth (m bgs)	100	4100
Porosity	0.02	0.2
Horizontal Permeability ($\log_{10}$ m^2)	-14	-10
Anisotropy ($\log_{10}$ K_h/K_v)	0	3
Initial Aquifer Salinity (mass fraction)	0	0.015
Leak Salinity (mass fraction)	0.015	0.05
CO_2 Leak Rate ($\log_{10}$ kg/s)	-9	1.5
Brine Leak Rate ($\log_{10}$ kg/s)	-9	1.5

Random samples of the model input parameters were selected using Latin Hypercube Sampling (Iman et al. 1981), assuming that each parameter is uniformly distributed. The parameter space was divided into ten depth intervals of 400 m each, and 6250 samples were generated for each depth interval, for a total of 62,500 samples. Figure 2 shows a pair plot illustrating the distribution of parameter values for one of the depth intervals. The number of parameter samples is so large and evenly distributed that the datapoints appear as a nearly solid block of color.

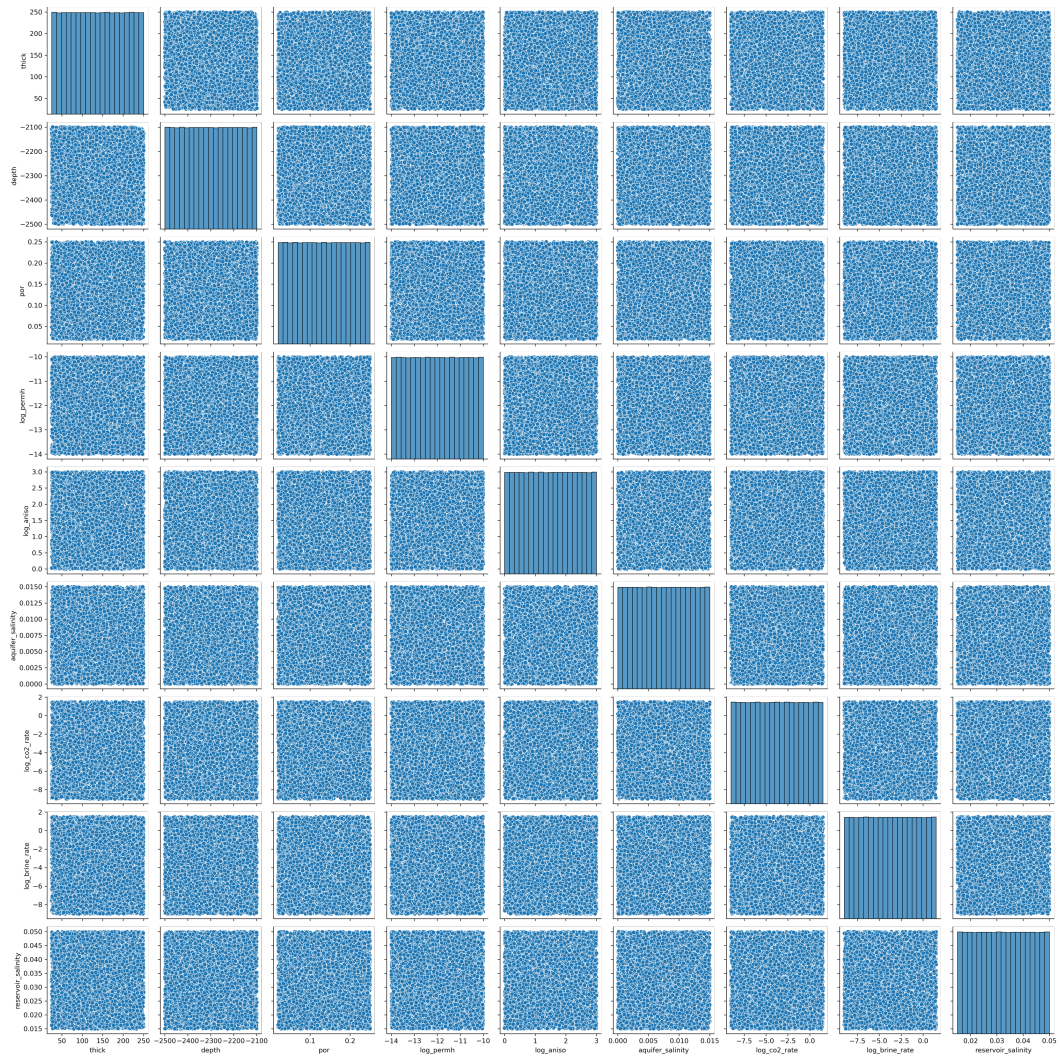

Figure 2. Pair plot for input parameters of simulations used to train the aquifer model for the depth interval 2100 to 2400 m, showing the uniform distribution of values in 6250 samples.

The sample set of input parameters for each depth interval was divided into three subsets: training, validation and test. The training dataset is the subset of data that is used to fit the model. The validation dataset is the subset of data that is used for an unbiased evaluation of a model fitted on the training dataset while tuning model hyperparameters (For example, see Section 5.0 for a discussion of adjustments to the model training rate). The testing dataset is the subset of data used for an unbiased evaluation of a final model fitted on the training dataset.

An unbiased evaluation means that the model is validated and tested on data that has not been seen during training. Randomly splitting the entire dataset into three subsets is a common method for unbiased evaluation. In this case, 80% of the data was randomly selected to be in the training dataset, while the validation and testing datasets each received 10% of the data. This is commonly referred to as an 80-10-10 split, where 80% + 10% +10% = 100%.

3.0 STOMP Simulations

The 62,500 STOMP simulations were conducted using the input parameters from Section 2.0. Simulations were performed using STOMP-CO2 (isothermal multiphase flow of CO_2 and brine). Parameters were substituted into a template input file using a python script. The python script also performed the train-valid-test split as described in section 2.0 using Scikit-learn (Pedregosa et al. 2011). Both the template input file and the python script are listed in Appendix A.

The initial temperature and pressure in the simulations were assumed to be a function of depth. Hydrostatic initial pressures and a geothermal gradient of 1.2 °F/100 ft (21.7 °C/km) (Vaught 1980) were assumed. Simulations were run for a simulation time of 70 years, and model output saved at 0, 1, 2, 5, 10 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65 and 70 years.

To encompass impact plumes for both very small and very large CO_2 or brine leaks, a radial grid of 50 miles (80,467 meters) in radius was used. Grid radii ranged in size from 3.24 m to 5,877 m in the horizontal direction. Ten vertical grids were used, each one-tenth of the aquifer thickness, for a total of 1000 nodes. For each of the leakage scenarios, distributions of dissolved CO_2 and dissolved salt were calculated.

As an example of STOMP simulation output, Figure 3 shows predicted dissolved CO_2 mass fraction in an aquifer after 70 years of CO_2 leakage at a rate of 2.56 kg/s and brine leakage at a rate of 9.6×10^{-6} kg/s. In another example, Figure 4 shows dissolved salt mass fraction in an aquifer after 70 years of brine leakage at a rate of 4.6×10^{-9} kg/s and brine leakage at a rate of 16.6 kg/s.

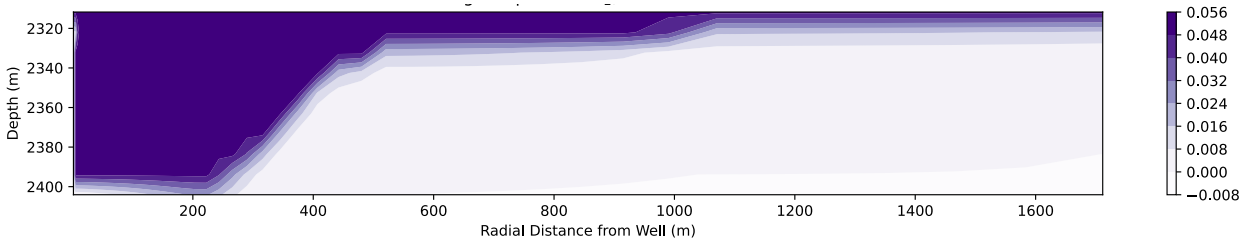

Figure 3. Dissolved CO_2 mass fraction after 70-year CO_2 leak at rate of 2.56 kg/s.

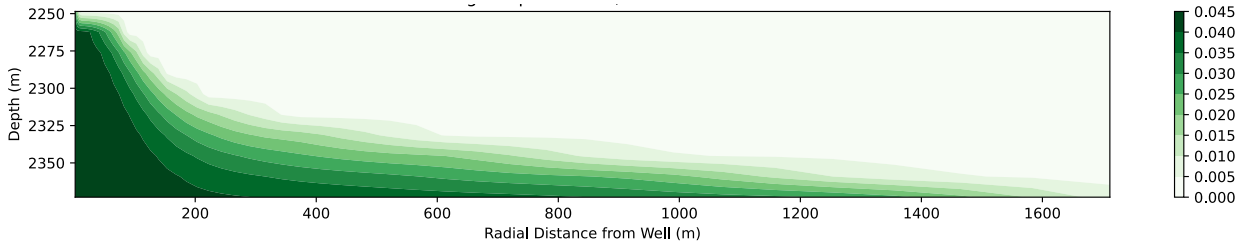

Figure 4. Dissolved salt mass fraction after 70-year brine leak at rate of 16.6 kg/s.

4.0 Impact Delineation

To delineate the impact of a leak on an aquifer, a threshold concentration must be set. The generic aquifer component allows the user to specify the threshold values for dissolved CO_2 mass fraction and dissolved salt mass fraction. These values may range between 0 and 1. However, it should be noted that the leak dissolved salt mass fraction is limited to a range of 0.015 to 0.05, so these are practical limits for the dissolved salt mass fraction threshold.

The impact metrics are volume, average radius and average thickness of the impact plume in the aquifer. An example grid of CO_2 mass fraction predictions is shown in Figure 5. If the user sets the threshold for CO_2 mass fraction to 0.04, then the grid cells that are shaded black will be considered a part of the impact plume. Volumes for each cell in the grid are shown in Figure 5.

- To calculate the plume volume, the node volume for each cell above the threshold is summed.
- To calculate the average thickness of the plume, the thickness of the plume in each column with cells above the threshold is averaged.
- To calculate the average radius of the plume, the radius of the plume in each row with cells above the threshold is averaged.

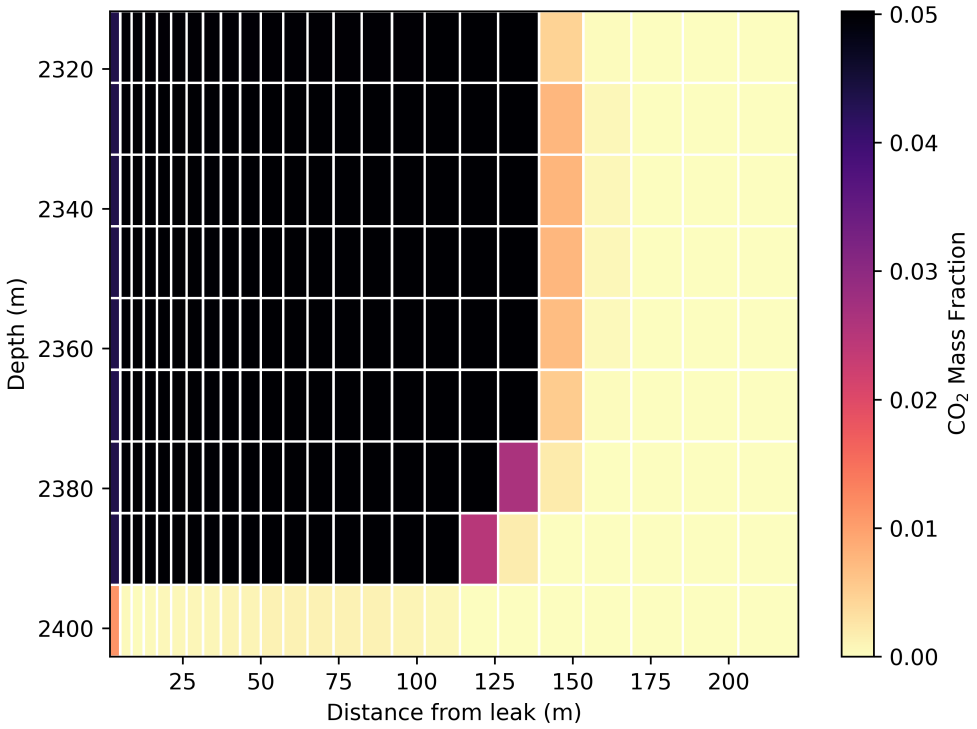

Figure 5. CO_2 Mass Fraction in each grid cell of an example STOMP simulation.

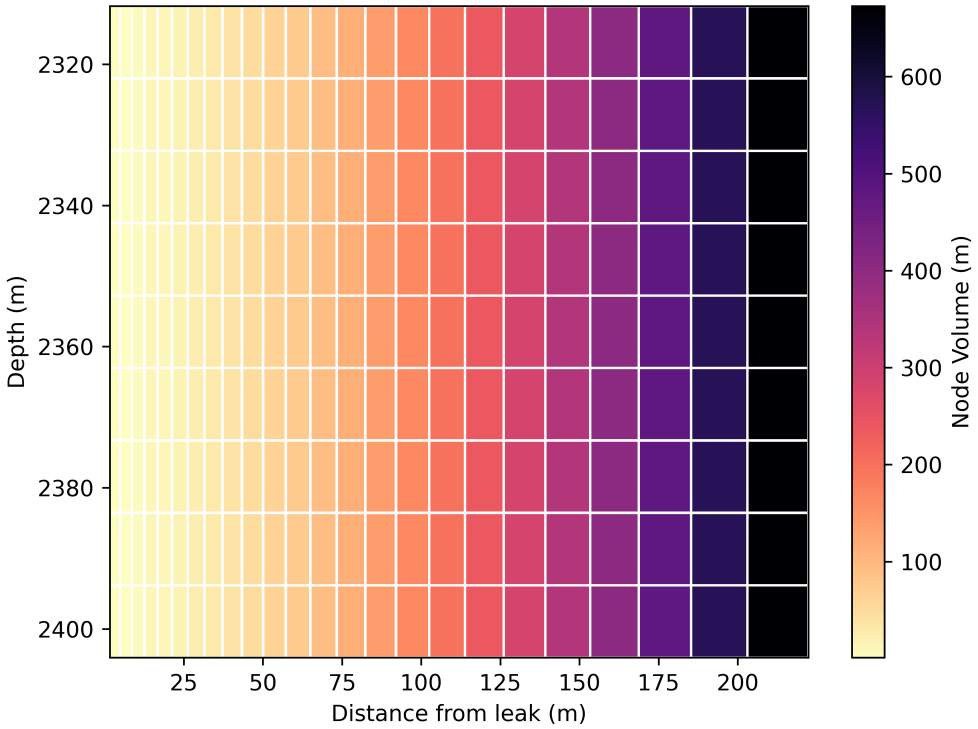

Figure 6. Node volume for each grid cell close to the leak.

5.0 Training

Deep learning surrogate models were trained to predict dissolved CO_2 and dissolved salt mass fraction in an aquifer in response to CO_2 and brine leakage, like the results shown in Figure 3 and Figure 4.

Pix2Pix is a Generative Adversarial Network, or GAN, model (Isola et al., 2017) that has been shown to have wide applicability to image-to-image translation problems with minimal modification. The similarity between image-to-image translation and the input-output fields utilized by flow models has been recognized, and the Pix2Pix method has been applied to predicting CO_2 plume migration in two-dimensional horizontal heterogeneous formations by Zhong et al. (2019). Using example code as a starting point (Google, 2021), the method has been extended to reproducing outputs of the STOMP simulator based on a set of input parameters. The neural networks were implemented in Tensorflow 2.5 using Keras.

The method uses two competing neural networks, a generator and a discriminator. The generator architecture, shown in Figure 7, is a modified U-Net (Ronneberger et al. 2015) regressor. Each encoder block is 2D Convolution → Batch normalization → Leaky ReLU. Each decoder block is Transposed 2D Convolution → Batch normalization → Dropout (applied to the first 3 blocks) → ReLU. Skip connections pass information directly between similarly-sized encoder and decoder layers. The discriminator architecture (Figure 8) is a classifier that classifies a patch of the model output as either real (STOMP output) or fake (generator output). Each discriminator block is 2D Convolution → BatchNorm → Leaky ReLU.

All inputs were on a 100 x 10 grid. Input layers were padded to achieve dimensions that are powers of two (128 x 16) to enable symmetric convolution and deconvolution. Cropping layers are applied to output so that predictions are on the original 100 x 10 grid.

Neural networks were trained separately for predicting dissolved CO_2 and dissolved salt. The training batch size was 1. There were ten model inputs for the dissolved CO_2 and dissolved salt deep learning models (Table 1). Given the STOMP grid dimensions of 100 x 10, the deep learning model inputs were a 4-dimensional tensor of size [batch size, 100, 10, 10] and model outputs were a 4-dimensional tensor of [batch size, 100, 10, 1].

Isola et al. (2017) used the Adam optimizer for rapid convergence. However, use of this aggressive optimizer can lead to mode collapse of the GAN, wherein the generator predicts the same solution for every time step. Zhong et al. (2019) solved this problem using the SCP optimizer and only updating the discriminator every 5 epochs. However, that solution required thousands of training epochs. To avoid mode collapse, the original Adam optimizer was used with a reduced training rate for the discriminator ($2x10^{-6}$) which was two orders of magnitude lower than that for the generator ($2x10^{-4}$). This modification resulted in acceptable predictions in 100 training epochs and avoided mode collapse.

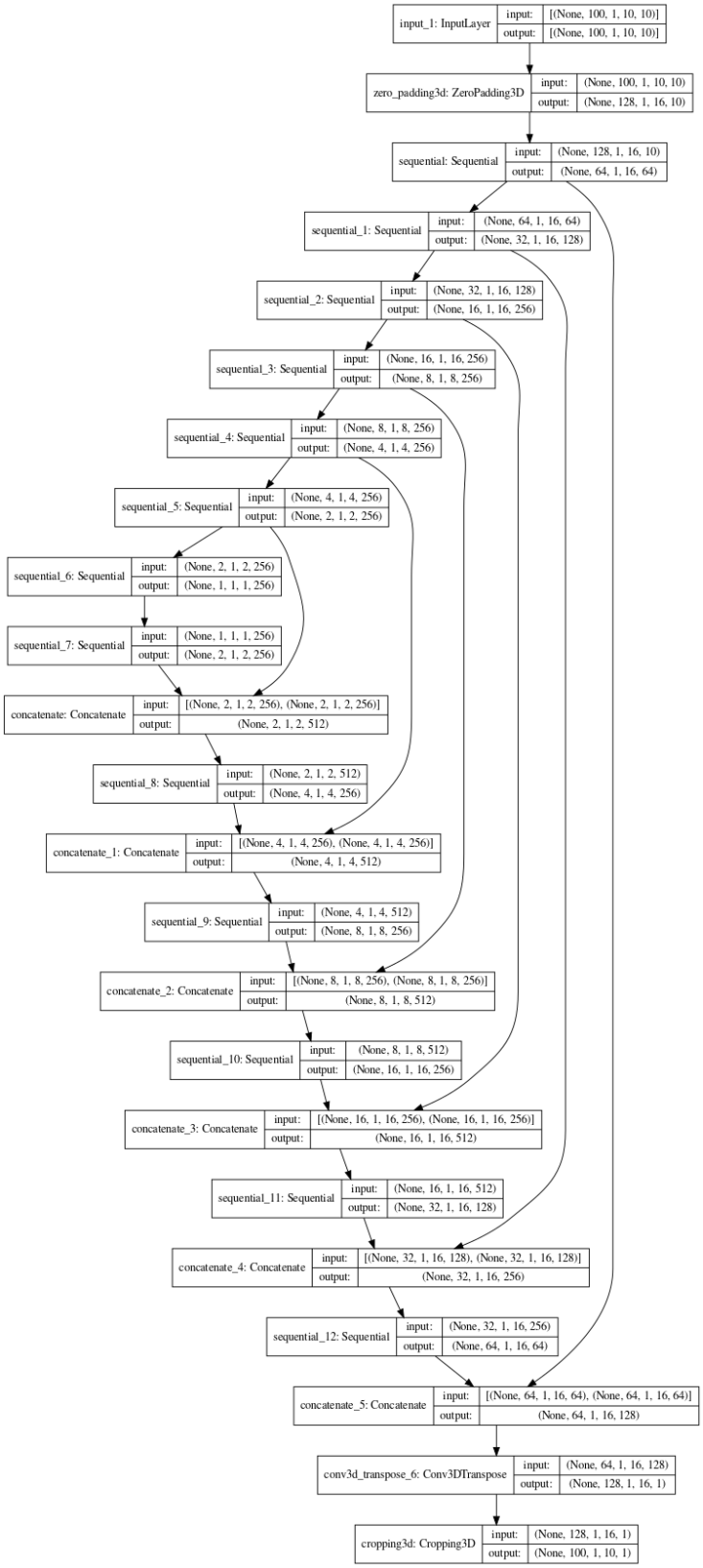

Figure 7. Generator Architecture

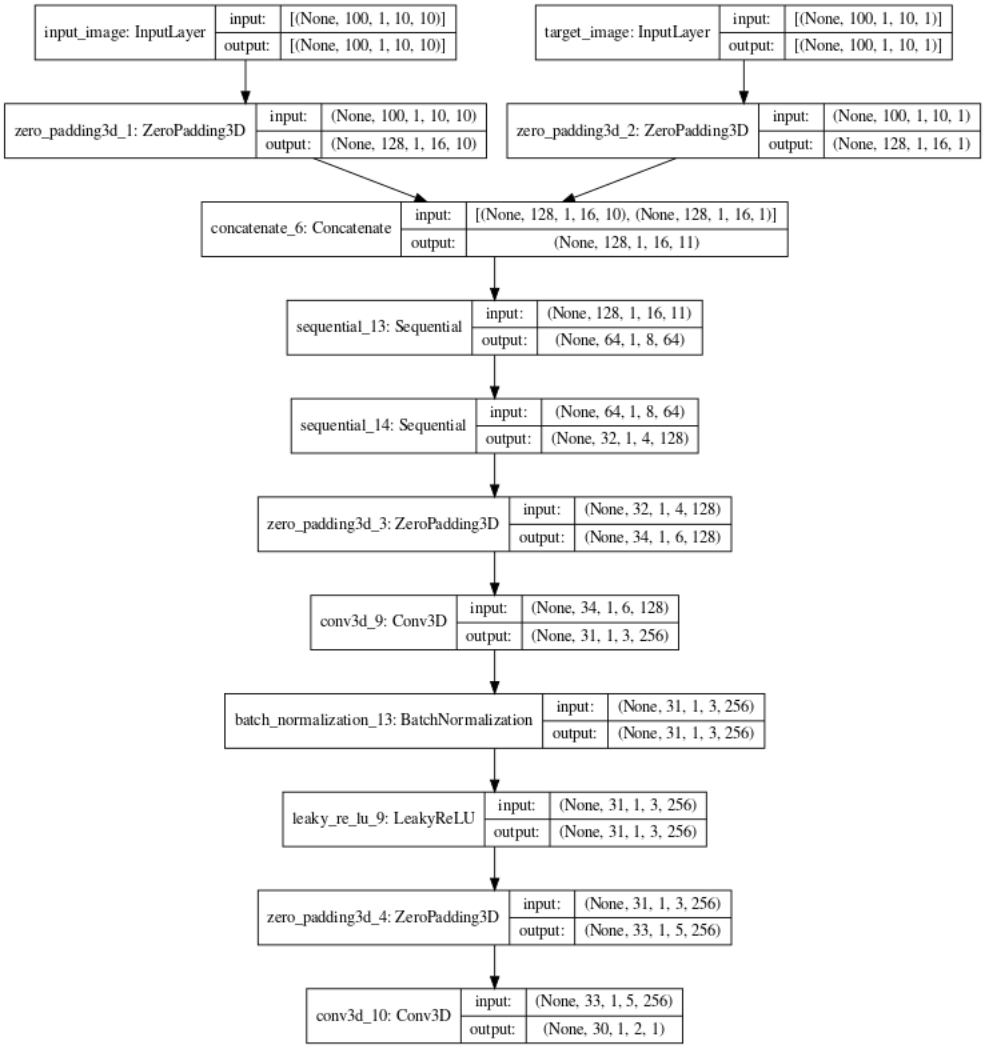

Figure 8. Discriminator Architecture

Figure 9 shows that on average the mean absolute error (L1 loss) between the dissolved salt mass fractions predicted by the deep learning model decreases with each training epoch.

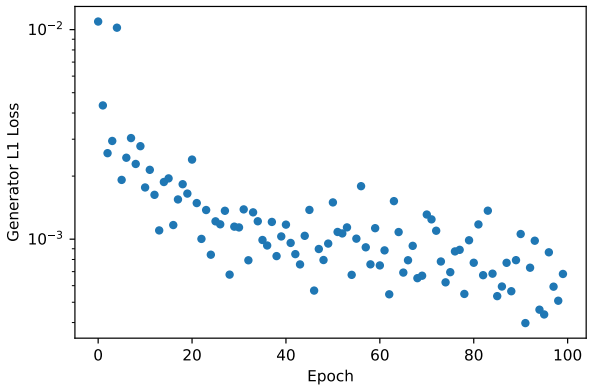

Figure 9. Generator L1 Loss during Training (2100-2499m salt mass fraction)

The generator loss function is equal to the GAN loss + lambda * L1 loss, where the L1 loss is the mean absolute error between STOMP simulation results and the machine learning model prediction. Lambda is a weighting factor to ensure that the two terms in the loss function are of the same order of magnitude. Values of 100 for lambda were used. The GAN loss is sigmoid cross entropy (binary classification), indicating how often a result generated by the machine learning model is classified correctly. The discriminator loss is equal to the sum of two binary classification losses, one for classifying “real” images (STOMP predictions) correctly and another for classifying “fake” images (generator predictions) correctly. If either the generator GAN loss or the discriminator loss gets very low it’s an indicator that one model is dominating the other, and the combined model is not training successfully. The value log(2) = 0.69 for the generator and 2 x log(2) = 1.395 for the discriminator are good reference points for these losses, as it indicates a perplexity of 2, where the discriminator is on average equally uncertain about the two options. The loss functions for the discriminator (Figure 10) and the generator (Figure 11) were monitored during training using Tensorboard to ensure that they remained close to these values.

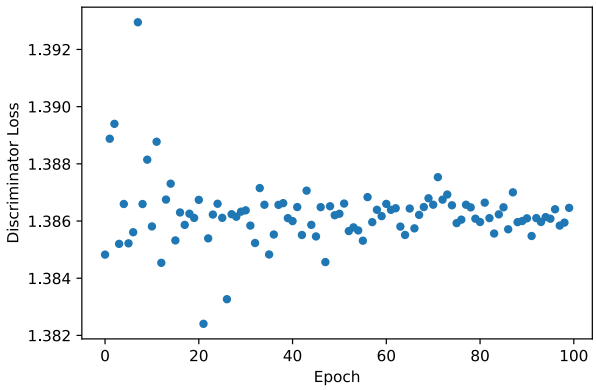

Figure 10. Discriminator Loss during Training (2100-2499m salt mass fraction)

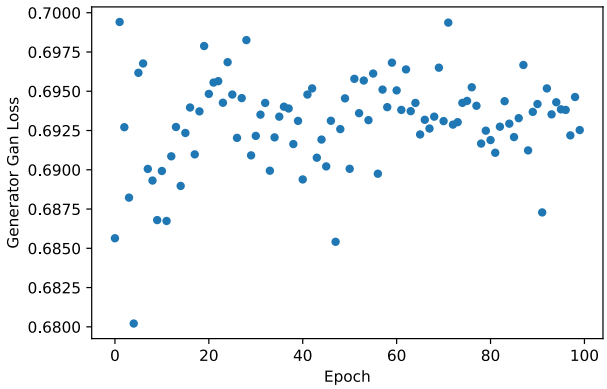

Figure 11. Generator GAN Loss during Training (2100-2499m salt mass fraction)

6.0 Validation

During training, the model was saved every 20 epochs. Each saved model was run against the validation dataset representing 10% of the data. L1 loss is the mean absolute error between STOMP simulation results and the machine learning model prediction. The generator L1 loss for each depth interval for CO_2 mass fraction is shown in Figure 12.

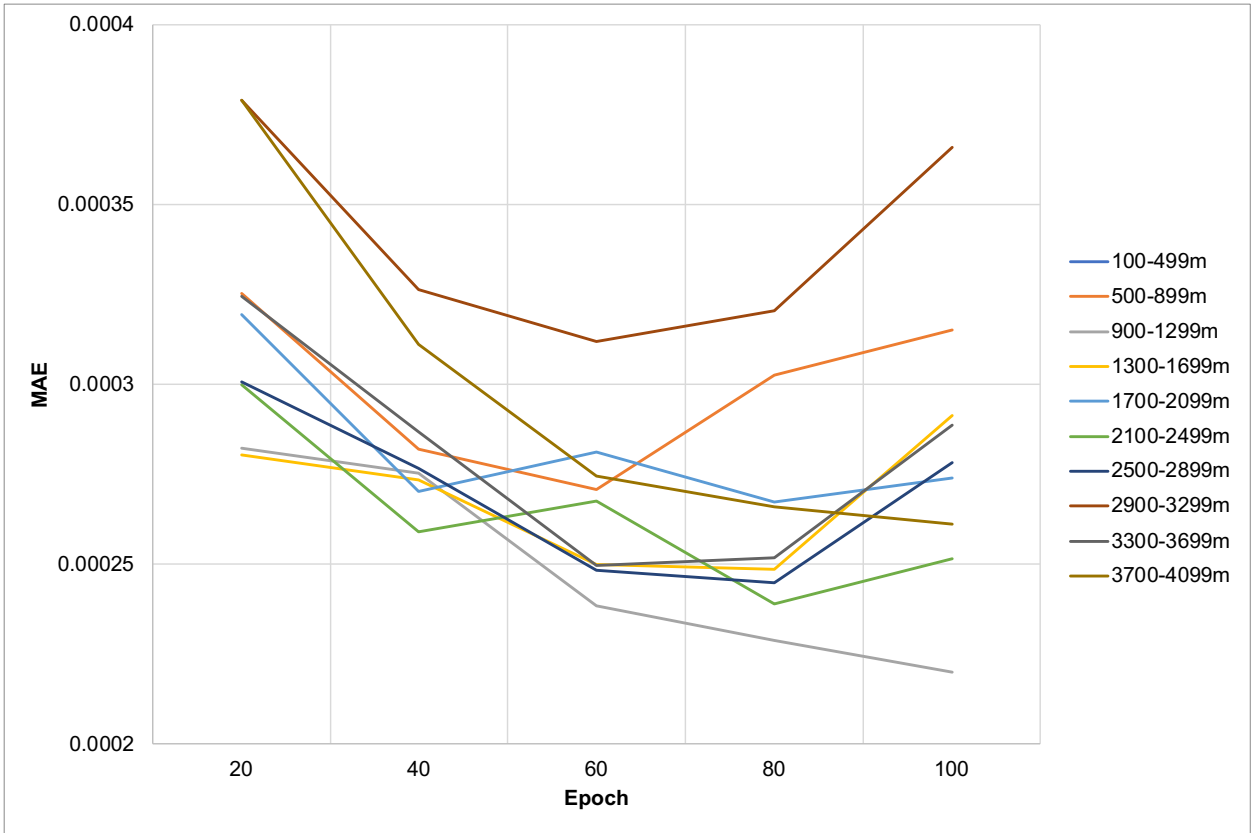

Figure 12. Generator MAE during Validation (CO_2 mass fraction).

As long as the validation loss continues to decrease during training, the model is not overfitting. For each depth interval, the model with the lowest validation error is saved for further testing. For example, for the 900-1299m depth interval, the validation loss is lowest for the model saved after 100 training epochs, so this model is used for further testing. However, for the 2900-3299m depth interval, the validation loss is lowest for the model saved at 60 training epochs, so that model is used for further testing. A similar selection process was used to select the best models for testing aqueous salt models (Figure 13).

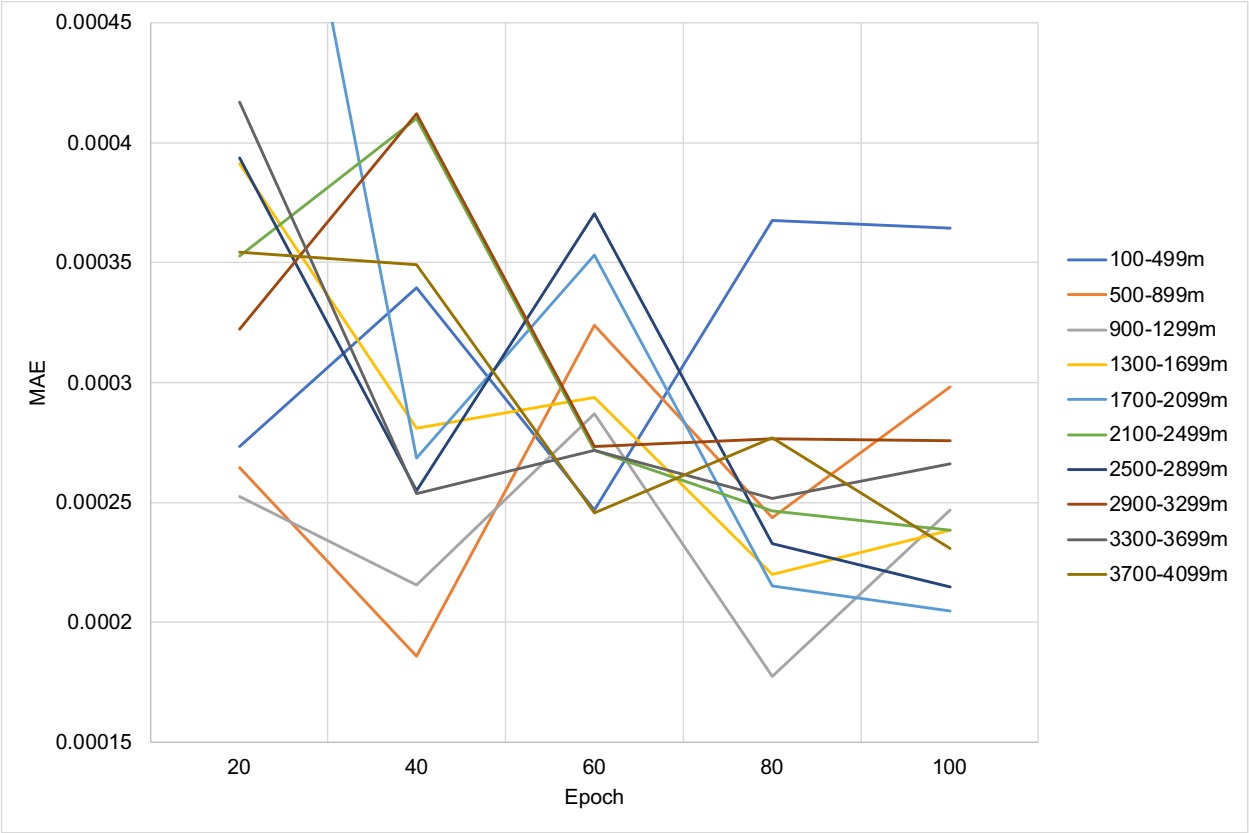

Figure 13. Generator MAE during Validation (Salt mass fraction).

7.0 Testing

The optimal models identified during validation (Section 6.0) were run on the test dataset consisting of 10% of the data. Example results for a large (mostly) CO_2 leak are shown in Figure 14. The absolute error is generally low except for a few spots on the edge of the plume. The same is true for different example results of a (mostly) salt leak shown in Figure 15.

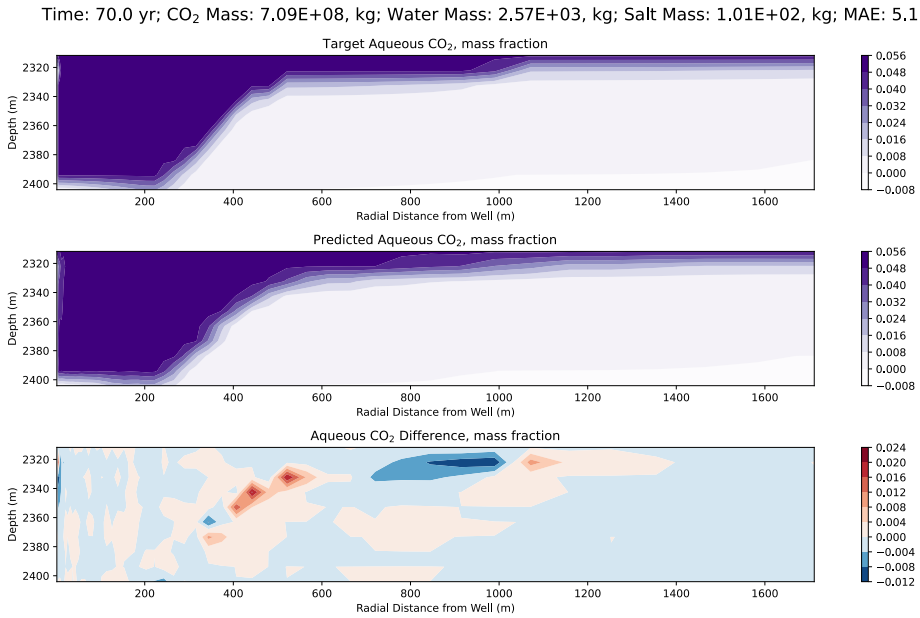

Figure 14. Comparison of target and predicted CO_2 mass fraction after leak consisting of 7.09×10^8 kg CO_2 and 2.57×10^3 kg water containing 1.01×10^2 kg salt.

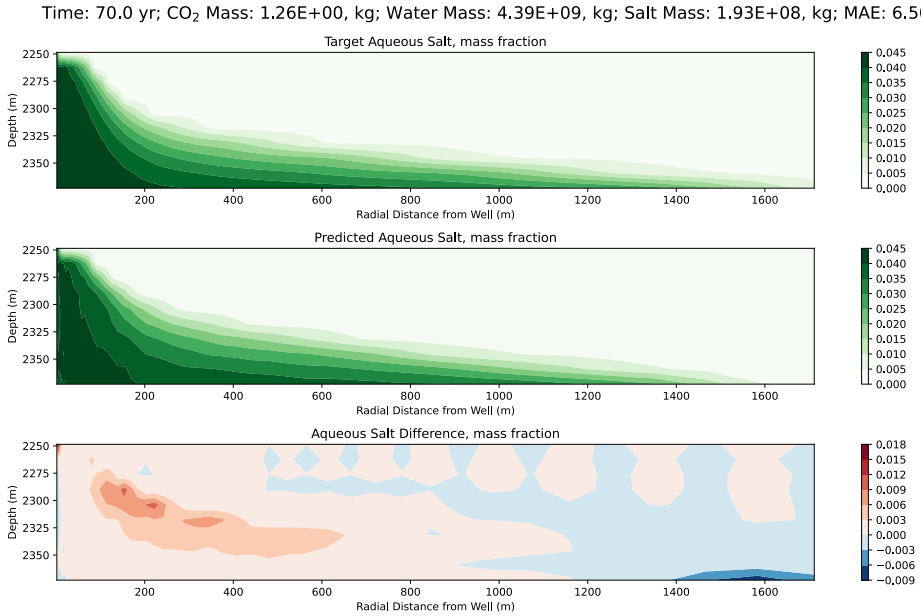

Figure 15. Comparison of target and predicted salt mass fraction after leak consisting of 1.26 kg CO_2 and 4.39×10^9 kg water containing 1.93×10^8 kg salt.

The testing dataset for each depth interval consists of 625 simulations. With a grid of 100x10 = 1000 cells and 17 saved time steps, this results in 10,625,000 different points in space and time to compare for each of the 10 depth intervals and 2 types of prediction (CO_2 and salt). Scatterplots comparing the target STOMP results and the predictions of the machine learning model for each depth interval are shown in Figure 16 for aqueous CO_2 mass fraction and in Figure 17 for aqueous salt mass fraction. The aqueous CO_2 mass fraction range up to values on the order of 10^{-2} and aqueous salt mass fractions range up to values on the order of 10^{-1}.

The mean absolute error between mass fractions predicted by STOMP and the deep learning model for all points in each depth interval are shown in Table 2. The mean absolute error was calculated using the following python functions:

```
def_error(actual: np.ndarray, predicted: np.ndarray):
    """ Simple error """
    return actual - predicted

def mae(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Absolute Error """
    return np.mean(np.abs(_error(actual, predicted)))
```

The mean absolute error between the aqueous CO_2 and aqueous salt are two to three orders of magnitude lower, on the order of 10^{-4}, with the worst fit being for CO_2 in the shallowest depth interval. Since in the 100-499m depth interval the large variability of thermodynamic properties of free phase CO_2 is expected as the conditions are close to its critical point, modeling is more difficult to fit a single model.

Table 2. Mean Absolute Error (MAE) for aqueous mass fraction predictions in each depth interval.

Depth Interval	MAE, Salt Mass Fraction	MAE, CO_2 Mass Fraction
100-499m	2.82E-04	5.56E-04
500-899m	1.98E-04	2.58E-04
900-1299m	2.03E-04	2.68E-04
1300-1699m	1.98E-04	2.61E-04
1700-2099m	2.13E-04	2.74E-04
2100-2499m	2.37E-04	2.80E-04
2500-2899m	1.90E-04	2.84E-04
2900-3299m	3.15E-04	3.24E-04
3300-3699m	2.06E-04	2.46E-04
3700-4099m	2.70E-04	3.05E-04

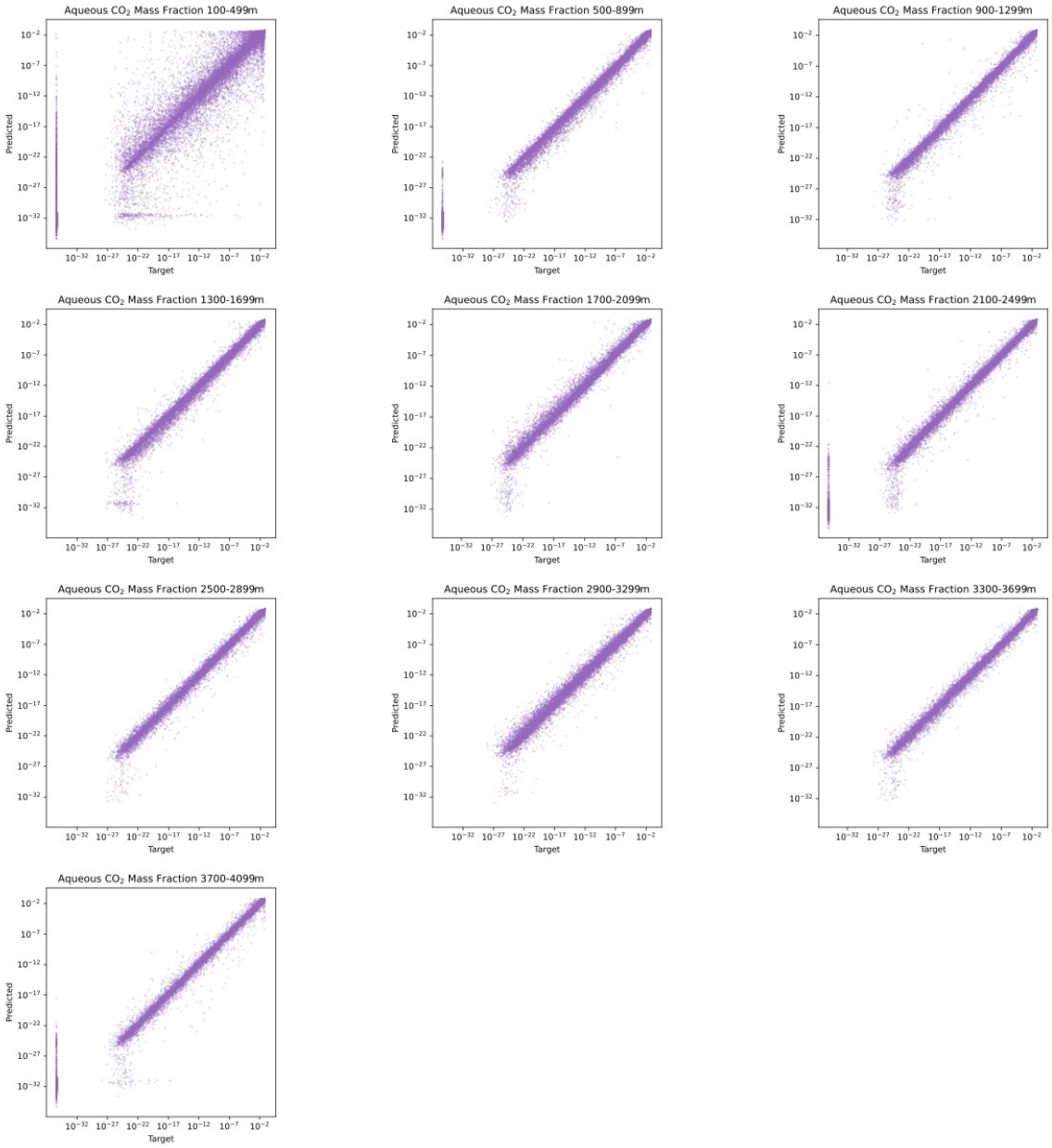

Figure 16. Comparison of target and predicted CO_2 mass fraction for all grid locations and times in the test dataset.

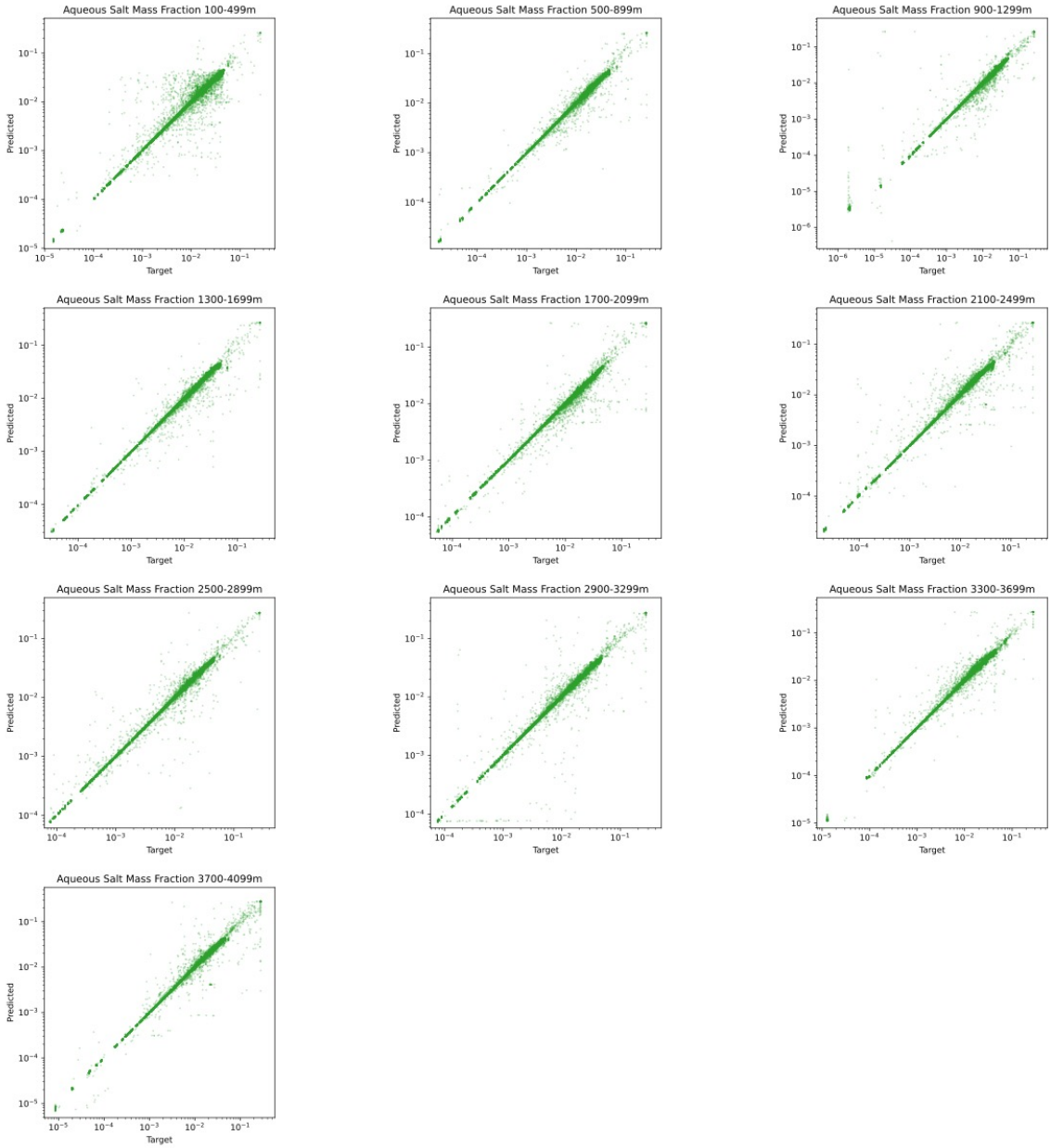

Figure 17. Comparison of target and predicted salt mass fraction for all grid locations and times in the test dataset.

As seen in Figure 14 and Figure 15, the machine learning model seems to predict the shape of CO_2 and salt plumes accurately. To confirm this, the plume volume was determined as described in Section 4.0, for both CO_2 and salt for both the STOMP simulation and the machine learning model at all time steps in the test dataset for each depth interval. Assuming a threshold mass fraction of 0.02, the predicted plume volumes range in size up to 10^{10} m^3 for aqueous CO_2 (Figure 18) and up to 10^9 m^3 for aqueous salt (Figure 19). The mean absolute error between the target and predicted plume volumes (Table 3) is on the order of 10^7, which is several orders of magnitude lower than the maximum plume size.

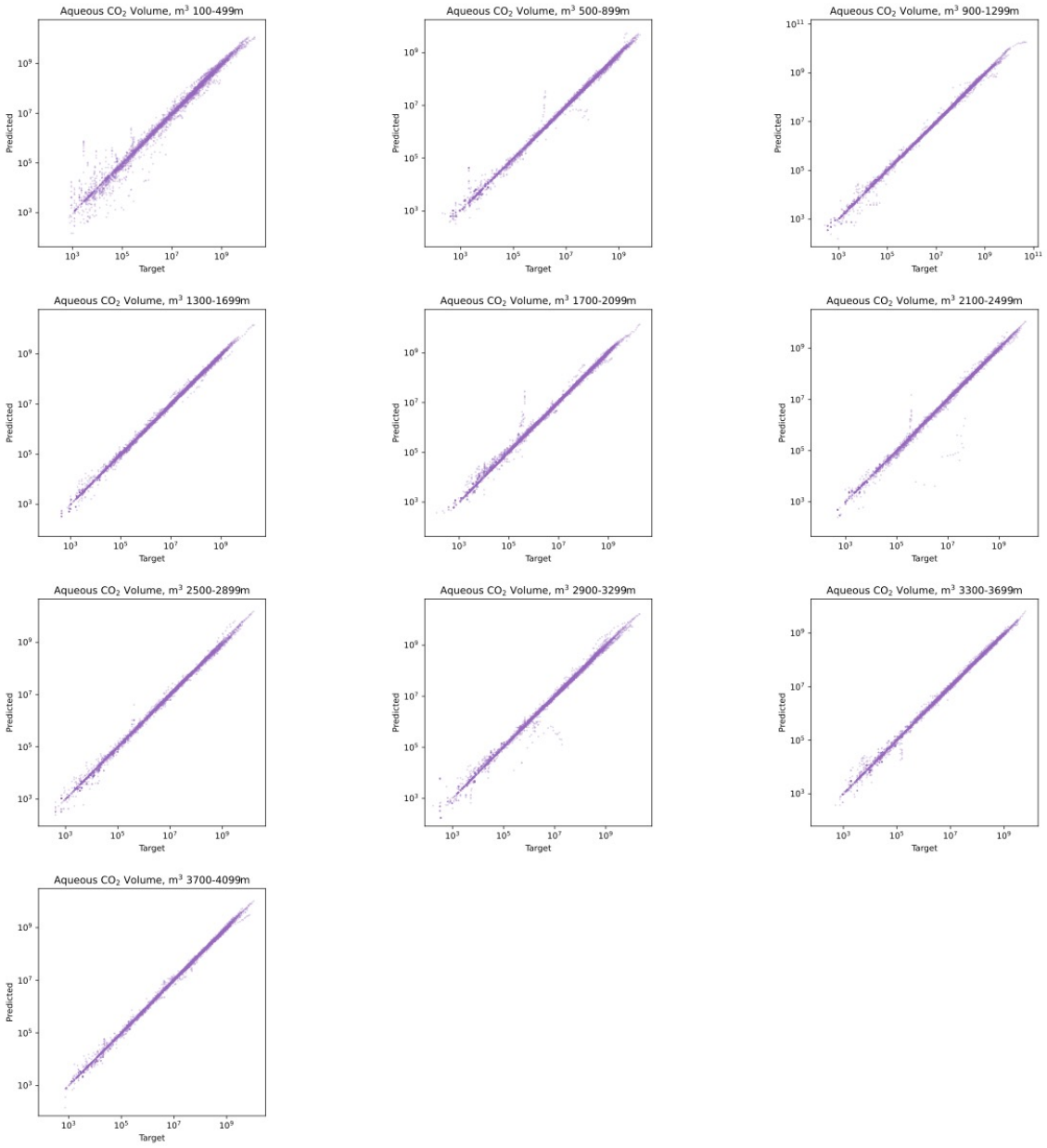

Figure 18. Comparison of target and predicted plume volume (m^3) assuming a threshold CO_2 mass fraction of 0.02.

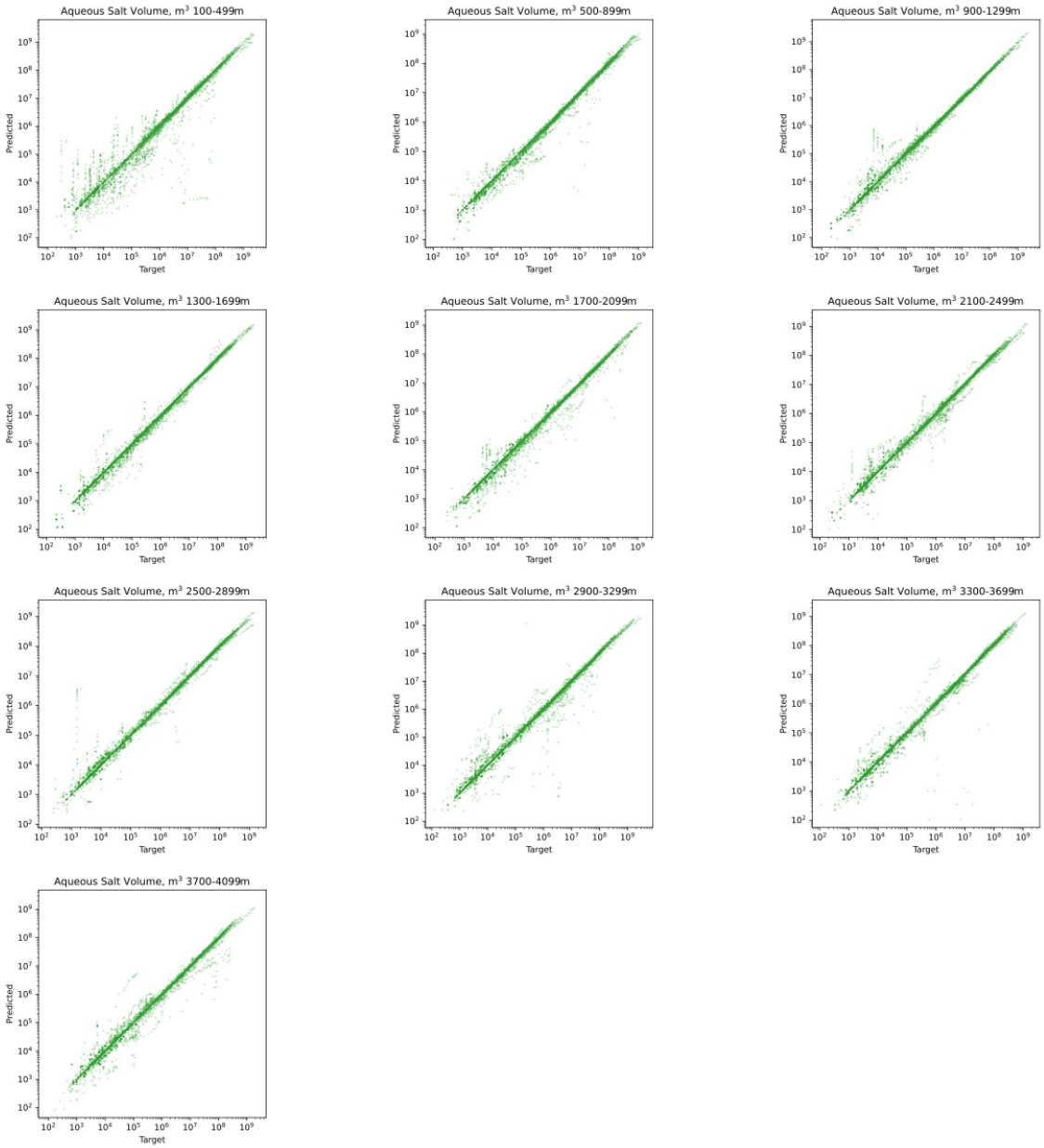

Figure 19. Comparison of target and predicted plume volume (m^3) assuming a threshold salt mass fraction of 0.02.

Table 3. Mean Absolute Error (MAE) for plume volume predictions in each depth interval.

Depth Interval	MAE, CO_2 Impact Volume (m^3)	MAE, Salt Impact Volume (m^3)
100-499m	3.00E+07	2.54E+06
500-899m	1.16E+07	1.60E+06
900-1299m	3.03E+07	1.41E+06
1300-1699m	1.07E+07	1.46E+06
1700-2099m	1.18E+07	1.28E+06
2100-2499m	9.73E+06	1.64E+06
2500-2899m	1.07E+07	1.71E+06
2900-3299m	2.18E+07	1.80E+06
3300-3699m	5.49E+06	1.54E+06
3700-4099m	1.14E+07	1.57E+06

Because the mass fractions and plume volumes have such different ranges, it is difficult to compare the accuracy between their models. For this comparison, a different error metric is used, normalized mean square error (NRMSE). The NRMSE facilitates the comparison between models with different scales, and was calculated using the following python functions:

```
def_error(actual: np.ndarray, predicted: np.ndarray):
    """ Simple error """
    return actual - predicted

def mse(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Squared Error """
    return np.mean(np.square(_error(actual, predicted)))


def rmse(actual: np.ndarray, predicted: np.ndarray):
    """ Root Mean Squared Error """
    return np.sqrt(mse(actual, predicted))


def nrmse(actual: np.ndarray, predicted: np.ndarray):
    """ Normalized Root Mean Squared Error """
    return rmse(actual, predicted) / (actual.max() - actual.min())
```

The results shown in Figure 20 show that the NRMSE for CO_2 volume predictions are 3-5 times lower than for CO_2 mass fraction predictions. The NRMSE for salt volume predictions are on the same order as the NRMSE for salt mass fractions. So in general, the accuracy of the plume predictions is either better or about the same as the pointwise mass fraction predictions.

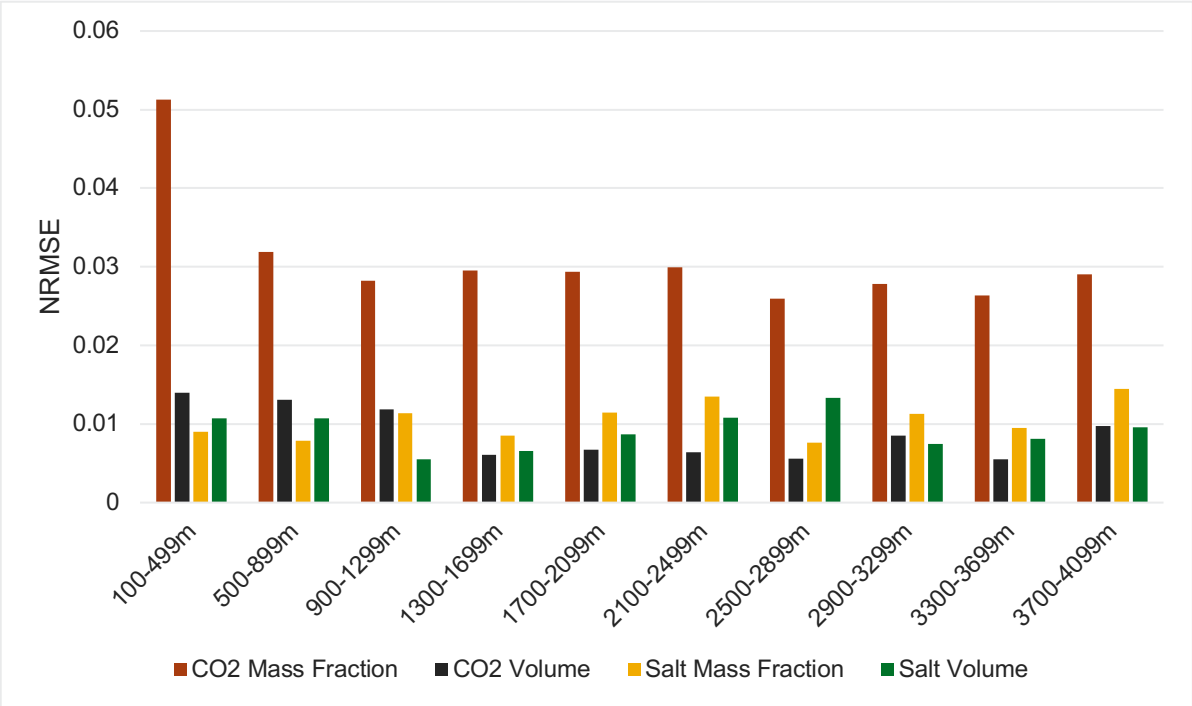

Figure 20. Normalized Root Mean Square Error comparison for all deep learning model predictions in test dataset.

8.0 Output Processing for DREAM

The Designs for Risk Estimation and Management (DREAM) tool was developed to assist in determining optimal placement of monitoring devices to detect carbon dioxide (CO_2) leakage from storage formations. DREAM optimizes across user-provided output from subsurface leakage simulations with the objective of identifying monitoring schemes that minimize time to first detection of user-specified leakage indicators. DREAM employs a simulated annealing approach that searches the solution space by iteratively mutating potential monitoring schemes built of various configurations of monitoring locations and leak detection parameters. This approach has proven to be orders of magnitude faster than an exhaustive search of the entire solution space (Yonkofski et al. 2020).

An example python script named *iam_sys_analytical_mswell_generic_lhs.py* can be found in the NRAP-Open-IAM examples/scripts folder and is listed in Appendix B. The script builds a system model consisting of an analytical reservoir component, a multisegmented wellbore component, and a generic aquifer component. Twenty-five simulations are run using Latin Hypercube Sampling. The results of these simulations are then converted into an HDF5 file format suitable for further analysis with DREAM.

The generic aquifer component generates predictions on a 100x10 cylindrical grid, centered on each well. DREAM expects simulation results to be on a 3D structured grid. A utility gridding module named enmesh.py (included with NRAP-Open-IAM) facilitates this conversion. Finally, the results for each of the 25 results are written to HDF5 files in the required format.

9.0 Conclusions

The new Generic Aquifer component of NRAP-Open-IAM calculates the concentrations of dissolved salt and dissolved CO_2 surrounding a leaking legacy well, and estimate the volume of an “impact plume” where concentration changes exceed user-specified thresholds. The input parameters were selected to cover a wide range of groundwater aquifers and leakage rates. The generic aquifer model was developed using a generative adversarial deep learning network, trained using a large synthetic dataset of STOMP multiphase flow simulations. The deep learning model predictions of dissolved salt and dissolved CO_2 in the aquifer compare well to the original STOMP simulation results. The extent of aquifer impacted by leaking CO_2 or brine is calculated using a user-defined mass fraction threshold. The aquifer impact volumes calculated based on the deep learning model compare well to those calculated based on STOMP simulation results. Gridded observation results from the generic aquifer component of NRAP-Open-IAM can be converted to HDF5 format files to use as input for monitoring design with the DREAM code.

10.0 References

Bacon DH, NP Qafoku, ZX Dai, EH Keating, and CF Brown. 2016. "Modeling the impact of carbon dioxide leakage into an unconfined, oxidizing carbonate aquifer." *International Journal of Greenhouse Gas Control* 44:290-299. 10.1016/j.ijggc.2015.04.008.

Bacon DH, CMR Yonkofski, CF Brown, DI Demirkanli, and JM Whiting. 2019. "Risk-based post injection site care and monitoring for commercial-scale carbon storage: Reevaluation of the FutureGen 2.0 site using NRAP-Open-IAM and DREAM." *International Journal of Greenhouse Gas Control* 90:102784. 10.1016/j.ijggc.2019.102784.

Bloomfield JP, MA Lewis, AJ Newell, SE Loveless, and ME Stuart. 2020. "Characterising variations in the salinity of deep groundwater systems: A case study from Great Britain (GB)." *Journal of Hydrology: Regional Studies* 28. 10.1016/j.ejrh.2020.100684.

Carroll SA, M Bianchi, K Mansoor, L Zheng, Y Sun, N Spycher, and J Birkholtzer. 2016. *Reduced-Order Model for Estimating Impacts from CO2 Storage Leakage to Alluvium Aquifers: Third-Generation, Combined Physical and Chemical Processes*. NRAP-TRS-II-009-2016, U.S. Department of Energy, National Energy Technology Laboratory, Morgantown, WV.

FutureGen Industrial Alliance I. 2013. *Underground Injection Control Permit Applications for FutureGen 2.0 Morgan County Class VI UIC Wells 1, 2, 3, and 4 – Supporting Documentation*. FG-RPT-017-Revision 1, Jacksonville, Illinois.

Iman RL, JC Helton, and JE Campbell. 1981. "An Approach to Sensitivity Analysis of Computer Models: Part I—Introduction, Input Variable Selection and Preliminary Variable Assessment." *Journal of Quality Technology* 13(3):174-183. 10.1080/00224065.1981.11978748.

Keating E, D Bacon, S Carroll, K Mansoor, YW Sun, LE Zheng, D Harp, and ZX Dai. 2016. "Applicability of aquifer impact models to support decisions at CO2 sequestration sites." *International Journal of Greenhouse Gas Control* 52:319-330. 10.1016/j.ijggc.2016.07.001.

NETL. 2015. *Carbon Storage Atlas, 5th Edition*. National Energy Technology Laboratory, Pittsburgh, PA.

NETL. 2018. Carbon Storage Research, *11/1*.

Pawar RJ, GS Bromhal, SP Chu, RM Dilmore, CM Oldenburg, PH Stauffer, YQ Zhang, and GD Guthrie. 2016. "The National Risk Assessment Partnership's integrated assessment model for carbon storage: A tool to support decision making amidst uncertainty." *International Journal of Greenhouse Gas Control* 52:175-189. 10.1016/j.ijggc.2016.06.015.

Pedregosa F, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. 2011. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research* 12:2825-2830.

Romanak KD, RC Smyth, C Yang, SD Hovorka, M Rearick, and J Lu. 2012. "Sensitivity of groundwater systems to CO2: Application of a site-specific analysis of carbonate monitoring parameters at the SACROC CO2-enhanced oil field." *International Journal of Greenhouse Gas Control* 6:142-152. 10.1016/j.ijggc.2011.10.011.

Ronneberger O, P Fischer, and T Brox. 2015. "U-Net: Convolutional Networks for Biomedical Image Segmentation." *Medical Image Computing and Computer-Assisted Intervention, Pt Iii* 9351:234-241. 10.1007/978-3-319-24574-4_28.

USEPA. 2010. Federal Requirements Under the Underground Injection Control (UIC) Program for Carbon Dioxide (CO2) Geologic Sequestration (GS) Wells. EP Agency. 237. Federal Register.

Vasylkivska V, R Dilmore, G Lackey, YQ Zhang, S King, D Bacon, BL Chen, K Mansoor, and D Harp. 2021. “NRAP-open-IAM: A flexible open-source integrated-assessment-model for geologic carbon storage risk assessment and management.” *Environmental Modelling & Software* 143. 10.1016/j.envsoft.2021.105114.

Vaught TL. 1980. *Assessment of the geothermal resources of Illinois based on existing geologic data*. United States. 10.2172/6773972

Yonkofski CM, JM Whiting, BZ Huang, and AC Hanna. 2020. *Designs for Risk Evaluation and Management (DREAM) Tool User’s Manual, Version: 2020.01-2.0*. NRAP-TRS-III-001-2020, National Energy Technology Laboratory, Morgantown, WV.

Appendix A – STOMP Input

A.1 STOMP Input File Template

```
~Simulation Title Card
1,
Generic Aquifers,
D.H. Bacon,
Pacific Northwest National Laboratory,
03 Mar 2021,
11:11 AM PDT,
1,
CO2 and brine leakage into aquifers

~Solution Control Card
Normal,
STOMP-CO2E w/isothermal,
2,
0,s,100,yr,0.001,s,0.01,yr,1.05,16,1.e-06,
100,yr,170,yr,0.001,s,0.001,yr,1.05,16,1.e-06,
999999,
Variable Aqueous Diffusion,
Variable Gas Diffusion,
0,

~Grid Card
Cylindrical,
100,1,10,
0.00,m, 3.24,m, 6.73,m, 10.49,m, 14.56,m, 18.94,m, 23.67,m, 28.77,m, 34.27,m,
40.20,m, 46.60,m, 53.51,m, 60.96,m, 69.00,m, 77.67,m, 87.02,m, 97.11,m,
107.99,m, 119.73,m, 132.40,m, 146.06,m, 160.80,m, 176.70,m, 193.85,m, 212.35,m,
232.31,m, 253.84,m, 277.07,m, 302.12,m, 329.15,m, 358.31,m, 389.76,m, 423.69,m,
460.29,m, 499.77,m, 542.37,m, 588.31,m, 637.88,m, 691.35,m, 749.03,m, 811.25,m,
878.37,m, 950.77,m, 1028.88,m, 1113.14,m, 1204.04,m, 1302.09,m, 1407.87,m,
1521.97,m, 1645.06,m, 1777.84,m, 1921.08,m, 2075.60,m, 2242.29,m, 2422.11,m,
2616.08,m, 2825.33,m, 3051.06,m, 3294.57,m, 3557.25,m, 3840.61,m, 4146.29,m,
4476.05,m, 4831.77,m, 5215.50,m, 5629.45,m, 6076.00,m, 6557.72,m, 7077.37,m,
7637.94,m, 8242.66,m, 8895.00,m, 9598.71,m, 10357.83,m, 11176.74,m, 12060.13,m,
13013.09,m, 14041.10,m, 15150.06,m, 16346.35,m, 17636.85,m, 19028.97,m,
20530.72,m, 22150.73,m, 23898.32,m, 25783.53,m, 27817.19,m, 30011.01,m,
32377.58,m, 34930.52,m, 37684.51,m, 40655.36,m, 43860.17,m, 47317.36,m,
51046.79,m, 55069.92,m, 59409.86,m, 64091.57,m, 69141.96,m, 74590.06,m,
80467.20,m,
0.0,deg,1,deg,
?bottom,m,?dz,m,

~Rock/Soil Zonation Card
1,
Aquifer,1,100,1,1,1,10,
```

~Mechanical Properties Card
Aquifer, 2650, kg/m^3, ?por, ?por, 1e-06, 1/m, Millington and Quirk,

~Hydraulic Properties Card
Aquifer,?permh, m^2, ?permh, m^2, ?permv, m^2,

~Saturation Function Card
Aquifer,Brooks and Corey w/ Entrapment, ?psi,, ?lambda, ?srw, 0.2,

~Aqueous Relative Permeability Card
Aquifer,Burdine,,

~Gas Relative Permeability Card
Aquifer,Burdine,,

~Salt Transport Card
Aquifer, 1.0, ft, 0.1, ft,

~Solute/Porous Media Interactions Card
Aquifer, 1.0, ft, 0.1, ft,

~Thermal Properties Card
Aquifer,Parallel,2.38,W/m K,2.38,W/m K,2.38,W/m K,930,J/kg K,

~Species Link Card
2,
H+,pH,
Total_CO2(aq),Aqueous CO2,

~Initial Conditions Card
Hydrostatic,32.0,MPa,-3169.5,m,23,C,0,m,-0.0217,C/m,?aquifer_salinity,-2700.0,m,0,1/m,

~Boundary Conditions Card
3,
East,Aqueous Unit Gradient,Gas Initial Condition,Salt Outflow,
100,100,1,1,1,10,1,
0,s,,,,,,,,,,,,,,,,,,
Bottom,Aqueous Zero Flux,Gas Zero Flux,Salt Zero Flux,
1,100,1,1,1,1,1,
0,s,,,,,,,,,,,,,,,,,,
Top,Aqueous Zero Flux,Gas Zero Flux,Salt Zero Flux,
1,100,1,1,10,10,1,
0,s,,,,,,,,,,,,,,,,,,

~Source Card
2,
CO2 Leak
Gas Mass Source,Water Relative Humidity,1,1,1,1,2,9,1,
100,yr,?source_pressure,MPa,?co2_rate,kg/s,0.0,,,,,,
Brine Leak

Aqueous Mass Source, Dissolved Salt Mass Fraction, NULL,1,1,1,1,2,9,1,
100,yr,?source_pressure,MPa,?brine_rate,kg/s,?reservoir_salinity,,,,,,

~Surface Flux Card
2,
Aqueous Mass Flux,kg/s,kg,East,100,100,1,1,1,10,
Gas CO2 Flux,kg/s,kg,East,100,100,1,1,1,10,

~Output Options Card
1,
1,1,5,
1,1,year,m,deg,6,6,6,
8,
Integrated CO2 Mass,kg,
Integrated CO2 Aqueous,kg,
Integrated CO2 Gas,kg,
Integrated CO2 Mass Source,kg,
CO2 Aqueous Mass Fraction,,
Gas Pressure,MPa,
Temperature,C,
Aqueous Salt Mass Fraction,,
17,
100,yr,
101,yr,
102,yr,
105,yr,
110,yr,
115,yr,
120,yr,
125,yr,
130,yr,
135,yr,
140,yr,
145,yr,
150,yr,
155,yr,
160,yr,
165,yr,
170,yr,
23,
X Node Centroid,m,
Y Node Centroid,m,
Z Node Centroid,m,
Rock/Soil type,,
Diffusive Porosity,,
x-intrinsic permeability,m^2,
z-intrinsic permeability,m^2,
Gas Saturation,,
Temperature,C,
Aqueous Pressure,MPa,
Gas Pressure,MPa,

CO2 Aqueous Mass Fraction,,
Aqueous Salt Mass Fraction,,
Aqueous Density,kg/m^3,
xnc Aqueous Vol,m/yr,
ync Aqueous Vol,m/yr,
znc Aqueous Vol,m/yr,
CO2 Mass Source Int,kg,
H2O Mass Source Int,kg,
CO2 Mass Source Rate,kg/s,
H2O Mass Source Rate,kg/s,
Salt Mass Source Int,kg,
Salt Mass Source Rate,kg/s,

A.2 Python Script to Generate STOMP Input Files

```
import glob
import numpy as np
import os
import string
import re
import argparse
import pandas as pd
from sklearn.model_selection import train_test_split

def get_trailing_number(s):
    m = re.search(r'\d+$', s)
    return int(m.group()) if m else None

def make_run_dirs(df, template_dir, out_dir):
    input_list = []

    def substitute_parameters(parametername,parametervalue):
        findstring = '?' + parametername
        for m,line in enumerate(input_list):
            if findstring in line:
                line_list = [x.strip() for x in line.split(',')]
                for n,i in enumerate(line_list):
                    if findstring in i:
                        line_list[n]=str(parametervalue)
                input_list[m] = ",".join(line_list)

    # input permeability in m^2
    # output Brooks Corey parameters psi, lambda and residual saturation
    def rel_perm(permeability):
        if permeability < 4.06e-14:
            return (4.116,0.83113,0.059705)
        elif permeability >= 4.06e-14 and permeability < 2.28e-13:
            return (1.573,0.62146,0.081005)
        elif permeability >= 2.28e-13 and permeability < 9.01E-13:
            return (1.45,1.1663,0.070762)
        else:
```

```
            return (1.008,1.3532,0.044002)

    cmd = "mkdir -p " + out_dir
    os.system(cmd)

    for i,sample in df.iterrows():

        # copy the template folder
        dir = out_dir + "/run" + str(i+1)
        print('Making directory ', dir)
        cmd = "rm -rf " + dir
        os.system(cmd)
        cmd = "cp -r " + template_dir + " " + dir
        os.system(cmd)

        # read input template into a list
        template = dir + "/input.template"
        with open(template, 'r') as file_path:
            input_list = file_path.readlines()
        input_list = [x.strip() for x in input_list]
        file_path.close()

        # substitute parameters
        layers = 10
        dz = str(layers) + '@' + str(sample['thick']/layers)
        substitute_parameters('dz',dz)
        #substitute_parameters('depth',sampl['depth'])
        bottom = sample['depth'] - sample['thick']
        substitute_parameters('bottom',bottom)
        #salinity = -2.50e-05 * sample['depth'] - 1.11e-02
        #substitute_parameters('salinity',salinity)
        por = sample['por']
        substitute_parameters('por',por)
        permh = 10.**sample['log_permh']
        substitute_parameters('permh',permh)
        permv = permh / 10.**sample['log_aniso']
        substitute_parameters('permv',permv)
        bc_psi, bc_lambda, bc_srw = rel_perm(permh)
        substitute_parameters('psi',bc_psi)
        substitute_parameters('lambda',bc_lambda)
        substitute_parameters('srw',bc_srw)
        co2_rate = 10**sample['log_co2_rate']
        substitute_parameters('co2_rate',co2_rate/(layers-2)/360) # 1 degree
        brine_rate = 10**sample['log_brine_rate']
        substitute_parameters('brine_rate',brine_rate/(layers-2)/360)
        source_pressure = 1-sample['depth']*1000*9.81/1E+6 # hydrostatic
pressure
        substitute_parameters('source_pressure',source_pressure)
        aquifer_salinity = sample['aquifer_salinity']
        substitute_parameters('aquifer_salinity',aquifer_salinity)
        reservoir_salinity = sample['reservoir_salinity']
        substitute_parameters('reservoir_salinity',reservoir_salinity)

        # print out the new input file
        inputfile = dir + '/input'
```

```
        thefile = open(inputfile, 'w')
        for line in input_list:
            print(line, file=thefile)

if __name__ == "__main__":

    # input arguments
    parser = argparse.ArgumentParser(description='Make simulation
directories')
    parser.add_argument('--sample', help='sample file to read')
    parser.add_argument('--template', help='template folder to copy')
    args = parser.parse_args()

    df = pd.read_csv(args.sample, delim_whitespace=True)

    y = range(1,len(df)+1)

    df_train, df_rem, y_train, y_rem = train_test_split(
        df, y, train_size=0.8, random_state=42)
    df_val, df_test, y_val, y_test = train_test_split(
        df_rem, y_rem, test_size=0.5, random_state=42)

    base=os.path.basename(args.sample)
    out_dir = os.path.splitext(base)[0]

    make_run_dirs(df_train, args.template, out_dir+'/train')
    make_run_dirs(df_val, args.template, out_dir+'/valid')
    make_run_dirs(df_test, args.template, out_dir+'/test')
```

Appendix B – Example NRAP-Open-IAM script generating DREAM Files

```
'''
This example couples the simple reservoir, multisegmented wellbore and
generic aquifer models. The saturation/pressure output produced by simple
reservoir model is used to drive leakage from a single multisegmented
wellbore
model, which is passed to the input of an adapter that provides well
coordinates, |CO2| and brine leakage rates and cumulative mass fluxes to the
generic aquifer model. HDF5 files for input to DREAM are created.

Example of run:
$ python iam_sys_reservoir_mswell_generic_lhs.py
'''

import sys,os
sys.path.insert(0,os.sep.join(['..','..','source']))
import numpy as np
from openiam import SystemModel, AnalyticalReservoir
from openiam import MultisegmentedWellbore, GenericAquifer, RateToMassAdapter
import matplotlib.pyplot as plt
from scipy.interpolate import RegularGridInterpolator
import openiam.enmesh as en
import h5py

if __name__ == "__main__":
    # For multiprocessing in Spyder
    __spec__ = None
    # Define keyword arguments of the system model
    num_years = 10
    time_array = 365.25*np.arange(0.0,num_years+1)
    sm_model_kwargs = {'time_array': time_array}   # time is given in days

    # Create system model
    sm = SystemModel(model_kwargs=sm_model_kwargs)

    # legacy well location
    xloc = 200
    yloc = 200

    # Add reservoir component
    ares = sm.add_component_model_object(AnalyticalReservoir(name='ares',
parent=sm,
        injX=0., injY=0., locX=xloc, locY=yloc))

    # Add parameters of reservoir component model
    ares.add_par('numberOfShaleLayers', value=3, vary=False)
    ares.add_par('shale1Thickness', value=100.0, vary=False)
    ares.add_par('aquifer1Thickness', value=100.0, vary=False)
    ares.add_par('shale2Thickness', value=100.0, vary=False)
    ares.add_par('aquifer2Thickness', value=100.0, vary=False)
    ares.add_par('shale3Thickness', value=500.0, vary=False)
```

```
    ares.add_par('injRate', value=1.0, vary=False)

    # Add observations of reservoir component model
    ares.add_obs_to_be_linked('pressure')
    ares.add_obs_to_be_linked('CO2saturation')
    ares.add_obs('pressure')
    ares.add_obs('CO2saturation')
    ares.add_obs('mass_CO2_reservoir')

    # Add multisegmented wellbore component
    ms =
sm.add_component_model_object(MultisegmentedWellbore(name='ms',parent=sm))
    ms.add_par('logWellPerm', min=-14.0, max=-12.0, value=-13.0)

    # Add linked parameters: common to reservoir and wellbore components
    ms.add_par_linked_to_par('numberOfShaleLayers',
ares.deterministic_pars['numberOfShaleLayers'])
    ms.add_par_linked_to_par('shale1Thickness',
ares.deterministic_pars['shale1Thickness'])
    ms.add_par_linked_to_par('shale2Thickness',
ares.deterministic_pars['shale2Thickness'])
    ms.add_par_linked_to_par('shale3Thickness',
ares.deterministic_pars['shale3Thickness'])
    ms.add_par_linked_to_par('aquifer1Thickness',
ares.deterministic_pars['aquifer1Thickness'])
    ms.add_par_linked_to_par('aquifer2Thickness',
ares.deterministic_pars['aquifer2Thickness'])
    ms.add_par_linked_to_par('reservoirThickness',
ares.default_pars['reservoirThickness'])
    ms.add_par_linked_to_par('datumPressure',
ares.default_pars['datumPressure'])

    # Add keyword arguments linked to the output provided by reservoir model
    ms.add_kwarg_linked_to_obs('pressure', ares.linkobs['pressure'])
    ms.add_kwarg_linked_to_obs('CO2saturation',
ares.linkobs['CO2saturation'])

    # Add observations of multisegmented wellbore component model
    ms.add_obs_to_be_linked('CO2_aquifer1')
    ms.add_obs_to_be_linked('CO2_aquifer2')
    ms.add_obs_to_be_linked('brine_aquifer1')
    ms.add_obs_to_be_linked('brine_aquifer2')
    ms.add_obs_to_be_linked('mass_CO2_aquifer1')
    ms.add_obs_to_be_linked('mass_CO2_aquifer2')
    ms.add_obs_to_be_linked('brine_atm')
    ms.add_obs_to_be_linked('CO2_atm')
    ms.add_obs('brine_aquifer1')
    ms.add_obs('CO2_aquifer1')

    # Add adapter that transforms leakage rates to accumulated mass
    adapt =
sm.add_component_model_object(RateToMassAdapter(name='adapt',parent=sm))
    adapt.add_kwarg_linked_to_collection('CO2_aquifer1',
        [ms.linkobs['CO2_aquifer1'], ms.linkobs['CO2_aquifer2']])
    adapt.add_kwarg_linked_to_collection('CO2_aquifer2',
```

```
        [ms.linkobs['CO2_aquifer2'], ms.linkobs['CO2_atm']])
    adapt.add_kwarg_linked_to_collection('brine_aquifer1',
        [ms.linkobs['brine_aquifer1'], ms.linkobs['brine_aquifer2']])
    adapt.add_kwarg_linked_to_collection('brine_aquifer2',
        [ms.linkobs['brine_aquifer2'], ms.linkobs['brine_atm']])
    adapt.add_obs_to_be_linked('mass_CO2_aquifer1')
    adapt.add_obs_to_be_linked('mass_CO2_aquifer2')
    adapt.add_obs_to_be_linked('mass_brine_aquifer1')
    adapt.add_obs_to_be_linked('mass_brine_aquifer2')
    adapt.add_obs('mass_CO2_aquifer1')
    adapt.add_obs('mass_brine_aquifer1')
    adapt.add_obs('mass_CO2_aquifer2')
    adapt.add_obs('mass_brine_aquifer2')

    # Add generic aquifer model object and define parameters
    ga = sm.add_component_model_object(GenericAquifer(name='ga',parent=sm))
    ga.add_par_linked_to_par('aqu_thick',
ares.deterministic_pars['aquifer1Thickness'])
    # ga.add_composite_par('depth',
    #     expr='ares.shale2Thickness + ares.shale3Thickness' +
    #     '+ ares.aquifer2Thickness')
    ga.add_composite_par('depth',
expr=ares.deterministic_pars['shale2Thickness'].name+
        '+'+ares.deterministic_pars['shale3Thickness'].name+
        '+'+ares.deterministic_pars['aquifer2Thickness'].name)
    ga.add_par('por', value=1.9652592824538797 63e-01, vary=False)
    ga.add_par('log_permh', value=-1.19146451590516555 5e+01, vary=False)
    ga.add_par('log_aniso', value=8.046003470121247947e-01, vary=False)
    ga.add_par('aquifer_salinity', value=1.267995018132549341e-02,
vary=False)
    ga.add_par('reservoir_salinity', value=4.159677791928499679e-02,
vary=False)
    ga.add_par('dissolved_salt_threshold', value=0.015, vary=False)
    ga.add_par('dissolved_co2_threshold', value=0.001, vary=False)

    ga.add_kwarg_linked_to_obs('co2_mass',
adapt.linkobs['mass_CO2_aquifer1'])
    ga.add_kwarg_linked_to_obs('brine_mass',
adapt.linkobs['mass_brine_aquifer1'])

    # Add observations (output) from the generic aquifer model
    ga.add_obs('dissolved_salt_volume')
    ga.add_obs('dissolved_co2_volume')

    # Define output folder to keep data files with gridded observations
    output_dir = 'dream_data'

    # Add gridded observations of the aquifer component
    ga.add_grid_obs('r_coordinate', constr_type='matrix',
output_dir=output_dir)
    ga.add_grid_obs('z_coordinate', constr_type='matrix',
output_dir=output_dir)
    ga.add_grid_obs('dissolved_co2_mass_fraction', constr_type='matrix',
output_dir=output_dir)
    ga.add_grid_obs('dissolved_salt_mass_fraction', constr_type='matrix',
```

```
output_dir=output_dir)

    print('-----------------------------------------------------------------
')
    print('                        UQ illustration ')
    print('-----------------------------------------------------------------
')

    import random
    num_samples = 25
    ncpus = 1
    # Draw Latin hypercube samples of parameter values
    s = sm.lhs(siz=num_samples,seed=random.randint(500,1100))

    # Run model using values in samples for parameter values
    results = s.run(cpus=ncpus,verbose=False)

    print('-----------------------------------------------------------------
')
    print('                  Write DREAM File ')
    print('-----------------------------------------------------------------
')

    # set boundaries of site grid
    top = ares.deterministic_pars['aquifer2Thickness'].value +
ares.deterministic_pars['shale2Thickness'].value +
ares.deterministic_pars['shale3Thickness'].value
    bottom = top + ares.deterministic_pars['aquifer1Thickness'].value
    grid = en.Grid(xmin=0, xmax=1000, ymin=0, ymax=1000, zmin=top,
zmax=bottom)

    # set constant grid spacing for vertical axis
    grid.get_axis('z').add_zone(min=grid.zmin,
max=grid.zmax).add_ticks(min_spacing=10., mul=0)

    # add wells
    grid.add_well(x=xloc, y=yloc, name='Well 1')

    # refine grid around wells
    grid.refine_grid_around_wells(min_x_spacing=5, min_y_spacing=5,
x_mul=1.5, y_mul=1.5)

    # get 3D mesh with radial distance from well
    xx,yy,zz = grid.get_vertices()
    cx,cy,cz = grid.get_centroids()
    cr = grid.get_radial_distance(xloc, yloc, cx, cy)
    positions = np.vstack(list(zip(cr.ravel(), cz.ravel())))

    # Read Gridded Observation Coordinates
    rcoord =
np.load(os.path.join(output_dir,'ga_r_coordinate_sim_1_time_0.npz'))['data']
    zcoord =
np.load(os.path.join(output_dir,'ga_z_coordinate_sim_1_time_0.npz'))['data']
    r_coord = rcoord[:,0]
    z_coord = np.flip(zcoord[0,:])
```

```
    # start simulation loop
    for sim in np.arange(1, num_samples+1):

        years = time_array/365.25
        steps = years.astype(int)

        # Open DREAM file
        hdf5 =
h5py.File(os.path.join(output_dir,'ga_sim_'+str(sim)+'.h5'),'w')

        # write grid and geologic data
        g1=hdf5.create_group('data')
        g1.create_dataset('porosity',
data=ga.deterministic_pars['por'].value*np.ones_like(cx),dtype='float32')
        g1.create_dataset('steps',    data=steps,dtype='float32')
        g1.create_dataset('times',    data=time_array,dtype='float32')
        g1.create_dataset('vertex-x',
data=np.array(xx)[:,0,0],dtype='float32')
        g1.create_dataset('vertex-y',
data=np.array(yy)[0,:,0],dtype='float32')
        g1.create_dataset('vertex-z',
data=np.array(zz)[0,0,:],dtype='float32')
        g1.create_dataset('x',
data=np.array(cx)[:,0,0],dtype='float32')
        g1.create_dataset('y',
data=np.array(cy)[0,:,0],dtype='float32')
        g1.create_dataset('z',
data=np.array(cz)[0,0,:],dtype='float32')
        g1['x'].attrs['units'] = 'm'
        g1['y'].attrs['units'] = 'm'
        g1['z'].attrs['units'] = 'm'
        g1['vertex-x'].attrs['units'] = 'm'
        g1['vertex-y'].attrs['units'] = 'm'
        g1['vertex-z'].attrs['units'] = 'm'
        g1['z'].attrs['positive'] = 'down'
        g1['vertex-z'].attrs['positive'] = 'down'

        # write gridded observations
        def write_obs(name, unit, positions, shape):

            def snake_to_camel(name):
                temp = name.split('_')
                res = ''.join(ele.title() for ele in temp)
                return res

            name2 = snake_to_camel(name)
            unit2 = snake_to_camel(unit)

            means = []
            mins = []
            maxs = []

            for step in steps:
```

```
                # read gridded observation files
                obs =
np.load(os.path.join(output_dir,'ga_'+name+'_'+unit+'_sim_'+str(sim)+'_time_'
+str(step)+'.npz'))['data']

                # Interpolate Gridded Observations to Site Grid
                interpolator = RegularGridInterpolator((r_coord, z_coord),
obs, bounds_error=False)
                interpolated = interpolator(positions)
                reshaped = interpolated.reshape(shape)

                means.append(np.mean(reshaped))
                mins.append(np.min(reshaped))
                maxs.append(np.max(reshaped))


                g2=hdf5.require_group('plot%i'%step)
                g2.create_dataset(name2,data=reshaped,dtype='float32')

                g2[name2].attrs['unit'] = unit2

            # calculate min,mean,max over all time steps
            g3=hdf5.require_group('statistics')
            g3.create_dataset(name2, data=np.array([
np.min(np.array(mins)),np.mean(np.array(means)),np.max(np.array(maxs))
]),dtype='float32')

        write_obs('dissolved_co2','mass_fraction',positions,cx.shape)
        write_obs('dissolved_salt','mass_fraction',positions,cx.shape)

        hdf5.close()

        # end sim loop
```

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

www.pnnl.gov