IMPERIAL

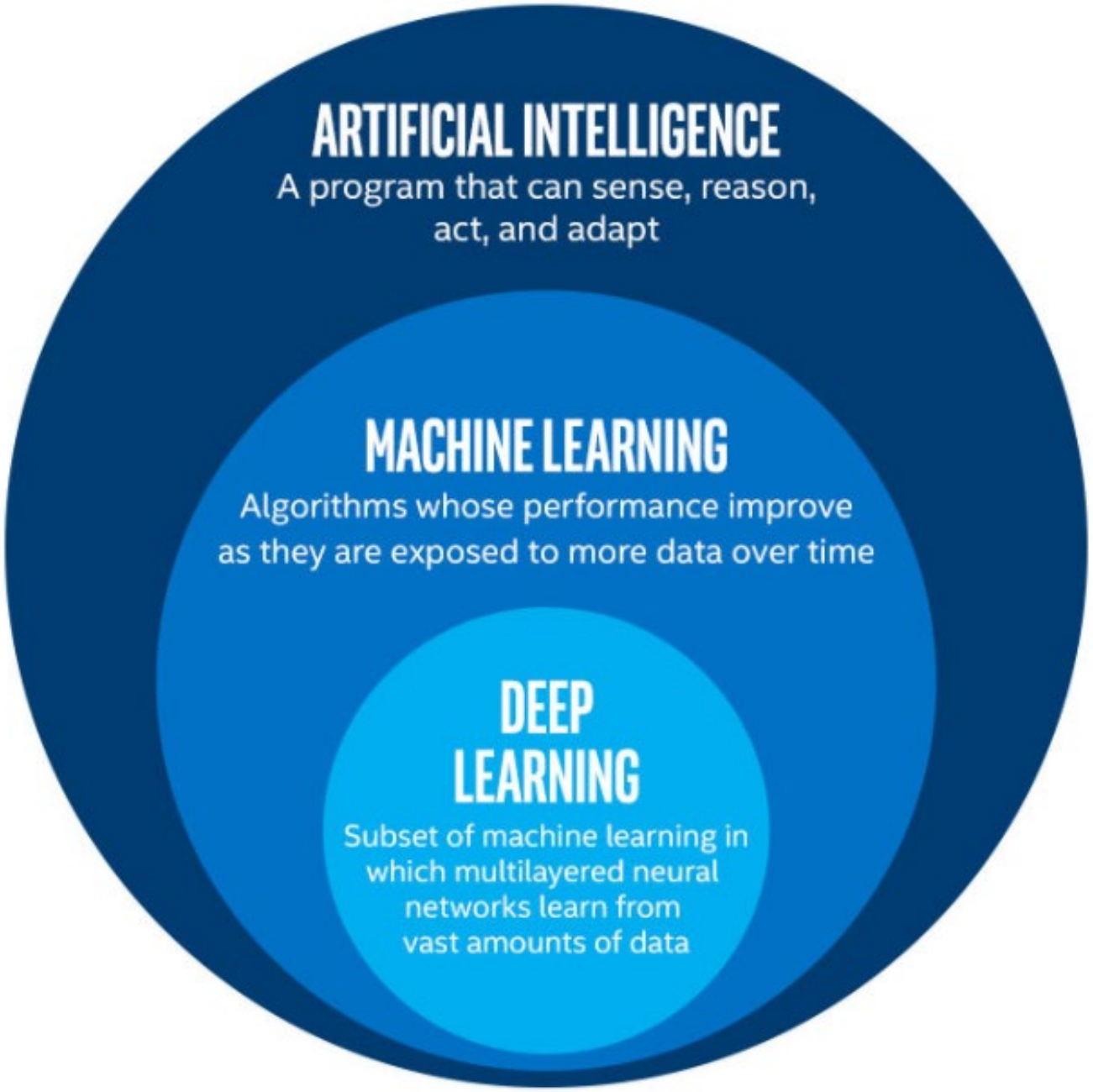
Physics-informed Machine Learning for Solving Forward and Inverse 3D Multi-layered Consolidation Problems

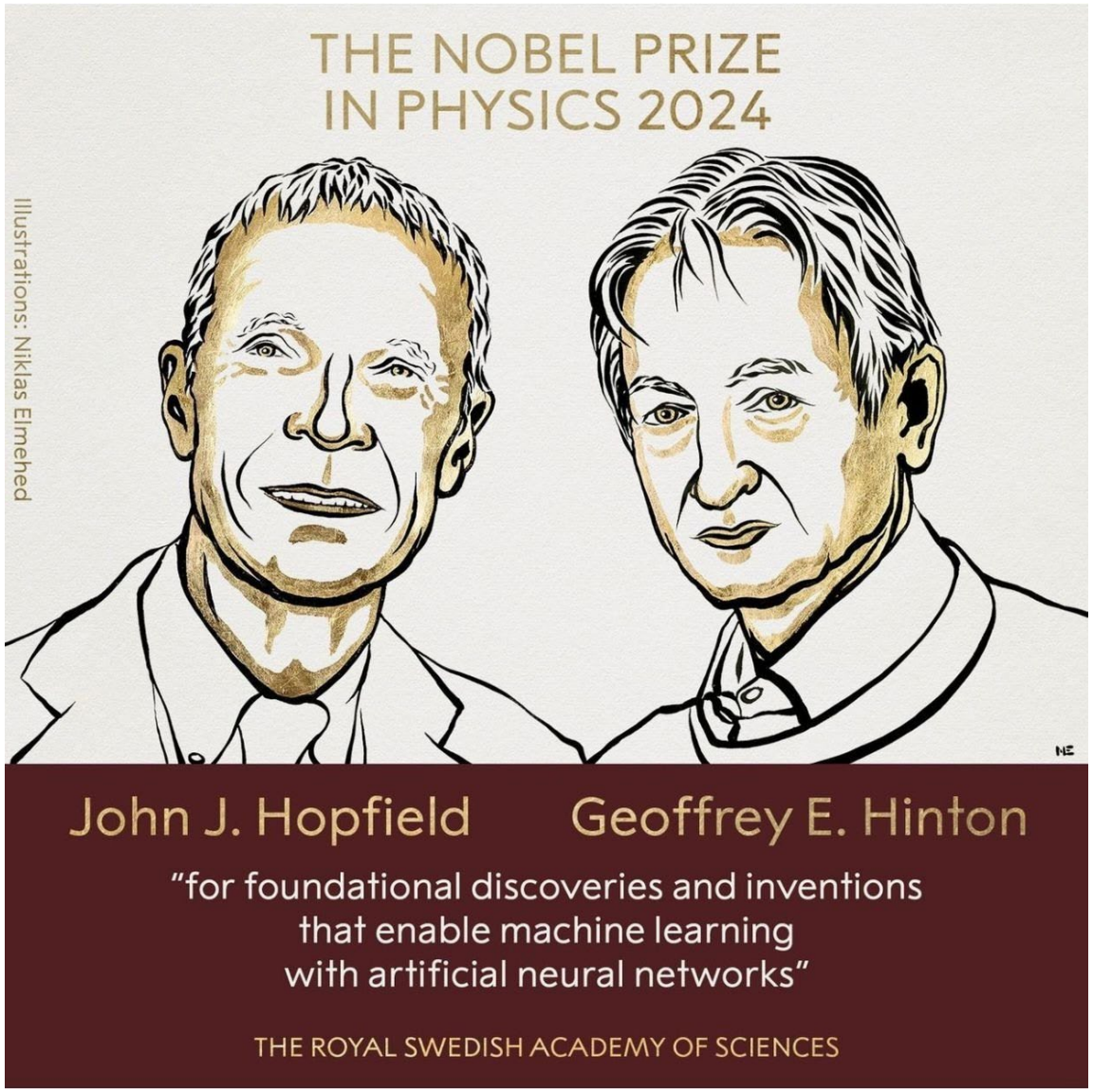
QUIRK London University Physics Conference

Umar Siddique (UCL)

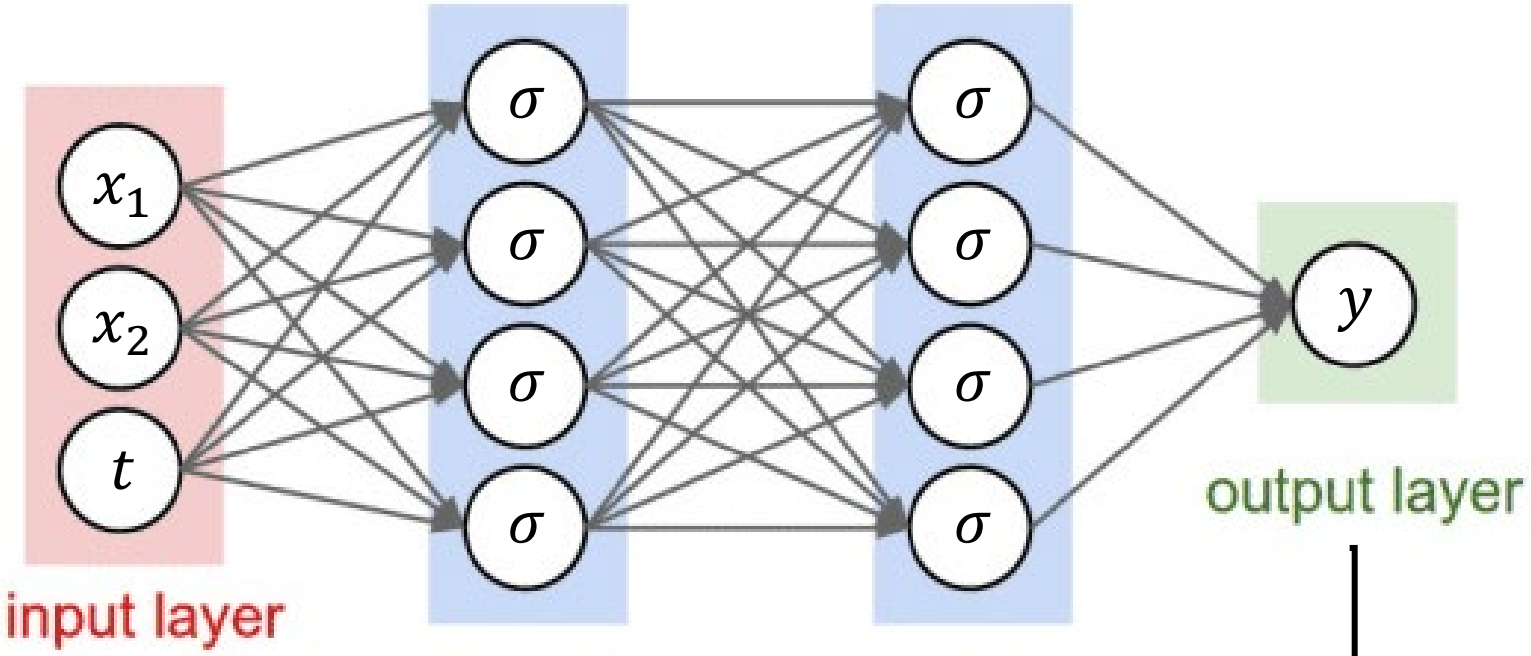
15/02/2025

Physics-informed Deep Learning


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data


THE NOBEL PRIZE IN PHYSICS 2024
Illustrations: Niklas Elmehed
John J. Hopfield
Geoffrey E. Hinton
"for foundational discoveries and inventions that enable machine learning with artificial neural networks"
THE ROYAL SWEDISH ACADEMY OF SCIENCES

Neural Network
Feed Forward

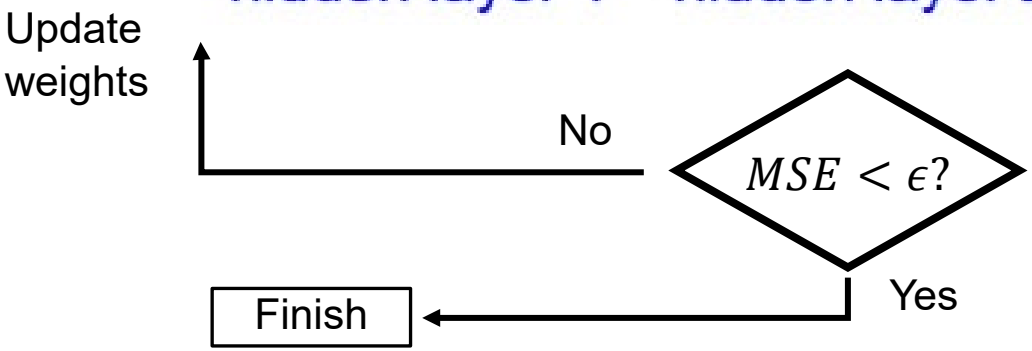


The feed-forward neural network is constructed and trained using input data. The model will train the **weights** by minimising the **loss function**

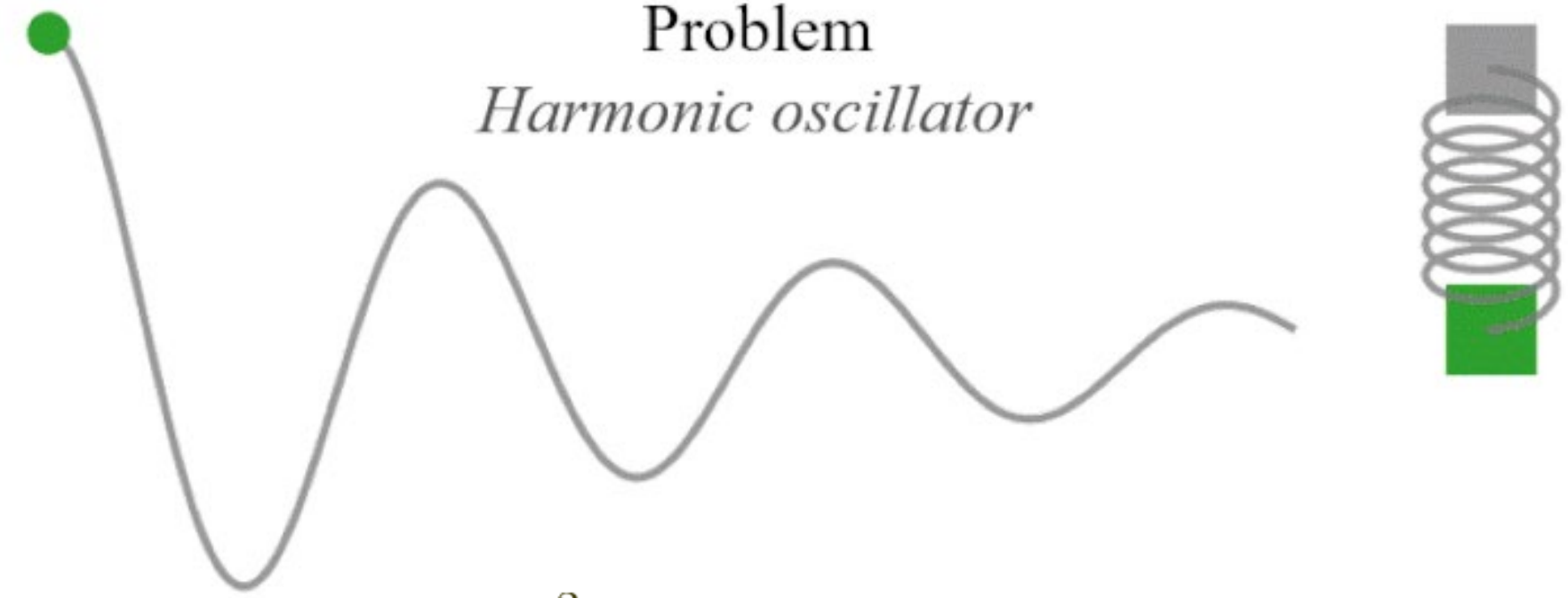
Neurons within these layers are interconnected, adjusting weights and biases through training to improve accuracy

Calculate the Loss

$$MSE = \frac{1}{n}\sum_{i}(y_i - \hat{y}_i)^2$$



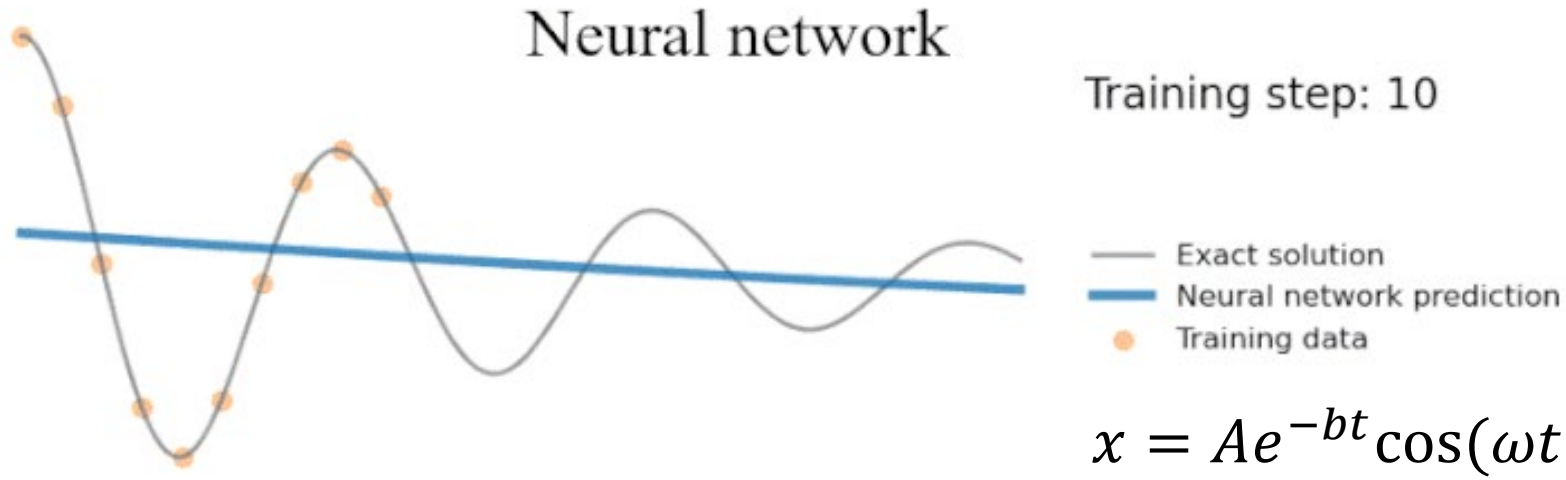
Harmonic Oscillator



$$m\frac{d^2u}{dx^2} + \mu\frac{du}{dx} + ku = 0$$

Source: https://benmoseley.blog/

Harmonic Oscillator



$$x = Ae^{-bt}\cos(\omega t + \phi)$$

$$m\frac{d^2u}{dx^2} + \mu\frac{du}{dx} + ku = 0$$

Forward Problem

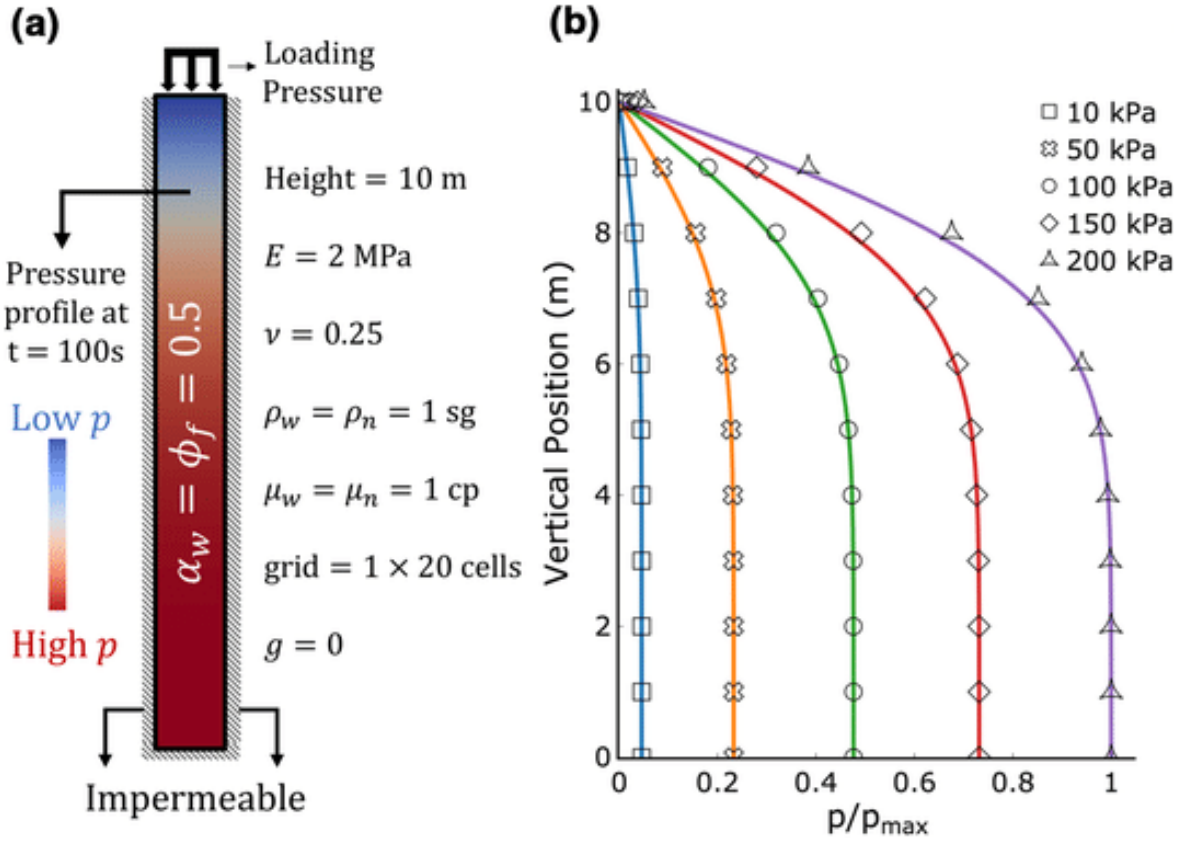
Developing a surrogate model to solve 3D consolidation problems

The Terzaghi Problem in 1D

Karl von Terzaghi introduced the idea in a series of papers in the **1920s** based on his examination of building consolidation on soil.

The problem is **one-dimensional**, the only gradient being in the **vertical direction.**

The purpose of the analysis is to predict the evolution of displacement, effective stress, and **pore pressure** throughout the soil as a function of time **following the load application.**



Pore pressure refers to the **pressure exerted by fluids** within the tiny spaces **("pores")** between soil particles or rock grains

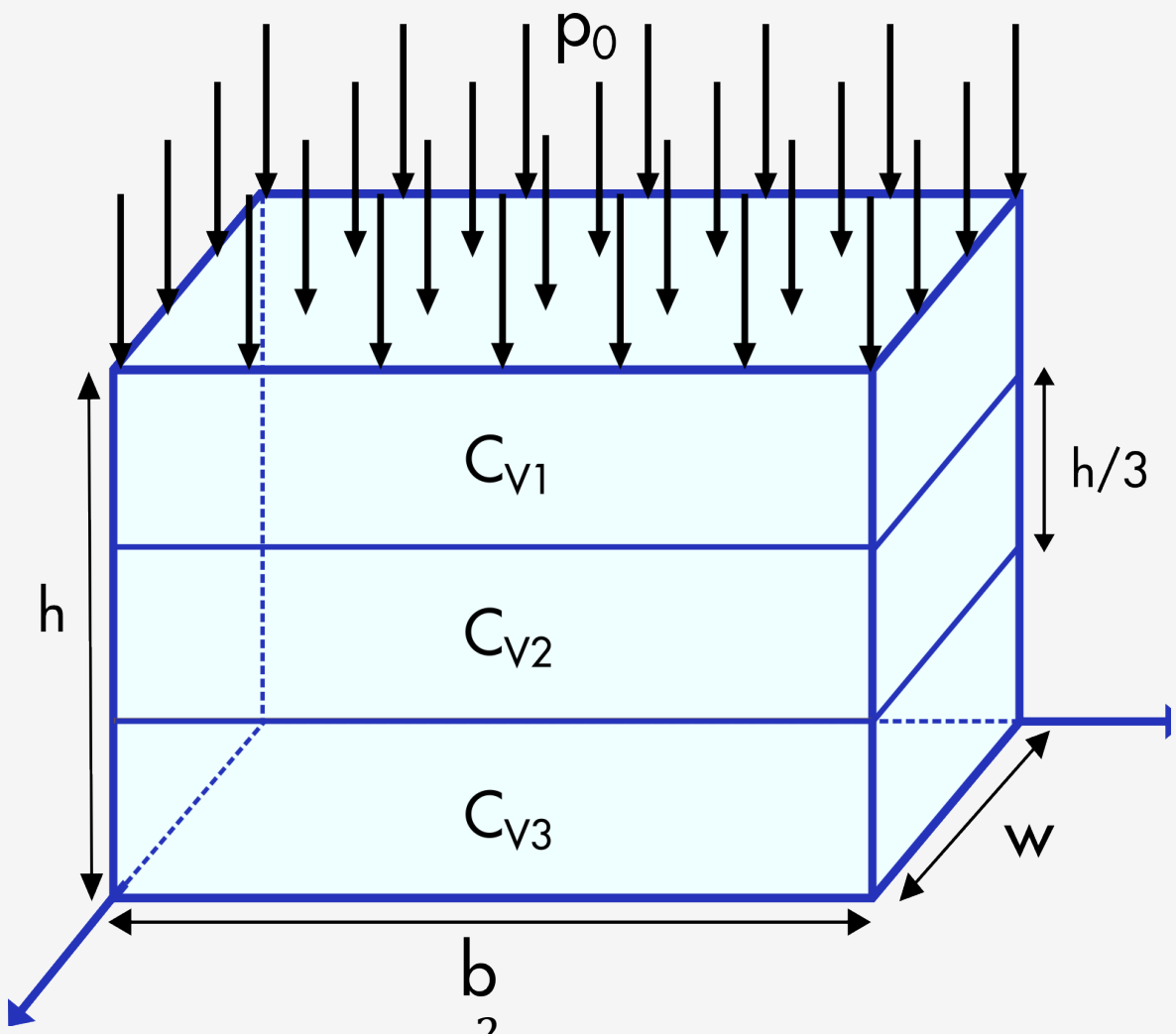
The Terzaghi Problem in 3D

Extension to the 1D Problem

Rendulic expanded the one-dimensional consolidation theory to two and three dimensions, introducing the Terzaghi–Rendulic theory.

This theory assumes that during consolidation under constant external loads, **the sum of normal stresses at any point in the soil remains constant.**

Consequently, the consolidation problem becomes analogous to the thermal diffusion problem, with its mathematical representation referred to as the **diffusion equation.**



$$c_{v1} \quad \text{for } \frac{2}{3} < z < 1,$$

$$c_{v2} \quad \text{for } \frac{1}{3} < z < \frac{2}{3},$$

$$c_{v3} \quad \text{for } 0 < z < \frac{1}{3}$$

The Terzaghi Equation in 3D

$$\frac{\partial u}{\partial t} = c_v\left(\frac{\partial^2 u}{\partial x^2} + \cancel{\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}}\right)$$

Solving the Terzaghi PDE

Finite Element Method (FEM)

Lattice Boltzmann Method

Fourier Series (1D Only)

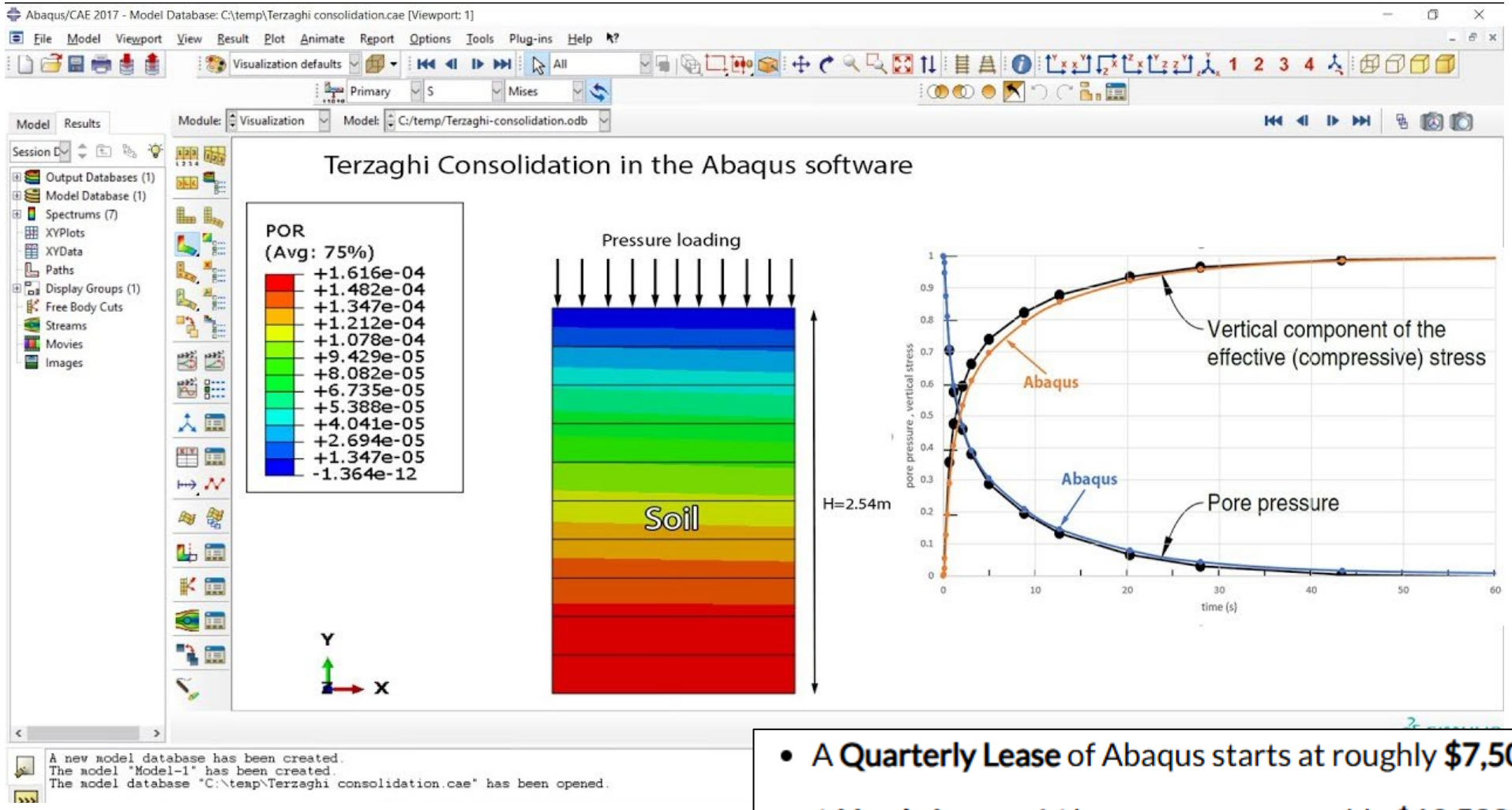
$$\frac{\partial u}{\partial t} = c_v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

Finite Volume Method (FVM)

Smoothed Particle Hydrodynamics

Finite Difference Method (FDM)

Abaqus



- A **Quarterly Lease** of Abaqus starts at roughly **$7,500**
- A **Yearly Lease** of Abaqus starts at roughly **$19,500**

Central-Order Difference Method

$$\frac{\partial u}{\partial t} = c_v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

$$\frac{\partial u}{\partial t} = \frac{u(x,y,z,t+\Delta t) - u(x,y,z,t)}{\Delta t}$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x+\Delta x,y,z,t) - 2u(x,y,z,t) + u(x-\Delta x,y,z,t)}{\Delta x^2}$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{u(x,y+\Delta y,z,t) - 2u(x,y,z,t) + u(x,y-\Delta y,z,t)}{\Delta y^2}$$

$$\frac{\partial^2 u}{\partial z^2} = \frac{u(x,y,z+\Delta z,t) - 2u(x,y,z,t) + u(x,y,z-\Delta z,t)}{\Delta z^2}$$

Central-Order Difference Method

$$\frac{\partial u}{\partial t} = c_v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

$$\frac{u(x,y,z,t+\Delta t) - u(x,y,z,t)}{\Delta t} = C_v\left(\frac{u(x+\Delta x,y,z,t) - 2u(x,y,z,t) + u(x-\Delta x,y,z,t)}{\Delta x^2}\right.$$
$$\left.\frac{u(x,y+\Delta y,z,t) - 2u(x,y,z,t) + u(x,y-\Delta y,z,t)}{\Delta y^2} + \frac{u(x,y,z+\Delta z,t) - 2u(x,y,z,t) + u(x,y,z-\Delta z,t)}{\Delta z^2}\right)$$

$$u(x,y,z,t+\Delta t) = u(x,y,z,t) + C_v\Delta t\left(\frac{u(x+\Delta x,y,z,t) - 2u(x,y,z,t) + u(x-\Delta x,y,z,t)}{\Delta x^2}\right.$$
$$\left.\frac{u(x,y+\Delta y,z,t) - 2u(x,y,z,t) + u(x,y-\Delta y,z,t)}{\Delta y^2} + \frac{u(x,y,z+\Delta z,t) - 2u(x,y,z,t) + u(x,y,z-\Delta z,t)}{\Delta z^2}\right)$$

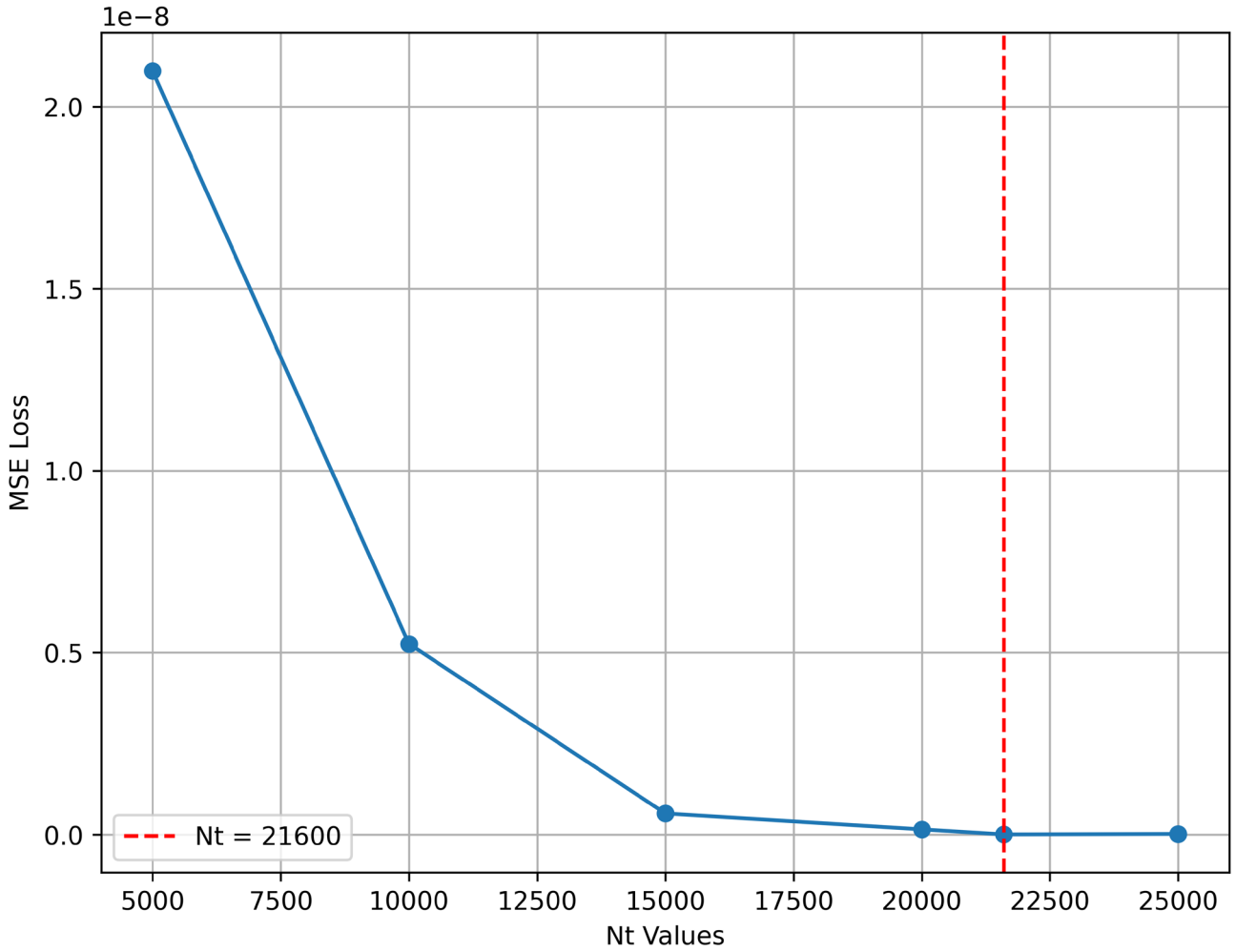
Computing Exact Solution

```
N = 60
Nt = 21600
Delta_x, Delta_y, Delta_z = 1/N, 1/N, 1/N
Delta_t = 1/Nt

x_range = np.linspace(0, 1, N)
y_range = np.linspace(0, 1, N)
z_range = np.linspace(0, 1, N)
t_range = np.linspace(0, 1, Nt)

# Time-stepping loop
for n in range(0, len(t_range) - 1):
    for i in range(1, len(x_range) - 1):
        for j in range(1, len(y_range) - 1):
            for k in range(1, len(z_range) - 1):
                cv = get_cv(z_range[k])
                u[i, j, k, n + 1] = u[i, j, k, n] + Delta_t * (
                    cv * ((u[i + 1, j, k, n] - 2 * u[i, j, k, n] + u[i - 1, j, k, n]) / Delta_x**2 +
                          (u[i, j + 1, k, n] - 2 * u[i, j, k, n] + u[i, j - 1, k, n]) / Delta_y**2 +
                          (u[i, j, k + 1, n] - 2 * u[i, j, k, n] + u[i, j, k - 1, n]) / Delta_z**2)
                )
```

The previous iterative formula is used to compute an exact solution (with boundary conditions) for a mesh grid of **60 x 60 x 60 points** from **0 to 1m.**

A large N_t value means $\Delta t = \frac{1}{N_t}$ is **negligible**, such that the solution **converges** to an exact solution, i.e. $u(x, y, z, t + \Delta t) \rightarrow u(x, y, z, t)$

Convergence Condition

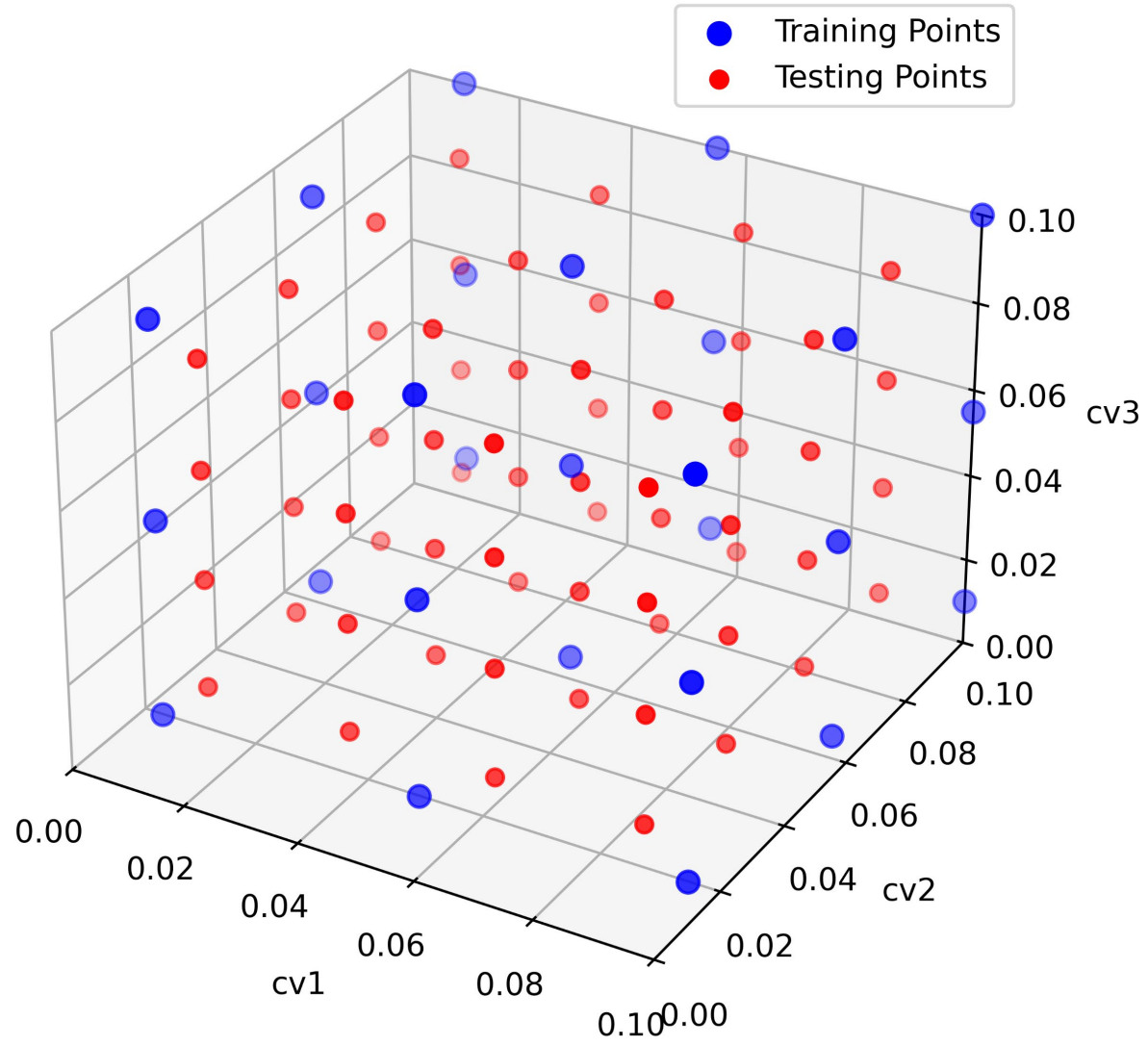


- The convergence condition ensures that as the number of time steps increase, the numerical solution **approaches the true solution of the differential equation.**

- We determine $N_t = \mathbf{21600}$ to be enough time steps to **achieve an 'exact' solution.**

- This ensures that the loss associated from the data constraint is **negligible.**

What data do we use for the model?



Training Region:

$$c_v \in [0.01, 0.055, 0.10]$$

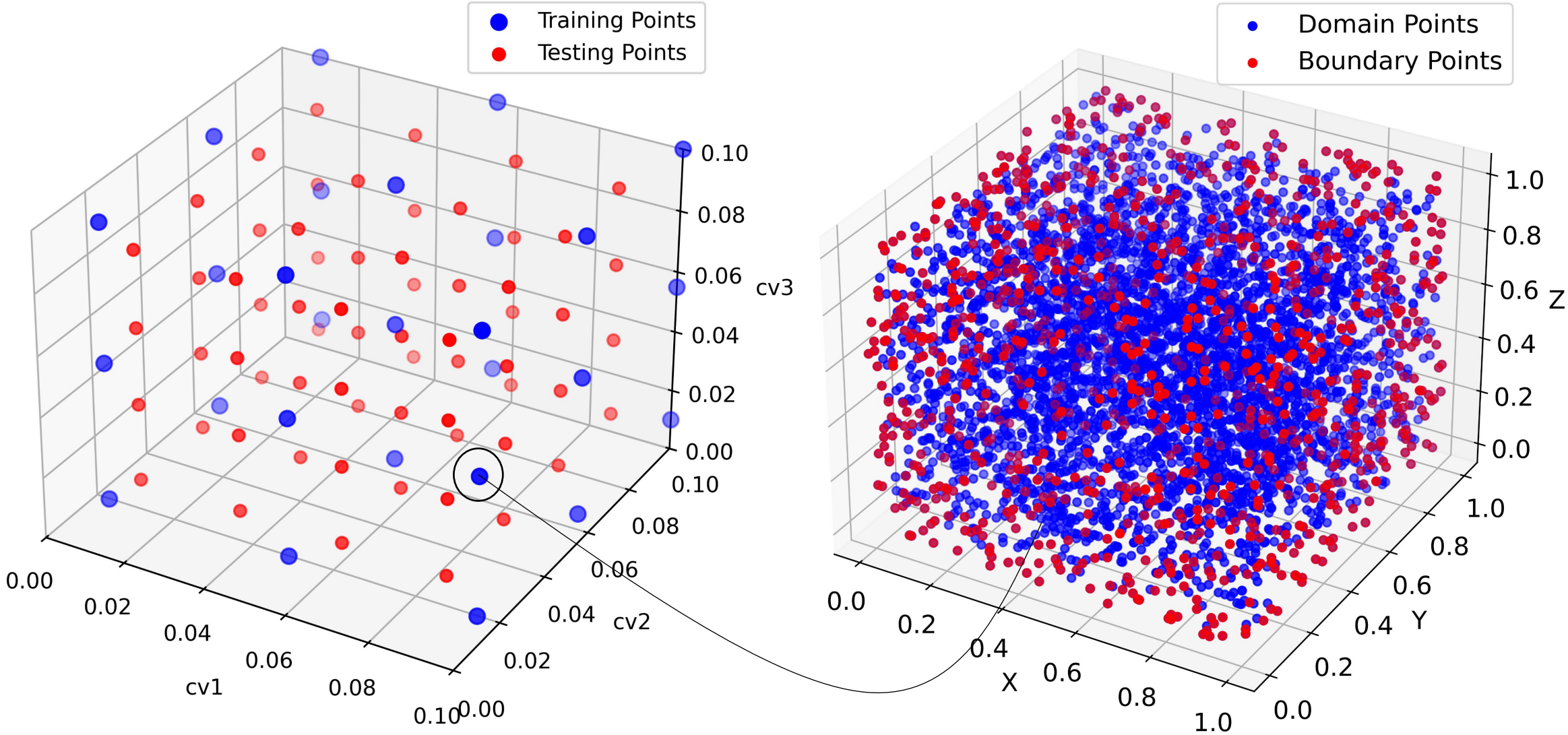
27 combinations

Testing Region:

$$c_v \in [0.015, 0.4, 0.065, 0.09]$$

64 combinations

What data do we use for the model?



What data do we use for the model?

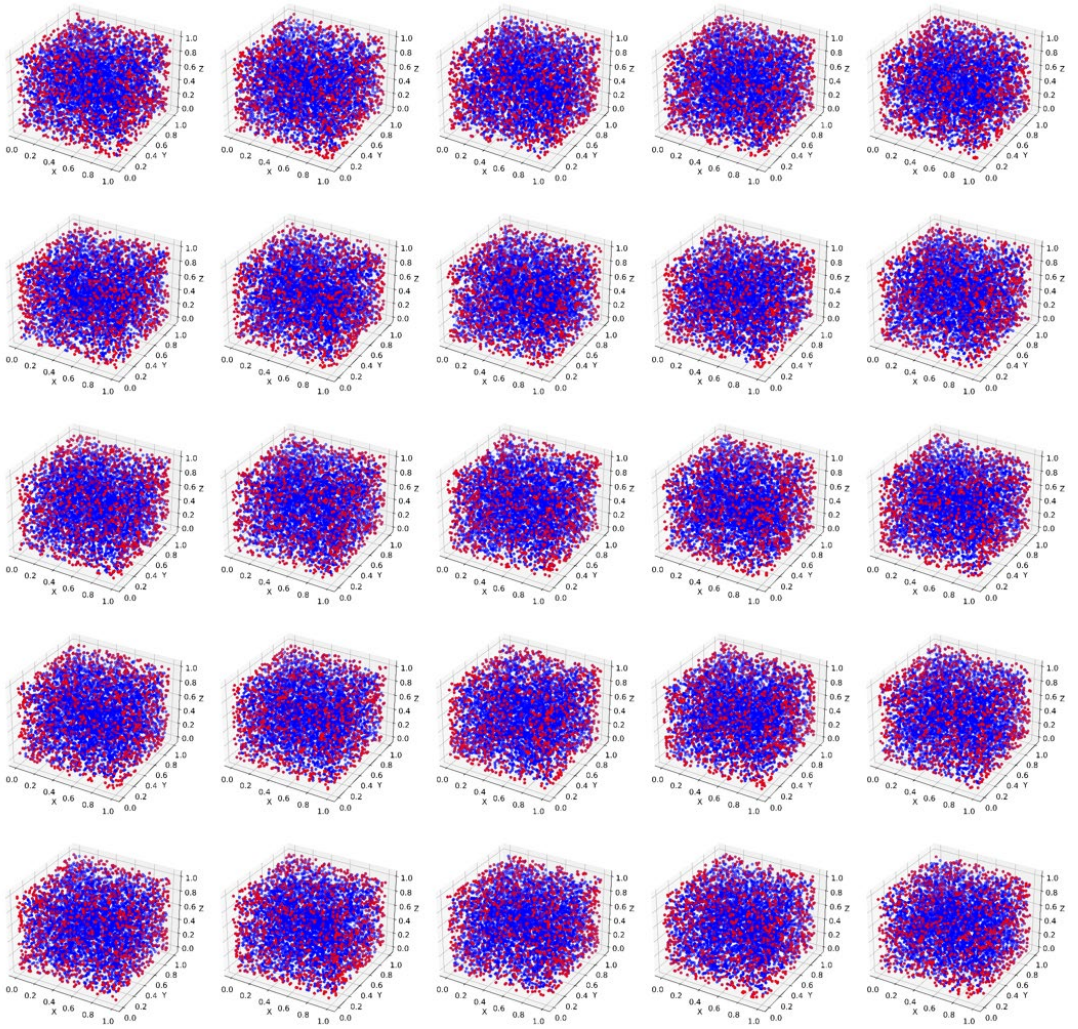
Each exact solution is computed in approximately **6.3 hours.** The file size of each solution is **36.45 GB.**

Each solution has ≈ 4.7 billion samples

$$60 \times 60 \times 60 \times 21600 = 4665600000$$

Each solution is then randomly sampled with **5000 points**, with a **higher concentration (20%)** of points selected at the boundary conditions.

This allows the model to correctly minimize the loss in both the domain and boundary regions.

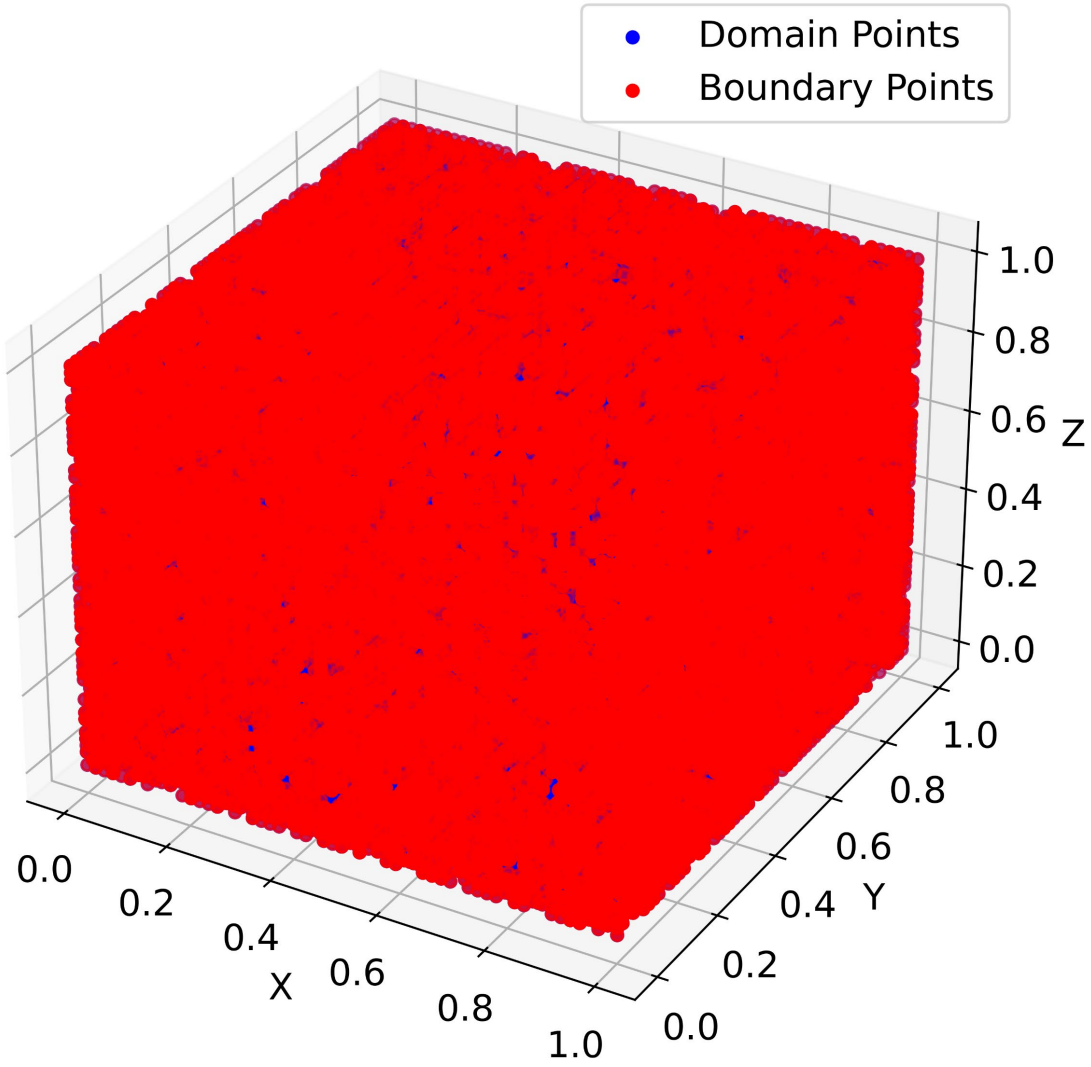


What data do we use for the model?

The **27** exact solutions are then **combined** to a single input data array which is then fed into the model for training.

The training data is therefore reduced from a total file size of **984 GB** to **8,440 KB**. The final training data has $27 \times 5000 = 135{,}000$ sample points.

The combined training data will fill **99.9%** of the domain space.



Boundary Conditions

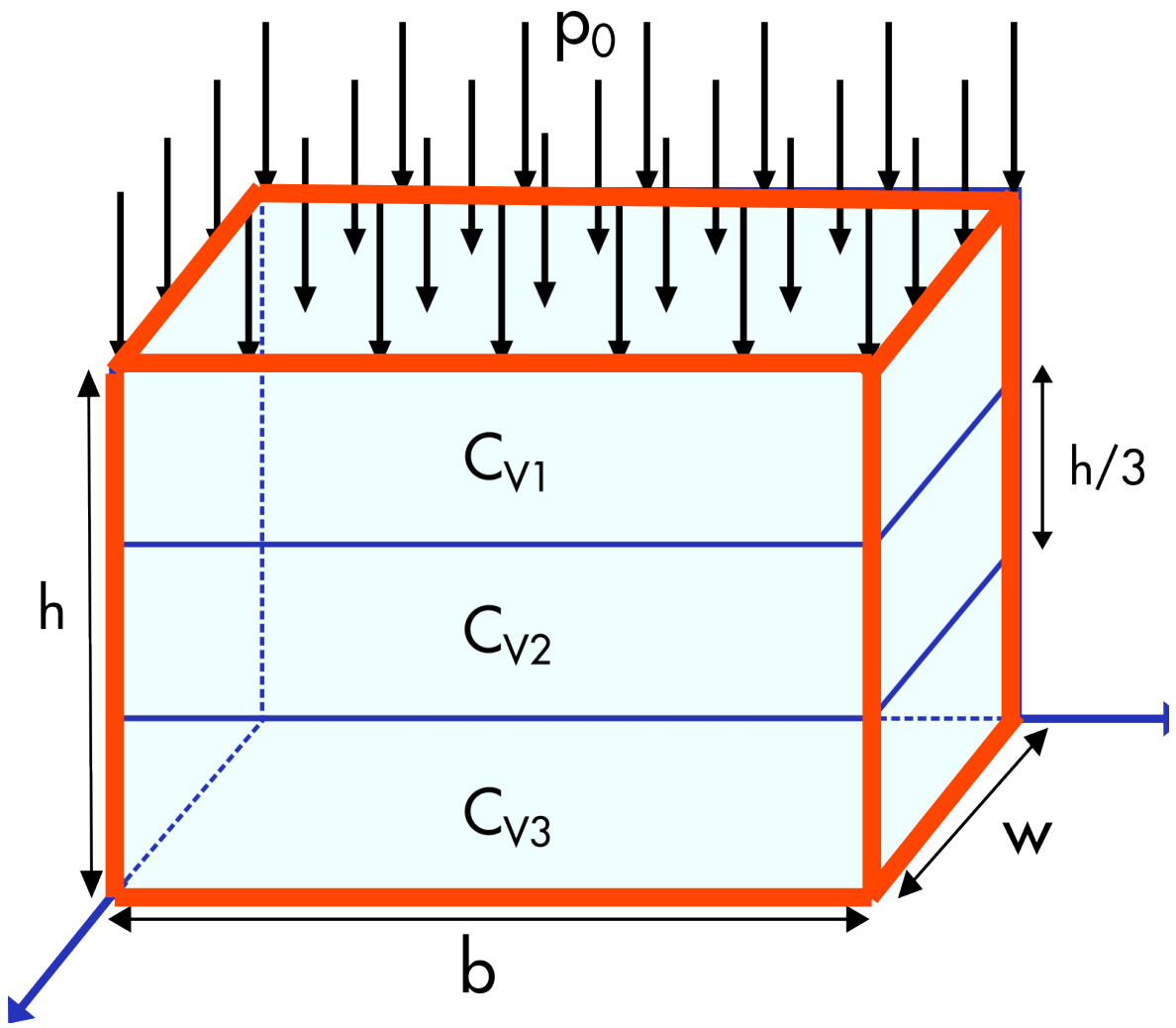
Double Drainage

$$u(x, y, z, 0) = p_0 \quad (1.0)$$

$$u(0, y, z, t) = u(b, y, z, t) = 0$$

$$u(x, 0, z, t) = u(x, w, z, t) = 0$$

$$\frac{\partial u(x, y, 0, t)}{\partial z} = \frac{\partial u(x, y, h, t)}{\partial z} = 0$$



Interface Conditions

Continuity of Flux

$$p_{top} = p(z_{int} + \epsilon) \; ; \quad p_{bot} = p(z_{int} - \epsilon)$$

Via Darcy's Law,

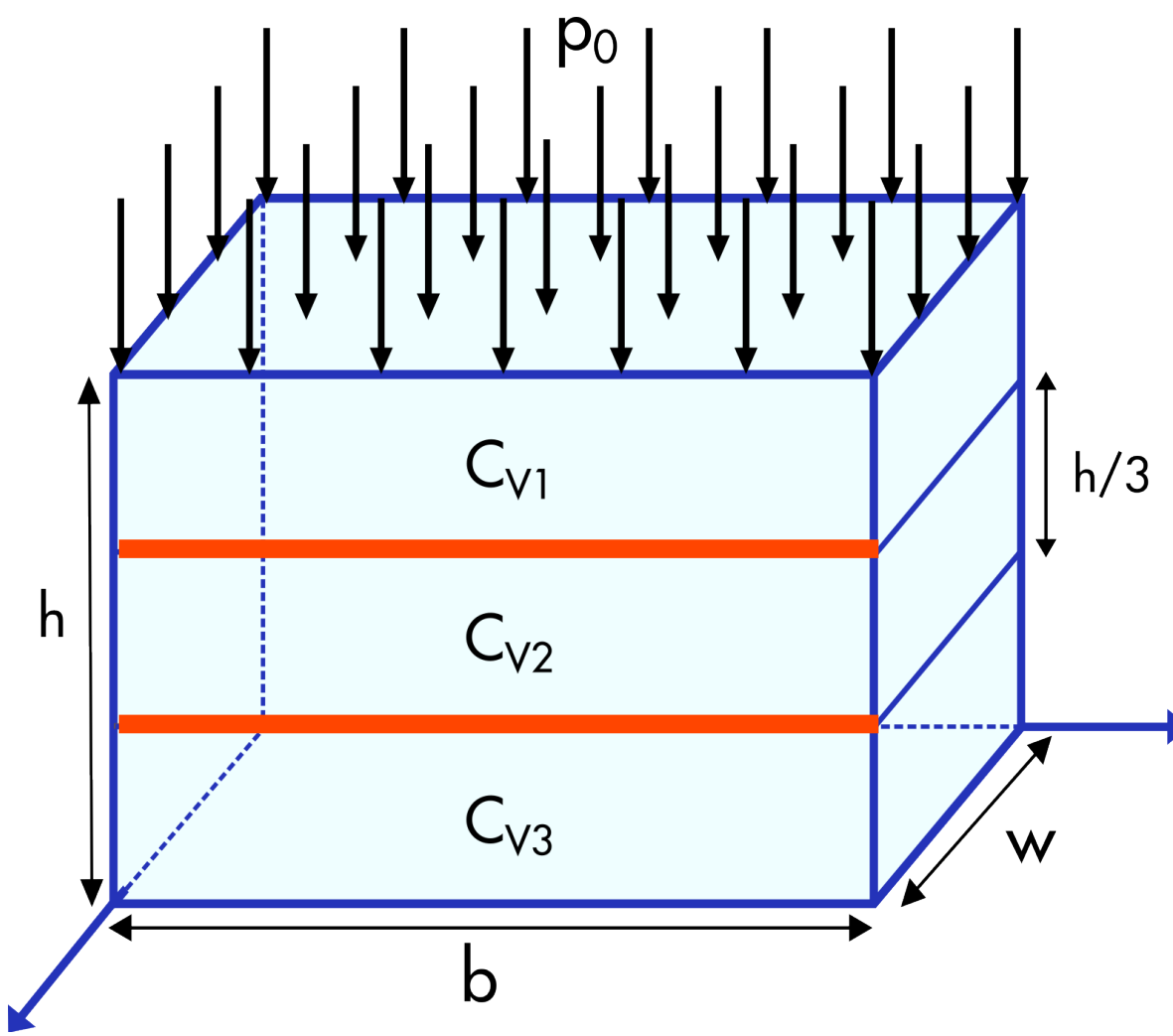
$$v = ki = k\frac{dP}{dz}$$

v: flow rate $[s^{-1}]$

k: soil permeability $[ms^{-1}]$

$$c_v = \frac{k}{m_v \gamma_w} \; , \; [c_v] = m^2 s^{-1}$$

$$\therefore k = c_v m_v \gamma_w \propto c_v$$



Interface Conditions

Continuity of Flux

$$\therefore v = ki \propto c_v \frac{dP}{dz}$$

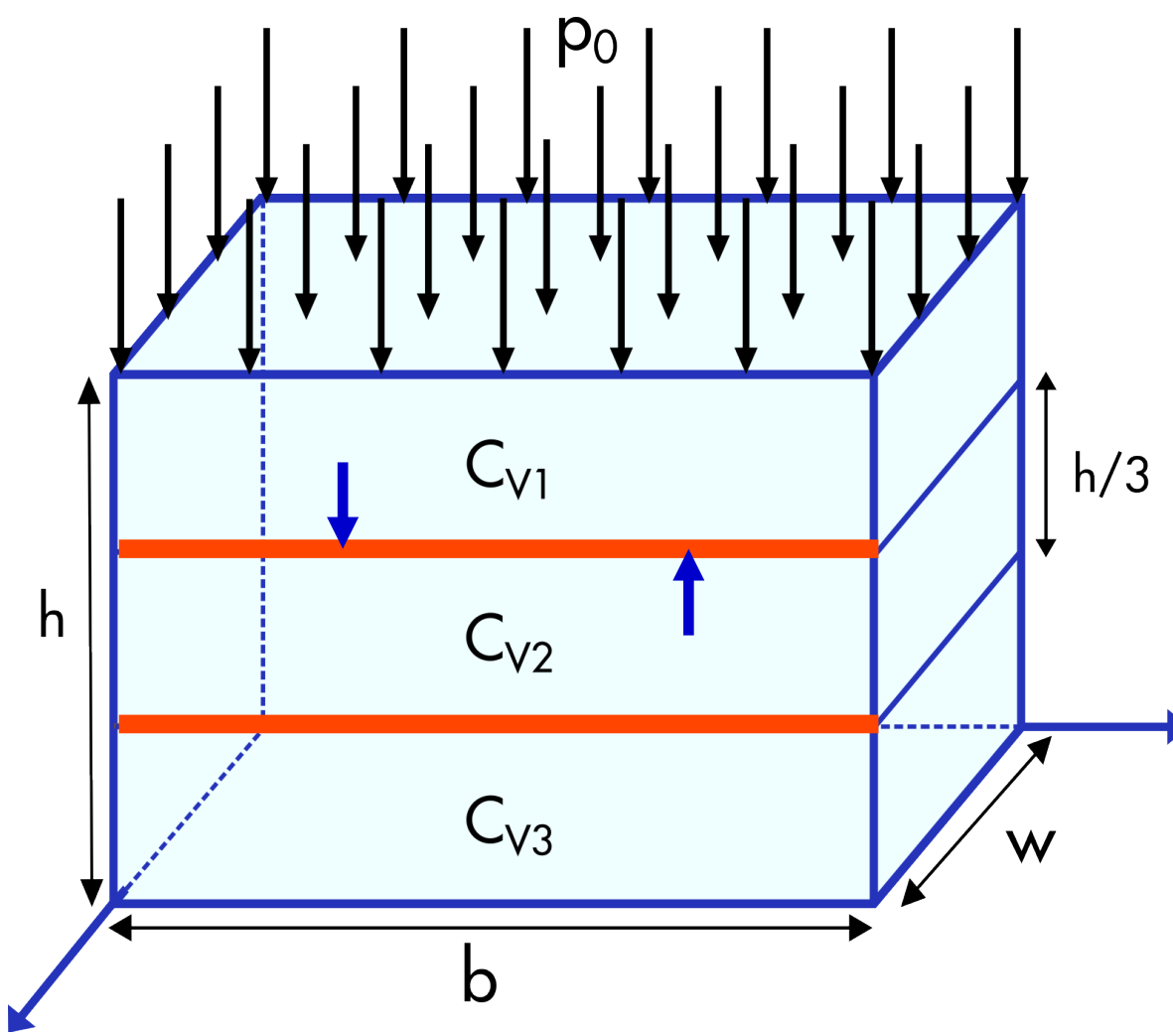
Equating the flow rates,

$$c_{v_{top}} \frac{P_{top} - P_{int}}{\cancel{\Delta z}} = c_{v_{bot}} \frac{P_{int} - P_{bot}}{\cancel{\Delta z}}$$

$$c_{v_{top}} P_{top} - c_{v_{top}} P_{int} = c_{v_{bot}} P_{int} - c_{v_{bot}} P_{bot}$$

$$P_{int} \left(c_{v_{bot}} + c_{v_{top}} \right) = c_{v_{top}} P_{top} + c_{v_{bot}} P_{bot}$$

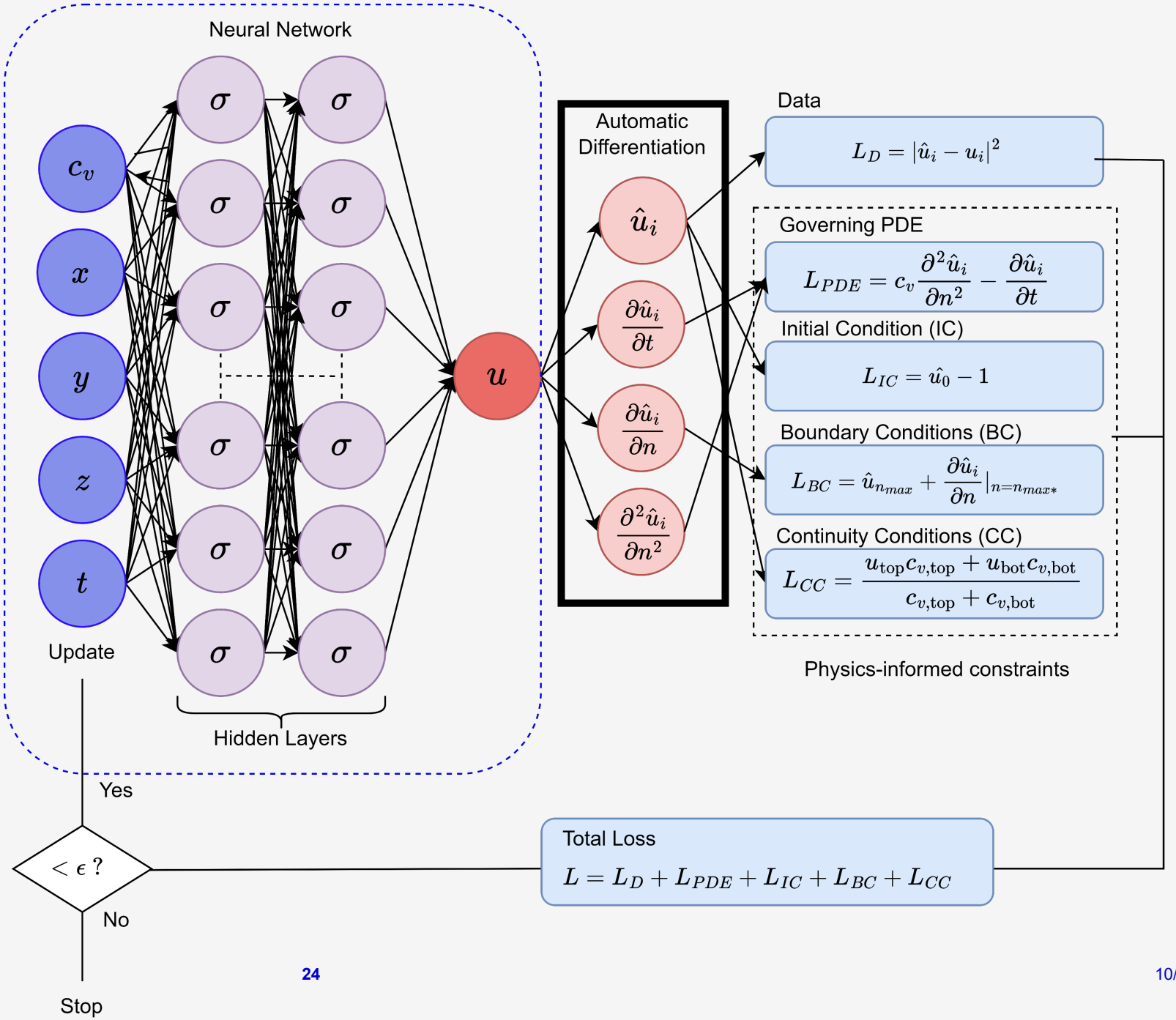
$$\therefore P_{int} = \frac{c_{v_{top}} P_{top} + c_{v_{bot}} P_{bot}}{c_{v_{top}} + c_{v_{bot}}}$$



Physics-informed neural network

The model is developed using **TensorFlow** and **Keras** backends. There are **4 hidden layers, with 40 neurons** in each layer.

We use the **'tanh'** activation function to introduce non-linearity and ensure values from 0 to 1.



High-Performance Computing at Imperial

What do we use it for?

High-Performance Computing at Imperial

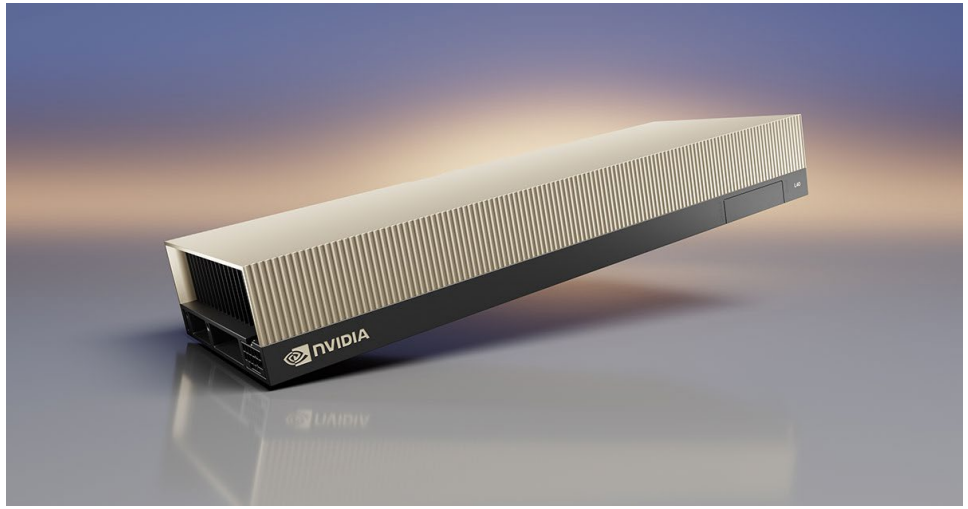
Nvidia GPUs

"Modern HPC data centres are key to solving some of the world's most important scientific and engineering challenges" **- Nvidia**

Specifications

NVIDIA L40S GPU

FP32	91.6 teraFLOPS
TF32 Tensor Core	366 teraFLOPS*
FP16	733 teraFLOPS*
FP8	1,466 teraFLOPS*
RT Core Performance	212 teraFLOPS
Max Power Consumption	350W



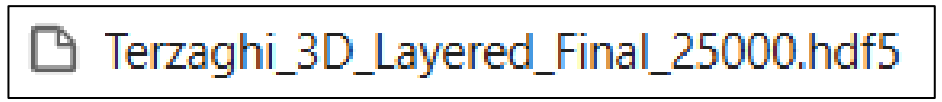
High-Performance Computing at Imperial

Submitting batch jobs

```
#!/bin/bash
#PBS -l select=1:ncpus=4:mem=64gb:ngpus=1:gpu_type=L40S
#PBS -l walltime=12:00:00

cd $PBS_O_WORKDIR

echo "Job started at $(date)"

eval "$(~/anaconda3/bin/conda shell.bash hook)"
source activate tf-gpu
python 3dlayeredtl.py --epochs 25000 > output.txt

echo "Job ended at $(date)"
```

```
Last login: Tue Feb  4 16:06:03 2025 from 2a0c:5bc0:89:1e01:ee61:b76f:1113:67b3
[us322@login-bi ~]$ qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
110251.pbs-7      rcs_mpiboffin    ge420                    0 Q v1_capability24
110479.pbs-7      AdaptorTrimming* agreen2                  0 Q v1_capability48
111634[].pbs-7    HaplotypeCaller* gjp15                    0 H v1_small72a
120670[].pbs-7    Linear2          hc1017                   0 B v1_medium72a
125782[].pbs-7    G1_Hp_simulatio* abeth                    0 B v1_medium24a
125783[].pbs-7    G1_Op_simulatio* abeth                    0 Q v1_medium24a
125784[].pbs-7    G1_Hp_simulatio* abeth                    0 Q v1_medium24a
130201.pbs-7      Gly.James.s1     jt2615                   0 Q v1_largemem72
130203.pbs-7      Gly.James.s1     jt2615                   0 Q v1_largemem72
132352.pbs-7      48_edge_to_edge* ewolpert          2101:54* R v1_medium72
132359.pbs-7      49_edge_to_wind* ewolpert          1553:32* R v1_medium72
132360.pbs-7      49_edge_to_wind* ewolpert          1362:28* R v1_medium72
132361.pbs-7      50_edge_to_wind* ewolpert          1268:58* R v1_medium72
132362.pbs-7      50_window_to_wi* ewolpert          1226:02* R v1_medium72
132363.pbs-7      50_window_to_wi* ewolpert          1132:45* R v1_medium72
```

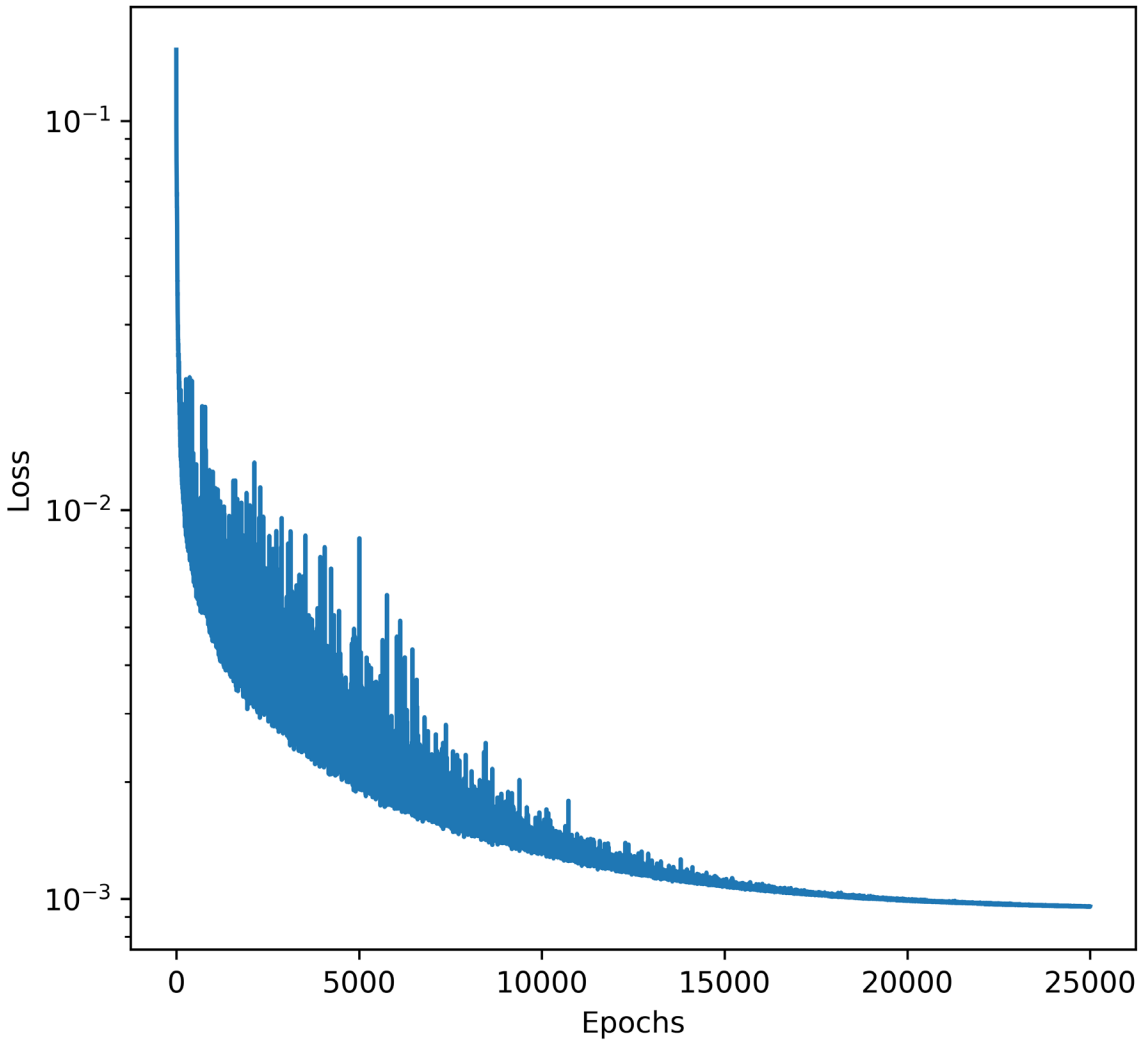
```
model = sn.SciModel(
    [xd, yd, zd, td, cv1, cv2, cv3],
    targets_3D,
    "mse",
    "Adam",
    load_weights_from = '/rds/general/user/us322/home/3DTLNew/weights/
    Terzaghi_3D_Layered_Final_25000.hdf5'
)
```



Training the Physics-informed Neural Network

Loss Function

- The neural network is trained using an **L40S GPU** for approximately **12 hours.**
- For reference, training this network using a 6-core CPU (Ryzen 5600x) **takes more than 100 hours.**
- We determine that a suitable loss is around $\mathbf{10^{-3}}$**,** such that there is no visible difference between the model's output and the exact solution.
- The model is trained for **25,000 epochs** (iterations) with a batch size of **500.**
- The learning rate of the **Adam optimiser** is exponentially decayed.



Training the Physics-informed Neural Network

Hyperparameter Tuning

- **Hyperparameters** must be appropriately determined for each problem.
- The model architecture can be too **complex** and **not offer any additional accuracy**.
- There must be a compromise between complexity and training time.

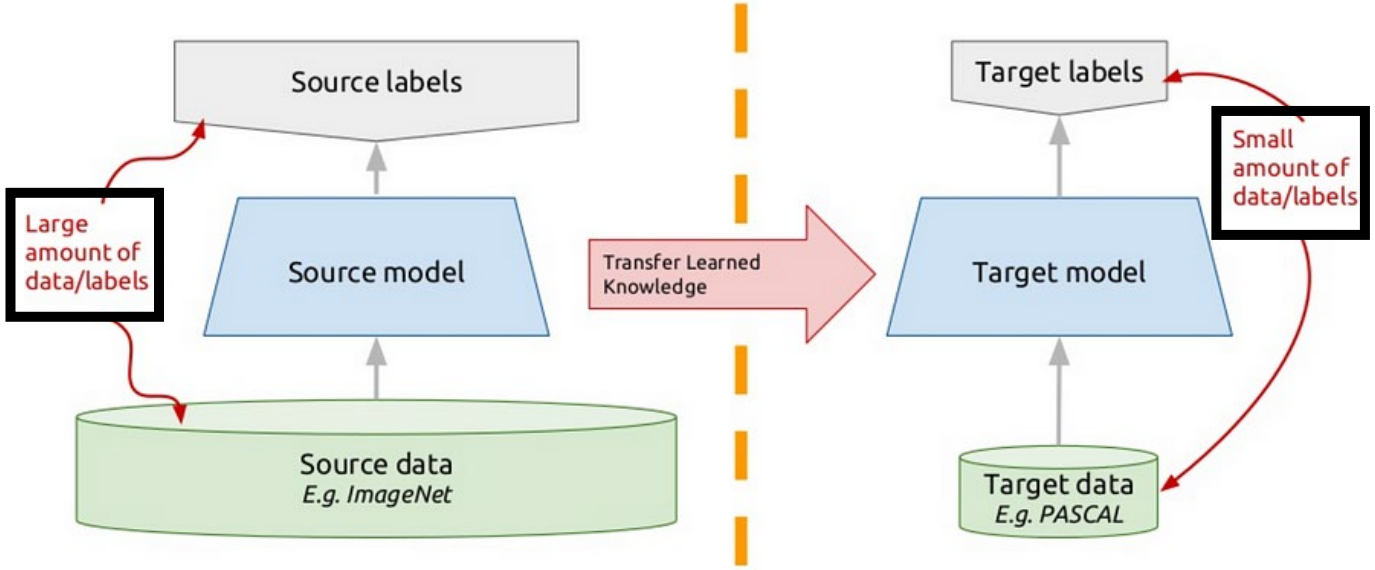
Layers \ Neurons		10			20			40		
time		t=0.20	t=0.40	t=0.80	t=0.20	t=0.40	t=0.80	t=0.20	t=0.40	t=0.80
1	L_2	2.1e-01	2.0e-01	1.9e-01	1.9e-01	1.4e-01	8.8e-02	1.7e-01	1.0e-01	6.5e-02
	MAE	9.3e-02			6.6e-02			5.5e-02		
2	L_2	9.7e-02	4.8e-02	5.1e-02	4.4e-02	2.3e-02	2.6e-02	2.9e-02	2.1e-02	1.7e-02
	MAE	3.6e-02			2.2e-02			1.8e-02		
4	L_2	2.7e-02	1.4e-02	1.6e-02	1.1e-02	8.2e-03	7.6e-03	9.6e-03	7.8e-03	8.7e-03
	MAE	1.6e-02			1.1e-02			9.0e-03		

What if I don’t have a GPU?
Transfer Learning

Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem.

It's a way to **reuse models** and knowledge to address new problems more **efficiently.**

This has been used in many image classification algorithms. If a model is good at classifying racoons, it can probably be used to classify tanukis.

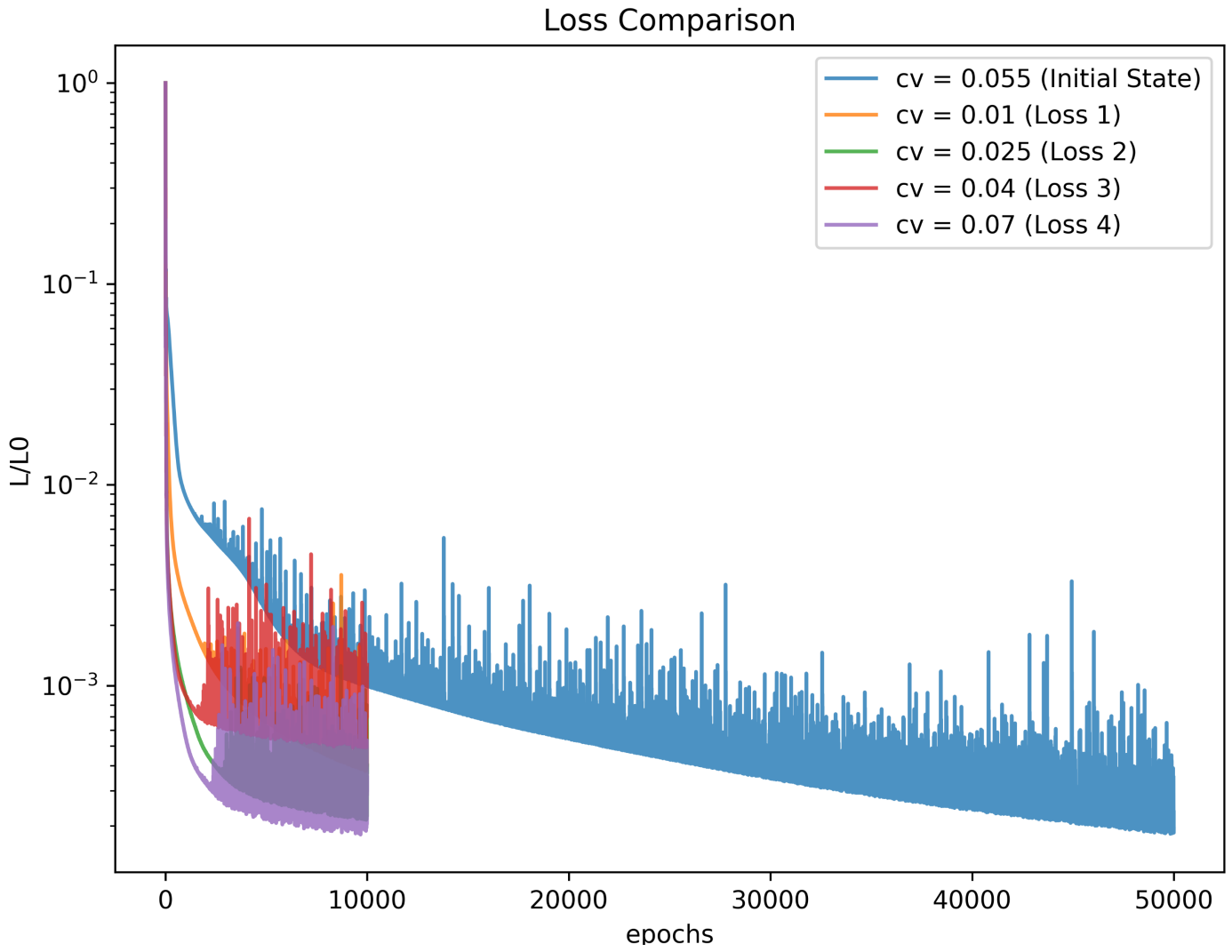


What if I don’t have a GPU?
Transfer Learning

The neural network is trained initially for **50,000 epochs** for the coefficient in the **midpoint** of the training region

The model is then **re-trained** for a new coefficient/parameter

Each subsequent training session leads to a lower loss value, with **quicker convergence**

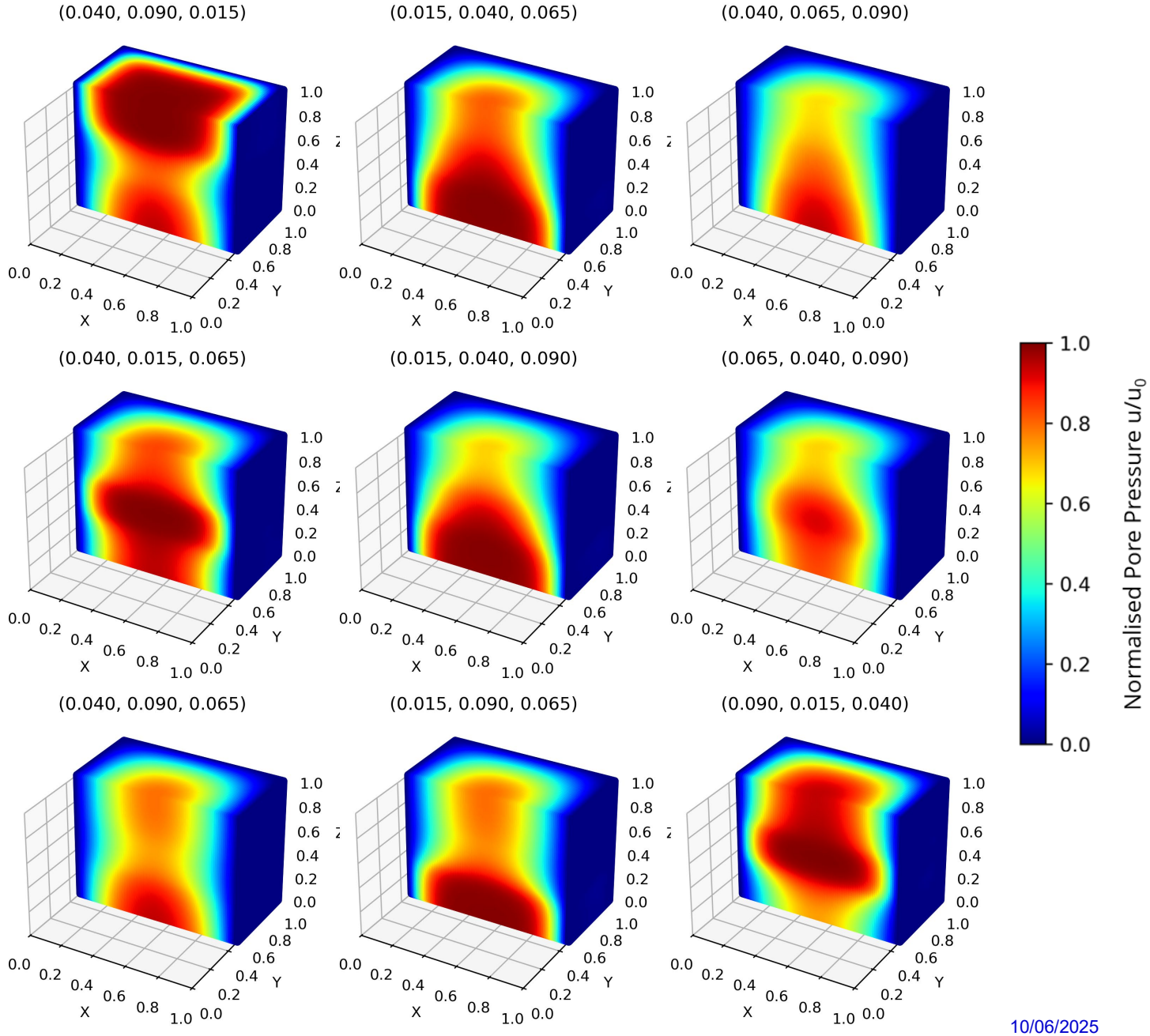


Results

Pore Pressure Plots

The following plots are outputted from the model for $\boldsymbol{t = 0.5}$ **years** and **1,000,000** samples. The $(c_{v_1}, c_{v_2}, c_{v_3})$ values are varied over the testing region.

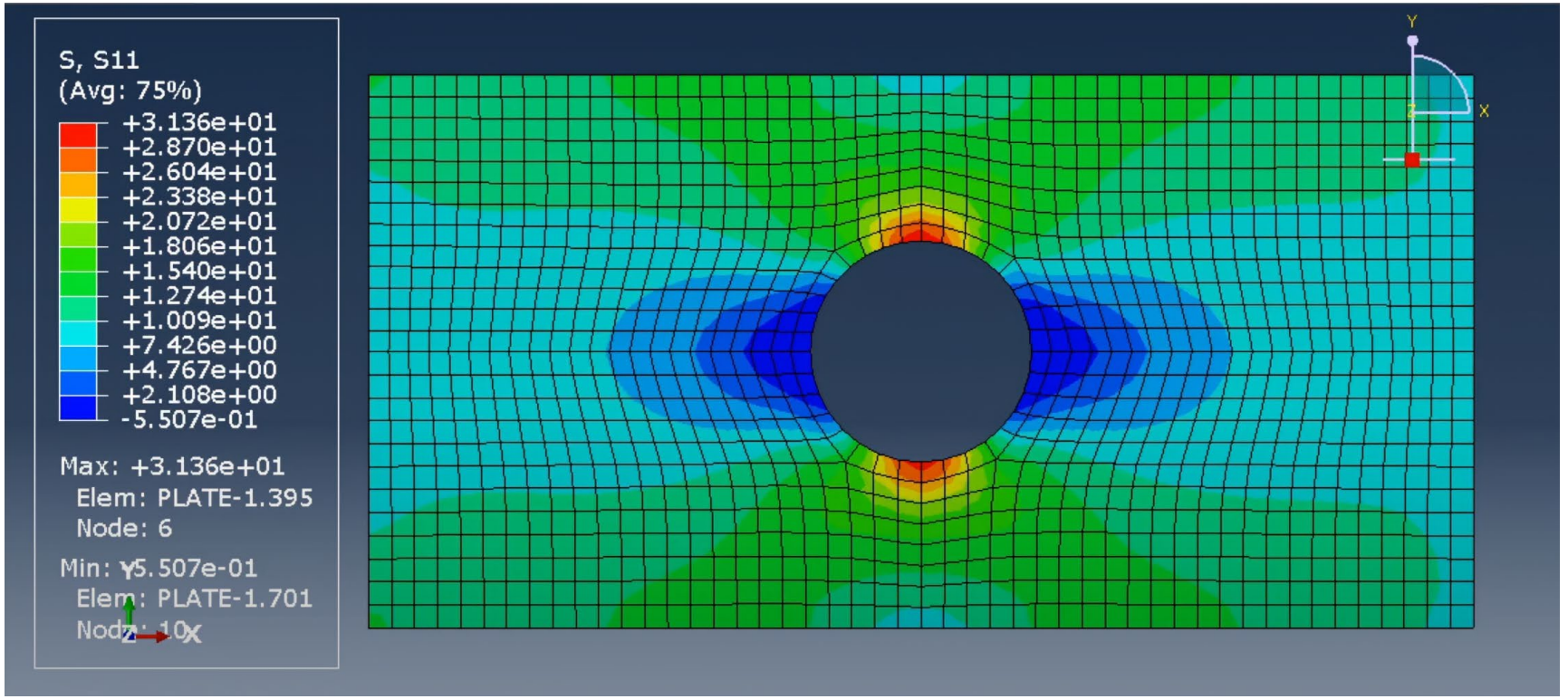
The model can also output a pore pressure for any grid size, or even a single point. The FEM solution **would need to compute the entire** solution first before evaluating a single point.



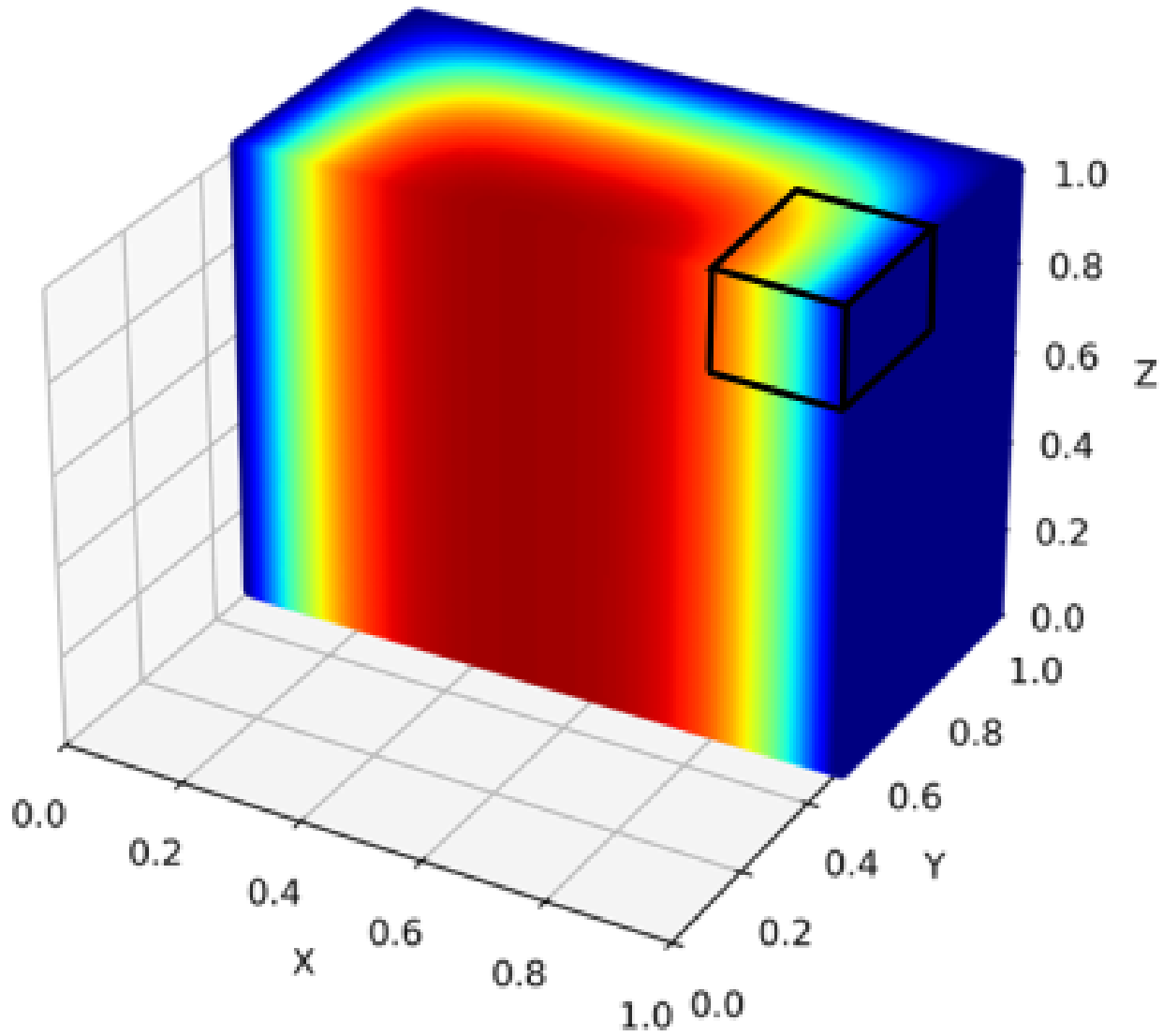
Mesh Constraint

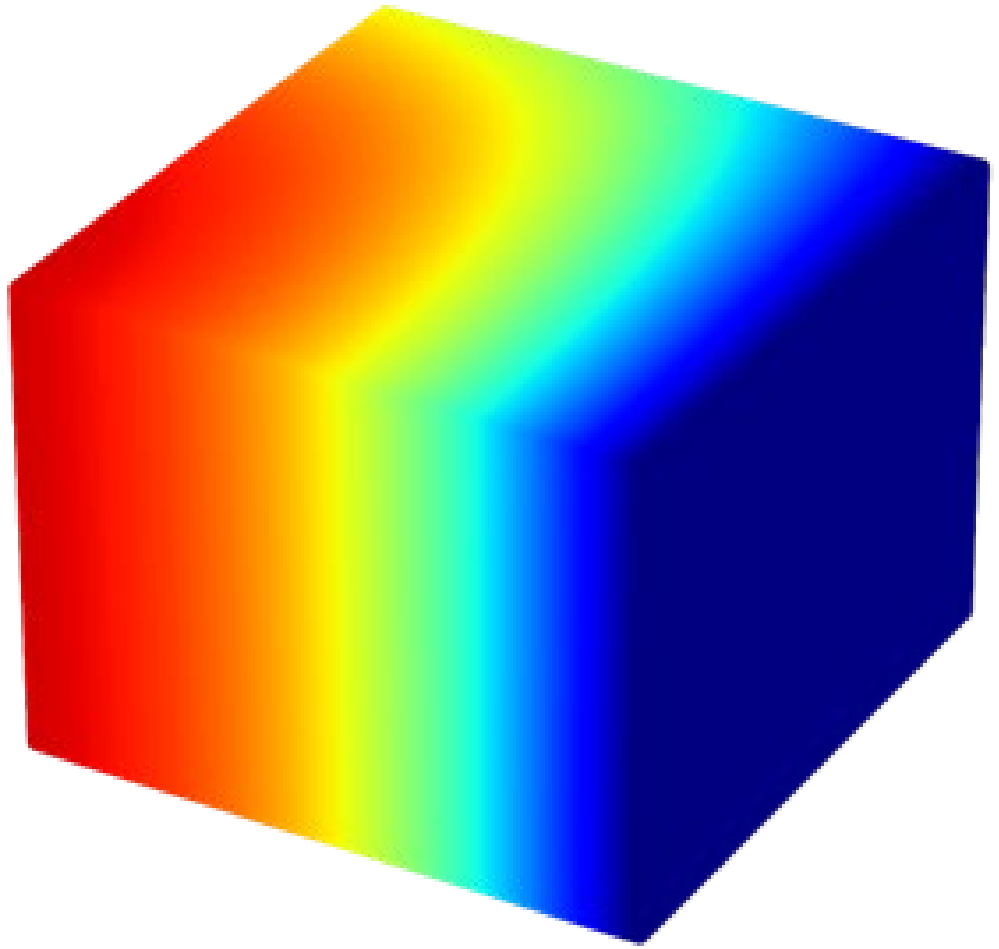
FEM solutions are **always constrained to a mesh.**

If a researcher would like to probe the value at a **specific point**, they must compute the FEM solution again.

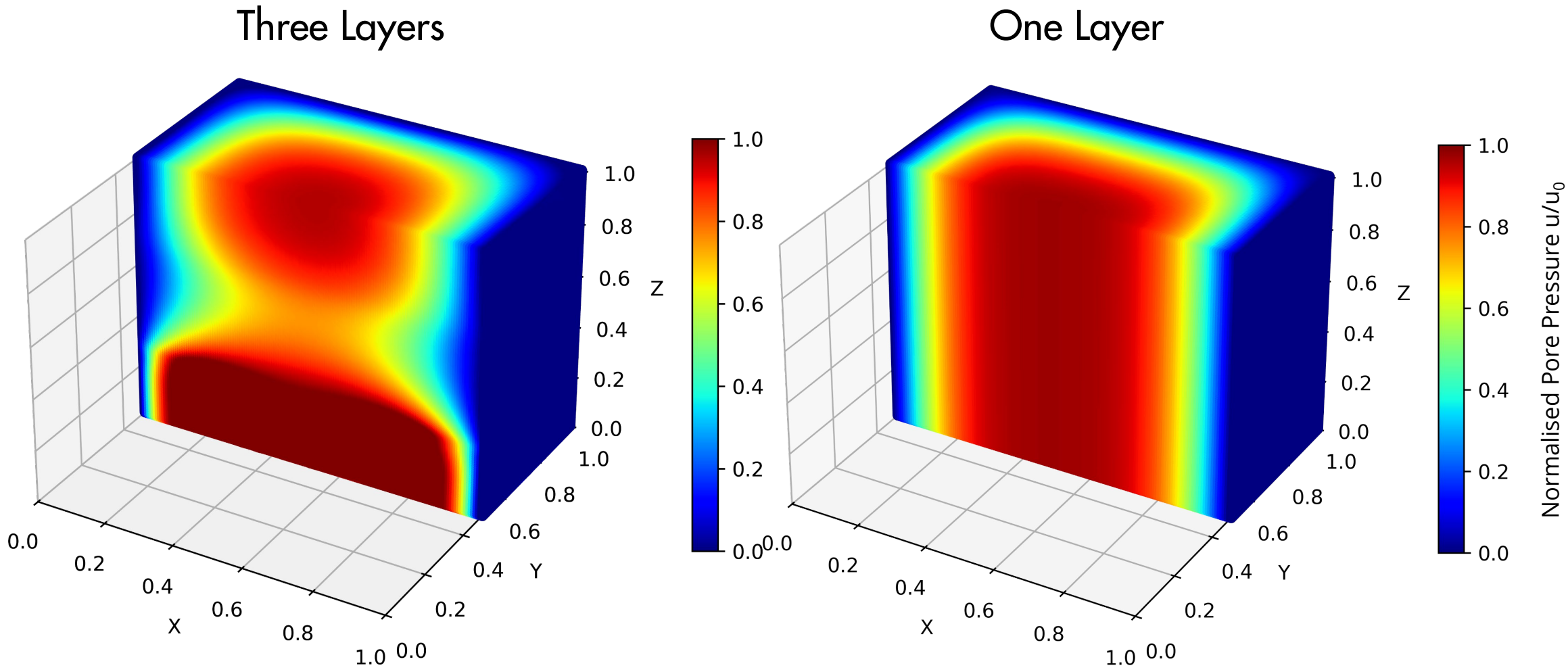


Mesh Constraint

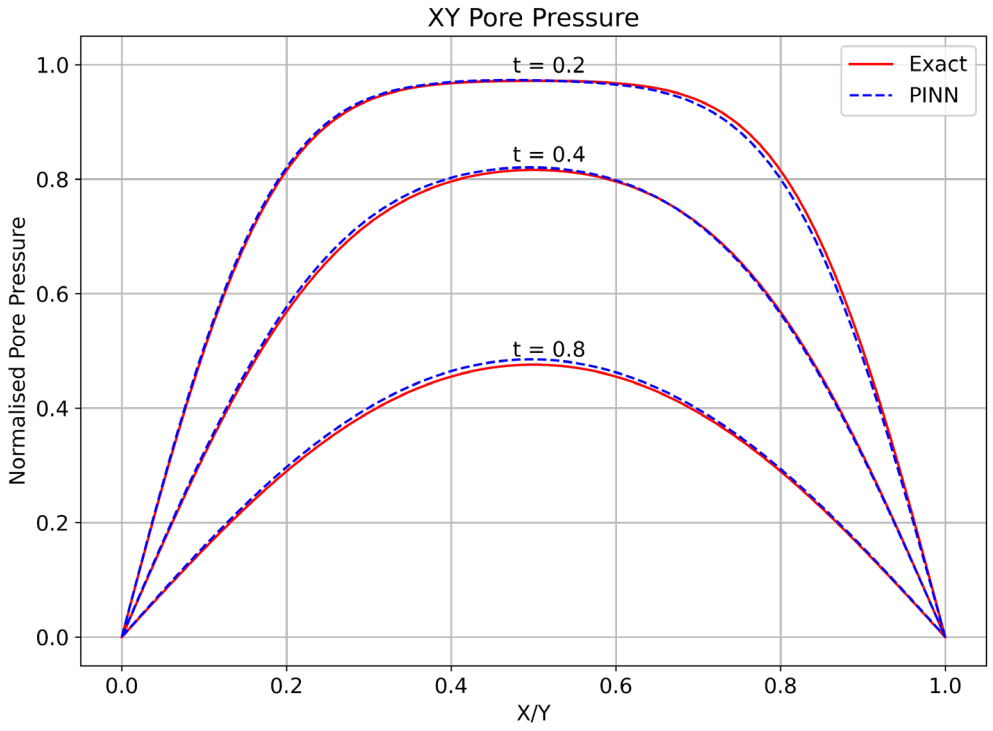


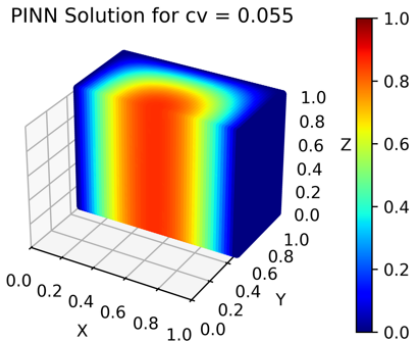


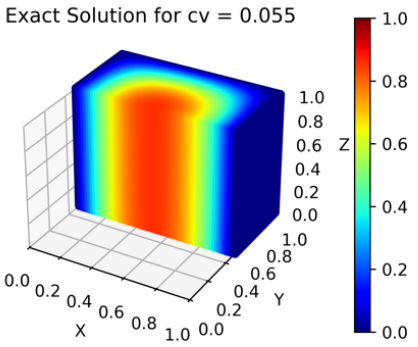
What about one layer?

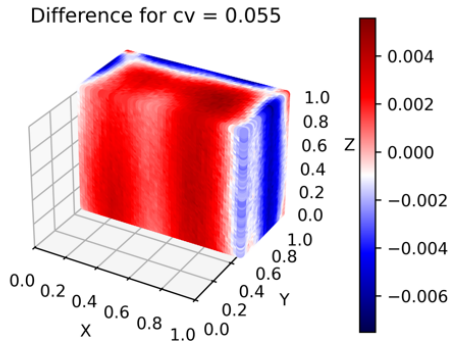


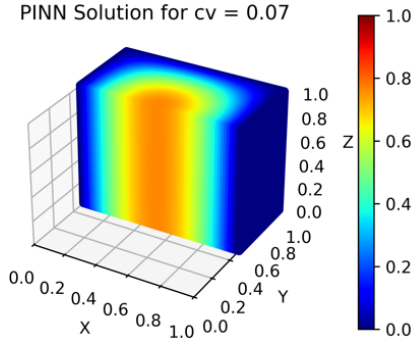
Comparisons

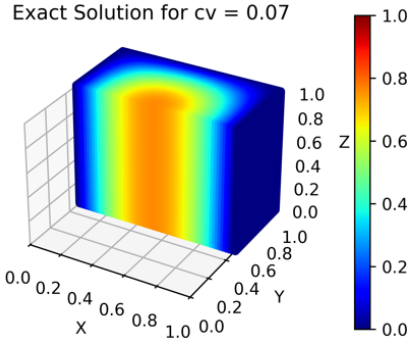


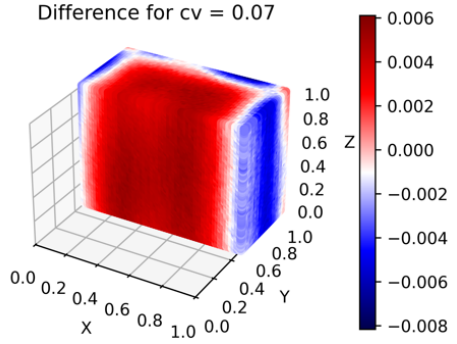


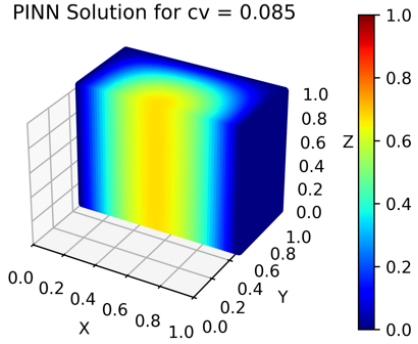


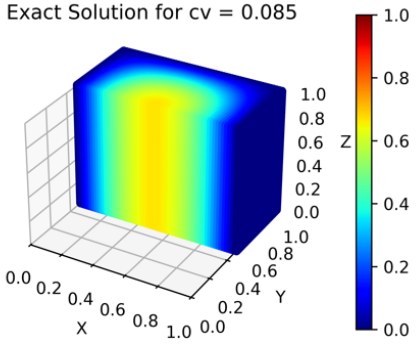


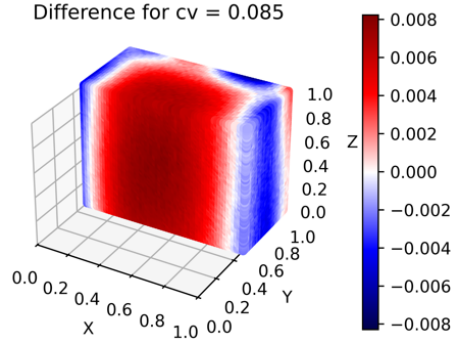


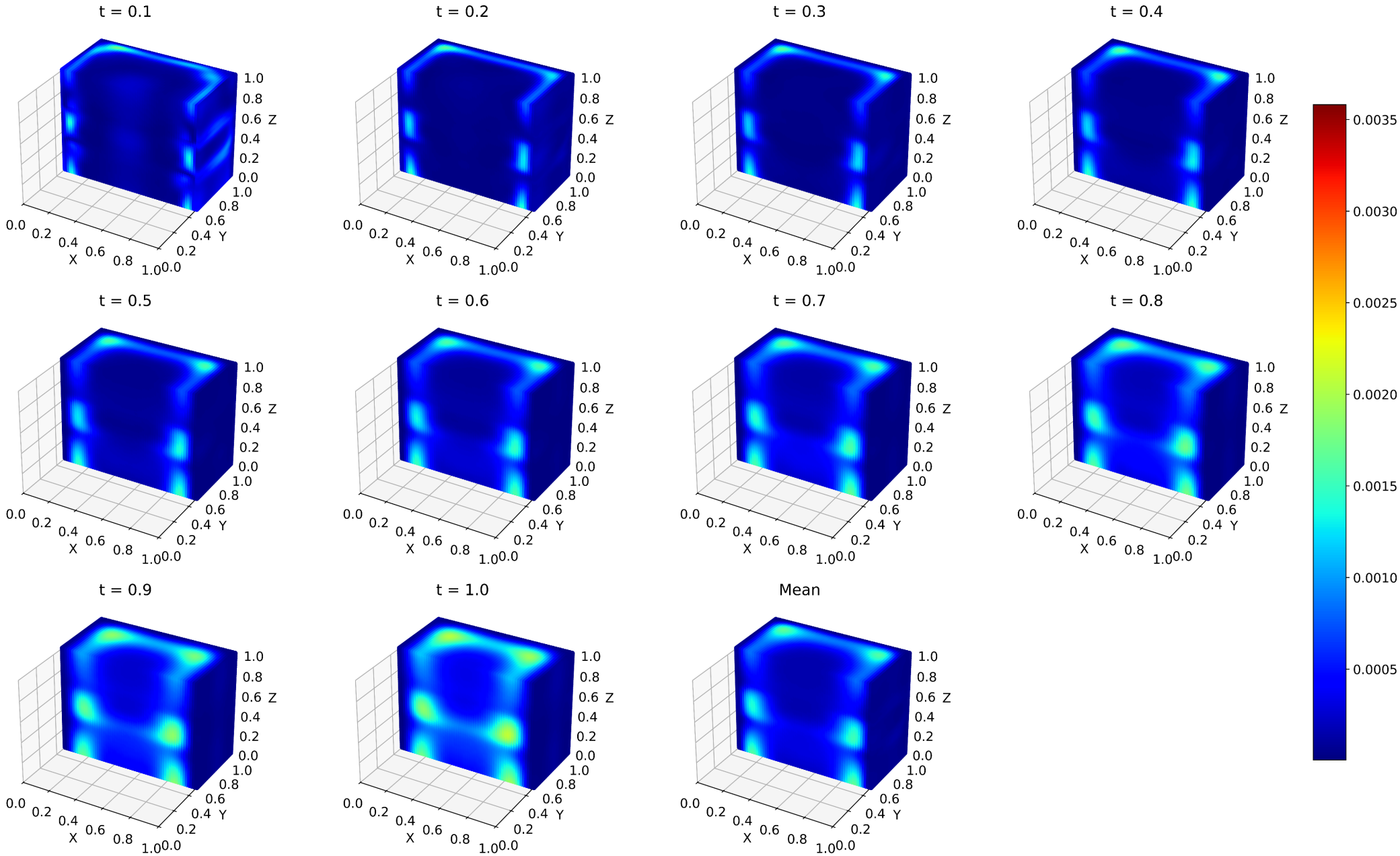




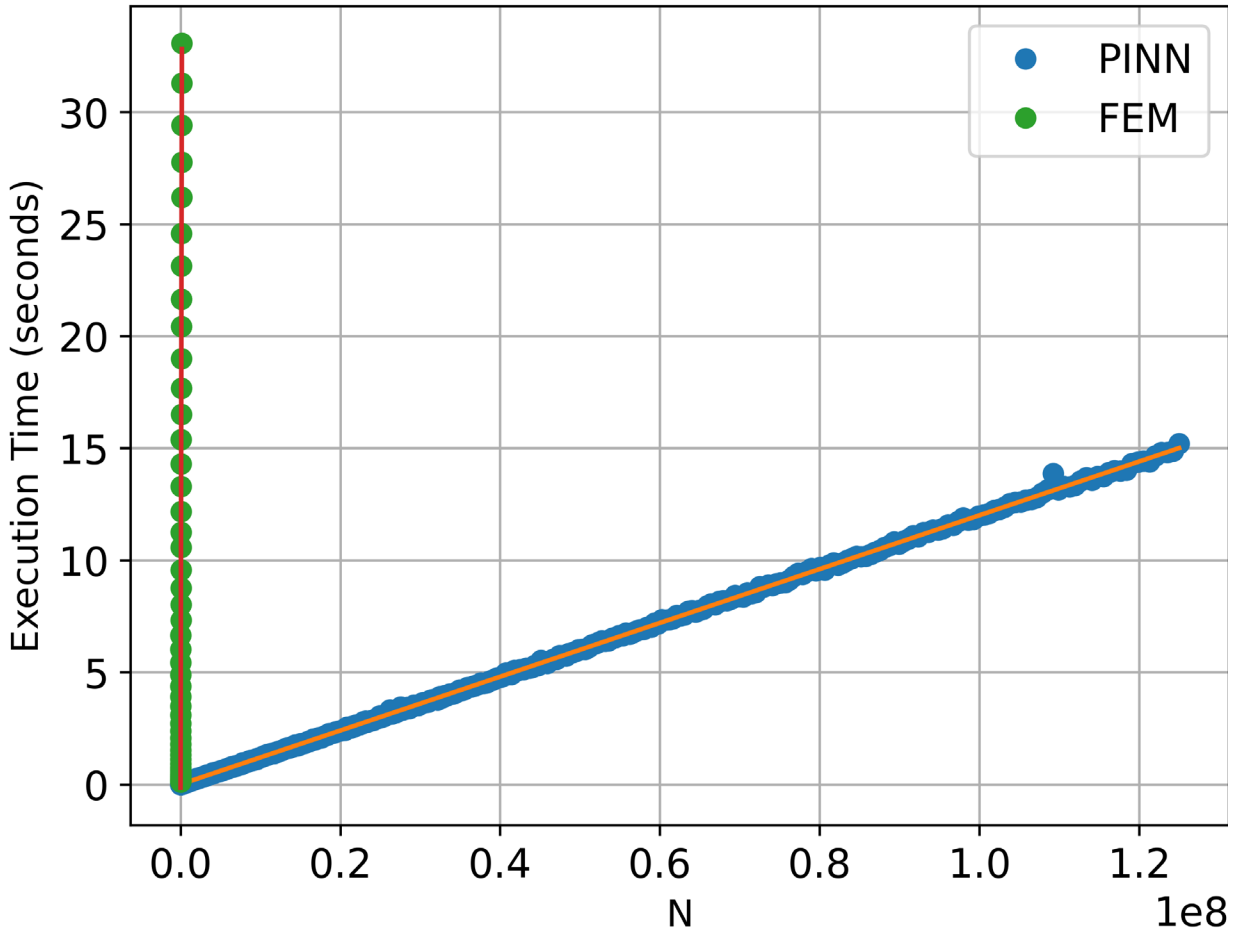


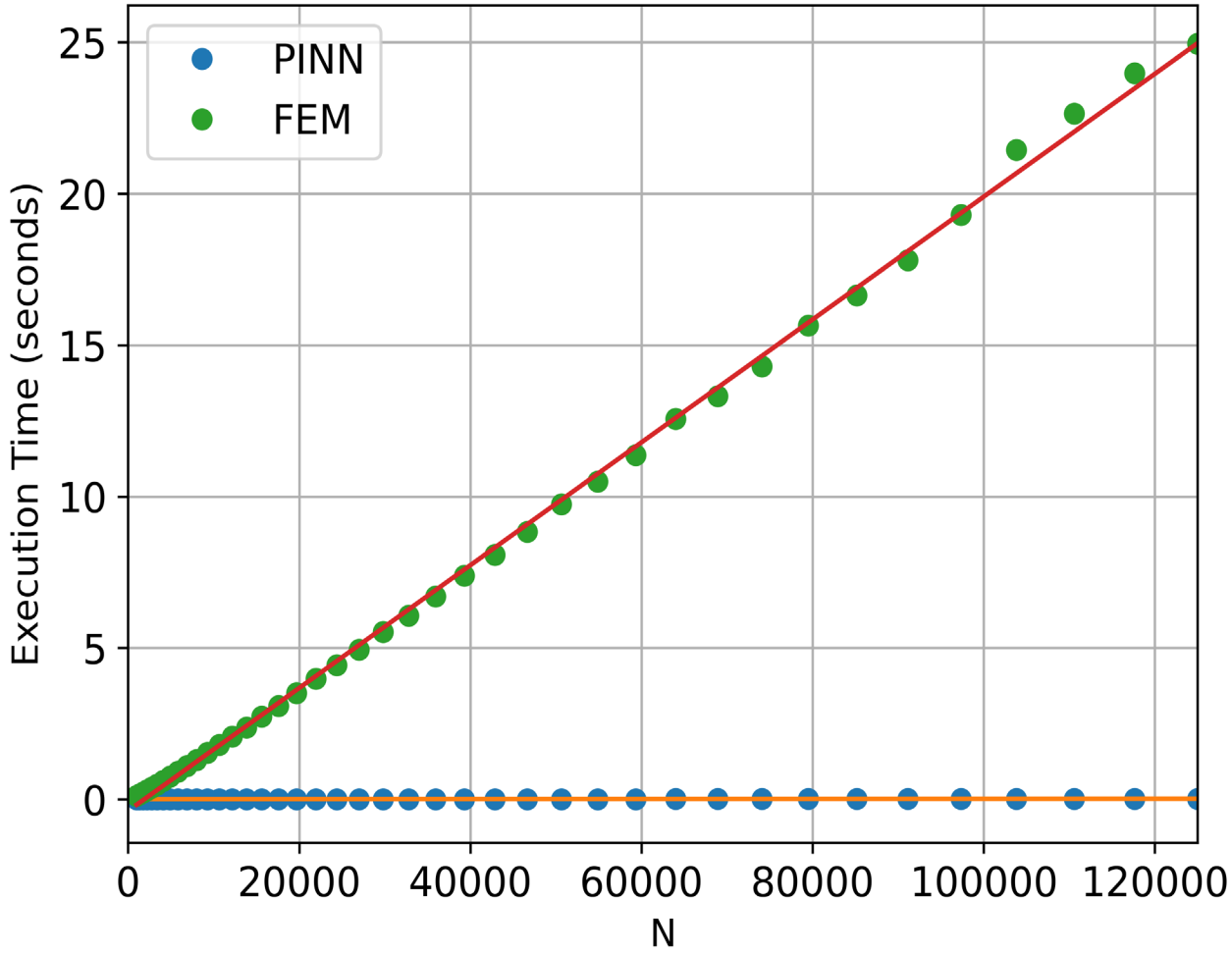


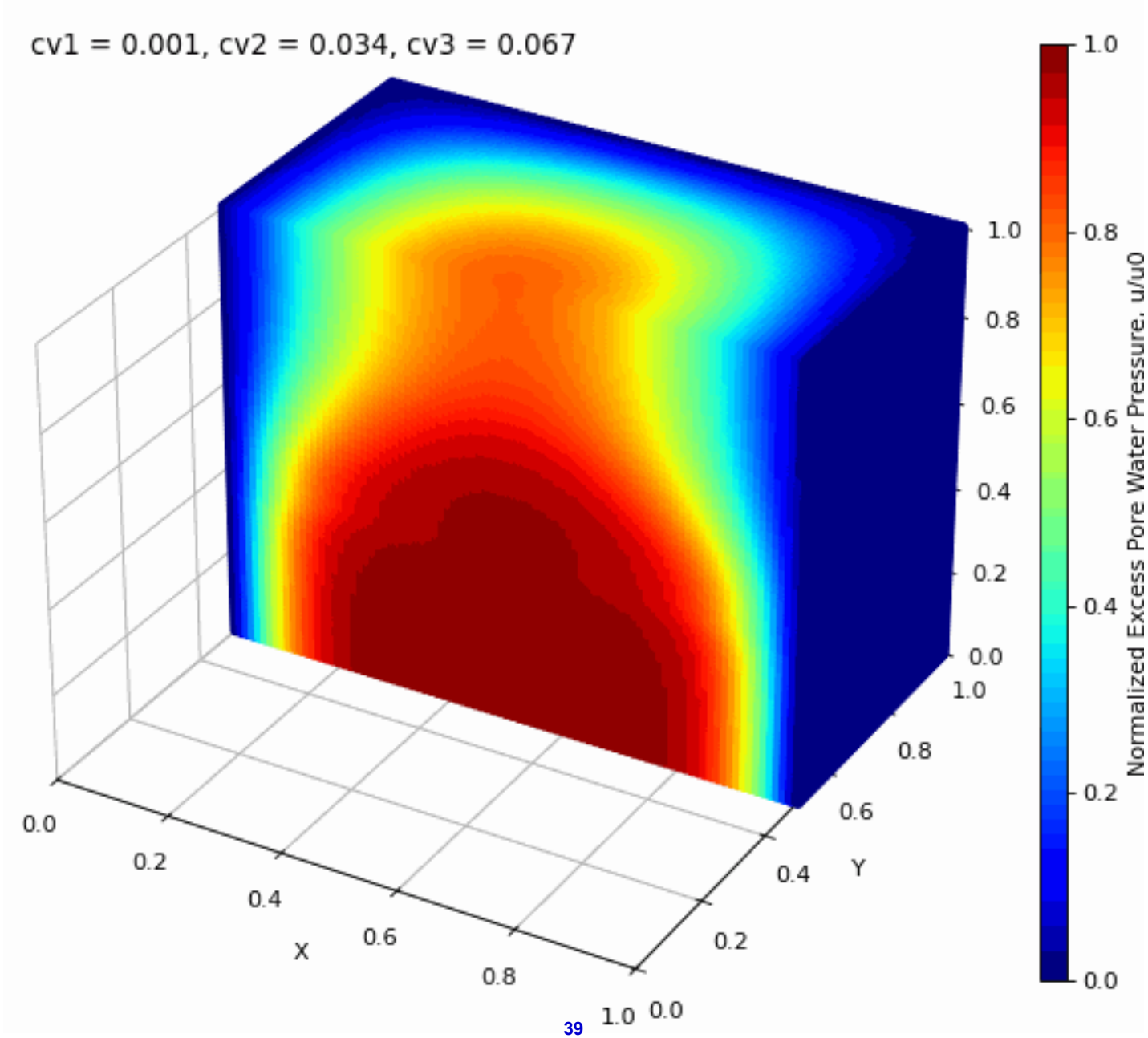


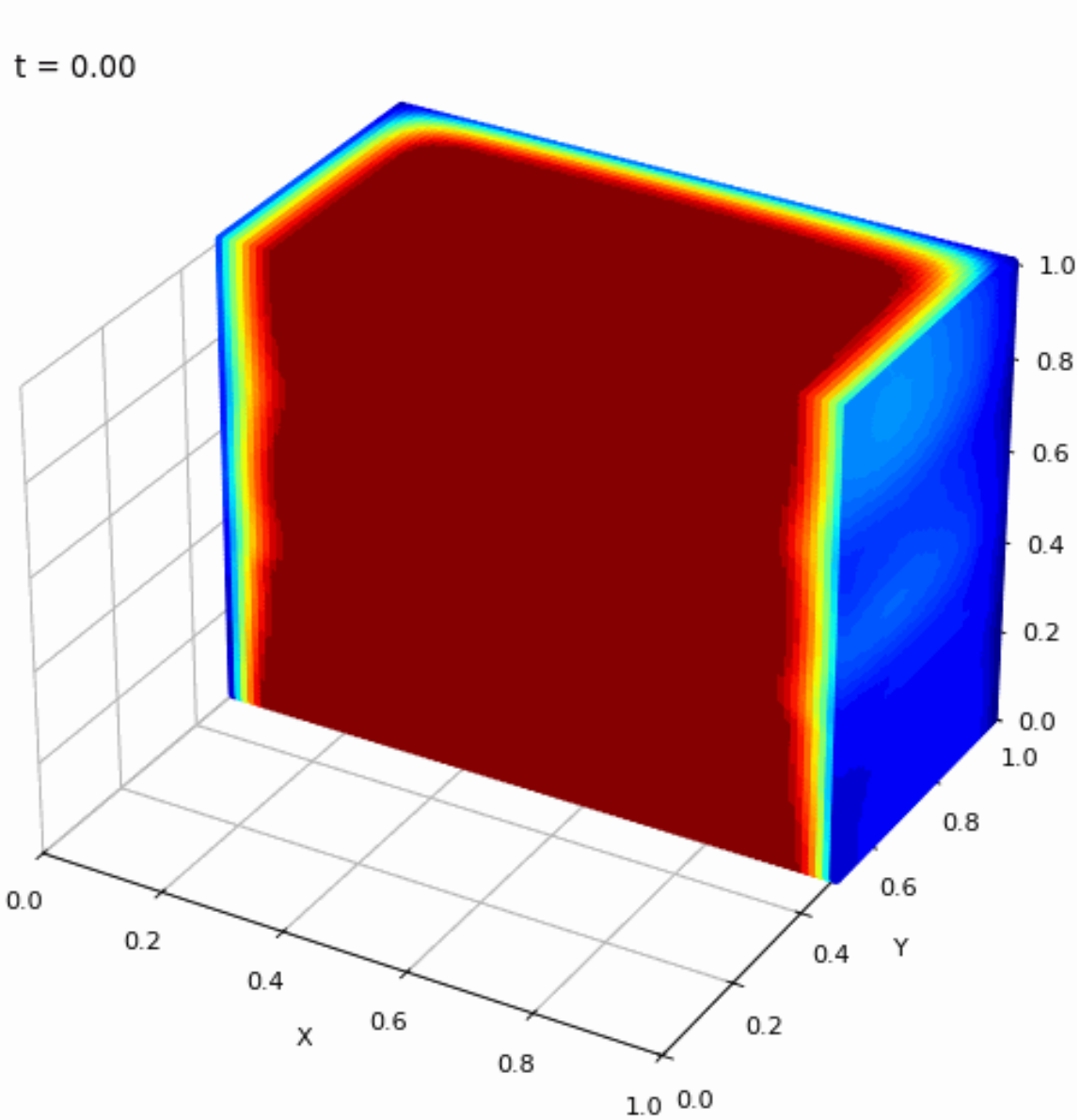

t = 0.1
t = 0.2
t = 0.3
t = 0.4
t = 0.5
t = 0.6
t = 0.7
t = 0.8
t = 0.9
t = 1.0
Mean
X
Y
Z
0.0
0.2
0.4
0.6
0.8
1.0
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005

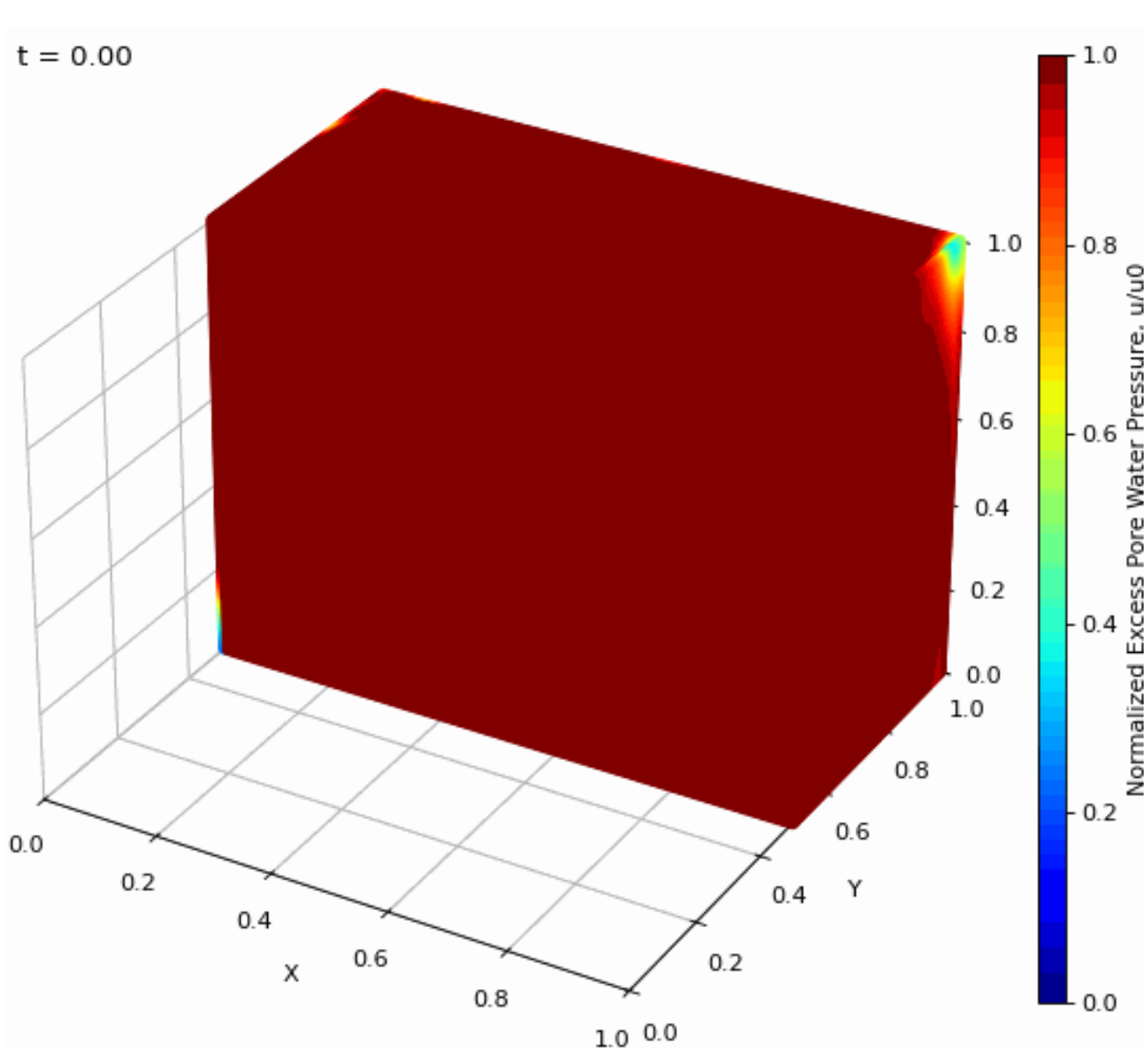
Execution Times



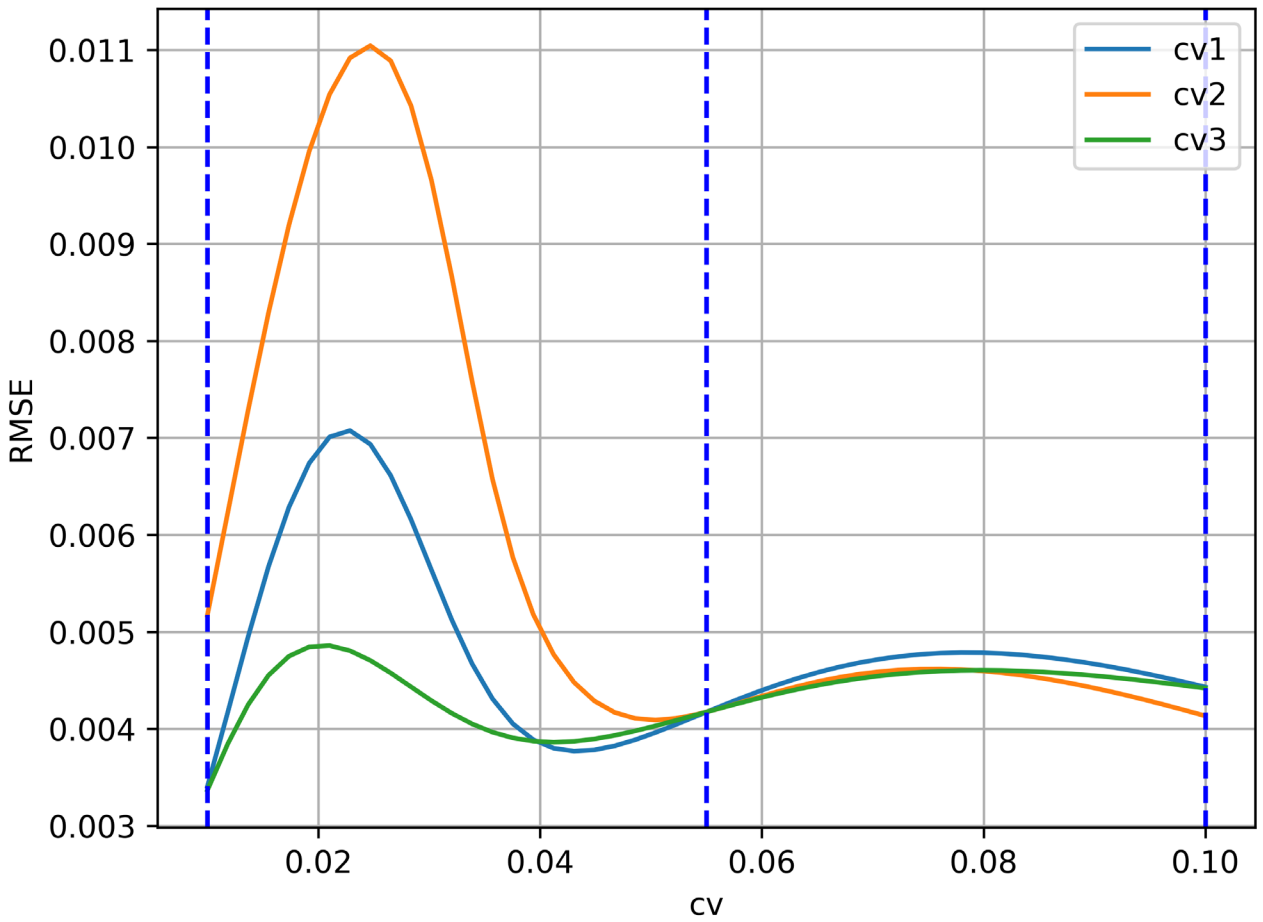



cv1 = 0.001, cv2 = 0.034, cv3 = 0.067
X
Y
Normalized Excess Pore Water Pressure, u/u0
0.0
0.2
0.4
0.6
0.8
1.0


t = 0.00
0.0
0.2
0.4
0.6
0.8
1.0
X
Y


t = 0.00
0.0
0.2
0.4
0.6
0.8
1.0
X
Y
Normalized Excess Pore Water Pressure, u/u0

Sensitivity Analysis



The input coefficients are **varied singularly** for each layer, keeping the other two coefficients **constant at 0.05**

The RMSE is calculated between the model's output and the exact solution.

The plot shows that the performance of the model **is poor for values further away** from the training points, as expected.

Inversion Problem

Predicting the coefficients of each layer in a given solution

Inversion Algorithm

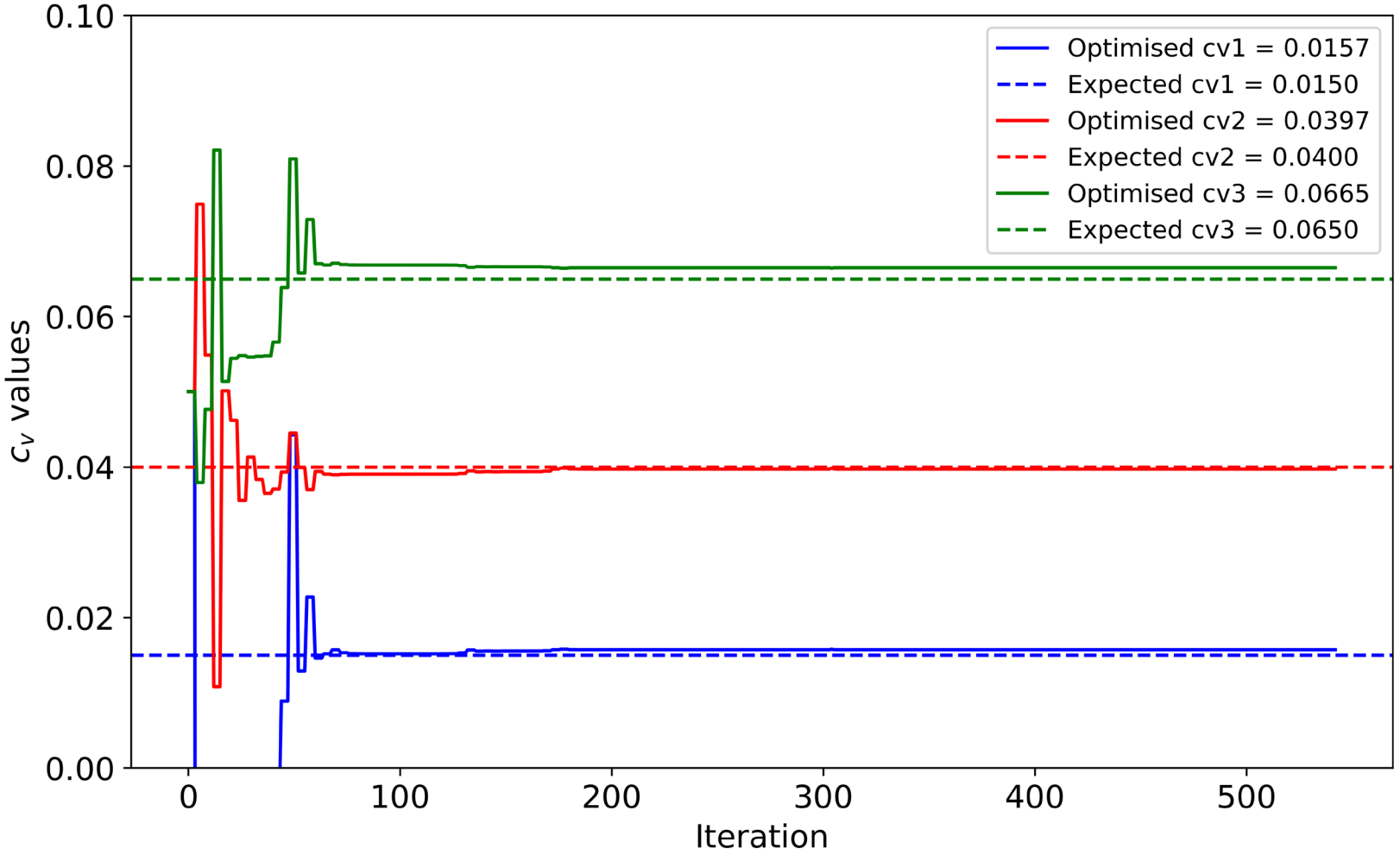
How can we use the model to predict unknown coefficients?

Algorithm 2 Three-Layered Inversion

1: Initialize empty lists `cv1_tested`, `cv2_tested`, and `cv3_tested` to store tested values
2: Load exact pore pressure data u_t.
3: **function** ERRORFUNCTION($[c_{v1}, c_{v2}, c_{v3}]$)
4: Append c_{v1} to `cv1_tested`, c_{v2} to `cv2_tested`, and c_{v3} to `cv3_tested`.
5: Evaluate the pore pressure u for $[c_{v1}, c_{v2}, c_{v3}]$.
6: Compute the Mean Squared Error, MSE $= \frac{1}{n}\sum_{i=1}^{n}(u_t[i] - u[i])^2$.
7: **return** the MSE.
8: **end function**
9: Define an initial guess for $[c_{v1}, c_{v2}, c_{v3}]$ as $[0.05, 0.05, 0.05]$.
10: Set optimization tolerances: `ftol` $= 10^{-8}$ and `gtol` $= 10^{-8}$.
11: Minimise the error using BFGS algorithm, updating $[c_{v1}, c_{v2}, c_{v3}]$.
12: Store the optimized values of c_{v1}, c_{v2}, and c_{v3} in `best_cv1`, `best_cv2`, and `best_cv3`.

Inversion

Multi-layered

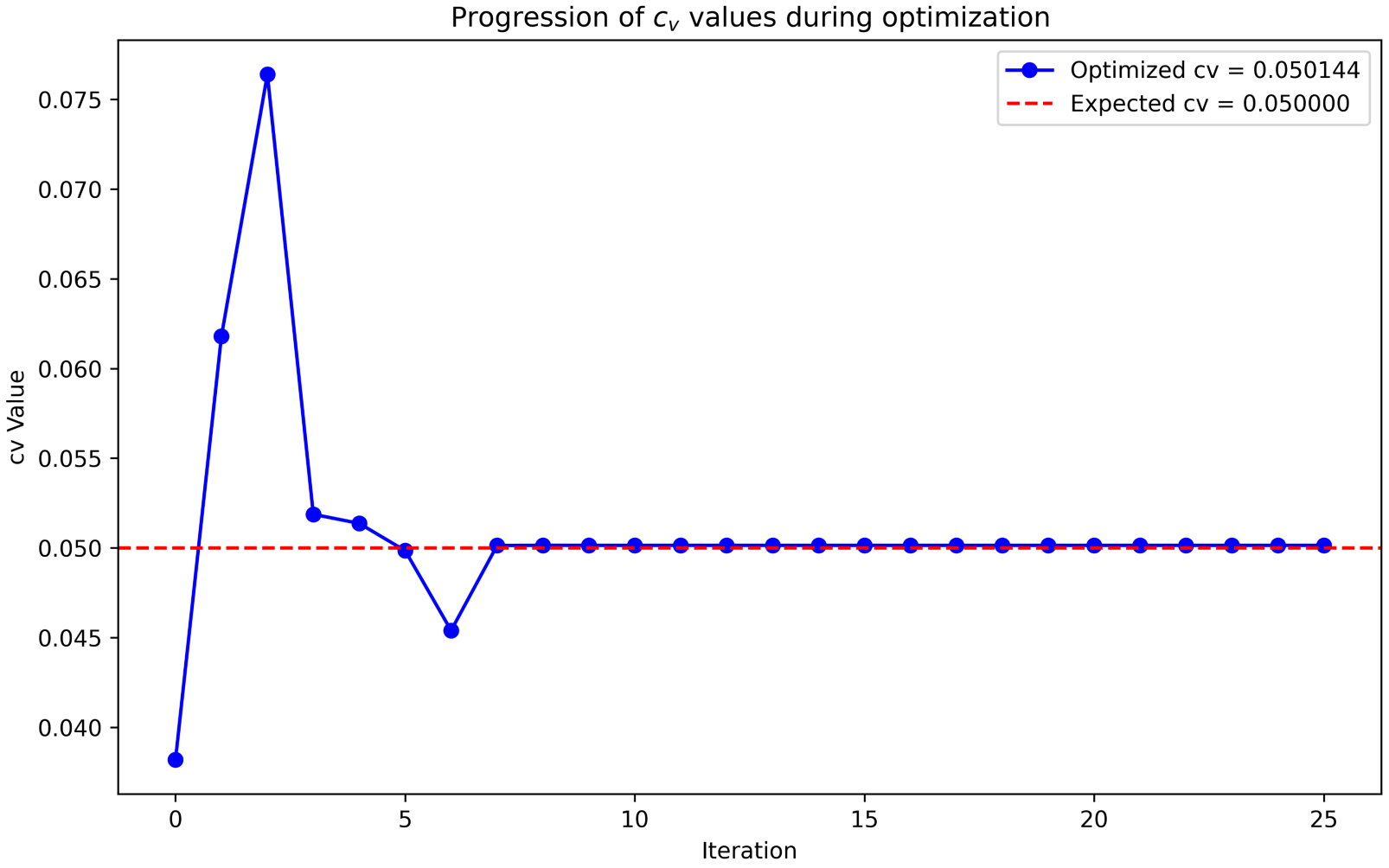


c_{v_1} accuracy: **96.34%**

c_{v_2} accuracy: **99.25%**

c_{v_3} accuracy: **97.69%**

Inversion
Single-layered



99.7% accurate!

Conclusion

What does this mean for civil engineers?

Our results show that the PINN model can **accurately (and quickly)** output pore pressure data and **predict unknown parameters** from exact solution data.

FEM software is typically **costly and challenging** to learn, while the PINN model can be easily tweaked by Python users for different drainage conditions.

This research can be extended to **more realistic and unsymmetrical examples** to aid civil engineers in making informed decisions in the real world.

IMPERIAL

Thank you!

Umar Siddique
15/02/2025