# Physics-informed Convolutional Neural Networks for Temperature Field Prediction of Heat Source Layout without Labeled Data*

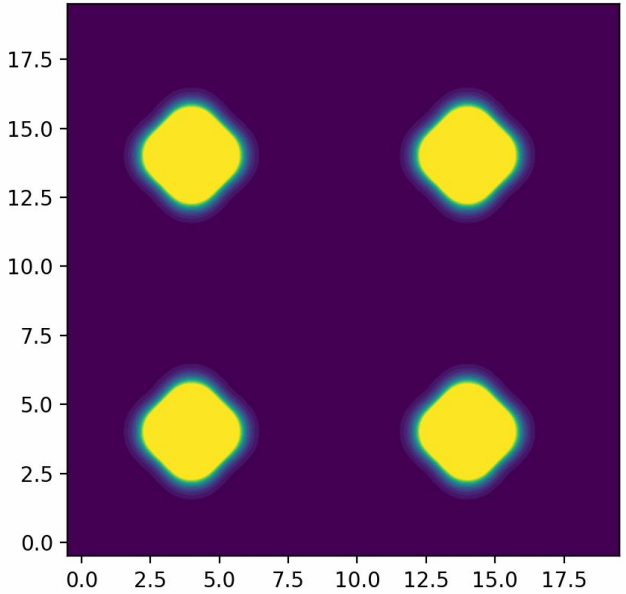
**Group 7**

Peter Knauss, Thomas Sievert, Wenzhong Hu, Aashay Arora

Github

# Motivation

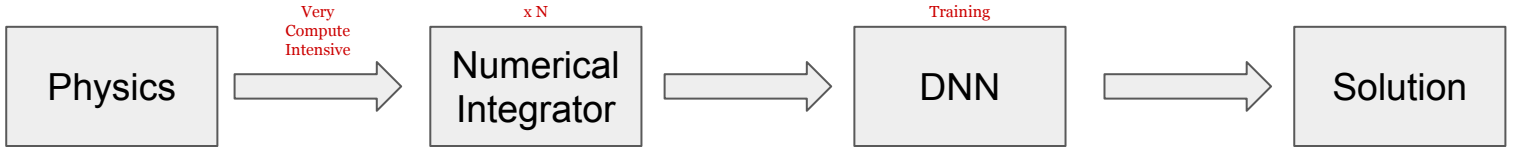
- Partial Differential Equations (PDEs) are used to model a wide range of physical phenomena, including fluid dynamics, heat transfer, and quantum mechanics etc.
- They can allow us to predict the path of a hurricane, the behavior of a fluid in a pipeline, and the thermal design and analysis of electronic systems. Therefore knowing how to solve them fast and accurately is very important.
- Although there are ways to solve them numerically, these are usually computationally expensive.
- What if we are able to teach a ML model to solve them faster?



## But How?

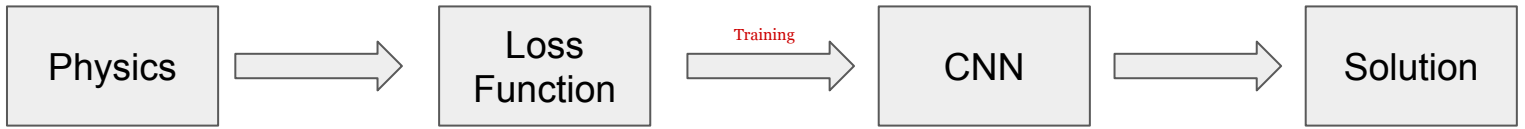
## Option 1: Supervised Learning

- Solve the PDEs using numerical methods and feed that into a DNN
  - Large amounts of training data are usually necessary to achieve reliable predictive performance.
  - Preparing sufficient simulation data is expensive and time-consuming




## Option 2: Unsupervised Learning - PINN

- Solve the PDEs by incorporating the physics within the loss function and then training the model



# Heat Equation

- The heat equation is a PDE that describes the rate of flow of heat in/out of a material, given the temperature of the environment.
- In 2+1 dimensions, it looks like

$$\partial_t T(x,y) = \nabla^2 T(x,y) + \phi(x,y)$$
$$(x,y) \in \Omega$$

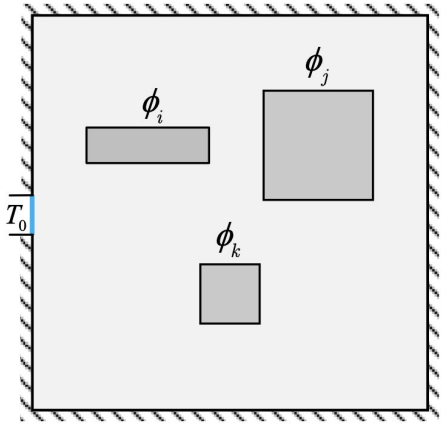
- We are looking for the steady-state solution i.e. the temperature field that satisfies the Poisson equation for the given heat layout + boundary conditions.

$$\nabla^2 T(x,y) + \phi(x,y) = 0$$

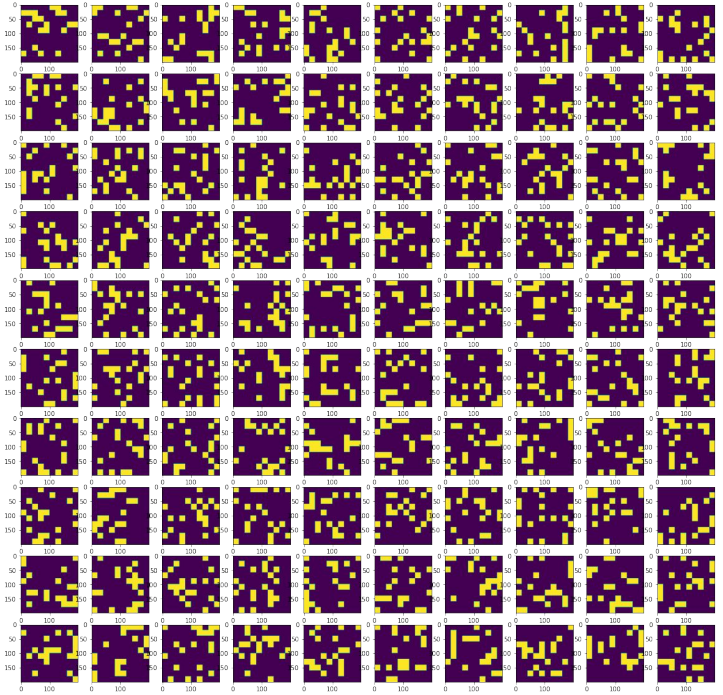
$$\phi(x,y) = \begin{cases} \phi_i(x,y) \in \Gamma_i, & i \in 1,2,3,...n \\ 0 & \text{else} \end{cases}$$

# Layout Setup

The training layout is a 2D conducting domain with several rectangular heat sources. An isothermal boundary of length δ and temperature $T_0$ in the bottom exchanges heat with external circumstances. Other boundaries are adiabatic.




We train on 10k such layouts.



To validate the performance of the model, we compare against the solution from FDM.

# Training the Model

# Two Types of Boundary Conditions

- Dirichlet BCs
  - fixed boundary condition

- Specifies the value of the function when imposed on PDEs:

$$y(x) = f(x) \quad \forall x \in \partial\Omega,$$

- f(x) = const. = To In problem

At isothermal sinks, T(x, y) = 298K.

# Two Types of Boundary Conditions

- Neumann BCs

- Specifies the value of the first derivative of the function when imposed on PDEs:

$$\frac{\partial y}{\partial \mathbf{n}}(\mathbf{x}) = f(\mathbf{x}) \quad \forall \mathbf{x} \in \partial\Omega,$$

Over adiabatic walls, the flux of heat is set to 0.

# Physics Informed Loss Function

- The Loss function is derived from the Poisson Difference Equation.
  - We write the Poisson equation with the differentials approximated by the corresponding differences to $O(\delta r^2)$, i.e.

$$\partial_{xx}T(x,y) = \frac{T(x+\delta x, y) - 2T(x,y) + T(x-\delta x, y)}{\delta x^2} \equiv \frac{T(x_{i+1}, y_j) - 2T(x_i, y_j) + T(x_{i-1}, y_j)}{h^2}$$

- With these differences, the Poisson Equation becomes,

$$4T(x_i, y_j) - T(x_{i+1}, y_j) - T(x_{i+1}, y_j) - T(x_{i-1}, y_j) - T(x_i, y_{j+1}) - T(x_i, y_{j-1}) = \frac{h^2\phi(x_i, y_j)}{k}$$

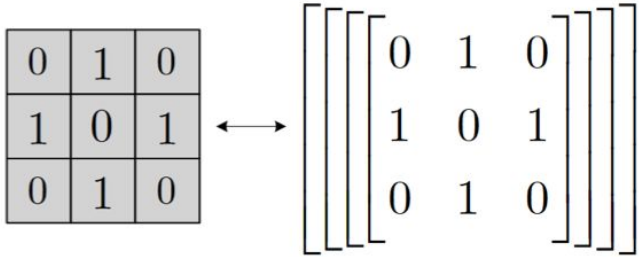
- Then the loss function is,

$$\mathcal{L} = \frac{1}{|D_I|}\sum_{(x,y)} \left|\left| T(x_i, y_j) - \frac{1}{4}T(x_i, y_j)' \right|\right|, \quad (x,y) \in D_I$$

where

$$T(x_i, y_j)' = T(x_{i+1}, y_j) + T(x_{i-1}, y_j) + T(x_i, y_{j+1}) + T(x_i, y_{j-1}) + \frac{h^2\phi(x,y)}{k}$$

The loss function kernel then looks like,

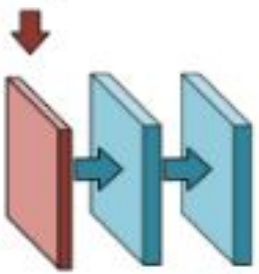


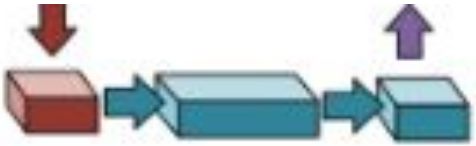

added to the temperature contribution from the heat sources.

The form of the loss function is consistent with Jacobi iteration. In each back propagation, the loss function guides the network to predict the temperature field approximating the value of the next iteration.

# CNN Architecture

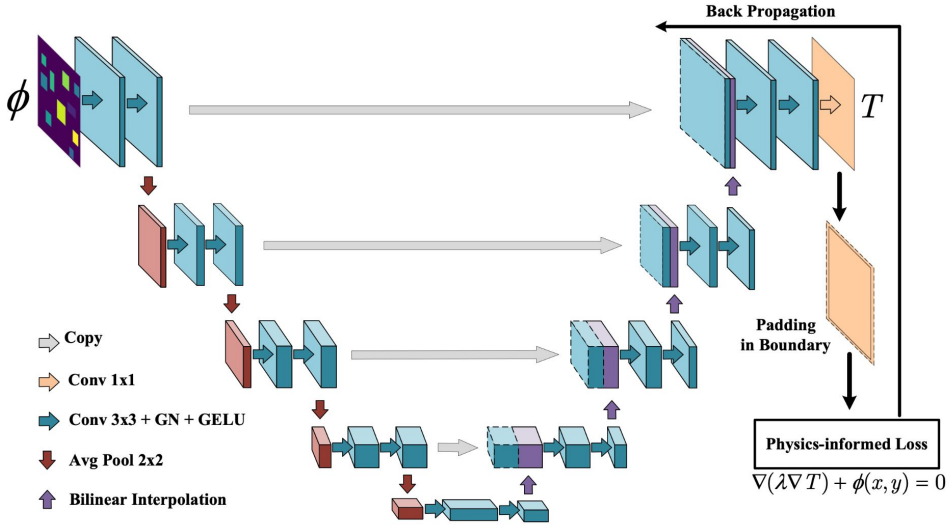
- Multi-layered U-Net style CNN architecture
- Built off of sets of encoding and decoding block with a flatten block in the middle
- Encoding Block: 2 sets of convolutions, normalizations, and activation functions. Pooling done between layers.
- Decoding Block: Same steps as Encoding Block, but with varying “middle channels” instead of same throughout and concatenated to the encoding block output.
- Flatten Block: “Transfer block” between encoding and decoding.



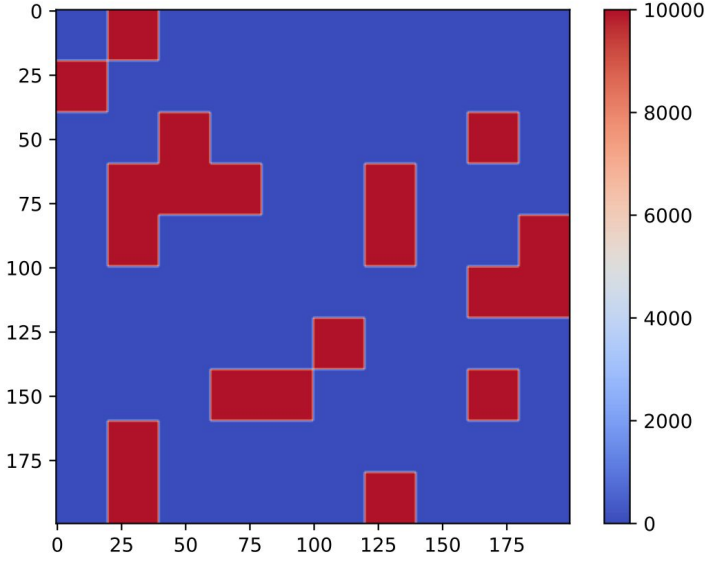


# U-Net Architecture

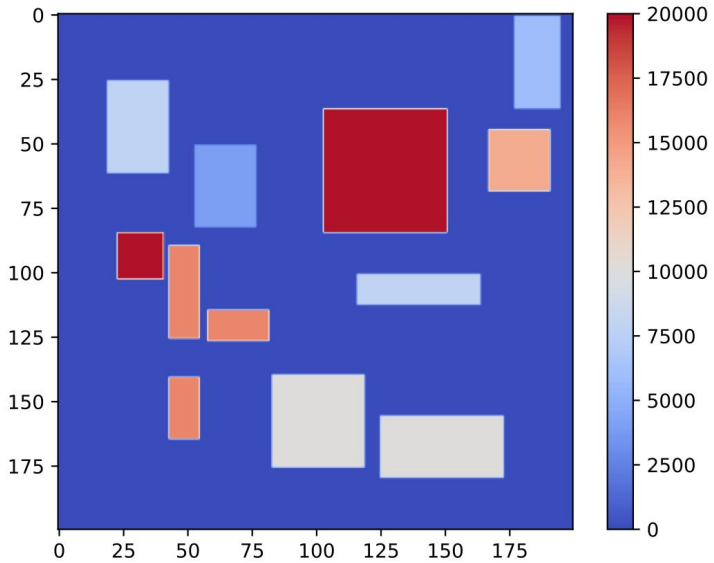
- Using layers of the encoding and decoding blocks, we built up a U-Net architecture very similar to the one shown in class.
- Our U-Net had 4 layers and a flatten block.
  - Encoding channel changes: 3 - 64 - 128 - 256- 512
  - Center channels: 512 - 1024 - 512
  - Decoding channel changes: 1024/512 - 512/256 - 256/128 - 128/64



# Dataset




(a) layout of case 1



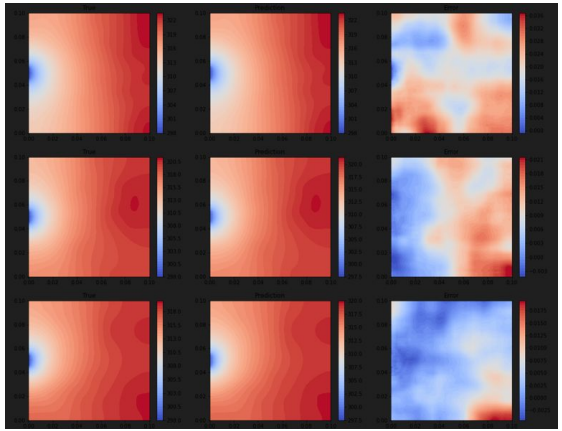

(c) layout of case 2

200 x 200 pixels

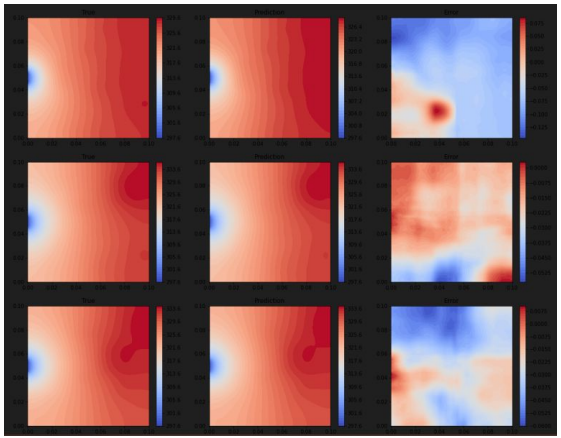
# Results

- Our results using unsupervised learning were well within our expected limit and close to the results of the paper we are looking at.
- We used MAE (mean absolute error) across every run to validate, while also checking CMAE - MAE on the components instead of over the whole domain.

<u>Simple Layout</u>
MAE - 0.0108
CMAE - 0.0106
Max - 0.0269



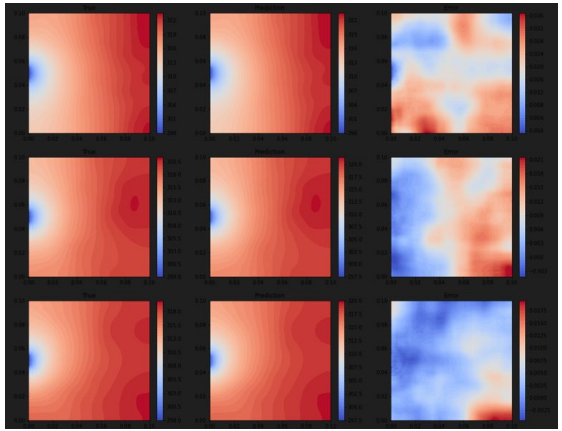
<u>Complex Layout</u>
MAE - 0.0262
CMAE - 0.0262
Max - 0.0752



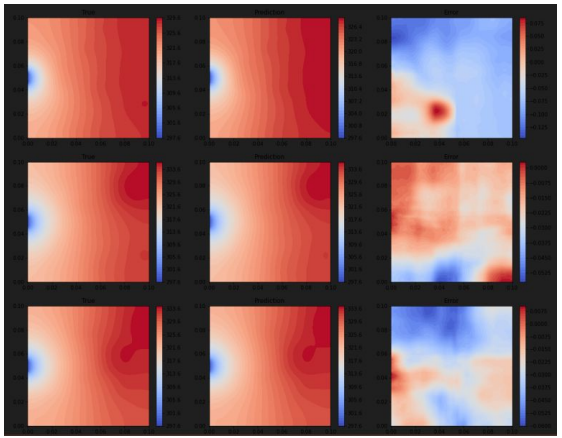
# Unsupervised Learning

- Our results using unsupervised learning were well within our expected limit and close to the results of the paper we are looking at.
- We used MAE (mean absolute error) across every run to validate, while also checking CMAE - MAE on the components instead of over the whole domain.

<u>Simple Layout</u>
MAE - 0.0108
CMAE - 0.0106
Max - 0.0269



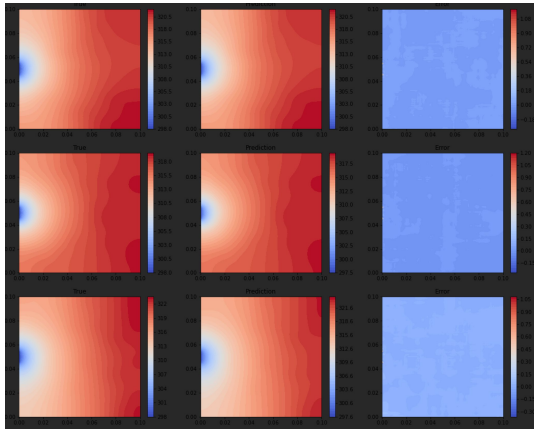
<u>Complex Layout</u>
MAE - 0.0262
CMAE - 0.0262
Max - 0.0752



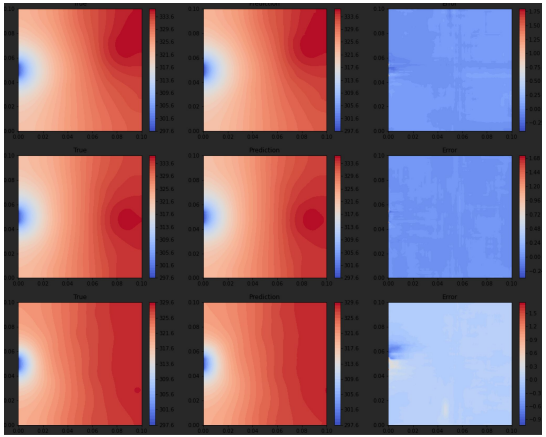
# Supervised Learning

- Structure of error-map much simpler for SL than UL.
- SL predictions are consistently ~0.25K too cold as compared to the FDM truth.
    - Open question: Why is it biased towards colder temperatures?

<u>Simple Layout</u>
MAE - 0.0058
CMAE - 0.0061
Max - 1.164

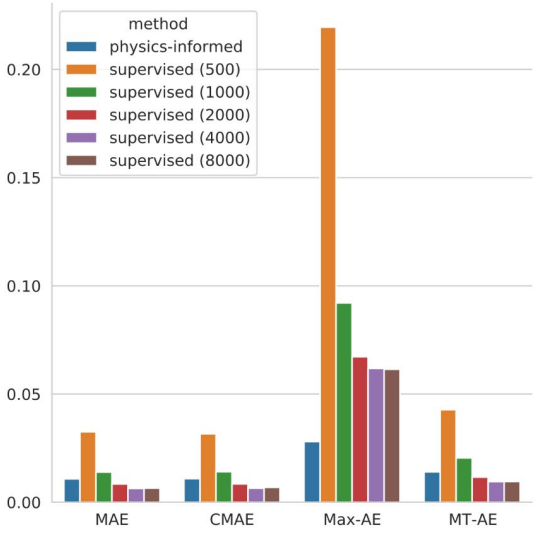


<u>Complex Layout</u>
MAE - 0.0179
CMAE - 0.0178
Max - 1.844

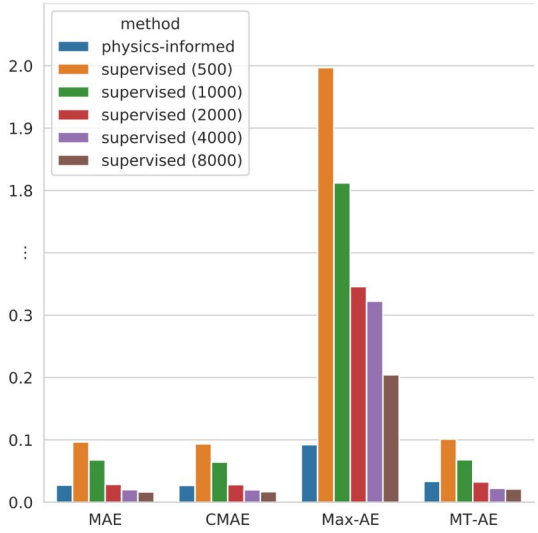


# Comparison with Supervised Learning

- UL performance between 2k-SL and 4k-SL.
- Max-AE worse for all sizes of SL, suggests SL learns global approx., but has little accuracy locally.



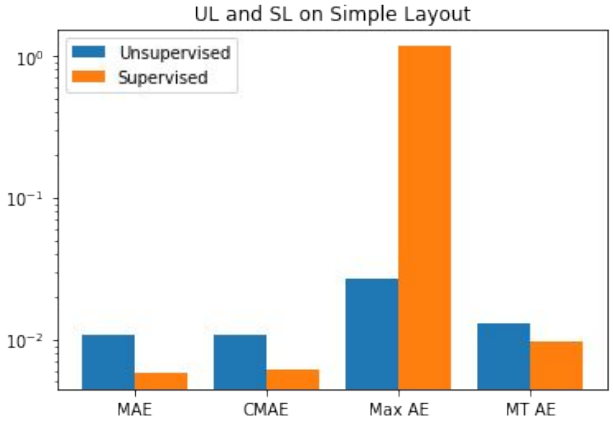

(a) case 1

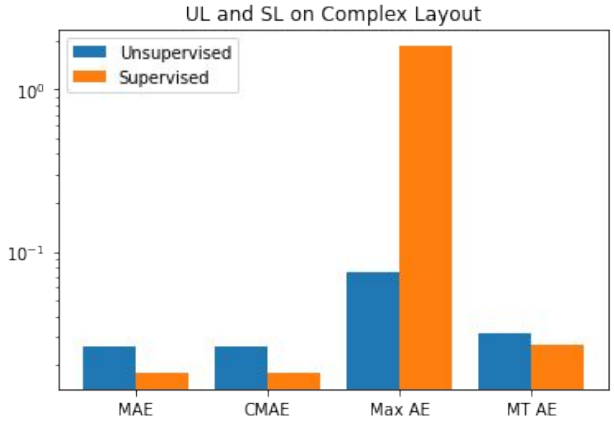



(b) case 2

# Comparison with Supervised Learning

- UL performance between 2k-SL and 4k-SL.
- Max-AE worse for all sizes of SL, suggests SL learns global approx., but has little accuracy locally.



# Summary and Conclusion

- We create a physics-informed CNN for temperature field prediction given a heat source layout.
    - A loss function derived from the heat difference equation guides the network to map intensity distributions to solution functions without any labelled data.
    - Impose hard Dirichlet and Neumann BCs to make the network converge.
    - Use a model based on the UNet framework.
- Performance of unsupervised learning model is similar to data-driven and numerical techniques (MAE < 0.3 K)
- The method can be generalized to other PDEs as well as more complicated domains.

# Work in Progress

- Implement the model in Tensorflow and compare.
  - Some of torch functionality is not available in tensorflow, so we do some hand-waving.
    - For e.g. instead of reflected padding (only in torch), we use zero padding in tf.
  - Will be interesting to benchmark the two frameworks for PINNs
  - Hand-waving is just an ablation study.

# Team Member Contributions

- Peter
  - U-Net building
  - UL & SL implementation
- Thomas
  - UL implementation
  - TF port
- Wenzhong
  - FDM
  - Loss Function
- Aashay
  - Overall coordination
  - FDM
  - Loss function
  - TF Port

# Backup

# Jacobi Iteration

Jacobi iteration is a numerical method used to solve linear equations by iteratively approximate solution to the system of equations by successively updating each component of the solution based on the current value.

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (L + U)\mathbf{x}^{(k)})$$

# Performance Metrics

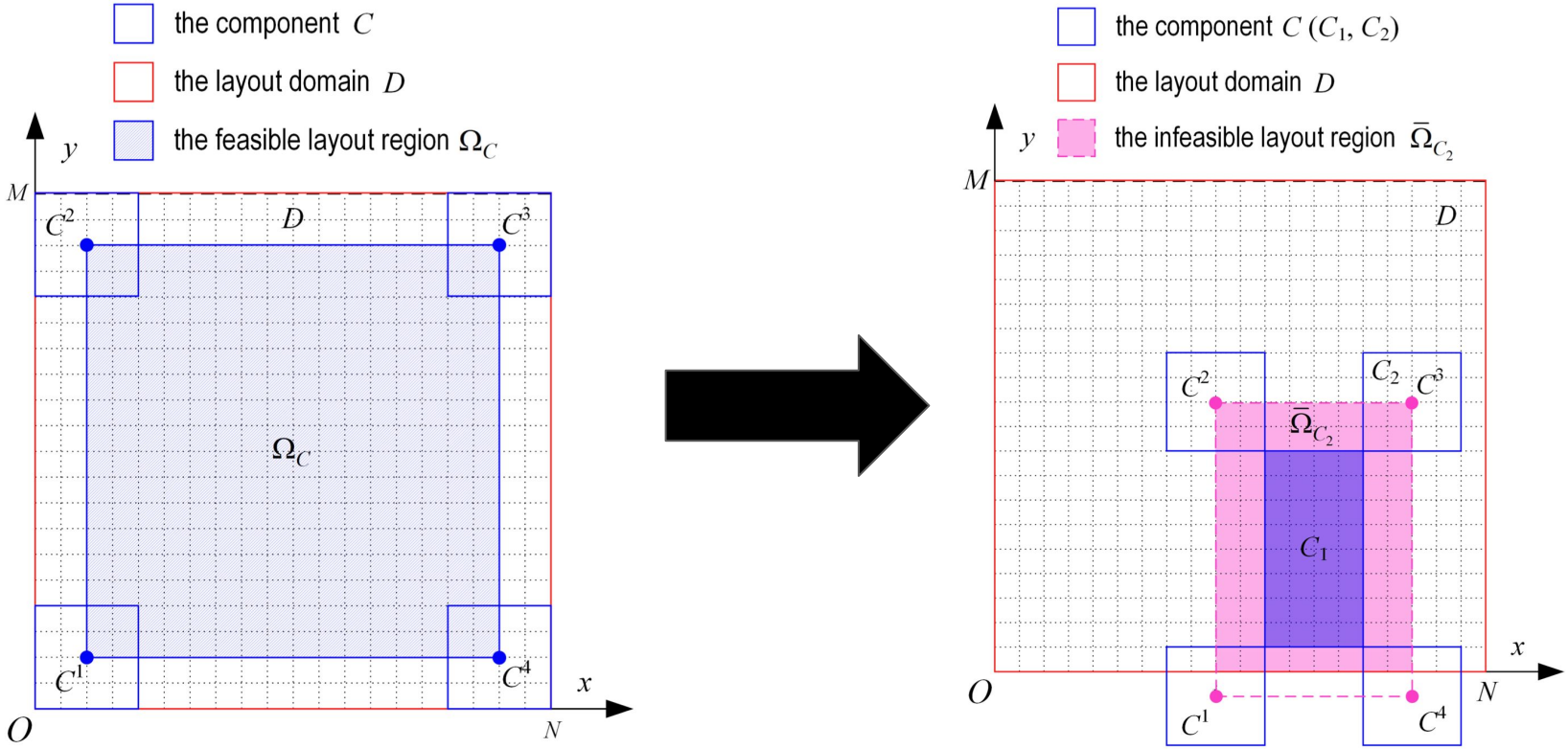
$$\text{MAE} = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |\hat{Y}_{ij} - Y_{ij}|$$

$$\text{CMAE}_i = \frac{1}{sum(M_i)} \sum \left( M_i \otimes |\hat{Y} - Y| \right)$$

$$\text{Max AE} = \max_{i,j} |\hat{Y}_{ij} - Y_{ij}|$$

$$\text{MT-AE} = |\max(\hat{Y}) - \max(Y)|$$

# Data Generation



# Data Generation