Multi Scale flow in Porous media

SAMBa IRP

Amin Sabir

Team members: Sam Williams, Kamran Arora

January - May 2024

Contents

1 Introduction **3**

2 Mathematical problem setup and motivation **3**
2.1 Problem setup . . . 4
2.2 Motivation for neural networks . . . 4

3 Methodology **5**
3.1 Fourier neural operators (FNO) . . . 5
3.2 Convolution neural networks (CNN) . . . 7

4 Results **8**
4.1 FNO: Fibre images to velocity maps . . . 8
4.2 CNN toy example: Random velocity maps to mean velocity . . . 9
4.3 CNN: velocity maps to $\mathbf{K}$ . . . 10

5 Conclusions and future work **11**
5.1 Continuing this research . . . 11
5.2 Connection between the strands . . . 11

A Code availability and model descriptions **14**
A.1 Parameter descriptions . . . 14
A.2 Model descriptions . . . 14
A.3 FNO model . . . 14
A.4 Mean velocity CNN model . . . 15
A.5 Multi-scale flow CNN model . . . 15

1 Introduction

Fluid flow through porous media is crucial in various industry problems including nuclear waste disposals, oil and gas recovery and composite material manufacturing [Pesavento et al. (2017)]. In this project, we are concerned with the case of composite materials with liquid composite moulding (LCM) technology [Bodaghi et al. (2019)] and predicting the permeability of these materials. A composite material is formed by combining other materials with different properties (see Figure 1) which enhances the overall material structure and makes it versatile to use. Common examples include reinforced concrete and carbon fibres [Clyne and Hull (2019)].

These materials are porous and have a property called permeability which indicates the ability of fluids to flow within the material medium. This property gives valuable information for the manufacturing process and in the case of LCM, where to inject resins within the material for their structural integrity and industrial use. Unfortunately, macroscopic permeability cannot be experimentally measured in these materials, so there has been research into predicting permeability variations while trying to reduce the possibility of defects in such materials.

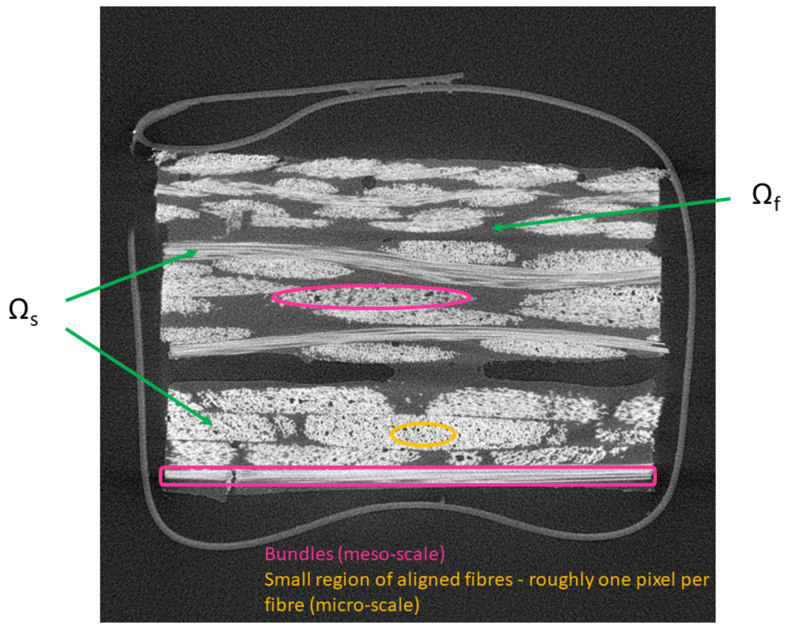
Ω_f
Ω_s
Bundles (meso-scale)
Small region of aligned fibres - roughly one pixel per fibre (micro-scale)

Figure 1: 2D slice of X-ray CT scan of a composite material. Bundles are outlined in pink. Fibres are outlined in orange. Regions of fluid flow are labelled in green.

Motivated by Chen (2023), where a fast Fourier transform (FFT) method was developed for the dual-scale flow problem (Section 2), this project aims to build on this work to help capture the uncertainty in the fibre orientations and help predict macroscopic permeability of a composite material. We attempt to do this in three ways:

1. Encapsulate the variation in local fibre orientation via a distribution and use this to generate random realisations of a composite material.
2. Implement a faster solver based on Finite Element Methods (FEM).
3. Construct a neural network (NN) to predict macroscopic permeability, given image data of the composite material.

The main focus of this report will be the last strand, in constructing a neural network to predict the macroscopic permeability for given fibre orientation data of the composite materials. The report is outlined as follows, there is an overview of the mathematical setup of the multi-scale flow problem with the motivation behind using a neural network (Section 2). The fundamentals of the neural networks and how they relate to the problem (Section 3) are then explored with results comparing against the performance of Chen's FFT solver (Section 4). Finally, we end with the future work for this strand, as well as other strands and how these all fit together in moving forward with the research (Section 5).

2 Mathematical problem setup and motivation

This project stems from the work done in Chen (2023), in which the author developed a numerical method to find the *macroscopic permeability,* $\mathbf{K}$, of a composite material based on a Fast Fourier Transform (FFT).

The composites we concern ourselves with can be considered at three scales:

- In the *micro-scale*, we have individual material fibres placed next to each other.
- In the *meso-scale*, the collections of fibres form *bundles*, oriented in different directions.
- In the *macro-scale*, the bundles are weaved together to create a dry composite material, into which resin can be injected.

2.1 Problem setup

Following Chen (2023), flow through a composite material can be separated into a clear region, Ω_f, and a porous region, Ω_s. This is the dual-scale setup of the multi-scale flow, where we consider flow going into the meso-scale and macro-scale only (Figure 2). We consider incompressible flow modelled via the Stokes equations in Ω_f and governed by Darcy's law in Ω_s.

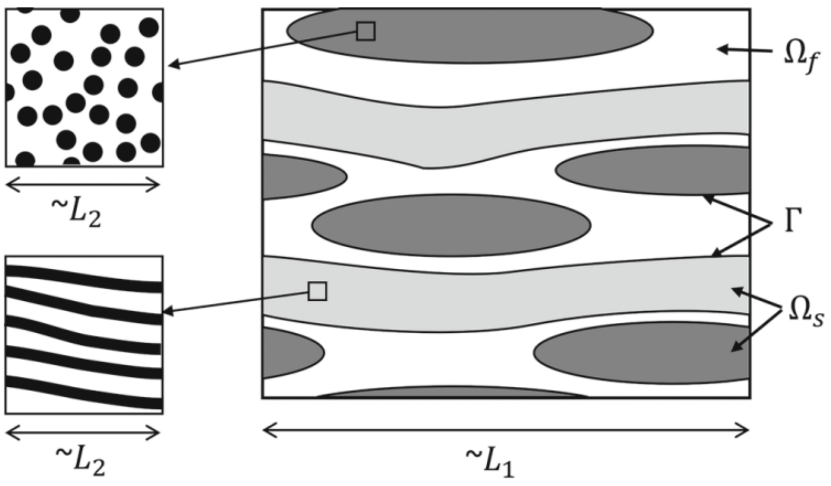

Figure 2: Schematic illustration of a dual-scale permeable domain, including clear fluid region (Ω_f) and permeable solid regions (Ω_s) [Chen (2023)]

A Brinkman term, β, is present in the porous region to account for local permeability within the bundle. By unifying the domains into $\Omega := \Omega_s \cup \Omega_f$, the flow can be described by the Stokes-Brinkman partial differential equation (PDE) [Durlofsky and Brady (1987)]

$$\begin{cases} \varphi(x)\Delta u(x) - \beta(x)u(x) - \nabla p(x) = 0, \\ \nabla \cdot u(x) = 0, \end{cases} \quad x \in \Omega. \tag{2.1}$$

Here we have defined

$$\varphi(x) = \begin{cases} \mu, & x \in \Omega_f, \\ \mu_e, & x \in \Omega_s, \end{cases} \quad \text{and} \quad \beta(x) = \begin{cases} 0, & x \in \Omega_f, \\ \mu \mathbf{k}_s^{-1}, & x \in \Omega_s, \end{cases} \tag{2.2}$$

where μ is the dynamic viscosity, μ_e is Brinkman's effective viscosity and $\mathbf{k}_s$ is the local permeability tensor. We also take periodic boundary conditions on the pressure and velocity.

Given a macroscopic pressure gradient $G = \langle \nabla p \rangle_\Omega$ (a loading condition), the PDE can be solved for a velocity field u. The macroscopic velocity $U = \langle u \rangle_\Omega$ can then be computed. Here $\langle \cdot \rangle_\Omega$ is the average value over the domain Ω. Darcy's law in macroscopic form relates these two quantities using the macroscopic permeability tensor $\mathbf{K}$ via

$$U = -\frac{1}{\mu}\mathbf{K} \cdot G. \tag{2.3}$$

Therefore solving the PDE for u allows us to calculate the macroscopic permeability. Conversely, we can load the problem with the macroscopic velocity and compute the macroscopic pressure gradient from the solution of the PDE. Darcy's law in macroscopic form can then again be used to compute $\mathbf{K}$.

Often (2.1) is difficult to solve analytically depending on the setup. This is the case with our setup as we have different fibre orientations in Ω, coupled with the periodic boundary conditions, making it difficult to obtain an exact solution. This leads us to consider solving this problem computationally with Strand 3, using neural networks.

2.2 Motivation for neural networks

Given image data of composite materials, our goal is to estimate their respective macroscopic permeability tensors, $\mathbf{K}$. Chen (2023) proposes a PDE solver with the use of FFT. This solver is computed several times for different orientations of the fibres in the material. This method requires extensive use of a high-performance computing (HPC) system. Due to its large computational cost, we want to use the HPC system effectively and consider a more efficient alternative approach.

Improving the accuracy of this FFT solver via classical numerical methods can impose a large computational cost and there is a trade-off as a result. Deep learning models involving neural networks (NNs) can drastically reduce the computational time required and are used in many cases such as the use of a deep surrogate model on PDEs [Deveney et al. (2019)]. It has been shown for many classes of ODEs and PDEs that the number of parameters required for NNs are significantly smaller and can still achieve similar or better results than for classical numerical methods [Lagaris et al. (1998), Bhattacharya et al. (2021)].

NNs are computational models formed of interconnected neurons or nodes organised in layers. NNs can be trained to learn patterns in data from their input layers and to adjust the connections (weights) in the intermediary (hidden) layers to produce a meaningful output for the final layer. Along the way, information is processed with linear operators and non-linear activation functions.

For solving the Stokes-Brinkman problem, the governing equations (2.1) are non-linear PDEs with complex dynamics. NNs have shown great success approximating non-linearity and complicated relationships through the use of mathematical operations and non-linear activation functions to establish such relationships. NNs are capable of handling high-dimensional data, which in our case would be the velocity and pressure fields in 2D and 3D domains. This allows the NN to capture the intricate flow patterns and variations in the porous media [Almajid and Abu-Al-Saud (2022)].

Once trained on reasonable data, NNs can generalise and perform well on unseen data, adapting to different problem settings and domain geometries. This is particularly useful in the Stokes-Brinkman problem where there could be different boundary conditions and porous media properties that could change from the setup in (2.1). Once a robust NN is established for our problem with periodic boundary conditions, it could be adaptable to different setups where some numerical methods may struggle [Peng et al. (2021)].

We see that neural networks have been very powerful in various scenarios, motivating us to implement them for our problem setup.

3 Methodology

In this section, we discuss the setup of the neural networks, namely how we can combine two exciting deep learning approaches in tackling the problem. We first discuss the method setup and then explore the two approaches.

Here we have a two-part setup (Figure 3):

1. Map the image data of the fibres directly to their respective velocity maps
2. From their velocity map calculate the macroscopic permeability tensor **K**,

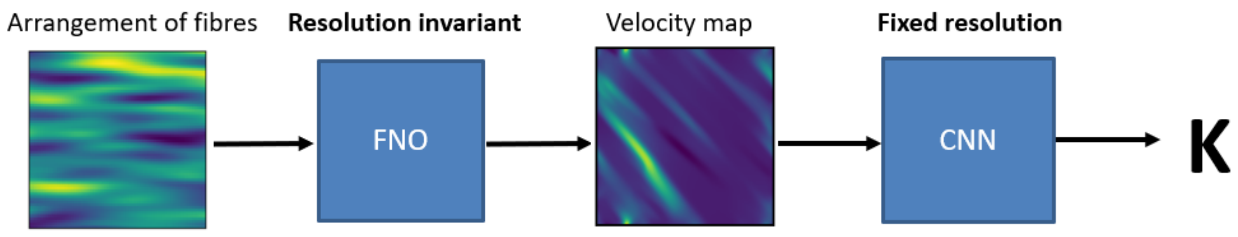

Figure 3: Plan to express permeability tensor **K** for a given fibre image

Both of these methods are supervised learning problems. For 1, we have the inputs as the images of the fibres and the output as their velocity maps. For 2, we take these velocity maps as the inputs and their outputs as **K**. The inputs and ground truth outputs are taken from Chen (2023) FFT solver. These approaches will involve the use of model architectures called a Fourier neural operator (FNO) and a convolutional neural network (CNN), explained further in Sections 3.1 and 3.2.

We note here that there is a slight modification from what we originally planned during Semester 1. Previously, we proposed using the image data to calculate **K** solely by the use of a CNN and with the scope to do it alternatively with a FNO. However, Chen recently developed an FNO that could reasonably complete the first task (1), so we wanted to build on top of this with the CNN. The advantage of doing this is further discussed in Section 3.1, with the main idea that FNO is resolution-invariant where once trained on one resolution can predict different resolution data. This setup can be used to produce a better dataset to use for CNN after.

3.1 Fourier neural operators (FNO)

For the first part (1) we have an image-to-image problem. This can be tackled by a promising neural network approach called neural operators (NOs). NOs are neural networks that learn the solution operators for PDEs [Li et al. (2020b)]. They take the initial or boundary conditions of a solution and the network tries to output a candidate solution that solves the PDEs directly.

Standard NN approaches like a CNN, map from an infinite to a finite dimension space and are mesh-dependent. The main advantages of NOs over standard NN approaches are that NOs are mesh-independent and learn a mapping or link between two infinite-dimension spaces, from a finite collection of observations of input-output pairs for this mapping. Unlike standard NNs that are fixed on the discretisation of training data, NO can adapt to various discretised data without needing it to be retrained. Note that numerical methods such as FEM are mesh-dependent and we will discuss later how Strand 2 with FEM could be used as training data for the mesh-independent FNO.

FNO's regime is beneficial, however because of its resolution-invariant nature, the training process is often slow but in the long-term can be more fruitful than the standard NNs as discussed.

3.1.1 Framework of Neural Operators

Standard NNs consist of linear transformations and non-linear activation functions. Similarly, NOs consist of linear operators and non-linear activation operators. Let v be the input vector (image of fibres) and u be the output vector (velocity map). Then we have

$$u = (K_n \circ \sigma_n \circ \ldots \circ K_2 \circ \sigma_1 \circ K_1)\, v \tag{3.1}$$

for n layers of a neural network where K_i is the i^{th} convolution (linear) layer and σ_i is the i^{th} activation function for $1 \leq i \leq n$.

For NO, it is a similar formation but K is seen as a kernel-convolution integral operator formulated from (2.1)

$$(\mathcal{L}_a u)(x) = f(x) =: \nabla p(x), \quad x \in \Omega \tag{3.2}$$

where $\mathcal{L}_a(u)(x) := \varphi(x)\Delta(u(x)) - \beta(x)u(x)$ with $a(x) = g(\varphi(x), \beta(x))$ for some function g. Under general conditions, we can define a Green's function $G_a : \Omega \times \Omega \to \mathbb{R}$ which solves (3.2). Therefore

$$u(x) = \int_\Omega G_a(x, y) f(y)\, dy. \tag{3.3}$$

Here G_a is continuous where $x \neq y$, since $\mathcal{L}_a$ is uniformly elliptic. We can replace G_a with a kernel function κ that takes inputs $x, y, a(x)$ and $a(y)$. The solution for the velocity over the clear fluid and porous regions Ω from (2.2) becomes the kernel-convolution integral operator

$$u(x) = \int_\Omega \kappa(x, y, a(x), a(y)) f(y)\, dy. \tag{3.4}$$

We then use (3.4) as an iterative solver algorithm forming the NO. Note we lift $u(x) \in \mathbb{R}^d$ to a high dimensional representation $v(x) \in \mathbb{R}^n$ through a standard feed-forward neural network where n is the dimension of the hidden representation. We see this alongside the full NO steps in Algorithm 1 and Figure 4.

Algorithm 1: Neural operator algorithm

```
Initial velocity $u_0(x) = (x, a(x))$ where $a(x) = g(\varphi(x), \beta(x))$
Lift by NN: $v_0(x) = P(x, a(x))$ where $P$ is a feed-forward NN
For $t = 0, 1, \ldots, T-1$.
    $v_{t+1}(x) = \sigma\left(W v_t(x) + \int_\Omega \kappa_t(x, y, a(x), a(y)) v_t(y)\, dy\right)$, where $\sigma$ is an activation function
Projected back by NN: $u(x) = Q(v_T(x)))$ where $Q$ is a feed-forward NN
```

The parameter T represents the total number of layers in the FNO, which is specified in the results section (Section 4 and Appendix A). Note here that W is the bias term.

The kernel-convolution integral can be very complex and time-consuming to compute. The inputs and outputs of PDEs are continuous functions. Using a local convolution kernel (from a CNN) would discretise the functions, not capturing their full behaviour. On the other hand, these continuous functions with a convolution kernel in Fourier space are more efficient to use. This is where the Fourier transforms come in; forming the Fourier neural operator (FNO). Other formulations of the NO are available including graph and wavelet NOs [Kovachki et al. (2023)] but due to the setup of the problem with periodic boundary conditions and the non-linear PDE nature, the FNO is chosen.

3.1.2 Fourier layer

We have the kernel-convolution integral operator in Algorithm 1

$$(K(a)v_t)(x) := \int_\Omega \kappa_t(x, y, a(x), a(y)) v_t(y)\, dy, \quad 0 \leq t \leq T-1, \quad \forall x \in \Omega. \tag{3.5}$$

If we let $\kappa_t(x, y, a(x), a(y)) = \kappa_t(x - y)$ where t represents the $(t+1)^{\text{th}}$ layer, we can use the convolution theorem where (3.5) in the spatial domain is equivalent to point-wise multiplication in the Fourier domain. From Figure 4(b), we see this in action following three steps:

1. Fourier transform ($\mathcal{F}$) on $v_t(x)$
2. Linear transform (R) on $\mathcal{F}(v_t)(x)$ to preserve the lower Fourier nodes
3. Inverse Fourier transform ($\mathcal{F}^{-1}$) on $R \cdot \mathcal{F}(v_t)(x)$ to get back to the spatial domain

Therefore (3.5) becomes

$$(K(a)v_t)(x) = \mathcal{F}^{-1}\left(R \cdot \mathcal{F}(v_t)\right)(x), \quad \forall x \in \Omega. \tag{3.6}$$

A bias term Wv and a non-linear activation function σ are applied to the result in the spatial domain to help recover higher frequency nodes that are lost by R in the Fourier layer. This Fourier construction helps speed up the overall algorithm and these Fourier layers are applied iteratively to generate a candidate velocity field u (Figure 4). We see these results in Section 4.1.

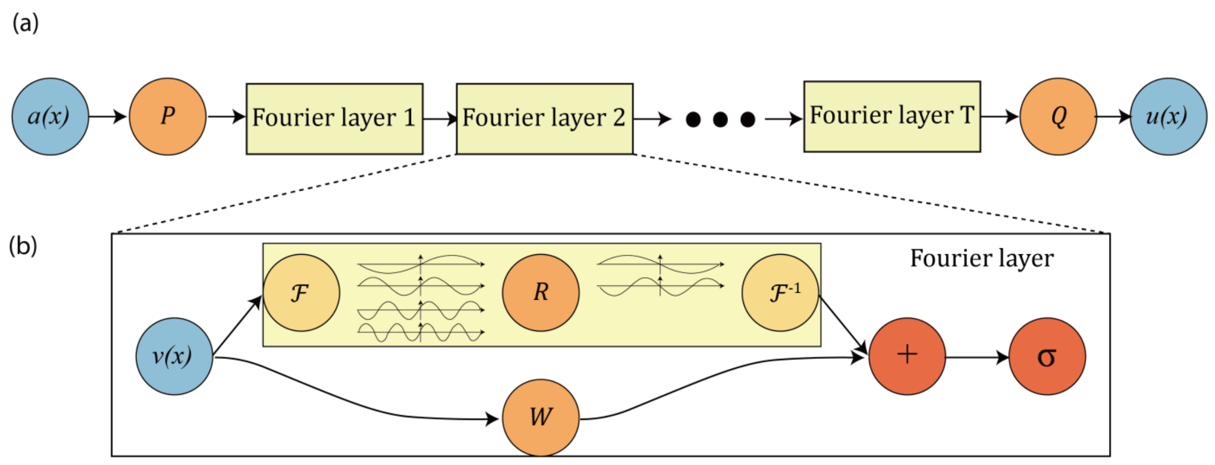

Figure 4: structure of a typical FNO - **(a)** architecture of a neural operator (P, Q are the feed-forward NNs), **(b)** Fourier layer [Li et al. (2020a)]

3.2 Convolution neural networks (CNN)

For (2), once a reasonable FNO has been constructed, we have an image-to-tensor problem. Where we are going from the velocity map to their respective macroscopic permeability tensor. We use a CNN for this task and in this section, we discuss the general setup of CNNs and optimisation techniques to improve overall performance. Later in the results (Section 4), we see the use of CNN in action for a toy example with mean velocity and the multi-scale flow problem.

3.2.1 CNN setup

In a CNN, there are 3 main types of layers used, see Figure 5.

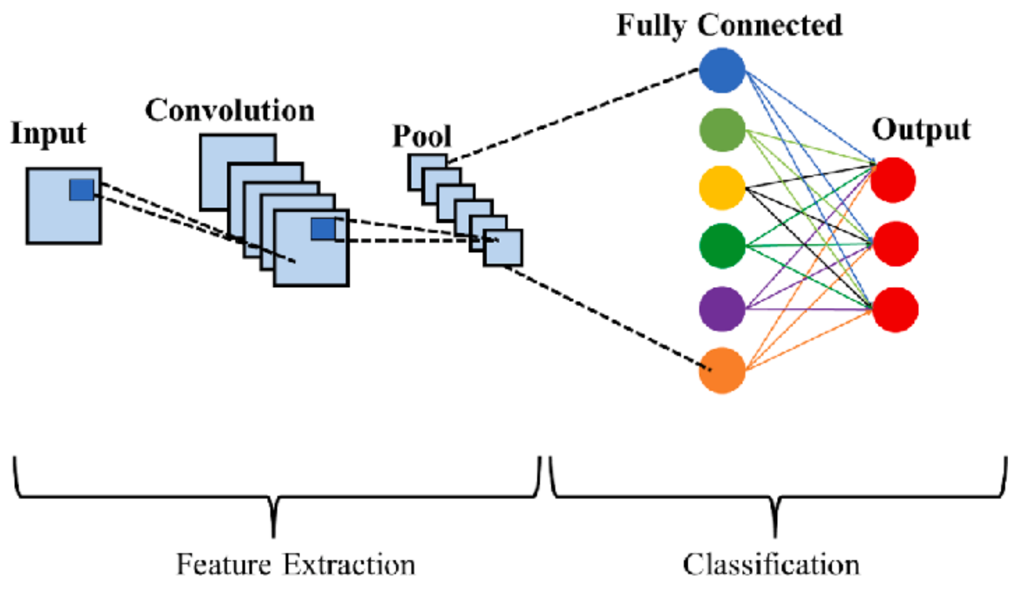

Figure 5: General structure for a CNN [Hadi et al. (2023)]

1. **Convolutional layer**
   This layer uses a filter or a kernel that is applied to the image input to extract particular properties (a feature map) based on what the filter is trying to find. For example, it could be detecting edges or patterns of an image. In our case, this is extracting the relevant properties to form $\mathbf{K}$. At the start of training these filters are randomly initialised and through backpropagation on the training set, the filters adapt to learn the structure they need to find [Narkhede et al. (2022)]. The layer creates a feature map for that image and is the first layer of the CNN.

2. **Pooling layer**
   Often placed between convolutional layers, this layer receives several feature maps and applies a pooling operation to each of them. This pooling operation essentially takes away the irrelevant data from a feature map, whilst preserving the important characteristics that will be needed later down the line. By doing this we reduce the number of parameters and calculations in the network. There are multiple pooling techniques including maximum pooling and average pooling, which either take the maximum value across a cell region or the average of all the values. Here we use max pooling [Zafar et al. (2022)]. Non-linearity is then introduced at the end via activation functions on the map. This allows the CNN to adapt well to different data more easily compared to linear models such as linear regression. Typically for CNNs, a Rectified Linear Unit (ReLU) activation function [Ide and Kurita (2017)] is used, so all negative values of the map become zero and we maintain only the positive values for the map.

3. **Fully-connected layer**
   This final layer, which is often the last layer of a NN as well as a CNN, is the fully-connected layer.

It connects all the preceding layer neurons together and is used to classify the image input with a classification label. In the regression model, instead of mapping the images to classification labels, we would map them to numerical values. We use the regression model to compute a **K** tensor, completing the process shown in Figure 3.

Usually there is a series of convolutional and pooling layers together until all the main features have been extracted and then the fully-connected layer is added on at the end.

3.2.2 Optimisation

CNNs can be trained in a variety of different ways to tackle machine learning tasks. For instance to optimise their hyperparameters, methods such as stochastic gradient descent or Adam optimisation [Kingma and Ba (2014)] can be used. To help prevent overfitting, dropout can be used, which randomly shuts down several neurons in the hidden layers, making the network more robust to new data coming in [Srivastava et al. (2014)]. In our case, some of these techniques will be used to improve performance.

4 Results

In this section, we discuss the results of using the methods from Section 3 in helping to tackle our problem. For FNO, a setup is performed on realistic fibre image data and is compared to Chen (2023) FFT solver. Following this we applied the CNN on these FNO velocity maps to generate their respective permeability tensors.

4.1 FNO: Fibre images to velocity maps

With the setup of the FNO (see Appendix A.3), we used the model on 800 fibre image samples each with 64×64 pixels. These images had different Brinkman terms β randomly generated for inputs, see Figure 6 as an example. To train the FNO and compare the performance, Chen (2023) FFT solver was used as a ground truth. Note that the FNO implementation and results were conducted mainly by Chen (2024). The code is available as a GitHub repository, see Appendix A.

There have been some promising results, with a range of different β representing different fibre orientations generating different regions for Ω (Figure 6). Applying the FNO model, we obtain a close comparison between the ground truth and FNO velocity maps (Figure 7). The model's training was very intensive though, requiring at least 15 minutes on an HPC system or at least over $3-4$ hours on a standalone laptop.

Before using a CNN, there was work to go immediately from the velocity map to **K** via the use of Darcy's law (2.3) with the loading of the macroscopic velocity (from the FNO output in x and y directions) and pressure gradient discussed in Section 2.1. This post-processing of the model outputs (64×64 pixels) yielded interesting results for the components of **K** (Figure 8). Although, the diagonal elements of **K** were predicted reasonably well, often the off-diagonal components were greatly underestimated for the 800 samples.

Overall, the performance of the FNO here shows great potential and flexibility.

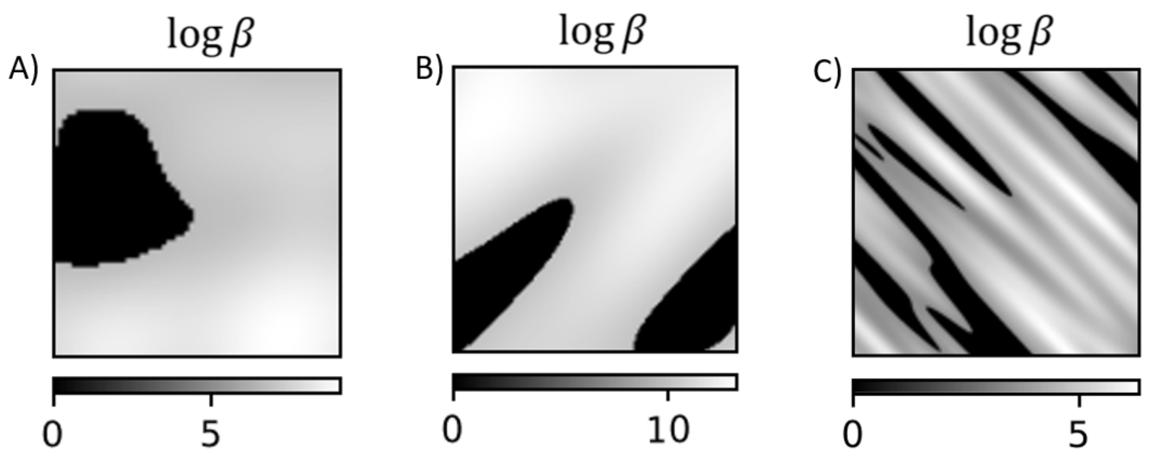

Figure 6: Examples of β input with flow intensity scale in Ω for A), B) and C)

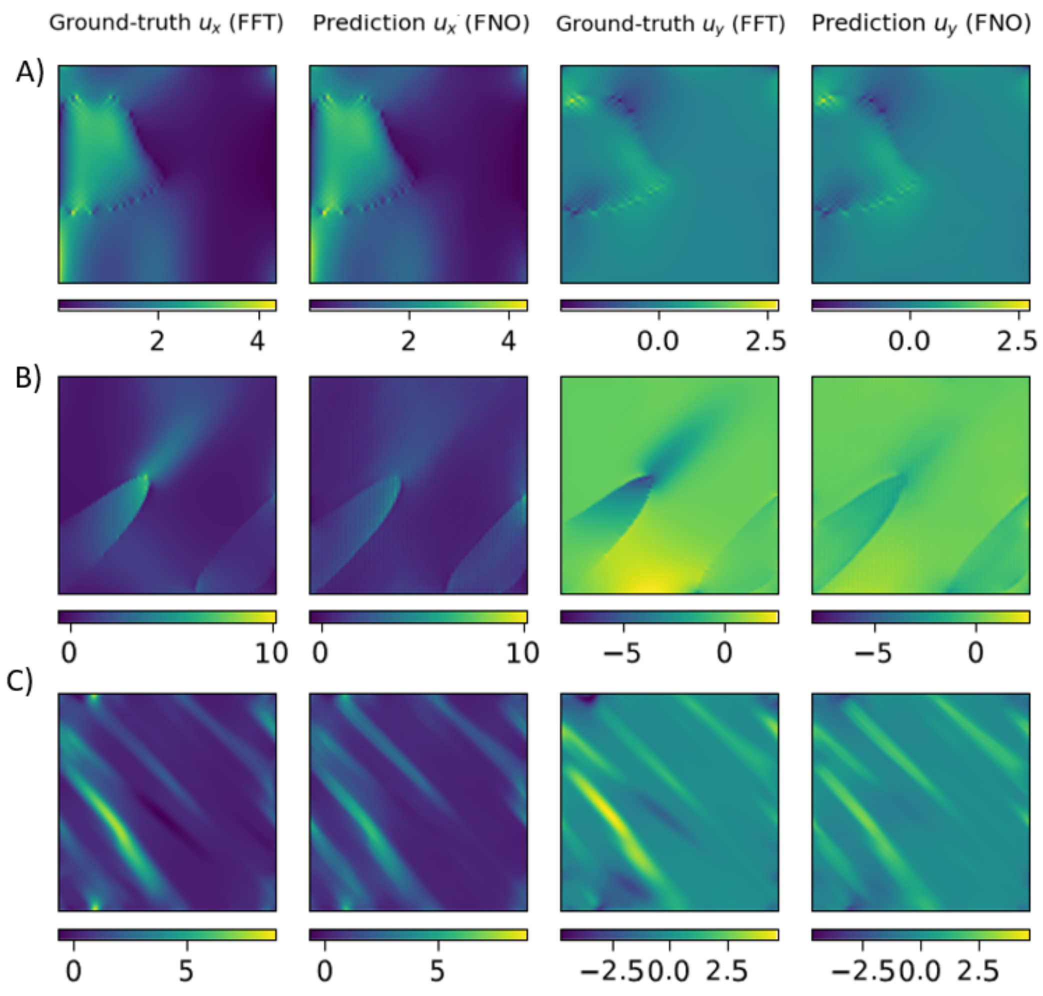

Figure 7: Velocity maps in x and y directions with their respective flow intensity scales in Ω (FFT vs FNO for Figure 6)

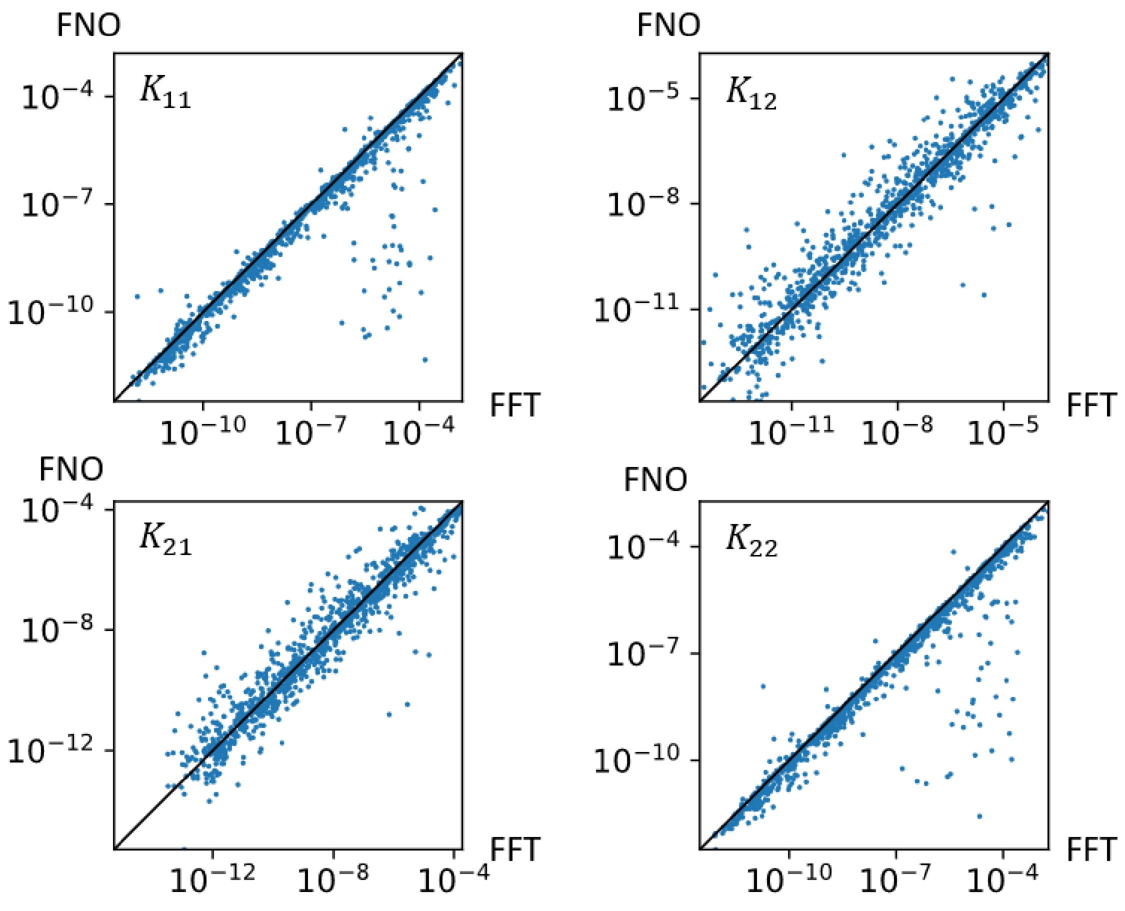

Figure 8: Components of $\mathbf{K}$ tensor for the 800 samples of 64×64 pixels (FNO vs FFT)

4.2 CNN toy example: Random velocity maps to mean velocity

Before tackling the main multi-scale flow problem, we tried the CNN on a simpler problem. We wanted to predict from randomly generated 64×64 velocity maps (Figure 9) their respective mean velocity. Note, here these velocity maps are not physically representative of the fibre orientation structure we have in Figure 6, but it can give us a base CNN model that could be extended for the main problem.

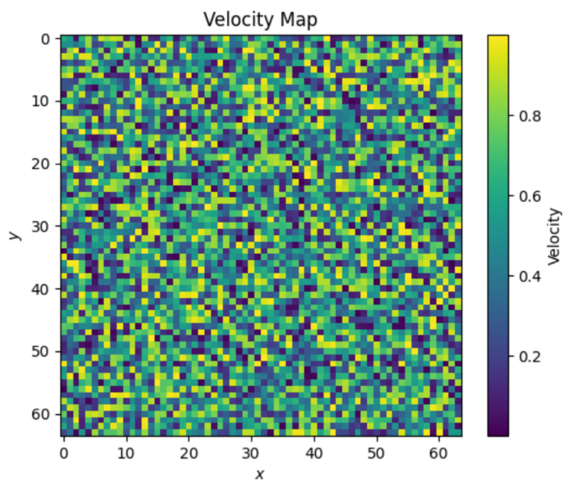

Figure 9: Example 64 × 64 velocity map

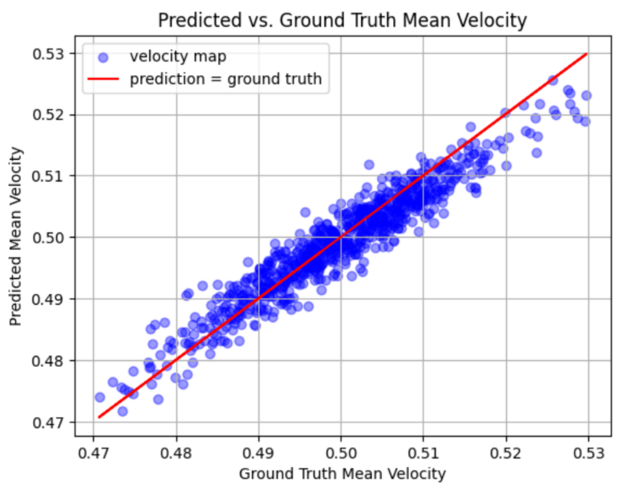

Figure 10: Comparison of 800 velocity maps with their predicted and ground truth mean velocities

We ran the model on 1200 samples (Appendix A). We see the model performs very well in Figure 10 and gives a basis to advance this CNN to model more complex relationships such as our main problem between velocity maps and their permeability tensors.

4.3 CNN: velocity maps to K

Following this toy example, we now focus on the main problem and use the 64 × 64 fibre images that the FNO used in the CNN setup. Similar to the post-processing in Section 4.1, the CNN inputs are the velocity maps for x and y directions. The outputs for $\mathbf{K}$ were normalised. The CNN is constructed with the layers and parameters outlined in Appendices A and A.5. Note these results are more preliminary than Section 4.1.

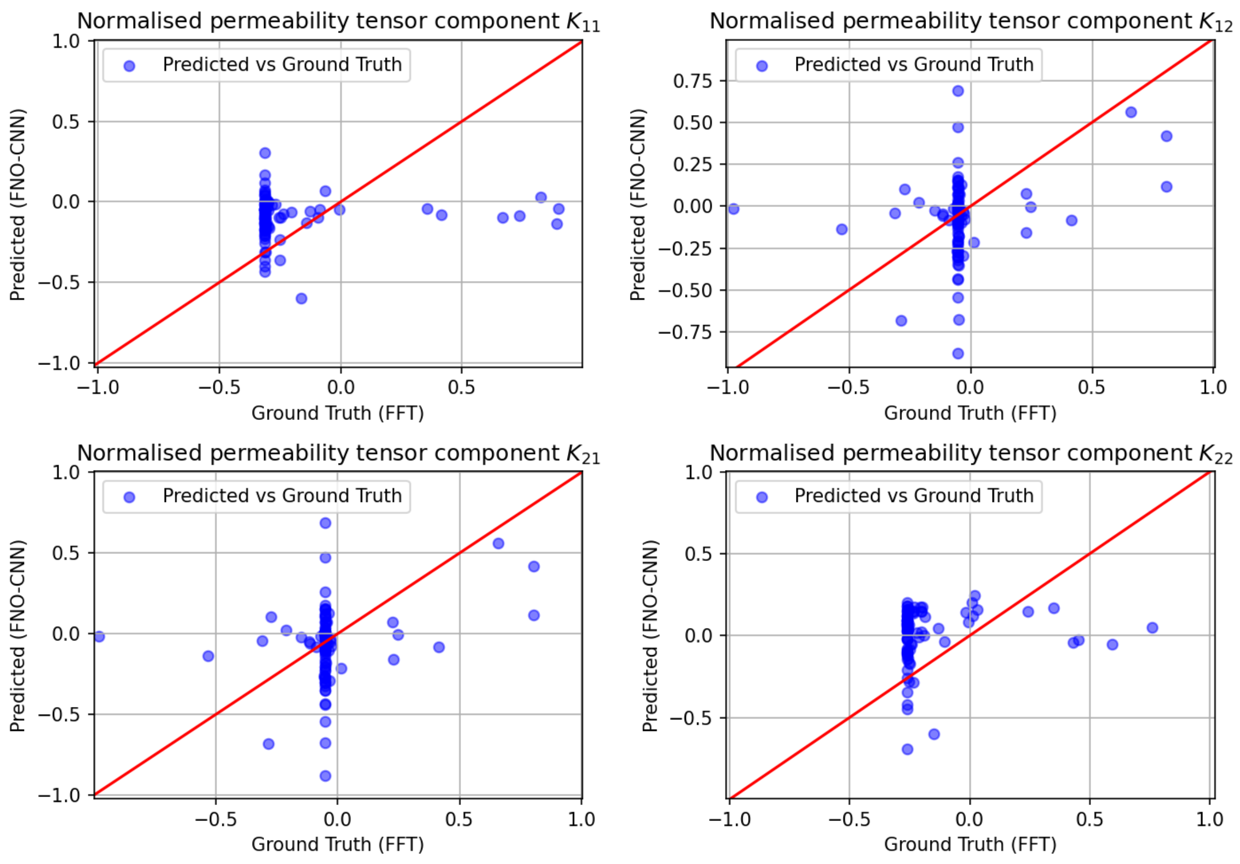

Figure 11: Components of normalised $\mathbf{K}$ tensor for the test 100 samples of 64 × 64 pixels (FNO vs FFT)

We see the results in Figure 11, where although there is a wide spread of predictions, the CNN struggles to predict $\mathbf{K}$ overall. This may be due to the nature of the data not having much variation and the complexity of the multi-scale problem. In Section 5.1, we look at ways to improve this model.

5 Conclusions and future work

In this report, we have discussed how neural networks can be used to help tackle the Stokes-Brinkman problem (2.1) with multi-scale flow for composite materials. In Section 3, we gave an overview of the neural networks that can be implemented for the problem. Following this in Section 4, there have been interesting developments in the FNO and CNN results with multiple directions for future research.

We now look ahead at these directions in both the NN strand and link with the work from the other two strands (1 and 2) for the wider problem.

5.1 Continuing this research

Regarding the NN strand (Strand 3), there are various opportunities to progress it further.

1. **Improve model performance and train/test on different resolution data.**
   For the FNO, the model hyperparameters (Appendix A.3) were chosen in line with an example using a simpler setup containing Darcy's flow [Li et al. (2020a)]. Although the model performance was respectable, we could tailor it to the more complex flow problem, testing a range of hyperparameters and using cross-validation.

   For the CNN, as this was more preliminary and an alternative to the post-processing of FNO (Section 4.1) there is a substantial amount of work to fine-tune the network. The toy example results (Section 4.2) show some promise but learning the non-linear relationship to a permeability tensor is complex. The available data from the FNO velocity map outputs may not be enough for the CNN, so more information may be required to predict $\mathbf{K}$ such as their pressure gradients. To incorporate more of the physics into the CNN, Darcy's law could be embedded in the network architecture to make it more physics-informed.

   Once a working CNN is implemented, we can train and test it out for different resolutions of fibre images. Due to the resolution-invariant of the FNO, we could try to test the FNO on higher resolutions than 64×64 pixels and consequently use this for our CNN to be able to predict more realistic fibre orientation arrangements.

2. **Pairing the FNO and CNN together an HPC pipeline - training as one unit (FNO-CNN).**
   The current setup involves the FNO and CNN being trained separately and independently. This increases the training time and computational cost. An alternative is to merge the training process of both networks and have them set up in a pipeline (FNO-CNN) that works in an HPC setting. How this would be constructed could be complicated, with deeper architectures more prone to expanding/vanishing gradients in calculations. Careful consideration would be required to implement the structure properly.

3. **Extend to different porous media problems with different geometries and consider other data-driven approaches.**
   The periodic nature of the problem (2.1) lends itself well to the use of FNO and FFT, as the Fourier space is well suited for the periodicity of functions. For different porous media problems which are not in a rectangular domain or do not have periodic boundary conditions, the FNO could face difficulties.

   Other neural operators could be used including graph and wavelet versions for these problems [Kovachki et al. (2023)]. A comparison can then be made across them and see which version could be used in each problem. Again, an HPC system would need to be set up to test the capabilities of these neural operators and to implement them individually would be tough.

5.2 Connection between the strands

We start with a brief restatement of this project's goals. Using the numerical method proposed in Chen (2023) as a starting point, our individual strands aim to improve the speed, efficiency and robustness of the solver, introduce randomness to model local fibre orientations and build a neural network capable of predicting macroscopic permeability, $\mathbf{K}$.

5.2.1 Progress in the other strands

In semester 1, we set out a road map (Figure 12) with the grand vision of how all the strands would link together to tackle the multi-scale problem. Throughout the year, there has been progress in conducting this vision.

For Sam's strand (Strand 1), there have been investigations into the fibre orientations for different resolutions of images taken from actual composite materials. Sam has been developing a model to replicate these orientations to specify the points and structures of the fibres through the theory of statistical manifolds and models such as a Gaussian process. The model aims to pinpoint the essential structural points and elements of where realistic fibres would be. Eventually, the goal is to use this model to generate realistic β inputs to use in the NNs instead of the randomly generated ones we use currently.

For Kamran's FEM strand (Strand 2), there has been some progression in developing a FEM solver to rival Chen's FFT solver. Kamran formed a PDE FEM solver in Firedrake [Ham et al. (2023)] converting (2.1) to a linear system and experimented on different problem setups that Chen (2023) proposed including the

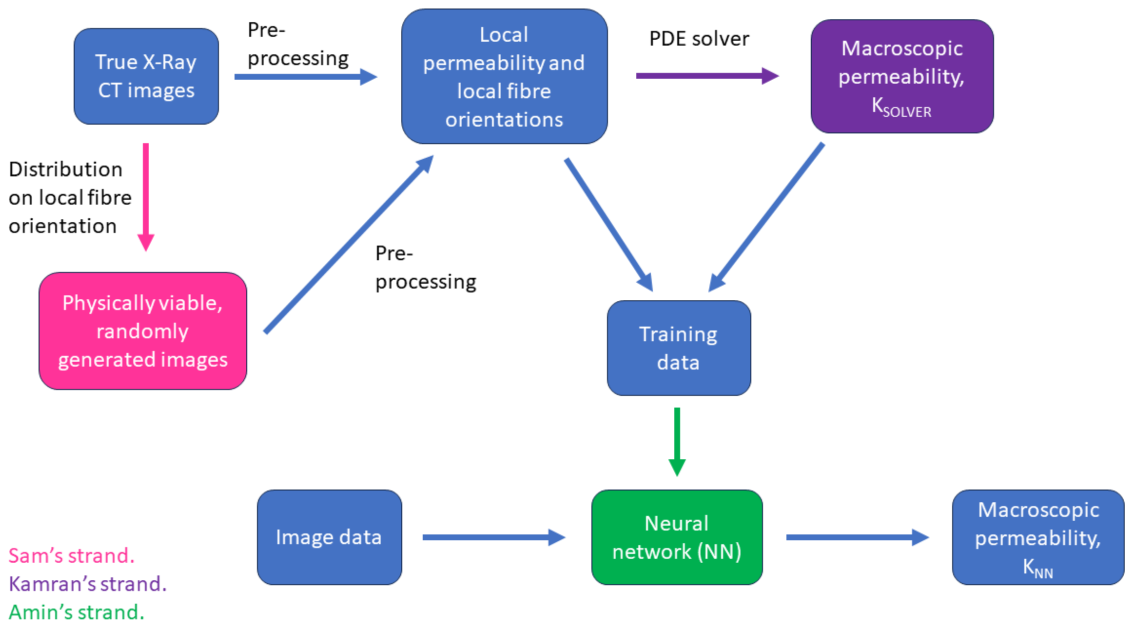
True X-Ray CT images
Pre-processing
Local permeability and local fibre orientations
PDE solver
Macroscopic permeability, K_{SOLVER}
Distribution on local fibre orientation
Physically viable, randomly generated images
Pre-processing
Training data
Image data
Neural network (NN)
Macroscopic permeability, K_{NN}
Sam's strand.
Kamran's strand.
Amin's strand.

Figure 12: Diagram of project elements and strand connections

parallel channels and square hole problems. A comparison was made between the solvers and there is still work to do to quantify which solver performs better overall. Ultimately, once there is a clear distinction as to which solver performs better, this will act as the ground truth to run both the FNO and CNN models.

5.2.2 Continuing research between the strands

Moving forward, there are a variety of tasks to tie in the work across all the strands to reach the goals outlined in Figure 12. The process starts with Sam's strand to Kamran's strand and finally to my strand.

Once a full comparison has been made between Kamran's FEM solver and Chen's FFT solver, the more accurate one will feed the ground truth data for training and testing in the neural networks. There would need to be careful implementation with the current setup of the FNO and CNN with the Firedrake FEM solver if this FEM works better than the FFT.

Upon further development of Sam's model to generate physically viable, random realisations of the composite material, we would want to generate the corresponding values for $\beta(x)$ that encodes the local permeability. These values, along with calculated local fibre orientations, can then be fed into Kamran's improved PDE solver to obtain values of macroscopic permeability, $\mathbf{K}$. This provides us with data triplets of the form (image, β, $\mathbf{K}$) on which the regression FNO-CNN can be trained. There is a possibility to implement the generative model for fibre orientations with the FNO-CNN as part of an encoder-decoder structure. Ultimately, this results in a combined neural network that can quickly and accurately predict macroscopic permeability, given just a raw image.

An ambitious goal is once this setup is working, we could use the values of $\mathbf{K}$ predicted, to fit a statistical distribution on $\mathbf{K}$ with the $\beta(x)$ values. This would give Sam's model the ability to generate more realistic fibre orientations that can be fed back into the solvers. Also, if there are a limited number of true images with their $\mathbf{K}$, one could generate similar images from neural networks to use as realistic synthetic data.

Overall, this framework across the three strands should be implemented within an HPC setting, to ensure that high computing power is used to tackle the multi-scale flow problem.

References

Almajid, M. M. and Abu-Al-Saud, M. O. (2022). Prediction of porous media fluid flow using physics informed neural networks. *Journal of Petroleum Science and Engineering*, 208:109205.

Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. (2021). Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157.

Bodaghi, M., Lomov, S., Simacek, P., Correia, N., and Advani, S. (2019). On the variability of permeability induced by reinforcement distortions and dual scale flow in liquid composite moulding: A review. *Composites Part A: Applied Science and Manufacturing*, 120:188–210.

Chen, Y. (2023). High-performance computational homogenization of stokes–brinkman flow with an anderson-accelerated fft method. *International Journal for Numerical Methods in Fluids.*

Chen, Y. (2024). High performance computing simulations at university of bath.

Clyne, T. W. and Hull, D. (2019). *An introduction to composite materials.* Cambridge university press.

Deveney, T., Mueller, E., and Shardlow, T. (2019). A deep surrogate approach to efficient bayesian inversion in pde and integral equation models. *arXiv preprint arXiv:1910.01547.*

Durlofsky, L. and Brady, J. (1987). Analysis of the brinkman equation as a model for flow in porous media. *Physics of Fluids*, 30(11):3329.

Hadi, M. U., Qureshi, R., Ahmed, A., and Iftikhar, N. (2023). A lightweight corona-net for covid-19 detection in x-ray images. *Expert Systems with Applications*, 225:120023.

Ham, D. A., Kelly, P. H. J., Mitchell, L., Cotter, C. J., Kirby, R. C., Sagiyama, K., Bouziani, N., Vorderwuelbecke, S., Gregory, T. J., Betteridge, J., Shapero, D. R., Nixon-Hill, R. W., Ward, C. J., Farrell, P. E., Brubeck, P. D., Marsden, I., Gibson, T. H., Homolya, M., Sun, T., McRae, A. T. T., Luporini, F., Gregory, A., Lange, M., Funke, S. W., Rathgeber, F., Bercea, G.-T., and Markall, G. R. (2023). *Firedrake User Manual.* Imperial College London and University of Oxford and Baylor University and University of Washington, first edition edition.

Ide, H. and Kurita, T. (2017). Improvement of learning for cnn with relu activation by sparse regularization. In *2017 international joint conference on neural networks (IJCNN)*, pages 2684–2691. IEEE.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2023). Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020a). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895.*

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020b). Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485.*

Narkhede, M. V., Bartakke, P. P., and Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1):291–322.

Peng, J.-Z., Aubry, N., Zhu, S., Chen, Z., and Wu, W.-T. (2021). Geometry and boundary condition adaptive data-driven model of fluid flow based on deep convolutional neural networks. *Physics of Fluids*, 33(12).

Pesavento, F., Schrefler, B. A., and Sciumè, G. (2017). Multiphase flow in deforming porous media: A review. *Archives of Computational Methods in Engineering*, 24:423–448.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Zafar, A., Aamir, M., Mohd Nawi, N., Arshad, A., Riaz, S., Alruban, A., Dutta, A. K., and Almotairi, S. (2022). A comparison of pooling methods for convolutional neural networks. *Applied Sciences*, 12(17):8643.

A Code availability and model descriptions

In the Results section (Section 4), we performed numerical simulations using the CNN and FNO. We provide a table (A.3 and A.5) of the model hyperparameters for each model as well as a table (A.1) giving a short description of what each parameter represents.

The code folder is available in `https://github.com/Amino21786/Multi-ScaleFlowIRP` called `Multi-ScaleIRP` that reproduces all the plots and results in his report, including the Python files with the construction of the NN models. In the folder, there is a `README.md` and a `dependencies.yml`, which outline what this code folder entails and a guide to download the relevant Python libraries required. The main Python files used were `dev_FNO_mdev.py`, `meanVelocityCNN.py` and `multiScaleFlowCNN.py` in Section 4. Other Python files were made to explore the CNN's capability including for predicting MNIST digits, fashion MNIST and predicting the velocity Hessian matrix for a (x, y) velocity matrix. Note the data folders including `R64`, `R128` and `dataCNN` are not in the Git repository due to their size, however they are available upon request.

A.1 Parameter descriptions

Parameter	**Description**
N_{training}	Number of training samples
η	Learning rate for the optimiser (Adams)
N_{epochs}	Number of epochs used for training
Rank	Percentage of parameters taken in the Fourier layers
Factorisation	Matrix/tensor factorisation for processing (Tucker factorisation used in FNO)
Kernel size	Size of convolutional filter matrix applied in the convolution layer (2D, $n \times n$)
Padding	Number of extra pixels added to the border of input images or feature maps
Stride	Number of pixels the convolution filter matrix moves across in each calculation

Table 1: Description of parameters in Tables 2 and 3

A.2 Model descriptions

Simulation	**Dataset**	**Network Layers**	N_{training}	N_{testing}	η	N_{epochs}
FNO (4.1)	Fibre image data (64×64 pixels, R64 folder) Chen (2024)	34 (T from Algorithm 1)	700	100	0.008	200
CNN (4.2)	Randomly generated velocity maps (64×64 pixels)	6	1000	200	0.001	10
CNN (4.3)	Velocity map Results from FNO (64×64 pixels, R64/resu folder)	7	700	100	0.003	200

Table 2: Model setups for FNO and CNN

A.3 FNO model

Input channels	**Output (final) channel**	**Projection channels**	**Rank**	**Factorisation**
3	2	64	0.42	Tucker

Table 3: Model parameters for FNO

A.4 Mean velocity CNN model

This architecture has six layers, consisting of two 2D convolutional and maximum pooling layers pairings (2D convolution → max pooling → convolution ...) with two fully connected layers at the end, to extract the mean velocity of the 64×64 velocity map.

Layer	**Input channels**	**Output channels**	**Kernel size**	**Stride**
2D convolution 1	2	2	4	-
2D convolution 1	2	2	4	-
Maximum pooling	2	2	4	2
Fully-connected 1	3136	128	-	-
Fully-connected 2	128	1	-	-

Table 4: Model parameters for mean velocity CNN

A.5 Multi-scale flow CNN model

This architecture has seven layers, consisting of three identical 2D convolutional and maximum pooling layers pairings (2D convolution → max pooling → convolution ...) with a fully connected layer at the end, to extract the 4 components of **K** tensor.

Layer	**Input channels**	**Output channels**	**Kernel size**	**Stride**
2D convolution	2	2	4	-
Maximum pooling	2	2	4	2
Fully-connected	50	4	-	-

Table 5: Model parameters for multi-scale flow CNN