# Estimating the Permeability of Porous Media with Fourier Neural Operators

Lukas Schröder

Master Thesis

Aufgabensteller: Prof. Dr. -Ing. Harald Köstler

Bearbeitungszeitraum: tba

# Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Masterarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 6. November 2024 ……………………………………………

# Acknowledgement

I want to thank all the people who helped me to realize this thesis. First, Professor Dr. -Ing. *Harald Köstler* for his idea to solve this problem with FNOs and feedback, then M. Sc. *Samuel Kemmler* and M. Sc. *Shubham Kavane* from the i10 chair for sharing their knowledge with me as well as their help in sharpening my findings. It was also great to have the opportunity to compare my work with B. Sc. *Impana Somashekar* who solves the same problem from a different angle in her thesis. The access to the *Alex* cluster from the *NHR@FAU* team was central to run my project and I want to thank them for that as well.

# Abstract

This work introduces and investigates a surrogate model for predicting the permeability of porous media based on Fourier neural operators (FNOs). The core concept is to predict the entire pressure field created by the input geometry and then to extract the resulting pressure difference afterwards to allow the network a closer proximity to the flow phenomena in physics. FNOs leverage the frequency domain to apply their weights, are independent of input resolution and thus provide good results in a short training time compared to other neural networks. Techniques like weight normalization, cosine learning rate scheduling with warmups and the Adam optimizer with weight decay are applied for improved stability and performance. Additionally, the improved factorized Fourier neural operators (FFNO) with separate weights and convolutions for every dimension and skip connections for deeper and smaller models is compared to the original approach. In a 3D dataset consisting of the flow around randomly shifted spheres simulated by the Lattice-Boltzmann method an R2-score of 99.91 with a 4-layer FFNO was achieved, proving the capabilities of the concept. To show how the model can handle small datasets consisting of complicated and diverse geometries, a second experiment was conducted on a different set. It achieved an R2-score of 88.54 with an increasing error for smaller permeabilities. When these extreme cases are filtered out, the model scores again above 90, showing that it is also able to represent more complex geometries. The Swin transformer served as a comparison on a 2D dataset, where it achieved a better result in a fraction of its time. Analyzing the results highlighted that when speed is important the 4-layer original FNO approach is optimal and that the 8-layer FFNO provides the best accuracy. If the effort of creating and training a dataset is worth it, depends on the number of times it is used afterwards, but in general, FNO based surrogate models bridge the gap of fast, but limited analytical solutions and universal, but slower numerical simulations.

TODO:

- Add Shubham thesis reference

# Content

# 1 Introduction

## 1.1 Motivation

When the permeability of a porous medium needs to be estimated, numerical simulation or experiments are required for a high degree of accuracy. Both require time and care to be conducted, making them undesirable when many similar geometries need to be estimated fast. So, a surrogate model, which condenses the problem to its relevant parameters to provide a quick estimation, is helpful here. The advancements in neural networks as well as their hardware during the last 10 years allow them to be used as one. These computer learning based models allow for a quick, but accurate estimation with a higher efficiency than starting every time a new numerical simulation as it learns from previous experiments. The twist here is to model the pressure filed, which is driven by partial differential equations and then extract the permeability from it. This stands in contrast to the pure machine learning approach, where the output would directly be the desired value, shifting this approach closer to the physical domain.

## 1.2 Topic and Goals

The thesis topic is to estimate the physical property of permeability of porous media with the Fourier neural operator (FNO). This is done by estimating the pressure field and calculating the permeability from the pressure difference between inlet and outlet. To fulfill this request, the following intermediate steps have been taken:

1. Selecting the geometries to test the method on
2. Establishing a numerical simulation of the media with the Lattice Boltzmann solver WaLBerla and lbmpy
3. Preprocess the data for the optimal training results
4. Train FNOs with different setups and evaluate the outcome

In the process of working on the realization, many questions came up that this work tries to answer:

- Are Fourier neural operators even capable of capturing the pressure field to a reliable degree?
- What scope of porous media can a model cover with what requirements?
- How to improve the performance by tuning the hyperparameters of the model as well as the training process?
- How is the efficiency and speed compared to other networks, numerical simulation and analytical solutions?
- In what cases are surrogate models based on neural networks useful?

## 1.3 Structure

To establish the baseline of knowledge required to understand the proceedings in this work, the second chapter will start with the fundamentals of flow in porous media. Then the Lattice Boltzmann method (LBM), which was used to create the datasets for the later training, will be explained. The focus will then shift to neural networks and especially the types that are useful for mimicking physical systems. This is meant only as an overview of the core concepts, but all relevant papers with more detailed explanations are referenced.

The next chapters follow the described procedure, by introducing the three different data sets which fulfill different purposes and how the flow field was simulated on them. The used architectures and all thoughts put into the training process follow afterwards. Then the results from the realizability test on the first set, the investigation into more complex geometries and comparison with the Swin transformer get presented. Finally, the results will be discussed, trying to answer the broader established questions.

The last chapter gives an opportunity to reflect on the results and presents new questions that would be exciting to follow in the future.

# 2 Fundamentals

This section introduces some of the core concepts relevant to the estimation of permeability. Starting with the physical aspect, continuing with computational simulation of it and ending with the basics of deep learning. This ordering is coincidentally historically as well.

## 2.2 Introduction to Flow in Porous Media

To get to the goal of predicting certain values of flow in porous media, it is important to establish a basic understanding of the underlying principles that drive these phenomena. A material with a matrix of solid particles and void is considered porous. Good examples are sponges, limestone or ground coffee. The ratio of void to the entire volume is called porosity $\varepsilon$. In the following it is assumed that only one fluid is present in the pore network, thus ignoring multiple phase situations.

Some assumptions about the fluid and flow must be made to describe the mechanics behind it easily. At first the fluid is assumed to be Newtonian, meaning that the viscosity tensor is constant. Also, the medium must be able to be represented by a volume of a certain size independent of its position, so it must be homogenous. We will see later, that more complex pore geometries, no longer homogenous, are a challenge to predict and calculate. Additionally, the flow is assumed to be not influenced by temperature gradients, steady state and incompressible.

Permeability is the resistance the material induces on flow of liquids through the medium and was put in relation to other quantities by *Darcy's Law* by *Henry Darcy* (Darcy, 1856). The law describes the volumetric flow rate $Q$ by the permeability $k$, the cross-section area $A$, the fluid viscosity $\mu$ and the pressure drop along length $L$. Figure 1 shows the experimental setup.

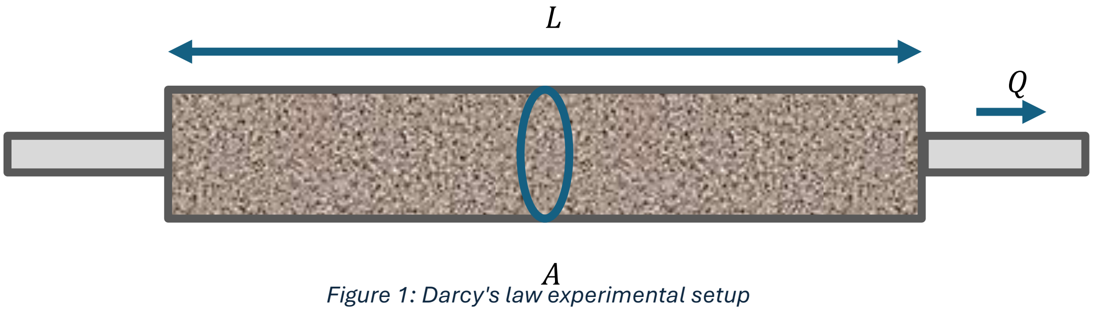

*Figure 1: Darcy's law experimental setup*

$$Q = \frac{-kA}{\mu}\frac{\Delta p}{L}$$

Equation 1: Darcy's law (Darcy, 1856)

*Darcy's law* does only apply to laminar flow and this work also sticks to laminar flow only. With the met assumptions, the conservative equations for the flow inside can be derived and used with numerical schemes to simulate the flow on computers. Numerical simulations are computationally expensive and so formulas to estimate the permeability for a certain media were established. This is relevant here, because one goal of the thesis is also predicting faster and thus a comparison between deep learning and these formulas will be made later. The *Kozeny-Carman* equation is a simple one of these ways for creeping flows (Bear, 2018).

$$\frac{\Delta p}{L} = \frac{150\mu}{\phi_S^2 d_p^2}\frac{(1-\ \varepsilon)^2}{\varepsilon^3}V_0$$

Equation 2: Kozeny–Carman equation (Carman, 1956)

The equation assumes a packed bed of spherical particles of sphericity $\phi_S$ and diameter of an equivalent spherical particle $d_p$. $V_0$ represents the empty velocity*.* When this is paired with *Darcy's law*, the *Kozeny* equation for permeability can derived. There is a whole research field dedicated to finding better estimation of $k$ based on more parameter, but this would go outside the extent of this work and needs reliable ways of estimating these different parameters too.

$$\kappa = \phi_S^2 \frac{\epsilon^3 d_p^2}{180(1-\varepsilon)^2}$$

Equation 3: Kozeny equation

Now, having established the basis of flow in these geometries, the focus will shift on the simulation of them. The *Lattice-Boltzmann* method is an alternative to finite volume solvers and is used here.

## 2.2 The Lattice Boltzmann Method

This method of computational fluid simulation (CFD) differs from the classical methods of discretizing macroscopic quantities like the momentum and mass equations on a grid directly. The focus of this method is the lattice mesh consisting of same sized cells and handling them as containing a fictional particles. Here its roots, the cellular automaton, show. Each cell contains a distribution of particles in $m$ directions for $n$ dimensions, from which the density and velocity of the cell can be calculated. The $DnQm$ scheme classifies the simulation by these parameters. A higher $m$ indicates generally a higher precision but entails more calculations. (Krüger, et al., 2017)

Then there are three steps to the algorithm (with varying order):

- **Collision step**

  The intracellular interaction of the densities is calculated here. This mimics the collision of the molecules inside of the cells. The value in direction $i$ can updated with the Bhatnagar, Gross and Krook model (Bhatnagar, Gross, & Krook, 1954).

  $$f_i^*(\vec{x},t) = f_i^*(\vec{x},t) + \frac{f_i^{eq}(\vec{x},t) - f_i(\vec{x},t)}{\tau_f}$$

  *Equation 4: Collision step with SRT*

  The equilibrium is calculated from the local pressure and velocities. $\tau_f$ is the relaxation rate and has to be chosen according to the kinematic viscosity of the fluid and also defines the temporal resolution. This is the simplest form of the collision operator and is called single-relaxation-time (SRT). Because of an observed nonlinear dependency of the viscosity and the numerical errors, there is the more complex two-relaxation-times (TRT) approach (Ginzburg, 2008). There, the coefficients are controlled by two relaxation rates, one for symmetric and one for non-symmetric components, complicating the collision step.

- **Boundary handling**

  In the setup process, an inflow, outflow and no slip cells are defined to behave in their described way. The rest of the cells are the fluid domain and are not touched during this step. The inflow is realized by adding a constant value to all pdfs in the desired direction. The outflow is handled by forcing the pressure to be constant by adapting the pdfs accordingly. At last, the no slip cells, representing the solid parts and channel walls, reflect their received momentum back to in the fluid domain and this is realized by interchanging their pdfs relative to the walls position. There are different methods based on the order of bounce-back for this.

- **Streaming step**

  This self-explanatory step sends the pdfs in their respective direction to the neighboring cell. This is the only extra-cellular step and needs consideration when calculating on distributed memory.

$$f_i(\vec{x} + \vec{e}_i, t + \delta t) = f_i^*(\vec{x}, t)$$

*Equation 5: Streaming step equation*

This lattice mesh allows the solver to be easily implemented in complex geometries like pore structures, compared to the other solvers that need careful meshing to preserve the correct behavior in complex wall driven flows. The setup of a low Reynolds number flow in helps avoiding some pitfalls of LBM in complex geometries in combination with turbulence. The second advantage is the simple, well-structured way the calculation can be executed. This enables high performance on small- and large-scale systems due to its easy way of parallelization. It was important for the later comparison between using a neural network and simulating new geometries directly to have state-of-the-art solver performance to get a fair result. The focus lies on realistic results and performance and not on research on new and improved ways of LBM in porous media.

This chapter is mostly based on *Lattice Boltzmann methods in porous media simulations: From laminar to turbulent flow* (Fattahi, et al., 2016). In the later chapter that deals with the setup used for creating the data for the network, the bespoke settings for LBM in porous media will be discussed. The most important aspects being the choice of the bounce-back operator and the collision model.

### 2.2.1 The waLBerla Framework and lbmpy

The framework *waLBerla* (widely applicable lattice–Boltzmann from Erlangen) is written at the *chair for computer science 10 (system simulation)* at the FAU in C++ for massively parallel applications of LBM (Feichtinger, Donath, Köstler, Götz, & Rüde, 2011). The emphasis is on modularity in combination with hardware specialized kernels. It can run from a single processor or graphics processing unit (GPU) to huge clusters of them with MPI and OpenMP. Apart from using more sophisticated C++ code to set up the problem, a python software *lbmpy* was developed at the same chair, that has its focus in easy adaptation and code generation for waLBerla (Bauer, Köstler, & Rüde, 2021). This allows for quick setup and incorporation in this project without the loss of performance. Both were used in creating datasets to train the networks.

## 2.3 Surrogate Models from Deep Learning

Surrogate models are needed when certain objectives of a system need to be optimized and it is not possible to calculate the outcome quickly in a direct manor. This model should allow the tuning of design parameters and act closely to the simulation to understand their impact on the system. The goal is to make sensitivity and optimization studies feasible. The art lies in extracting the underlying mechanisms to mimic the system reliably in the parts that act on the objectives. There are certain classical aspects of this process of creating the model (Queipo, et al., 2005):

- **Design of experiments**

  What values should my model account for?

  **Here**: permeability, pressure (and velocities for physically informed)

- **Numerical simulation at selected locations**

  Are my simulated datapoints representable for the model?
  Are the results correct?

  **Here**: LBM simulation, given range of datasets

- **Construction of the surrogate model**

  Which numerical, analytical or data science method should be chosen?
  What are the best hyperparameter settings for the respective model?

  **Here**: Analytical (*Kozeny-Carman* equation), Fourier Neural Operators or Swin transformers

- **Model validation**

  Do my model's results match the data? How far is the range of my model?

  **Here**: Validation datasets, scatter plots and Comparison with other approaches

These aspects are reflected already in the questions and goal part of the thesis in the previous chapter and are answered in the consecutive ones. Surrogate models are especially helpful in the domain of 3D CFD engineering because of the high computational effort even when the search domain range is small.

### 2.3.1 Deep Neural Networks in Machine Learning

The core concept behind neural networks (NN) is the put input data $v$ through several layers that apply changeable weights to the data to get a result $u$ that closely matches the target data. This is called the forward pass and consists usually of linear application of the weights in $K$ and non-linear activation function $\sigma$.

$$u = (K_l \circ \sigma \circ K_{l-1} \circ \sigma \circ \cdots \circ K_1)v$$

*Equation 6: Multi-layer perceptron as function*

The weights get updated in the backward pass based on the distance of the output and target data. This calculation of this is done with the help of the loss function. The amount of change enacted on the weights is calculated by the optimizer, derived from stochastic gradient descend, and the learning rate scheduler, that reduces the amount of allowed change during the training to come closer to the global optimum. The data that is fed into the network should be curated in such a way that is it the easiest for the model to learn the relevant parts and not some uncorrelated features. A go to example is images as input and categories of the displayed object as output. There are no limits to what can be learned as long there is some kind of "useful" correlation from the input to the output. No understanding of the objects displayed is needed to be coded into the model and afterwards only educated guesses can be made, why the outcome has been selected.

The range a model can represent reliably depends on the amount of data it is trained on as well as the number of trainable parameters it possesses. To increase the quality of these networks, more layers were appended to the architectures in the 2010s. Training these systems is called for this reason deep learning. The vanishing gradient problem, meaning simplified that the gradient of the weight in each layer is harder to trace back to the loss calculation because there are more layers to penetrate, is more prevalent in deeper networks. Certain ways of combating these new problems are presented now. Here, usually non-linear activation functions are placed in between linear layers to allow for a wider range of operations to be represented without more learnable parameters. Another way are skip connection to allow the data to pass the layers and be combined with the processed outcome, this is present in the later presented feed-forward blocks. It helps containing important features of the data even after many layers deep. These techniques are combined to deepen the original FNO in the Factorized FNO architecture. Now, the latest advancements in NN for physical simulations are presented. The Neural Operators (NOs) are in the forefront in this field, but also alternative advancements like Transformers will be introduced quickly.

### 2.3.2 Neural Operators (NOs)

Neural networks are overwhelmingly used for mapping finite dimensional spaces. The previous example shows this, it's a mapping from 2D to a set of objects. The physical properties like velocities and pressure are infinite dimensional and get only discretized on finite meshes to handle them computationally. The momentum and continuity equations from *Navier-Stokes*, for example, are mapping functions connecting these properties with operators, mainly partial differential equations (PDEs). Exactly these operators should be represented in the models of NOs. To eliminate the dependence of NNs on finite spaces an encoder, also called lifting layer and decoder or projector maps the concrete discretized data into the infinite codomain. The hidden layers in between apply an iterative kernel integration $K$ consisting of a sum of linear local weights $W$ and a nonlocal integral kernel operator $\kappa$ combined with a bias and nonlinearity afterwards. The global convolution is more suited for continuous function than the local convolutions used for images. How this cryptic parts realization looks like will be demonstrated in the next chapter with the FNOs.

$$K: v^*(x) = \int \kappa(x, y)v(y)dy + Wv(x)$$

*Equation 7: Kernel integration of NOs*

Another great advantage is that the network can be used on different discretization without retraining, it acts like a mathematical operator. Mathematically expressed, the operator $G$ maps from the function space $A$, like position in space, to a function space $U$, like pressure. The according mapping $G_\phi$ realizes this operator with the parametrization $\phi$ and uses the projection layers $Q$ and lifting layers $P$.

$$G: A \to U; \qquad G_\phi := Q \circ L \circ P$$

*Equation 8: NO function & mapping*

The highlight is stated in the original paper: “Neural operators are the only known class of models that guarantee both discretization-invariance and universal approximation.” (Kovachki, et al., 2021). For a stringent understanding of the theoretical background, the lecture of the paper is highly recommended, but more would exceed the more practical application of this work here. The usage of the NNs inside can vary, but generally should be able to capture the scope of the problem and allow for universal approximation. The paper suggests four types: graph neural operators, multi-pole graph neural operators, low rank neural operators, and Fourier neural operators. The last one will be explained in detail now.

### 2.3.3 Fourier Neural Operators (FNOs)

The FNO model describes how the layers in between the lifting layer and projection layer are handled (Li, et al., 2020). The trick is that the linear transformation happens in spectral space. The Fourier transformation (FT) takes values from a function as input and calculates the number and amplitude of the frequencies present in the signal. This opens another domain, called Fourier space, of the represented data and allows for a different perspective on the same input. Figure 2 shows the overall structure of the NN and one Fourier layer.

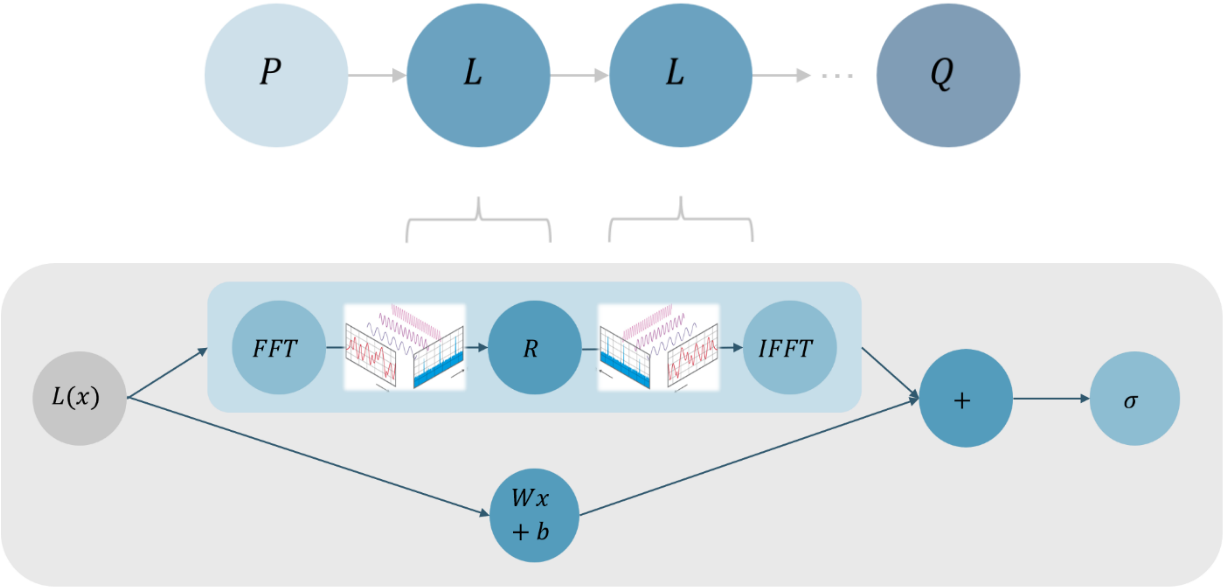

*Figure 2: Structure of the FNO model with one L block in detail*

The iterative updates happen in the $L$ blocks and are defined mathematically as follows.

$$x_{t+1} := \sigma\big(Wx_t + b\ + (K(x_t))\big)$$

*Equation 9: Update on x in the L block of FNOs*

Here $\sigma$ is the nonlinear activation function, $W$ is the application of local linear weights and biases $b$ and $K$ is the kernel integral operation. The operation is called integral, because it works on a sum of the whole domain, here implicitly done by applying the Fourier transformation. Now first to the definition of the transformation and its inverse. The function $f: D \to \mathbb{R}^n$ maps into $n$ dimensions and thus $j = 1, \ldots, n$. Figure 3 shows this transformation for the dimensions in time and one spatial one. Computationally, this is realized with the fast Fourier transform, which is implemented very efficiently on GPUs already.

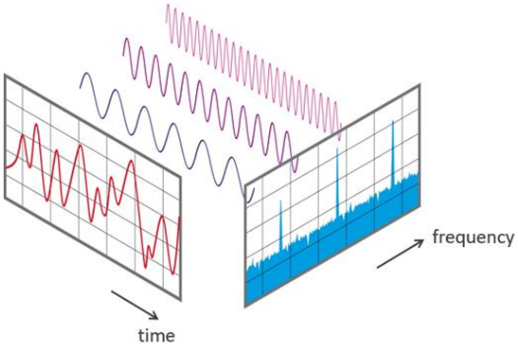

*Figure 3: Visualization of the Fourier transformation in time (source: https://www.nti-audio.com/de/service/wissen/fast-fourier-transformation-fft)*

$$(Ff)_j(k) = \int f_j(x)e^{-2i\pi\langle x,k\rangle}dx\,, \qquad (F^{-1}f)_j(x) = \int f_j(k)e^{-2i\pi\langle x,k\rangle}dk$$

*Equation 10: Fourier transformation and its inverse*

The convolution theorem is another puzzle piece to the kernel integral operator. Mathematically expressed, a convolution is the amount of overlap of one function as it is shifted over another. Convolutions are a common practice in NNs and usually are done by shifting a kernel, a small matrix window with learnable weights, over the whole domain of an image. This gets an image of similar size and this way the importance of local differences is highlighted. The convolution theorem states that a convolution of two signals ($u$ and $v$ here) is equivalent to a multiplication in Fourier space. The operator for a convolution here is "$*$".

$$r(x) = \{u * v\}(x) \triangleq \int u(\tau)v(x-\tau)d\tau = \int u(x-\tau)v(\tau)d\tau$$

$$= F^{-1}\{F(u) \cdot F(v)\}$$

*Equation 11: Convolution theorem*

So, to apply the kernel of weights in a convolution in discrete space, it is enough to multiplicate them after the FT. Let's assume $u$ are the learnable weights and it is sufficient to store only the Fourier space version $R = F(u)$ of them. Then the resulting operator is defined as:

$$K(x) := F^{-1}\{R \cdot F(x)\}$$

*Equation 12: Kernel integral operator*

The domain $D$ needs to be discretized with $m \in \mathbb{N}$ to allow it to run on a computer. Then the kernel operation results in the following form:

$$\big(R \cdot F(x)\big)_{k,l} = \sum_{j=1}^{n} R_{k,l,j}(Fx_t)_{k,j}$$

*Equation 13: Discretized Fourier convolution*

The size of R can be chosen as $R \in \mathbb{C}^{k_{max} \times d_v \times d_v}$ with $k_{max}$ Fourier modes. The FT inside the sum can be realized with the discrete Fourier transformation, which consists of one summand for every point in one dimension. So, by limiting $k_{max}$, the number of frequencies the Fourier space can represent and work on is limited from above. This truncates high frequencies of the PDF solution, which can be desirable to stabilize the result by filtering high frequency noise out. There is one $k_{max}$ for every dimension and the tuning of this parameter will be analyzed in the setup chapter.

The original paper suggests applying four of the $L$ blocks before projecting the result back in finite space (Li, et al., 2020). Later experiments show that adding more of them does not result in better results, so leaving less room for improvements and scalability. This and many other issues are tackled in the following improvement of FNOs called Factorized-FNOs.

### 2.3.4 Factorized Fourier Neural Operators (FFNOs)

*Tran* et al. proposed many advancements of FNOs in 2023 packed with plenty of helpful ways of stabilizing the training process to leverage the performance of FNOs substantially (Tran, Mathews, Xie, & Ong, 2021). In the center is the restructuring of the Fourier space usage and the accompanying $L$ blocks.

$$x_{t+1} := x_t + \sigma(\, W_2\sigma\,(\, W_1K(x_t) + b_1\,) + b_2)$$

*Equation 14: Update on x with the feed-forward block in FFNOs*

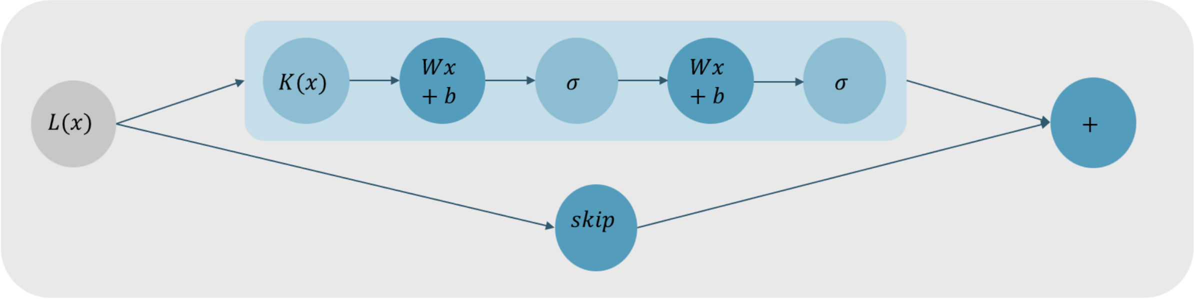

*Figure 4: Structure of one $L$ block of the FFNO model*

One of the two main changes is an additional linear weight application to control the result from the spectral convolution more precisely. Then the unaltered addition of $x_t$ allows for a better protrusion of the input to the deeper layers. This concept is hard to avoid since its appearance in Residual NNs in 2016 (He, Zhang, Ren, & Sun, 2016). The name giving change is the split of the FT in the dimension, reducing the overall computation and number of learnable weights by an order of magnitude (Tran, Mathews, Xie, & Ong, 2021), especially useful in higher dimensional problems. Figure 5 illustrates the split and summation of Equation 15 that calculates the 3D convolution with three 2D weight tensors $R_d$. When running on memory limited setups these weights can also be shared by every layer with moderate performance lost according to the paper.

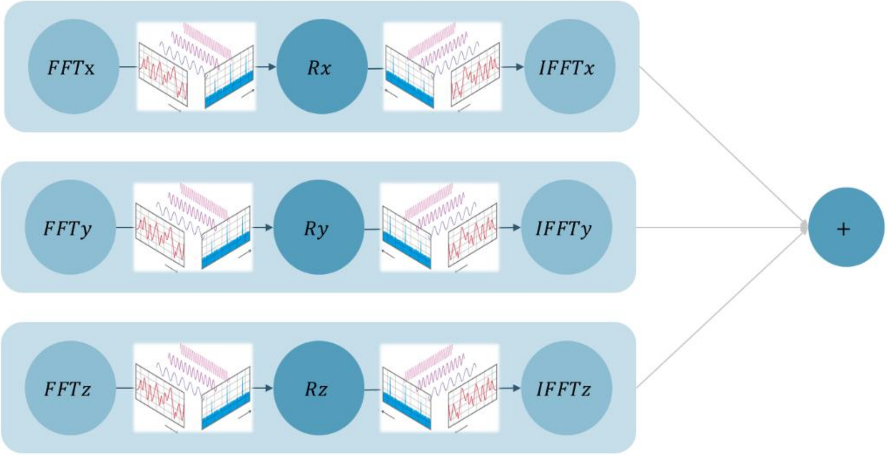

*Figure 5: Model of the factorized Fourier transformation*

$$K(x) := \sum_{d \in D} F_d^{-1}\{R_d \cdot F_d(x)\}$$

*Equation 15: Factorized kerel operator*

### 2.3.5 Physics Informed Neural Networks (PINNs)

Until now, the losses of the networks were derived from the distance from provided training data, hence a data-driven approach. But when modeling physical phenomena, the results should naturally be in harmony with the underlying PDEs of the problem. This led to the concept of physics informed NNs, which at least partially evaluate the loss as the fulfillment of the equations it should represent. To get the derivatives needed for the loss calculation, classical schemes like the finite volume or finite difference method or NN based automatic differentiation are required. Generally, this concept can be applied to every network, which is trying to PDEs. The advantage is the alleviation of the reliance on simulated datasets and thus making training on more geometries possible. In practice, the effectiveness of PINNs depends on the effort of implementing the loss calculation as well as increased training effort. In these cases, it is also advised to additionally validate the loss calculation scheme itself. When both data and calculated losses are used, it is called a hybrid approach. In this work, they were not utilized due to an easily available reliable simulation with LBM and the extra effort of implementing this type of loss and the increase in required output dimension. Nonetheless this approach would be interesting to combine with FFNOs.

### 2.3.6 Shifted Window Transformers (Swin)

Transformers were developed to model sequence data, with a significant impact on natural language processing (NLP). In NLP, the local and global relationships of words, also called tokens, to each other are of utmost importance. Self-attention mechanisms are central to assigning the relative relevance of tokens in their context in parallel. These networks can then produce new text based on input and their own previous output. At both ends of the training pass, they need a positional encoder and decoder that make them in fact an exotic type of neural operator (Kovachki, et al., 2021). Their central internal structure is complex and needs a large dataset to train a huge number of parameters but can also deliver phenomenal results and scale well with big datasets (Vaswani, et al., 2017).

The Vision transformer should mimic these achievements with image data. The tokes are here parts, so called patches, of the input images. Shifted windows transformer set to improve on vision transformers to reduce the training amount substantially, while increasing the general performance (Geng, Zhai, & Li, 2024). They tweak the application of the windows, by using them in different scopes and shapes and thus hierarchical. Their use case is in object detection, segmentation and classification. With the use of convolution for the token generation, they are no longer discretization-invariant and thus no NOs.

Their realization and training is more complex and thus transfer learning is usually used in use cases like in this research. This means that an already trained network on general data is used to initialize the weights before the training. Then the relevant data is used for finetuning the model. An analogy would be to first learn a new language by reading different types of text like news articles, books and lyrics before specializing in analyzing prose.

### 2.3.7 Deep Operator Networks (DeepONet)

Deep operator networks are an alternative to NOs, which also focus on the problem of a continuous operator in a discrete space. They consist of two parts, a branch net for encoding the input function with a certain number of sensor points and a second encoder, the trunk net for the location of the output function. The idea is built on the universal approximation theorem for operators, stating that this model can approximate any nonlinear continuous operator (Lu, Jin, Pang, Zhang, & Karniadakis, 2021).

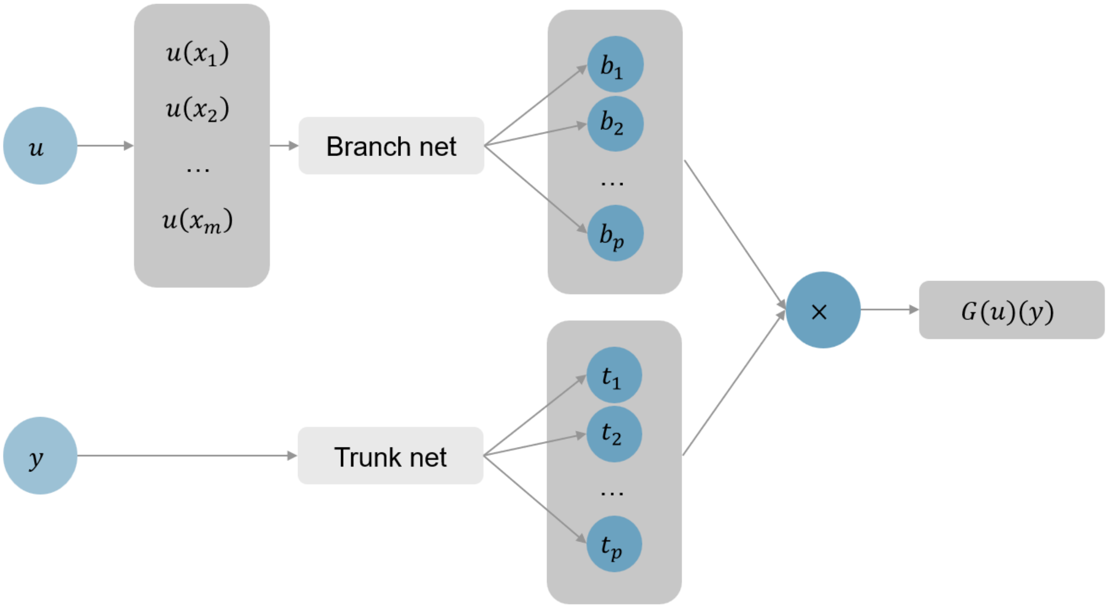

*Figure 6: Input function u(x), positions of the output function y and approximated function G(u)(y)*

In most papers referenced in this work they serve as a comparison to transformers or NOs, but generally perform better than baseline models, but worse than the contender in CFD cases. They could be seen as siblings to NOs, using some sort of encoding to deal with discretization, and are noted here for this reason.

### 2.3.8 Sinusoidal Representation Networks (SIREN)

As the name suggests, SIRENs are used for signal representation. This means that the model acts as a function that maps coordinates to RGB values of an image for example. Sinus as the periodic activation function and nonlinearity is the key for this architecture. The function is a derivative of itself and can therefore be trained on the derivatives of the searched function as well (Sitzmann, Martel, Bergman, Lindell, & Wetzstein, 2020). With special care of the initialization of the linear weights in between the activation function, this simple architecture is state-of-the-art in learning signal representations. This makes them the optimal candidate for the decoder part of NOs.

# 3 Data Creation

Three datasets were used during this work, that have all been created in different ways and serve different purposes as well. They will be described regarding content, purpose and distribution in the following chapter. The flow simulation of them is explained before finishing with discussing their processing to serve as input to the NNs.

## 3.1 Selection of porous media

The first set served the purpose of exploring the realizability of the project. Therefor an established code from Samuel Kemmler in *waLBerla*, which previously served the purpose of performance testing the systems (Kemmler, Rettinger, & Köstler, 2023), was used as basis. The simulated domain has $256x128x128$ cells and the central $80\%$ in x-direction is filled with spheres that also can permeate the boundary. Their size was set equally in range of $[15, 30]$ and the distance of the centers in range of $[size + 1, size + 8]$ resulting in 230 3D samples. Every other row was shifted about half the distance between the sphere centers in the y-z-plane to diminish easy channels for the flow. The rest was a buffer zone to allow for an undisturbed inflow with $0.00008$ lattice velocity and the fixed pressure outlet. The setup results in a Reynold number of $0.05$, so definitely a creeping flow where the Darcy law applies. Based on this approach, a second version was created with random shifts of the centers of the spheres in x direction in range $[-0.1, 0.1] * distance$ and in y-z direction in range $[-0.2, 0.2] * distance$, resulting in 280 samples. The regular spheres set was exclusively used for testing and replaced by the randomly shifted set in the final runs.

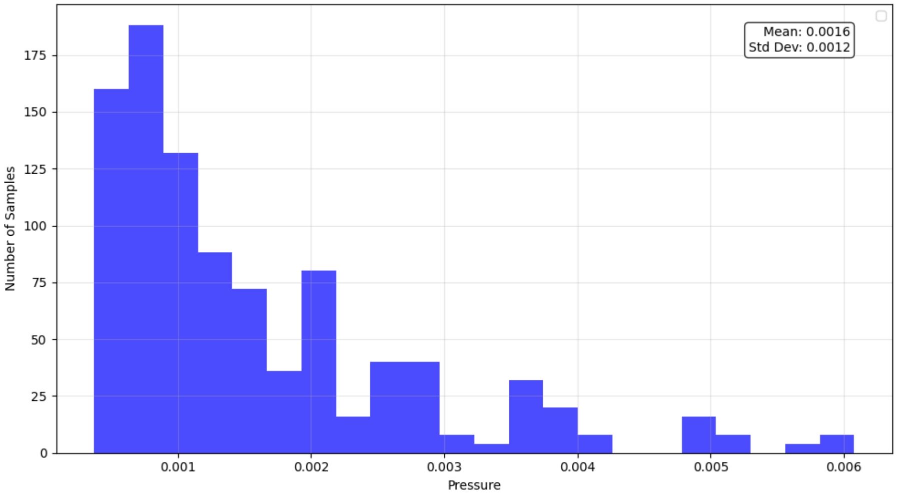

*Figure 7: Pressure difference distribution plot for randomized sphere set (notice scaling)*

Figure 7 reveals the skewed distributions of the pressure values towards the lower end. This is due to the nature of how the pressure buildup arises from small congestions in the flow domain, which is not linearly, but exponentially related to the porosity, like seen in Equation 2. To estimate the degree of this imbalance, every presented dataset's distribution is shown with the original range, mean and standard deviation as metrics.

The second dataset's objective was to estimate the range of porous media the model can handle. To this end, a dataset from the *Digital Rocks Project* (DRP), funded by a grant from the National Science Foundation in the U.S., was used. *Javier E Santos* et al. assembled this dataset from many projects in the DRP to get a great variety of complex flow (Santos, et al., 2021). To assure accurate and comparable simulation data, the provided flow data was neglected and an own simulation in lbmpy was made. 133 distinct geometries were simulated with a flow from the front and the back, but 14 of the simulation failed due to extreme narrow passages or no connection from front to back. The provided data was of size $256^3$ and was scaled down to $128^3$ to reduce computations and match the previous simulation. Then the subsequent simulation domain was $192x128x128$,with an added smaller buffer regions than before to reduce the size. In Figure 8 the even higher range and imbalance compared to the first set is obvious, marking the increased complexity to predict the values. Because of this, the next chapter discusses different scaling to soften the effect of this issue.

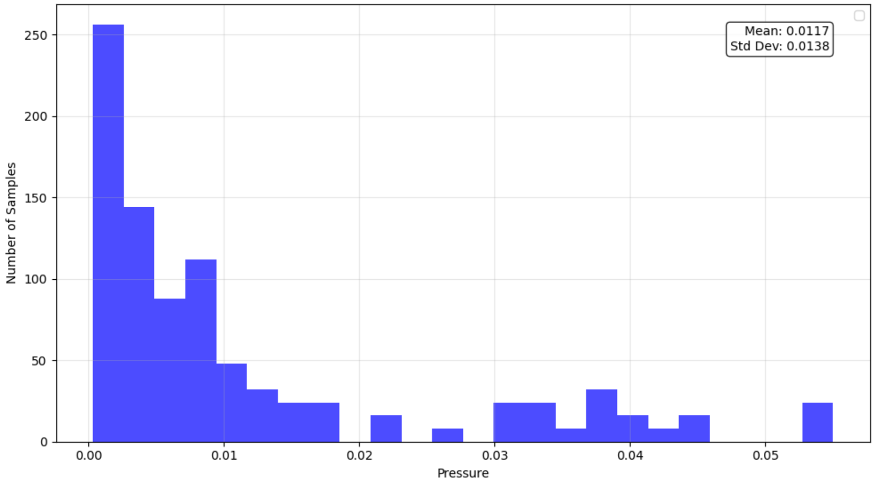

*Figure 8: Pressure difference distribution for the external dataset (notice scaling)*

To allow for a comparison with the 2D approach with swin transformers of Impana Somashekar, the last dataset was taken from her reference paper (Geng, Zhai, & Li, 2024). It contains 1000 with curved Voronoi polygons generated smooth sandstone, which form the standard set. Additionally, 600 rough sandstone images and 200 rock images are present as well, with more complexity. These images were all scaled to $512^3$ to preserve their fine details and simulated with buffer regions of 64 cells on the left and right side. The original set displays the least deviation and shows no "empty bins" like the previous sets.

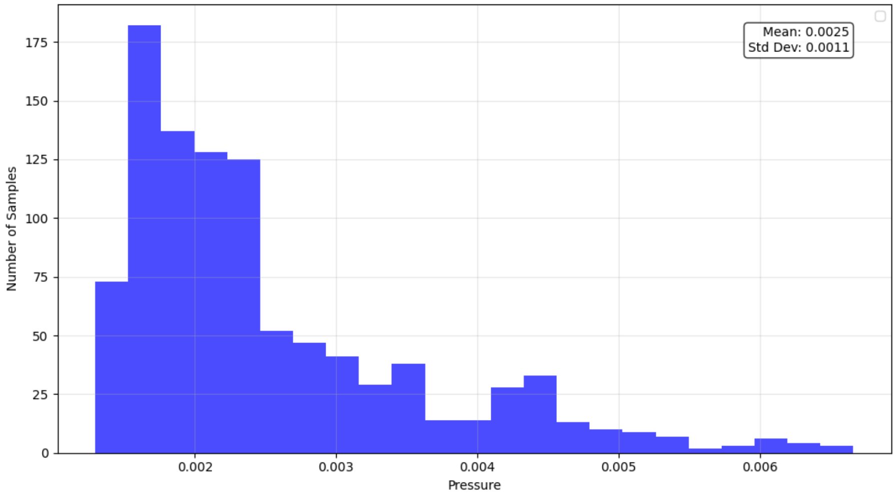

*Figure 9: Pressure difference distribution for the 2D smooth sandstone (notice scaling)*

## 2.3 lbmpy Simulation

It is important to note that the use case of these simulations is as physically based training data of the NNs and as a alternative method to it on its own. This purpose can be even fulfilled when the generated data is off to a certain degree compared with the most exact results possible. There is no reason to assume that the NNs shows a different quality in mimicking the simulation when fed with more physically accurate data. To this end, the simulations are set up to get good realistic results but without putting all effort in research of this field.

One other important aspect to the LBM was the performance. To get a fair comparison of the surrogate model and the numerical simulation, it was ensured to get substantial performance out of the setup. Fortunately, lbmpy supports running on GPUs with the programming language *CUDA* and thus both the NN training and the simulation could be run on the same *Nvidia A100 80GB* hardware for a fair comparison. The spheres dataset was simulated according to the settings already present in the code and are also optimized for particle movement simulation, which was not used here, and thus perform not ideal.

The calculation of the rest of the 2D and 3D geometries happened in *lbmpy*, because of its ease of use. The scaling was done with the *skimage.transform* function *resize* to the above described sizes and they were hand checked to ensure that every media was still represented well. Based on the reasoning from the paper from (Fattahi, et al., 2016), the settings were selected. They argument for the *D3Q19* scheme that it is still capable of delivering proper results for $Re \ll 1$ and thus was selected for a faster computation. The boundary setup is equivalent to the *waLBerla* setup with a velocity bounce back boundary condition with velocity of $0.00008$ lattice units as inlet, a fixed density of $1$ as outflow and the side of the channel as well as the solid geometries with a simple bounce back (BB) condition. A more complex BB boundary could have improved accuracy according to the paper, but the requirement of a wall distance function made this hard to realize in this scope. The domain was preset with the inlet velocity in x-direction for a faster convergence as well. The TRT function was chosen as a more accurate alternative to the in chapter 2.2 presented collision function with only one relaxation time (SRT). *Fattahi et al.* state that, “based on a symmetry argument, Ginzburg proposed a model based on two-relaxation-times, which can be seen as the minimal configuration that provides just enough free relaxation parameters to avoid non-linear dependencies of the truncation errors on the viscosity in the context of porous media simulations.” (Fattahi, et al., 2016). This is important, because the permeability should be assessed independent of the viscosity at play, which is not the case for SRT. Now both relaxation times need to be set accordingly. The first omega is set to $1.0$, to get a Reynolds number of around $0.06$ and the second one can be calculated by the flow characteristic “magic” number. Here, ${}^{3}/_{16}$ was chosen to mimic the BB of Poiseuille flow in a straight channel according to the referenced paper. With an optimized kernel, TRT can also be of the same performance as the SRT scheme, so there is no trade-off needed. The L2 norm of the pressure field served as an indicator if the simulation has reached its steady state to end the simulation.

Because the 2D dataset comes with simulated permeability values, a validation of the results can happen. Sadly, in their paper they do not provide enough information of the settings regarding domain size, inflow velocity or viscosity of the fluid to exactly mimic their simulation. Thus, an approximation of the constant factor that connects the pressure difference and the permeability based on Equation 1: Darcy's law had to be made. Basically, fitting both results by this factor as close as possible. The outcome of this can be seen in Figure 10, with the dashed line being prefect equivalence. There seems to be a trend of higher permeabilities to be predicted to be too low in the *lbmpy* simulation, but with a high overlap in the lower predicted values of under $1000\ mD$, where most of the sample are. Because of the lack of background information from the paper, it's tough to figure out who has the more accurate values, but at least most of the results are close and follow the same trend, validating the method used here to a limited extent. Regarding the question of the performance of neural networks, this is irrelevant, because there is no reason to assume that a model is worse at predicting slightly off physical values in comparison to exact ones. So to speak, it will learn the error from the data, so it can only be as good as the provided information.

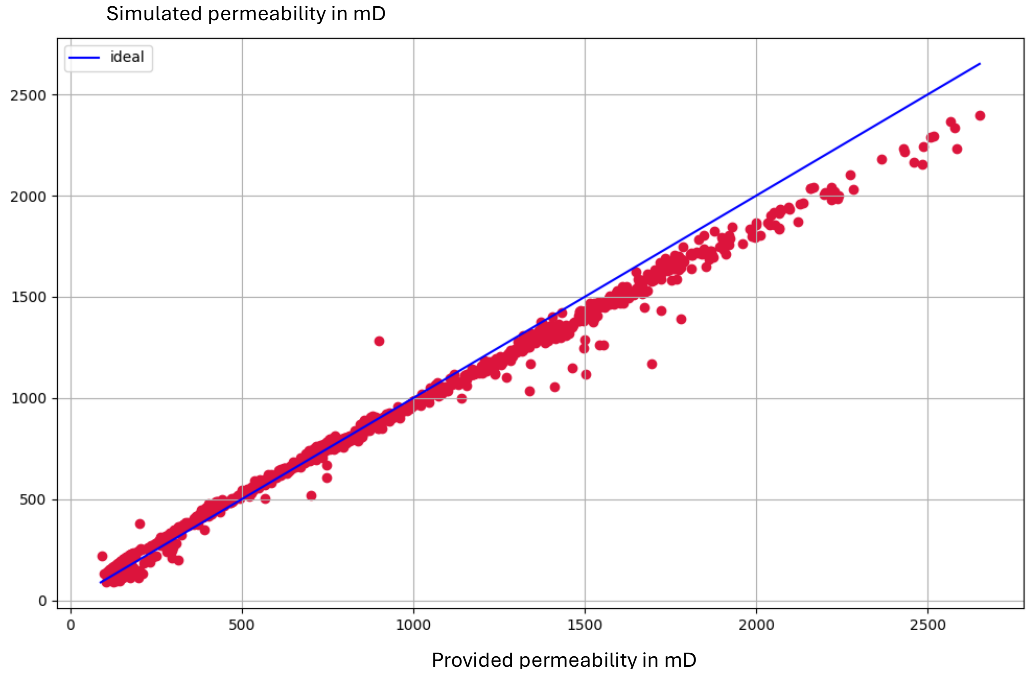

*Figure 10: Scatter plot of simulated and provided permeabilities*

## 2.4 Data Processing

Before its time to start training, the datasets need to be processed to optimize the learning. The first step was normalizing the pressure data to the range of $[0,1]$. NNs are designed to work in this range best to avoid too small or big gradients in the optimizer and to make use of the proper range of the activation function. The normalization was accomplished by a simple min-max normalization. In some complex geometries, extreme singular values can emerge, that are way beyond the inlet pressure, where naturally the highest values may occur. Therefore, moderate clipping was applied to combat this in the first dataset. Studying the Figures 7-9 reveals an uneven distribution of $\Delta p$ with a shift towards the lower end, which is bad for any training, because the model can develop "better knowledge" of the more frequent scenario and thus produces a higher error with the rarer cases. In other words, the networks' performance depends on the permeability, which is an undesirable quality, because the results are less predictable when the distribution of the tested samples is unknown. To combat this effect, the data set second and third set were exponentially transformed by power of $0.5$. How the normalized external datasets look afterwards can be seen in Figure 11. Take note that the plot shows the highest values in each sample, so the range of one sample is $[0, \Delta p]$. The results are renormalized to linear when evaluating to guarantee a fair comparison. This reformation showed a significant improvement and thus was applied to the sets shown in the results.

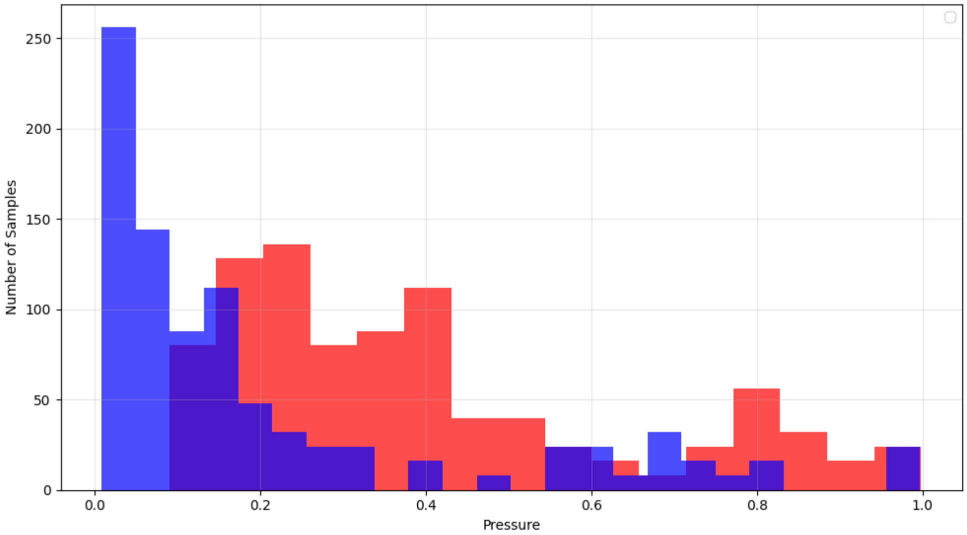

*Figure 11: Pressure difference distributions of the normalized linearly (blue) and exponentially (red) from the external standard data*

Additionally, a filter was implemented to cut off a certain percentile of samples from the top to analyze the effect of the outlier. The effect of this imbalance combating measures are discussed in the results section.

The required size to train a model properly is highly dependent on its architecture and on the complexity of the fed data. In general, it can be assumed that more training data leads to a better result until a certain threshold is reached, when the model's capacity is saturated. As well as that the higher the difference between the samples is the more data is needed with the same setup to cover all cases properly. The term for this is the generalizability of the model. This is why this thesis analyzes multiple datasets. In general, it's advisable to get as many samples out of the physical geometries as possible. Therefor each geometry of the external 3D dataset is simulated with the flow coming from the positive and negative x-axis. Also, every 3D simulation is fed in four times, each 90° rotated along the x-axis. If one of the rotations or reverse of a geometry is in the test or validation set, the other rotations with the same permeabilities are removed from the training set. All sets are stored in the *HDF5* format to ensure quick load times during training.

The geometry of the porous media is the input of the neural network and should be in the range of $[0,1]$ as well. The pore structure is simulated on a bigger scale than it is required to be sufficient for the neural network, so both are reduced to 50% of each dimension. To eliminate stark gradients between near the solid and fluid phase, the binary field was anti-aliased and smoothed to a float value in the necessary range during scaling. Other works on fluid dynamics modeling suggest using a signed distance function (SDF) as input instead of the binary field. The function assigns to every point the distance to the closest object and a negative value relative to the distance of its bounds inside of the objects. The wall distance is an important quantity in flow simulation, because walls slow down the flow near them due to friction, and therefore can inform the network with more relevant information than the phase only. After calculating the SDF for the packed sphere geometries and testing them as input, no significant improvement was found. This might be because the velocity field is more dependent on the walls than the pressure field. So, all later experiments were not run with SDFs, but this is no extensive result for all porous media and setups.

# 4 Setup & Training of the Fourier Neural Operator

This part finally illuminates the central peace of this thesis by discussing all settings involved and how it is technically realized in code. The structure is first the model settings in order of the forward pass transversal and then the training configurations.

## 4.1 FNO Setup

First, the layers of the original FNO are described and their chosen settings are given. Their central concepts have already been introduced during the fundamentals in chapters 2.3.2 and 2.3.3.

### 4.1.1 Lifting Layer

The lifting layer is the first transformation data undergoes when training. This is done with an application of one convolution layer with the kernel size of 1 and the output size of the layers represented spectral space. It's a mixing of the channels by the same weights for each point and is ignoring the spatial context. The original FNO paper suggests a channel size of 32 and this is applied here as well. Furthermore, the addition of the coordinate positions to the input before the lifting can increase the performance due to increased spatial awareness without local operations. The concept is identical to the later used feed forward blocks.

### 4.1.2 Spectral Convolution

The L blocks from Figure 4 are the central piece of the FNO and consist of the spectral convolution and a fully connected layer of the channels, presented in detail in the feed-forward block part (see Figure 13). One spectral layer contains complex weight matrices of the shape $[layers, layers, modes_x, modes_y, \dots]$ that are applied to the spectral domain values with the following multiplication in Einstein notation.

$$[in_{layers}, x_{trunc}, y_{trunc}, \dots] \cdot [in_{layers}, out_{layers}, modes_x, modes_y, \dots] = [out_{layers}, x_{trunc}, y_{trunc}, \dots]$$

*Equation 16: FNO frequency domain weight application*

Thus, the number of frequencies the spectral convolution can work on is set by modes. The rest of size $2 \cdot mode - size_{domain}$ is cut away like a low pass filter. The paper suggests setting the absolute modes to a quarter of the domain size in the respective dimensions, resulting in halving the independent modes, because already half of the modes are dependent because the FFT is applied to real valued data only (Li, et al., 2020). One test run was done to check if increasing the number of modes helps the performance and speed, which support the statement. It is important to note that theoretically the network is still able of representing high frequencies, because they can be recovered during the activation layers before the decoder (Li, et al., 2020). There are four blocks in the classic FNO model and different layer count runs are presented later.

### 4.1.3 Activation Function

When the data has been processed in the Fourier domain, it will be filtered with the activation function, mimicking the "firing rate" of neurons and realizing the universal function approximation theorem. When negative inputs are zeroed and positive ones let untouched, the result is the omnipresent rectifying linear unit (ReLU). The linear part lets the gradient flow through unaltered during backpropagation and the zero part helps only activating the relevant parts for the current sample. The gradient flow is especially important for deeper networks with long paths. The Gaussian error linear unit (GELU) improves on the ReLU, by smoothing it with the gaussian cumulative distribution function and thus allowing for continuous differentiability and mitigating the "dying ReLU" problem

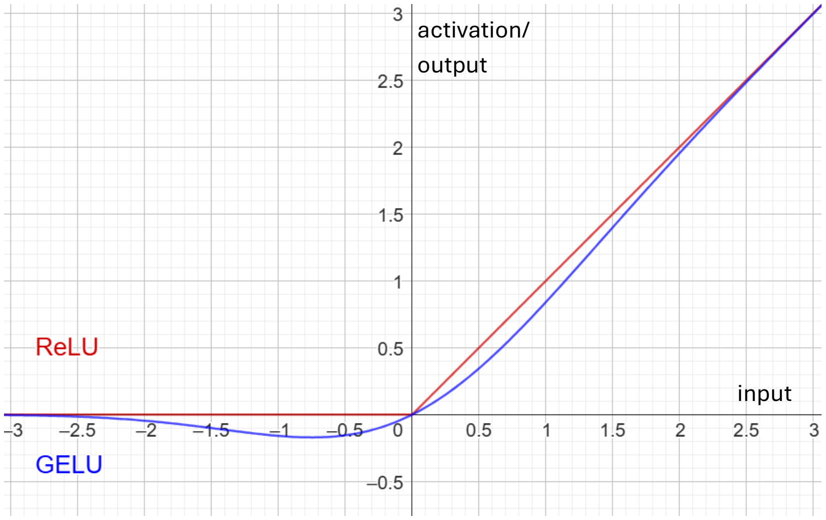

*Figure 12: Comparison of ReLU (red) and GELU (blue)*

by combating to much zero outputs by applying a small negative output. Generally, it has been shown that GELUs can improve upon the successful ReLU in robustness and expressiveness and so it was applied here in every setup (Hendrycks & Gimpel, 2016).

### 4.1.4 Projection Layer

At the end, the final layer projects the data back on the desired finite discretization. The SIREN Net was used for 3D data but failed for 2D cases. A simple feed forward block was used therefore with the image data. The networks consist of three layers for the SIREN and two for the 2D case with 32 channels as input and the single pressure field channel as output. The channel sizes of the hidden layer are increased by a factor of 4 for a more nuanced representation.

## 4.2 FFNO Changes & Optimizations

The greatest weakness of the classic FNO approach is found in the limit of its depth and the instability that requires careful tuning. First, the setup of the new factorized L block. The weights are reduced by one dimension and are only applied in the Fourier domain in one dimension at a time resulting in the following equation.

$$\begin{gathered}\left[in_{layers}, x_{trunc/full}, y_{full/trunc}, \dots\right] \cdot \left[in_{layers}, out_{layers}, modes_{x/y/z},\right] \\ = \left[out_{layers}, x_{trunc/full}, y_{full/trunc}, \dots\right]\end{gathered}$$

*Equation 17: FFNO frequency domain weight application*

The concept of filtering the high frequencies is present as well and the same rule of setting them to a quarter of the domain size applies here also.

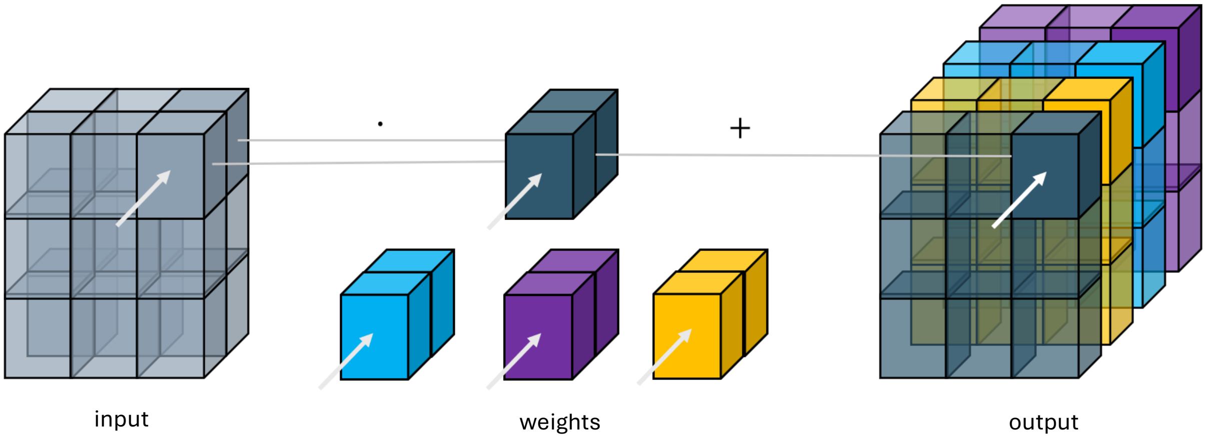

*Figure 13: 1x1 Convolution with 2 → 4 channels*

The output is directly handed to a feed-forward block, which combines several pairs of linear weights and activation functions and is inspired by the design of transformers. As seen in the decoder as well, the hidden layers in between scale the channels by a factor of four to also capture more refined features. The linear transformations are realized by applying a convolution with a kernel size of 1 to mix the dimensions like in the lifting layer. So, the same mixing for every point in the geometry with a fully connected network of the in and out channels like in the original FNO, but at a different position. This transformation is independent of the domain size as well as the spectral convolution, which works on only the lowest of the resolved modes of the frequencies, making the entire L block size independent. It is to note, that the de- and encoder don't have to share this attribute in general, but the layers used in this thesis fulfill this property as well. The proposed two layers turned out to be the best compromise between performance and model complexity.

To stabilize and accelerate the training process, *Tran et al.* apply weight normalization in their provided code. Usually, the weights $W$ are stored as a tensor, but for normalization purposes they need to be split in a direction part $V$ (plus its Euclidian norm $||V||$) and their weight $g$ so that $g$ is equivalent to the norm of $W$.

$$W = \frac{g}{||V||} V$$

*Equation 18: Weight split for normalization*

The optimization of both values can happen decoupled, and this decreases the risk of exploding weights and allows the optimizer to work on more “more angles” of the weights. Especially in deeper networks, internal covariant shift can happen. This phenomenon appears when the early layer shift the output into a bad distribution due to randomness in the data and initialization of the weights, that hinders the later layers to work on the data properly. So, batch normalization was introduced to maintain a stable distribution of the activations at each layer of the network directly by correcting them. As an alternative to batch normalization, weight normalization speeds up the convergence and increases the range of learning rates, where the training converges on, without introducing a significant computational overhead or a dependence on mini-batch statistics. Some more noteworthy improvements are the increased gradient flow, by combating vanishing and exploding gradients, and having a mild regularizing effect by constraining the weight space. A more detailed account of all the mechanisms at work can be found in the respective paper (Salimans & Kingma, 2016). In training FFNOs deeper than 4 layers for this project, it turned out to be a necessity to apply weight normalization to get any results at all.

## 4.3 Loss Functions

Having discussed how the information flows through the layers of the Fourier neural operators, it is now time to talk about the calculation of the difference between the output and the simulated target solution, the loss. With the choice of the function, we can guide the training of the network towards how the goal should be approximated, like a low variance results for example. First, a short overview over several relevant error functions and additionally the masking is discussed.

- **Mean Absolute Error**

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

*Equation 19: MAE*

The error is dependent on the scale of the data and thus makes it hard to differentiate datasets with different distributions.

- **Mean Squared Error**

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

*Equation 20: MSE*

Same dependence on scale as MAE, but with an amplification of bigger errors. This one was used for training on the pressure field.

- **Mean Absolute Percentage Error**

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - x_i}{y_i}\right|$$

*Equation 21: MAPE*

By relative scaling the errors, smaller pressure differences are highlighted and thus this was used to put the end results into perspective. Training with a relative loss function can result in worse results due to a high sensitivity of the small target results.

- **R² Score**

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

*Equation 22: R² score*

Also called the coefficient of determination, it is of relative nature as well. But here the squared error is set into proportions the variance of that data. This one is used for comparison with the swin transformer paper, as they use the same metric. Due to the squaring, it weights outliers more than the MAPE.

The unorthodox approach of modeling an entire field of data to then only extract one singular value from the front of the domain requires some reconsideration of the loss function. This led to the application of a masked or weighted loss to allow for a spatial differentializing. This mask is of the same shape as the output and assigns every point of the domain a relevance factor from $[0,1]$ with ascending importance. The first inlet layers are assigned $1$ to underline their worth and the rest is the inverse of the pore geometry scaled from $0.9$ for the fluid phase and $0.1$ for the solid phase. The mimicking of the pressure field naturally is only possible in the empty parts of the geometry, but the target data is still defined everywhere, making the masking necessary. It turned out to be important to have no regions where the loss is always zero independent of the output. During testing it happened that these regions put out increasingly big values that hindered the correct representation of the relevant parts, so the $0.1$ threshold for the solid phase was added.

## 4.4 Optimizer

The way the weights are updated in the backward pass from the loss is discussed. Stochastic gradient descent describes the method by which the network is optimized to reduce the loss $Q(w)$ by changing the weights $w$ according to their gradient $\nabla Q$. So, after every batch, the weights get changed inversely towards their gradient direction by a certain step size $\eta$, called learning rate. The next chapter discusses the evolution of this method in detail.

$$w := w - \eta \nabla Q(w)$$

*Equation 23: Stochastic gradient descend*

### 4.4.1 Adaptive moment estimation (Adam)

Over the years with the rising importance of machine learning it was iterated on this simple idea to improve the speed and quality of the training process. One important concept added was momentum, also called heavy ball method. The old gradients' directions and sizes are recognized during the new update to prevent the gradient to change to fast when faced with sparse outliers. Adaptive moment estimation (Adam) is the most popular of these improvements and iterates with the help of mass $\mathrm{m}$ (first momentum) and velocity values $\mathrm{v}$ (second momentum), that are calculated from the $\nabla Q$ and their previous values with forgetting factors $\beta$. The factors were set to their standard values $0.9$ for $\beta_1$ and $0.999$ $\beta_2$.

$$w := w - \eta \frac{m/(1-\beta_1)}{\sqrt{v/(1-\beta_2)} + \epsilon}$$

$$with \;\; m := \beta_1 m + (1-\beta_1)\nabla Q \; ; \;\; v := \beta_2 v + (1-\beta_2)\nabla Q^2$$

*Equation 24: Weight update in Adam*

The success of Adam can be attributed to its robustness regarding complex data while still being adaptive towards the idiosyncrasies of the individual training processes without requiring much hyperparameter tuning.

### 4.4.2 Weight Decay

Overfitting happens when the training data is approximated almost too well and the performance on unseen data like in the testing and validation is worse than expected and possible. To combat this, a technique called regularization was invented that prevents the weights from becoming too big, by minimizing not only the loss, but also the model's complexity (Loshchilov & Hutter, 2017). While the introduced weight normalization has a small regularizing effects already, the optimizer should be adapted for it as well by introducing an additional penalty term. So, the weights are reduced by a certain rate $\lambda$, called weight decay.

$$w := w - \eta\lambda w$$

*Equation 25: Weight decay*

*AdamW* tries to accomplish this without interfering with the adaptive nature of the momentum idea, so it was chosen as the final optimizer with a relatively decent weight decay of 0.0001.

### 4.4.3 Learning Rate Scheduler

The last ingredient of successful optimizing is the curve of the learning rate $\eta$. It should decrease over time like in sculpturing, where the chisel starts big to create the overall shape and then it gets smaller to work out the fine details. This is a measure to stabilize the training process and get the best results in the shortest time reasonable. As mentioned already, deeper FNOs are more prone to instabilities and thus *Tran et al.* suggest using a cosine learning rate with a warmup rate, where usually an exponential scheduling is used. Another addition can be the annealing technique warm restarts, so resetting the learning rate curve with a lower start point, but this was neglected in this work, because it would require more parameter tuning extending the scope of this work. The fact that much time is spend towards the ends of the range of $\eta$ with the cosine function seems to be a good fit for training FNOs fast and stable. Also, the warmup phase in the beginning, linearly increasing toward the top of the curve, helps with initializing the weights closer to their optimal position before starting the real optimizing.

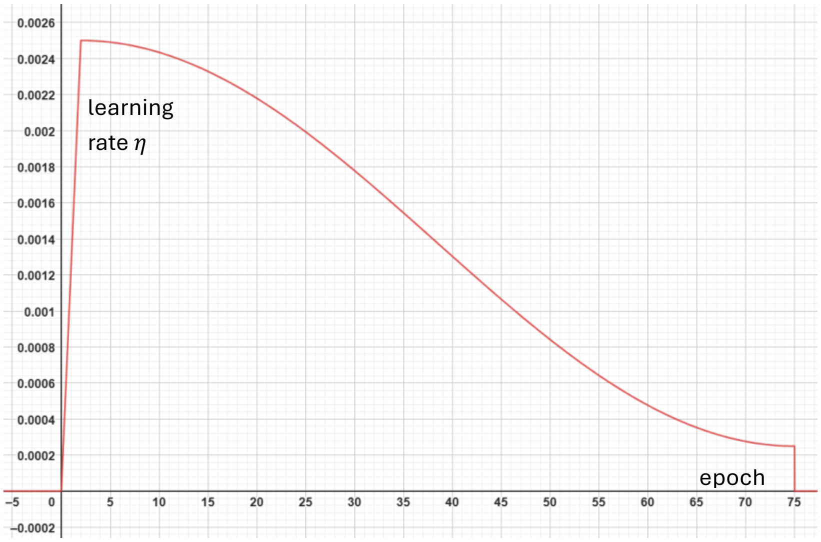

*Figure 14: Single cosine learning rate with Warmup*

The learning rate was set to $2{,}5 \times 10^{-3}$ with a warmup of two epochs and a minimum $\eta$ of 5% of the start value. The training was usually 50 epochs long.

## 4.5 Code

This chapter has illuminated so far what techniques and settings have been employed to get the most stable and fastest training by combating overfitting and vanishing or exploding gradients with recent advancements in machine learning (ML). Of course, this has to be realized in code and thus a short overview of the applied frameworks and the origin of the code is presented. At first this project was realized inside of *Nvidia’s* own framework *Modulus*, but its increased efforts in modification and performance issues made a shift to an independent code fruitful.

The basis is constructed with *PyTorch* in *Python*, that can run on GPUs with *CUDA* and has nearly every ML operation already implemented. To fulfill the Fourier space multiplication, a function expressed in the Einstein notation takes a considerable amount of computational effort, thus the package *opt_einsum* instead of the *PyTorch* function was used in the factorized case to improve the performance (Smith & Gray, 2018).

Some features required hand-crafted coding, like the learning rate scheduler, the masked loss function, the trainer as well as the data in- and output and processing. The models for the FNOs and the Siren Net are a patchwork of already available code from the papers’ *GitHub*, *Nvidia Modulus’* implementations and custom modifications to fit it all together. For some simple, but tedious parts, the large language model (LLM) *Claude 3.5 Sonnet* from *Anthropic* was used to generate, streamline and comment auxiliary parts. It is to note that the central important parts were written by hand and that rigorous reviewing of generated code and text is necessary, because the LLM’s output is not reliable.

The code can be publicly accessed from *GitHub* (github.com/NuxNux7/Permeability-from-FNO).

# 5 Results

With everything established, the different experiments and their results can be examined. This chapter is separated into three parts. First the realizability study, then the test of generalizability with a small set of 133 distinct complex geometries and finally a comparison with the Swin transformer with a 2D set.

## 5.1 Realizability with Packed Spheres

First, the original FNO model with different layer counts, the Siren Net as decoder, weight normalization and the cosine learning rate scheduling was run to establish a baseline. The loss plot during training reveals if the model converges at all and how stable the training works. Figure 15 displays the training loss and shows the convergent behavior that is present, when the model can predict the training set well without any perturbations during the end of the training.

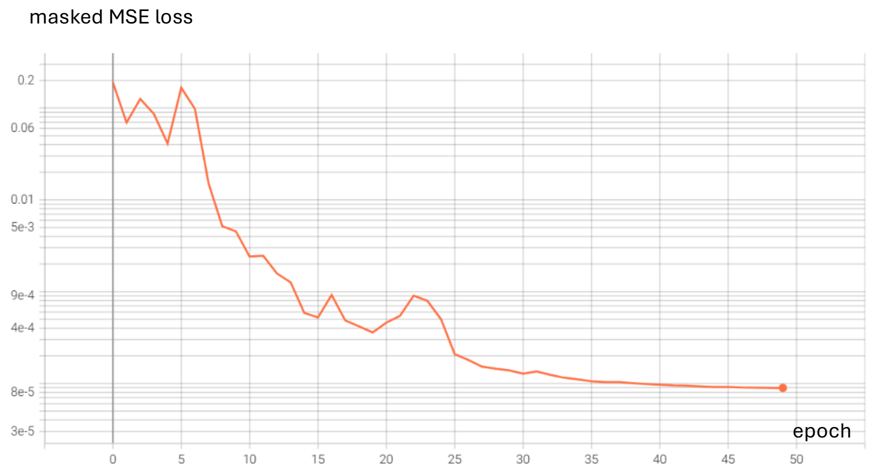

*Figure 15: Masked MSE loss of the training set of the original FNO model with 4 layers*

In contrast to the training loss plot, Figure 17 of the error of the pressure prediction in the testing set shows more fluctuations due to the smaller area over that the loss is calculated. But the convergence to the final values shows that no overfitting happens and respectively that the training routine is successful in learning the desired values reliably.

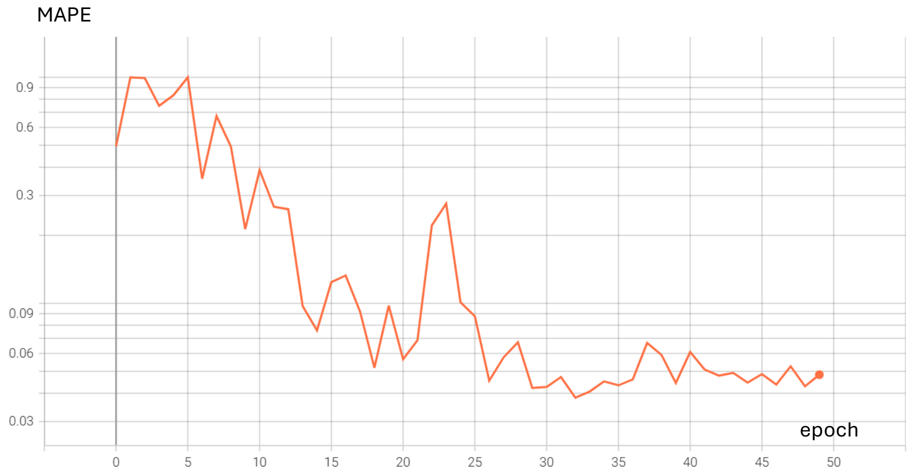

*Figure 17: MAPE of the Inlet Pressure of the Testing Set of the Original FNO Model with 4 Layers*

The validation set mirrors the distribution of the training set, so most samples are below 25% of the maximum encountered inlet pressure. In Figure 16 we can see how well the network can predict the values and, more importantly, that the error is independent of the value of the target, because the distances from the center line are about equal along its length.

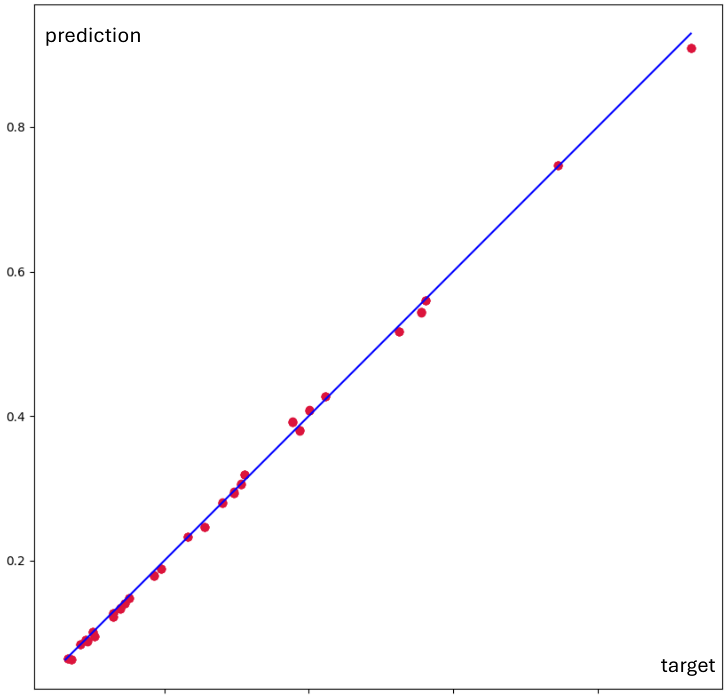

*Figure 16: Scatter plot of the inlet pressures from the 8-layer factorized model on the shifted spheres dataset*

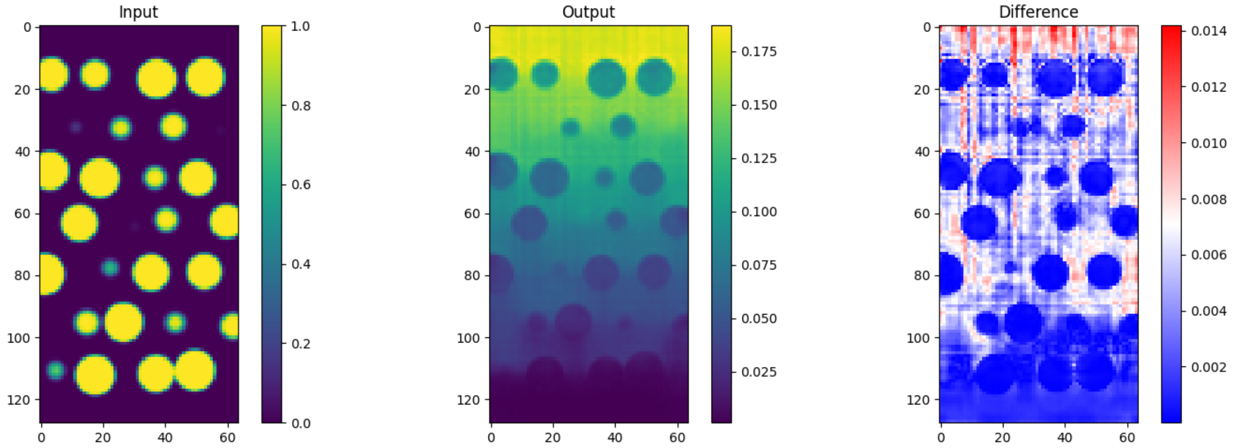

*Figure 18: Slices from spheres validation set results*

Table 1 displays the MAPE, R2 and MAE of the pressure difference as well as the worst samples absolute error (AE) together with the MSE and R2 of the entire pressure fields. Please note that all the error norms depend on the distribution of the data to differing degrees, making it hard to exactly compare them in between different datasets.

| Factorized | Layers | MSE field ($10^{-2}$) | MAE ($10^{-2}$) | MAPE | % $R^2$ Score | Max AE ($10^{-2}$) | Time per epoch in s | # Parameter |
|---|---|---|---|---|---|---|---|---|
| False | 2 | 0.039 | 1.20 | 4.04 | 99.18 | 6.86 | **24.28** | 1.34E+08 |
| False | 4 | 0.017 | 1.11 | 3.80 | 99.35 | 5.93 | 44.66 | 2.68E+08 |
| False | 6 | 0.040 | 1.46 | 4.80 | 98.84 | 7.01 | 54.75 | 4.03E+08 |
| False | 8 | 0.021 | 1.34 | 5.26 | 99.17 | 5.56 | 74.12 | 5.37E+08 |
| | | | | | | | | |
| True | 2 | 0.054 | 1.25 | 5.28 | 99.42 | 4.10 | 48.23 | **3.00E+05** |
| True | 4 | **0.006** | **0.51** | **2.02** | **99.91** | **1.44** | 102.32 | 5.80E+05 |
| True | 6 | 0.009 | 0.73 | 3.23 | 99.80 | 2.29 | 131.28 | 8.59E+05 |
| True | 8 | 0.008 | 0.56 | 2.66 | 99.88 | 1.98 | 171.56 | 1.14E+06 |

*Table 1: Results of the validation set of the shifted spheres dataset*

It is obvious from the high $R^2$ scores above 99% that all tested models are sufficient at representing the pressure difference in the ranges from the test and train set. The discernment of these models is hard to make because of their high performance and small differences. To test whether the differences are from random deviations caused by different initialization and ordering of the batches or from real improvement, the one setup was run 4 times with the same settings. From the range of the $R^2$ score of $[99.09, 99.57]$, we can see that the differences in inlet pressure prediction between the tested models in Table 1 cannot be clearly attributed to the setup changes. Only the entire field prediction seems to be less affected by random deviations in the factorized case and so this value can be more illuminating of the model's setup performance. It is only logical that the inlet pressure value, which is derived from only a small part of the domain, is more affected by random perturbations than the average value of the entire domain.

| MSE field ($10^{-2}$) | MAE ($10^{-2}$) | MAPE | % $R^2$ Score | Max AE ($10^{-2}$) |
|---|---|---|---|---|
| 0.017 | 1.11 | 3.80 | 99.35 | 5.93 |
| 0.020 | 1.49 | 5.76 | 99.09 | 5.17 |
| 0.016 | 0.99 | 3.88 | 99.56 | 3.74 |
| 0.012 | 0.96 | 3.78 | 99.57 | 5.27 |

*Table 2: Different runs of the same 8-layer factorized architecture*

So, the second learning from this experiment is that for simpler geometries like packed spheres with random shifts, while all models deliver a high accuracy, the deepest FFNO can generate the best results, but also the 6 times faster 2-layer original FNO works competently. In the end, because of the smaller models being already sufficient for the task, the deeper ones cannot improve meaningfully on this problem. This behavior indicates a close proximity to the best performance that FNOs can deliver in this task.

## 5.2 Complex Geometries & Small Datasets

Having established the realizability of the concept, the question raises where the limits to this method are. For this purpose, the DRP set was tested with widely varying geometries from many different types of porous media that come with two caveats. First the big difference in pressure ranges between samples and secondly the small dataset size with only 119 distinct geometries, where 952 resulting samples could be generated from. Especially transformer-based models struggle with small datasets, and for this reason usually utilize pretrained models for surrogate modeling there. Because the model is trained on the data in its exponentially transformed version, the loss calculation during evaluation happens after reverting this rescaling. Meaning that the result in the original domain of the problem is presented, because the question of this thesis is centered around the surrogate model's performance of the physical problem and not how well the NN works on its own. The accuracy in the re-linearized domain is thus worse than in the one where the training happened, but still better than with training without any transformation.

All trained networks show the same convergent behavior as seen in the previous chapter, with the weight normalization and the cosine learning rate scheduling, instabilities do not appear any more compared to earlier test runs. Table 3 shows the results in the same fashion as previously and Figure 19 illustrates a sample and its performance.

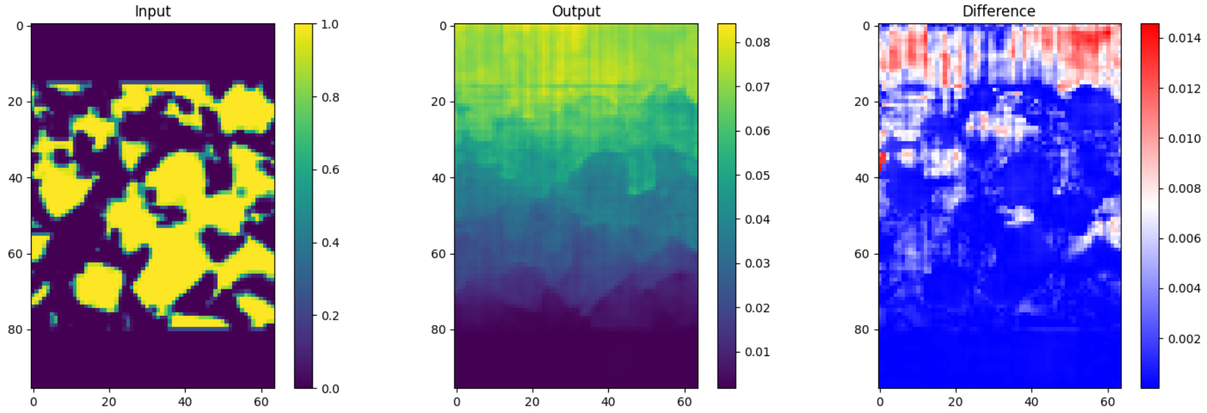

*Figure 19: Slice from the DRP validation set results*

| Factorized | Layers | MSE field ($10^{-2}$) | MAE ($10^{-2}$) | MAPE | % $R^2$ Score | Max AE ($10^{-2}$) | Time per epoch in s | # Parameter |
|---|---|---|---|---|---|---|---|---|
| False | 2 | 0.815 | 6.28 | 20.83 | 79.19 | 46.26 | **15.15** | 1.34E+08 |
| False | 4 | 0.902 | 6.56 | 29.25 | 79.85 | 52.77 | 24.97 | 2.01E+08 |
| False | 8 | 0.877 | 5.98 | 16.86 | 76.43 | 51.40 | 45.09 | 4.03E+08 |
| | | | | | | | | |
| True | 2 | 1.249 | 7.46 | 31.86 | 73.11 | 59.94 | 27.21 | 2.68E+05 |
| True | 4 | 1.061 | 6.00 | 13.68 | 68.19 | 70.02 | 49.50 | 5.14E+05 |
| True | 8 | 0.862 | **4.42** | **13.17** | **88.54** | **37.48** | 102.87 | 1.01E+06 |
| True | 12 | **0.662** | 5.40 | 16.79 | 77.57 | 65.52 | 139.31 | 1.50E+06 |

*Table 3: Results of the validation set from the DRP set*

Let's first discuss the most striking difference between this and the first experiment, the maximum absolute error. These range between $0.7$ and $0.37$ where the range of the predicted values itself is $[0,1]$, so an enormous error. Usually, these cases are geometries where only a small channel connects the inlet and outlet, like seen in Figure 21, with a target value in the upper half of the range. The scatter plot Figure 20 of the validation set shows how there is a correlation between predicted value and variability highlighted by the dashed line.

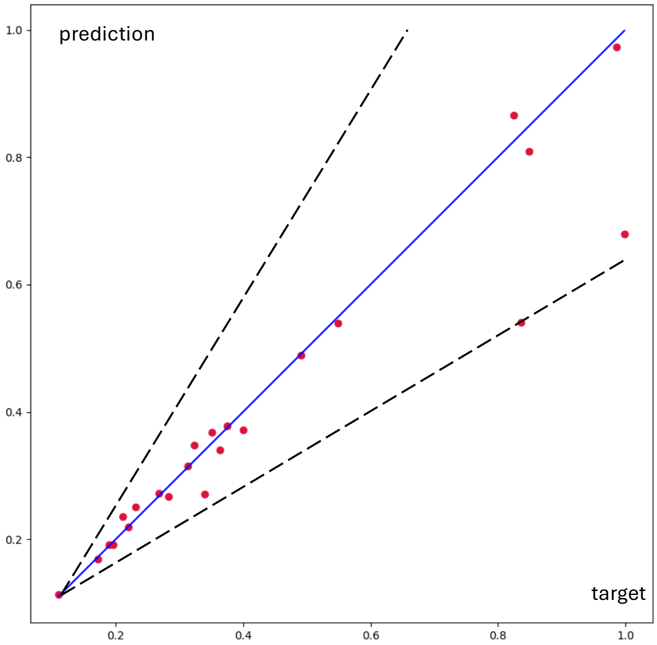

*Figure 20: Scatter plot of the inlet pressures from the 8-layer factorized model on the DRP dataset*

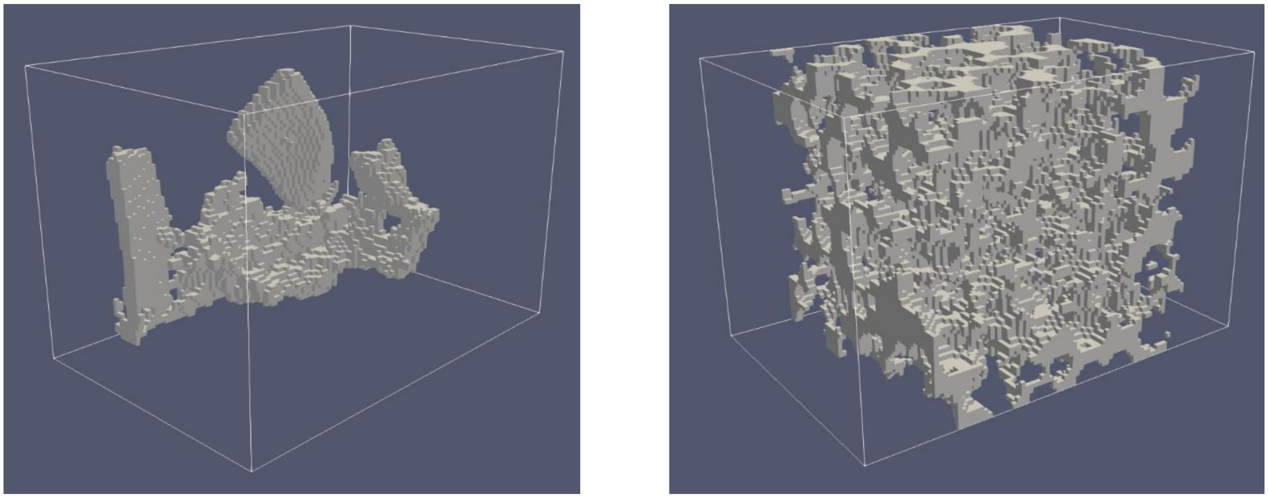

*Figure 21: Fluid domains from the validation set with the worst sample (left, MAPE: 29.6) and one average sample (right, MAPE: 1.69)*

The error metrics are driven mainly by these outliers and due to the MSE loss function dominating the gradients during training, hindering the fine tuning of other geometries. This is underlined by the difference between the 8-layer and the 12-layer FFNO run, where the shallower model performs significantly better in the inlet prediction, because its highest MAE sample is half of the deeper model while the more stable field prediction overall is worse. Other runs of the 8-layer setup show a higher maximum AE value and thus perform more in between the 4- and 8-layer configuration, showing again the high volatility introduced by these outlier examples. As previously, the field MSE shows the performance improvement of deeper FFNO model, where the standard FNO model does not improve from more layers after 4. Still, the inlet prediction underlies high inter-run and lower inter-layer differences which, again, makes it hard differentiating the setups. But in general, a small advantage of deeper and slower FFNOs can be made out. Nonetheless it is noteworthy that the models converge at all for a complex dataset like this with the relatively low number of samples, by still providing a competent assignment of inlet pressure for most of the samples. Its robustness prevents the hard to predict outliers from disturbing the training to that degree, that all samples are predicted badly.

To further investigate the effects of the sparse high inlet pressure samples, a second filtered dataset was created that contains the lowest 90% of all samples. Due to the distribution of the set, the new one only covers the pressure range up to around 60% of the previous. *Table 4* shows the results of the models on the new filtered, renormalized and scaled set.

| Factorized | Layers | MSE field ($10^{-2}$) | MAE ($10^{-2}$) | MAPE | % $R^2$ Score | Max AE ($10^{-2}$) | Time per epoch in s | # Parameter |
|---|---|---|---|---|---|---|---|---|
| False | 4 | 0.339 | 6.10 | 35.57 | 83.56 | 23.75 | **26.25** | 2.01E+08 |
| False | 8 | 0.324 | 5.57 | 44.46 | 84.86 | 27.56 | 47.20 | 4.03E+08 |
| | | | | | | | | |
| True | 4 | 0.191 | 3.80 | 19.03 | 90.73 | 20.75 | 50.67 | 5.14E+05 |
| True | 8 | **0.106** | **3.06** | **16.60** | **94.62** | **12.34** | 96.75 | 1.01E+06 |
| True | 12 | 0.128 | 3.57 | 17.47 | 92.97 | 18.68 | 123.80 | 1.50E+06 |

*Table 4:Results of the validation set from the filtered DRP set*

All models perform generally better on the subset compared to the full one, most obviously in the highest MAE samples, further supporting the thesis, that the high-pressure values limit the performance. The last scatter plot still displays the described correlation of the error, but to a lesser extent and with a lower mean value as well. The best $R^2$ score with 94,62% is again in the 8-layer FFNO model and now also with the lowest error in the entire field. Also, all factorized versions are better than the FNO in this table above the 2-layer setting. Due to the also 10% smaller and newly chosen verification set, the comparison between this and the unfiltered set is not perfect and thus the perturbations should be

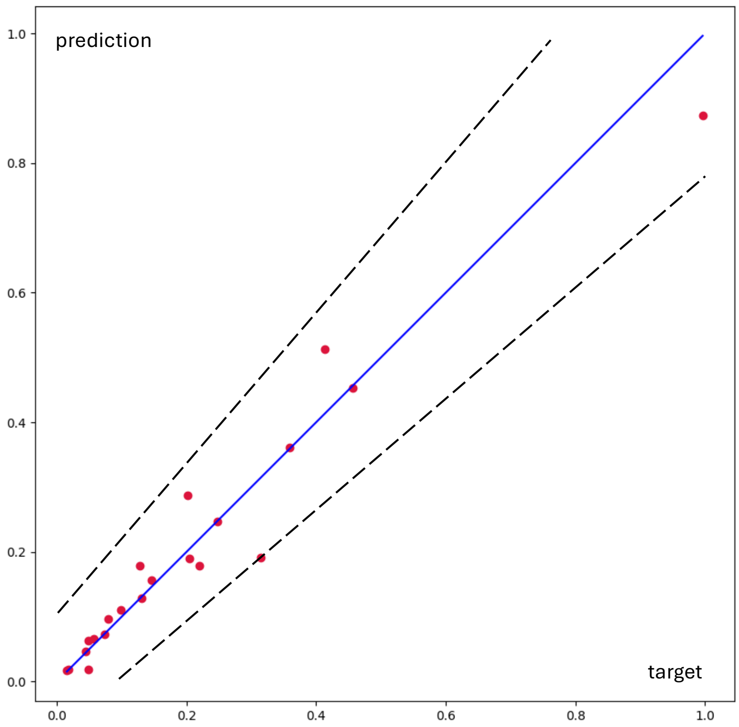

*Figure 22: Scatter plot of the inlet pressures from the 8-layer factorized model on the filtered DRP dataset*

accounted for. Nonetheless, more runs with a different verification set have confirmed the general better performance of the filtered setups.

One question that remains being open is, does the performance increase if the set sizes increase, a metric called scalability. Due to the scope of this thesis and the limited DRP overview set, this was not analyzed here, but the discussion at the end, will touch this question again.

## 5.3 2D Comparison with the SWIN Transformer

Finaly it’s time to compare the FNO against some competitor from the machine learning field, the swin transformer. The paper from (Geng, Zhai, & Li, 2024) illustrates applying the model on the here discussed problem of predicting permeability, but in 2D. Apart from the missing dimension, the setup is identical to the 3D setup, apart from the encoder, which is only a feed-forward block with a factor of 4 for the hidden channels and without an activation function. The SIREN did not deliver good results during testing. Also, their approach directly learns the permeability and does not go the route of predicting the entire field. It can be interpreted as a less physical based procedure and closer to the pure ML approach.

| Type | Layer | Dataset | MSE field ($10^{-2}$) | MAE ($10^{-2}$) | MAPE | $R^2$ Score | Max AE ($10^{-2}$) | Time per epoch in s |
|---|---|---|---|---|---|---|---|---|
| original | 2 | standard | 1.553 | 2.08 | 13.76 | 92.41 | 20.83 | **3.49** |
| original | 4 | standard | 1.374 | 1.68 | 8.44 | 94.06 | 28.29 | 4.51 |
| original | 8 | standard | 1.699 | 1.60 | 7.94 | 94.16 | 27.32 | 7.76 |
| | | | | | | | | |
| factorized | 2 | standard | 0.829 | 1.27 | 7.55 | 97.70 | 9.61 | 5.63 |
| factorized | 4 | standard | 0.442 | 0.86 | 5.33 | **98.78** | 10.68 | 8.62 |
| factorized | 8 | standard | **0.403** | **0.84** | 4.65 | **98.78** | **10.11** | 16.10 |
| factorized | 12 | standard | 0.530 | 0.93 | 4.97 | 98.50 | 13.63 | 23.85 |
| factorized | 16 | standard | 0.563 | 0.85 | **4.28** | 98.50 | 13.42 | 31.48 |
| | | | | | | | | |
| factorized | 8 | full (filtered 99%) | 1.249 | 1.46 | 15.89 | 94.37 | 18.42 | 28.09 |
| factorized | 8 | full | 0.658 | 0.70 | 17.02 | 94.50 | 23.00 | 31.36 |
| | | | | | | | | |
| factorized | 8 | rocks | 4.847 | 3.44 | 54.81 | 74.95 | 16.75 | 5.32 |
| factorized | 8 | rocks (on full) | 5.661 | 3.46 | 26.45 | 93.29 | 23.00 | 31.36 |
| factorized | 8 | noisy | 11.377 | 6.03 | 50.16 | 45.17 | 36.18 | 11.50 |
| factorized | 8 | noisy (on full) | 0.139 | 0.60 | 35.38 | 47.99 | 4.69 | 31.36 |

*Table 5: Results of the validation set on the 2D datasets*

Table 5 shows all validation set results of the 2D dataset with the first two columns being the artificially created porous media geometries, then the combined dataset of all 2D media, standard, rock images and noisy geometries and at the end the additional datasets itself. All factorized models show finally a significantly better performance than any original model with the 8-layer approach as best setup with an $R^2$ score of 98.78%. The maximum derivation is $10.11\%$ and occurs in the sparse high-pressure sample as seen in Figure 23, continuing the trend of difficulties in this region as in 3D.

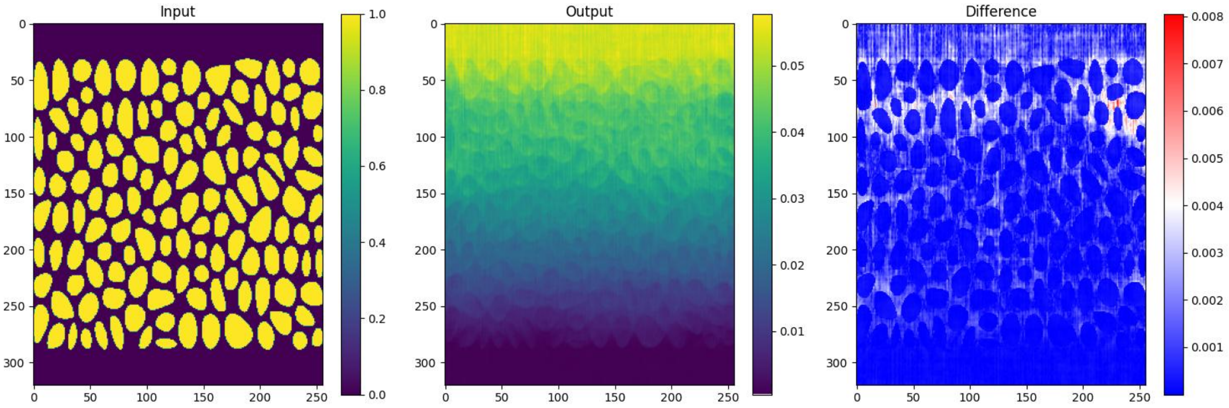

*Figure 24: Sample from the standard 2D set results*

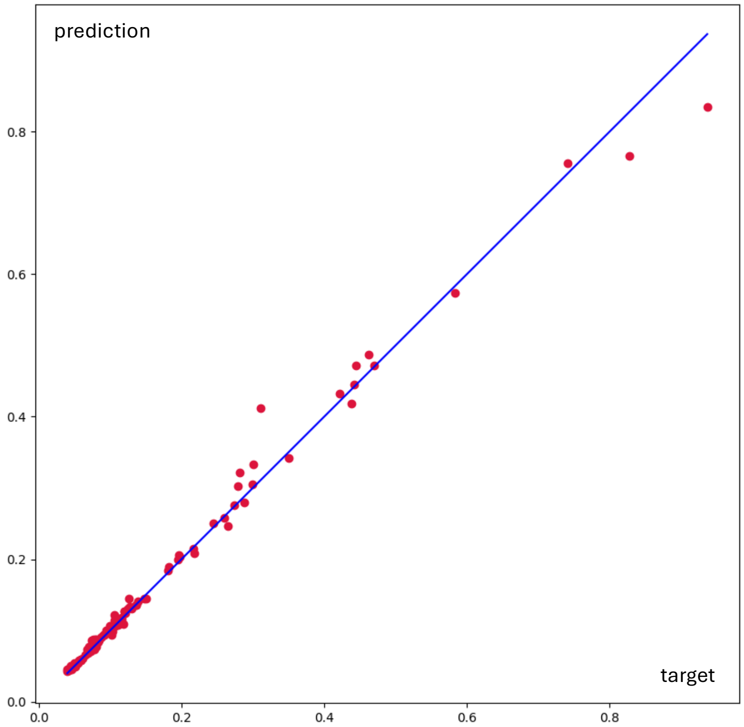

*Figure 23: Scatter plot of the inlet pressures from the 8-Layer factorized model on the standard 2D dataset*

The performance remains good with an $R^2$ score $94.50\%$ when all sets are combined, but the MAPE increases significantly, showing that the prediction of the smaller pressure samples are predicted worse. When looking at the small datasets of noisy ($\sim 500$ samples) and rock data (~200 samples), the error increases. To mitigate the risk of bad performance due to limited samples, another test run was performed with only the noisy or rock samples of the full validation set on the model also trained on the full set. Here, both cases performed better than when they are trained on only their limited set, but note that the min-max normalization of the sets is different and that the absolute metrics cannot be compared directly. The rock performance of an $R^2$ score between 74.95% and 93.29% on the full shows that the network still delivers an approximation of the inlet pressure, where the noisy falls completely apart with under 50%. The noisy edges of the geometries pose multiple challenges to the model, first higher gradients in the input of the network itself and secondly the high importance of the input resolution. The network does not see the full resolution the result data is simulated on, removing some fine-grained detail important for estimating the resistance to the slip of the walls.

| Type | Dataset | $R^2$ Score | time per epoch in s |
|---|---|---|---|
| SWIN transformer | standard | 98.14 | ~140 |
| SWIN transformer | rocks | 93.82 | ~80 |
| SWIN transformer | noisy | 92.35 | ~30 |

*Table 6: Results of the 2D dataset with the swin transformer*

Table 6 shows the results of the Swin transformer from their paper with their setup and data processing. They apply another method of reducing the imbalance and do not state if they calculate their final results of the validation set back in the original domain, as here. Both the transformer as well as the FFNO perform excellently on the standard dataset, but the FNOs leads marginally with 0.64 percentage points on the $R^2$ score. From their absolute error distribution, a maximum absolute error of ~12% can be extracted, which is worse than the 10.11% of the FNO and underlines its inferiority. When the FNO is trained on the full set in the rock case, the transformer is a bit better, but both still deliver competent results. Interestingly, the performance of the noisy images continues to be good on the competitor with a $R^2$ score of 92.35% where it fails on the operator. This might be due to the usage of a pretrained model of the transformer, where a smaller dataset matters less to performance, because a baseline of image interpretation is already given. The patch partitioning might also support the detection of the smaller details on the boundaries. The biggest difference between the two approaches is the speed of training, while the best performing 4-layer FFNO takes 9 seconds for one epoch, the transformer needs 140. This one order of magnitude difference can be explained by the high complexity and size of the transformer.

This chapter showed how the Fourier neural operator can predict the permeability of porous media in 2 and 3 dimensions successfully and on par with competing neural networks, with a faction of its training time. When the dataset contains a high complexity relative to its size, it still delivers useful results, but especially high-pressure geometry predictions and noisy geometries suffer. The next chapter will compile the learnings from this chapter, interpret and discuss them in detail.

# 6 Discussion

Now it's time to answer the broader remaining questions proposed at the beginning of this work: What knowledge can be gained from analyzing the runs about the settings of the neural network? In what cases are FNOs useful compared to other methods of estimating the permeability?

## 6.1 Hyperparameter Settings for FNOs

Many choices like the number of modes and factors in the feed forward block have already been discussed in the 4th chapter and evaluated in separate test runs or publications. Because these settings are assumed to be universal to this task, the focus of the experiments in the 5th chapter is on the type of the network, factorized or original, and the number of layers. Depending on the underlying dataset and dimensionality, the optimal settings can vary.

The choice of factorized over the original architecture is not as obvious as suggested in their proposal paper, because the original is more than twice as fast when trained with the same number of layers here, even if it has 3 orders of magnitude more learnable weights. Why this obvious difference in speed is present here, but not in the reference paper, is hard to evaluate, but it might be intrinsic to the combination of GPU and software and how well the kernel application in Fourier space is optimized. Additionally, the improvement of the accuracy is also not as clear. The best results in all datasets are from the factorized architecture, but the improvement is not always meaningful like in the spheres set with 99.35 vs 99.91 or that most factorized models performed worse on the unfiltered DRP set. Only in the 2D case, the factorized is clearly superior. The takeaway here is, while the factorized can be better than the simpler approach, it comes at a cost in speed, requiring careful weighting of priorities.

In theory, a deeper model with more layers can capture more complexities and thus improve accuracy. This only works when the problem is underrepresented in a smaller configuration and the gradients can flow through from back to front. The skip connections of the factorized FNO should allow for this, while the original misses such additional measures. However here, only some of this theory is reflected in the results. More than the proposed 4 layers of the original architecture worsen the performance in most cases tested and this can be well explained with vanishing gradients. Contrary to the paper, the FFNOs do not scale with increasing layer count either, with on average the 8-layer depth performing the best and unstable performance afterwards. The caveat with a derivative value like the inlet sections average is, that while the field accuracy is still increasing, the improvement is not always carried on to the desired value. This behavior might indicate that the model is sufficient at representing the result good enough and that more depth increases opportunities to learn wrong confounders. Generally, a more complex set should allow for a better scaling with increasing depth, but the here tested DRP set has not enough samples of the high outlier pressures and thus the model cannot be saturated as well.

It turns out that for the here tested sets, both approaches show plateauing at 4 and 8 layers while the difference between them is in a suboptimal relation to the runtime. Therefore, when speed is the main objective, the 2- or 4-layer original setup is ideal and when performance is key, an 8-layer factorized architecture is preferred.

## 6.2 Use Cases for Surrogate Models based on FNOs

The results from the last chapter already answered the realizability and limits of the FNOs, but now the broader question is about comparing it to alternatives to neural networks. First the numerical simulation with *lbmpy* as comparison. Simulating one sample of the DRP dataset on the same *Nvidia A100 80GB* GPU takes up to 414 seconds with an average of about half, depending on the convergence of the geometry. Just the inference of one sample, so when a model is already trained, it takes ~0.15 seconds depending on its configuration. The difference of 3 orders of magnitude is impressive, but this comparison is not fair. First, the LBM simulated data is more accurate in general and this accuracy does not depend on the size and complexity of a train dataset. Secondly, the model needs to be trained, which takes time, as well as the time to generate the data, usually with LBM itself. The training time for this example with an 8-layer factorized setup is around 7.5 seconds per sample. So, the improved speed of the network is bought with an overhead that is a formula of range of geometries the model should be able to tackle and how many samples should be analyzed afterwards.

$$Speedup = \frac{n_{test} * T_{LBM}}{n_{test} * T_{FNO_{inference}} + n_{train} * \left(T_{FNO_{train}} + T_{LBM}\right)}$$

Equation 26: Speedup of NN prediction over numerical simulation

Calculating the breakeven point of number of test samples for the DRP model for the FNO with the here used problem is ~275, when the model is trained on 250 geometries. Note that all these new test samples need to be represented somewhere in the range of permeabilities in the trainset to get a proper approximation. The energy consumed was not recorded exactly, but due to both methods running on the same GPU mostly close to its power limits, the energy relations are close to the time ones.

Another method of choice can be analytical formula. They require to be tailored to the underlying type of geometry analyzed, but when this is the case, they are incomparably faster than the previous types of methods. The first test with the shifted spheres is such a case, where the porosity and diameter of the objects in the *Kozeny-Carman* equation (Equation 2) deliver an estimate of the pressure gradient. The error plot in Figure 25 shows how the R$^2$ score of 95.04 is distributed on the set. So, when an analytical solution is available, it's a viable solution to get a very fast rough estimate of the problem, but for example the DRP set is too diverse to be solved by a formula like this one. There are certainly more elaborate approximation functions for a multiplicity of problems, but they always rely on an understanding of the idiosyncrasies of the set of geometries analyzed.

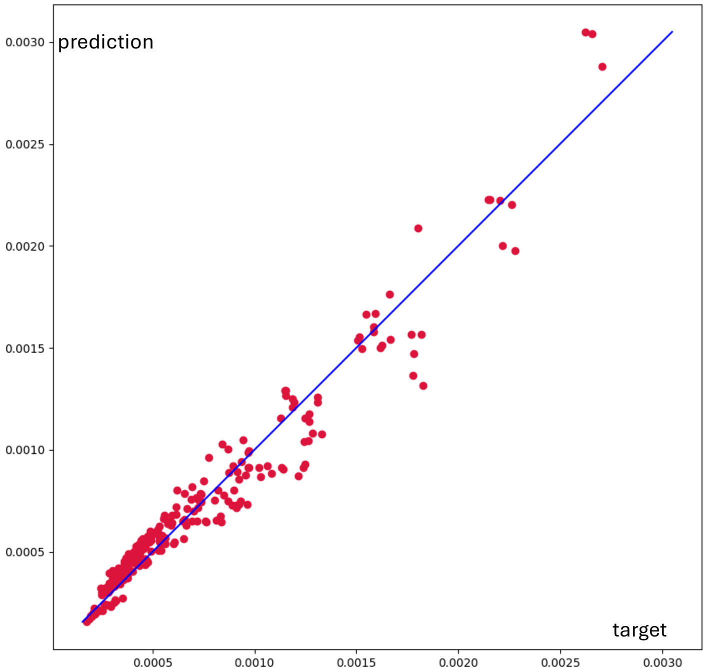

*Figure 25: Scatter plot of the results from the Kozeny-Carman equation*

The above-mentioned alternatives sort themselves nicely in the following diagram with performance at the y-axis and universality at the x-axis. Finally, it can be understood that the FNOs have their own sweet spot in cases where a lot of estimations are needed, that all fall into a known range of complexities.

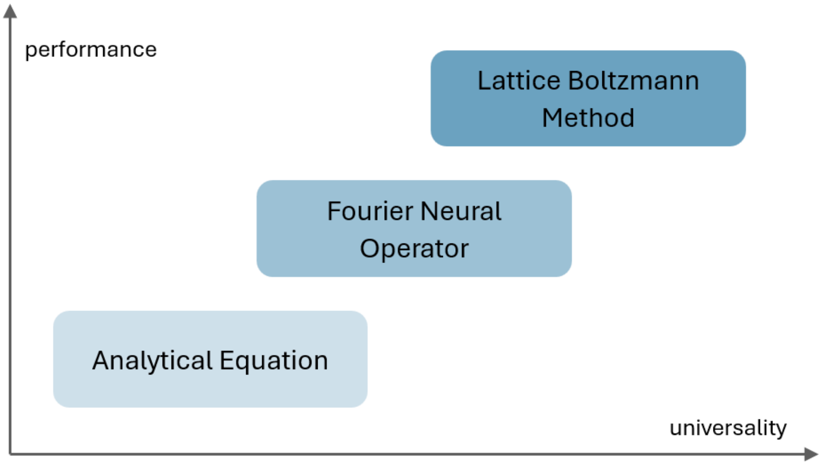

*Figure 26: Categorization of prediction methods*

# 7 Conclusion

## 7.1 Summary

This work captured all the steps necessary to predict the permeability of any geometry with neural networks. One core idea of the methodology is to predict the entire pressure field, which adheres to the Navier-Stokes equation and thus can be captured well by neural operators. Fourier neural operators utilize the frequency space to apply the weights to mimic the physical field efficiently and accurately and the cutoff of higher frequencies alleviate noise additionally. The data needed for training was simulated by the Lattice-Boltzmann method on three different datasets in 2D and 3D. To accomplish the best understandability of the data for the network, normalization as well as non-linear scaling was applied to be able to preserve details in the more common lower range of inlet pressures.

Then the tuning of the FNO and its proposed improvement, the factorized Fourier neural operator, was presented. The factorized architecture has separate weights and convolutions for every dimension and allows for a deeper network by introduction skip connections paired with other changes. The loss was calculated from a masked version of the MSE, which concentrates on the non-solid flow region of the geometry with special attention on the inlet section. It turned out the weight normalization and a cosine learning rate with a warmup phase increase stability while the AdamW optimizer provides modern weight adaptation.

Three tests evaluated the performance in different scenarios. First, the realizability was proven with a simpler 3D geometry of random shifted same sized spheres, where it achieved an excellent $R^2$ score of 99.88. Then a small and complex 3D dataset extracted from all types of porous media with 119 distinct examples to test the limits of FNOs. It turned out that the network was still able to deliver an $R^2$ score of 88.54 with a maximal absolute error of 0.37 with some rare and complicated outlier cases dominating the performance. Thus, the same setup on only the lowest 90% of pressures samples resulted again in a performance above an $R^2$ score of 90%. The last scenario checked was a generated 2D porous media that was tested in a different work on swin transformers with the same task. The FNOs improved over the competitors with 98.78 compared to the 98.14 of the swin with a decreased maximum absolute error while being significantly faster. Only when it comes to the small noisy set, the alternatives can maintain a decent result, while the FNOs are not able to represent the set anymore.

The final chapter discussed more broadly the use case of these surrogate models based on neural networks and how to set them up according to own requirements. While bigger in weights, the standard 4-layer FNO in 3D cases delivers a good performance at the best speeds with training times under 45 minutes on the spheres set. When the best accuracy is needed, the 8-layer FFNO is the choice with a training time above 2 hours. Probably the limitations of the train sets sizes and the comparative simple metric, which is the mean value of the pressure field at the inlet, cause stagnating results with deeper network with the factorized architecture, while other works on different problems can take advantage of its capabilities. The original FNO does not improve above 4 layers, because it lacks gradient preservation measures. When compared to analytic solutions and numerical simulations, the neural network surrogate model bridges the gap between the fast estimation of the formulas and the precision of the simulation, while allowing of a high degree of generality, when trained on a broad dataset. The caveat comes from its upfront cost of time and care and time towards generating the dataset. Therefore, an estimation of the range of input geometries needs to be made first and a certain threshold of samples generated afterwards to mitigate the efforts made during training. A rough calculation showed that at least 275 estimations need to be produced after simulation and training on a ~250 geometries dataset to be faster than the LBM simulation on the same hardware.

Finally, it can be concluded that FNOs are a viable solution for modeling the permeabilities of porous media and because of their robustness and performance a good choice of the plethora of neural networks for this task. If the training effort is worth it, still depends on the intensity a model is used afterwards, thus it is an addition to numerical simulation and analytical estimation, but no replacement.

Now, the conclusion can be summed up as answers to the five formulated goals from the beginning:

- Are Fourier neural operators even capable of capturing the pressure field to a reliable degree?

  Yes, the first experiment showed a converging behavior and achieved an $R^2$ score of above 99%.

- What scope of porous media can a model cover with what requirements?

  A wider range of geometries was tested in the second experiment, which still was able to deliver reliable results unless in some extreme underrepresented cases. So, if enough samples of different geometries are provided, wide scopes can be tackled.

- How to improve the performance by tuning the hyperparameters of the model as well as the training process?

  In these tested cases, the original FNO with 4-layers has the best speed-to-performance ration, but the 8-layer FFNO enables the best results.

- How is the efficiency and speed compared to other networks, numerical simulation and analytical solutions?

  Compared to Swin transformer, they perform about the same in a fraction of its time. The FNO surrogate model is a compromise between the high speed and specificity of analytical solutions and the universality and slow speed of numerical simulations.

- In what cases are surrogate models based on neural networks useful?

  When many of samples of geometries that fall into a predictable range of permeabilities need to be evaluated quickly or many similar cases have already been analyzed, the model training is useful.

## 7.2 Limitation & Future Work

While answering many questions from the beginning of this work, new ones appeared that stayed unanswered because of the scope of this work. While multiple hyperparameter settings like the type of optimizer and normalizations have been tested aside from this report and in other papers, they haven't been discussed in detail and in every combination. The focus remained on the layer count and factorized versus original and not mode and channel depth. Also, the question of scaling the performance to broader and bigger datasets, to get closer to the generalizability of LBM has not been tackled. This is the setting where transformers made great achievements in deep learning. Additionally, it would be interesting to see the difference between predicting the pressure field and then deriving the permeability compared to directly training on this one singular value like the swin in more detail. One aspect of NOs that stays untouched here is the invariance to resolution, thus it would be interesting to leverage this property to achieve a higher accuracy or speedup during training by varying the input resolution according to the goal.

The concept of physically informed neural networks is attractive was well. To add it to FNOs, the loss needs to calculate it from the output by applying the Navier-Stokes equations. For this to work, the velocity fields need to be predicted also, increasing the weight size and epoch time. Possible benefits of this would be reducing the need to simulate the training data beforehand and thus more flexibility.

The research field of neural network in physical simulation is quickly evolving and for this reason, the application of recent advancements to classical problems is fruitful and interesting. But because of improved numerical and computational capabilities, also the traditional simulation methods remain of great interest.

# References

Bauer, M., Köstler, H., & Rüde, U. (2021). lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods. *Journal of Computational Science*(49). Retrieved from https://doi.org/10.1016/j.jocs.2020.101269

Bear, J. (2018). *Modeling phenomena of flow and transport in porous media*. Springer International Publishing.

Bhatnagar, P. L., Gross, E. P., & Krook, M. (1954). A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*(94), 511-525. Retrieved from https://doi.org/10.1103/PhysRev.94.511

Carman, P. C. (1956). Flow of Gases Through Porous Media. *Chemical Engineering Research and Design*.

Darcy, H. (1856). *Les fontaines publiques de la ville de Dijon*. University of Gent: Victor Dalmont. Retrieved from https://gallica.bnf.fr/ark:/12148/bpt6k624312/f1n657.pdf

Fattahi, E., Waluga, C., Wohlmuth, B., Rüde, U., Manhart, M., & Helmig, R. (2016). Lattice Boltzmann methods in porous media simulations: From laminar to turbulent flow. *Computers & Fluids*(140), 247-259. Retrieved from https://doi.org/10.1016/j.compfluid.2016.10.007

Feichtinger, C., Donath, S., Köstler, H., Götz, J., & Rüde, U. (2011). WaLBerla: HPC software design for computational engineering simulations. *Journal of Computational Science, 2*(2). Retrieved from https://doi.org/10.1016/j.jocs.2011.01.004

Geng, S., Zhai, S., & Li, C. (2024). Swin transformer based transfer learning model for predicting porous media permeability from 2D images. *Computers and Geotechnics, 168*, 106-177. Retrieved from http://dx.doi.org/10.1016/j.compgeo.2024.106177

Ginzburg, I. (2008). Consistent lattice Boltzmann schemes for the Brinkman model of porous flow and infinite Chapman-Enskog expansion. *Phys. Rev. E, 6*(77), 066704. Retrieved from https://doi.org/10.1103/PhysRevE.77.066704

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770-778. Retrieved from https://doi.org/10.1109/CVPR.2016.90

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*. Retrieved from https://doi.org/10.48550/arXiv.1606.08415

Kemmler, S., Rettinger, C., & Köstler, H. (2023). Efficient and scalable hybrid fluid-particle simulations with geometrically resolved particles on heterogeneous CPU-GPU architectures. *Computational Engineering, Finance, and Science*. Retrieved from https://doi.org/10.48550/arXiv.2303.11811

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *Journal of Machine Learning Research*(24), 1-97. Retrieved from https://doi.org/10.48550/arXiv.2108.08481

Krüger, T., Kusumaatmaja, H., Kuzmin, A., Shardt, O., Silva, G., & Viggen, E. M. (2017). The Lattice Boltzmann Method. *Springer International Publishing*. Retrieved from https://doi.org/10.1007/978-3-319-44649-3

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020). Fourier Neural Operator for Parametric Partial Differential Equations. *arXiv preprint*. Retrieved from https://doi.org/10.48550/arXiv.2010.08895

Loshchilov, I., & Hutter, F. (2017). Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*(5). Retrieved from https://doi.org/10.48550/arXiv.2010.08895

Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence, 3*(3), 218-229. Retrieved from https://doi.org/10.1038/s42256-021-00302-5

Queipo, N., Haftka, R., Shyy, W., Goel, T., Vaidyanathan, R., & Tucker, P. (2005). Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*(41), 1-28. Retrieved from https://doi.org/10.1016/j.paerosci.2005.02.001

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*(29). Retrieved from https://doi.org/10.48550/arXiv.1602.07868

Santos, J. E., Yin, Y., Prodanovic, M., Gigliotti, A., Lubbers, N., & Khan, H. J. (2021, July 9). *3D Collection of Binary Images*. (D. R. Project, Editor) Retrieved from https://www.digitalrocksportal.org/projects/374

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., & Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 7462-7473. Retrieved from https://proceedings.neurips.cc/paper/2020/hash/53c04118df112c13a8c34b38343b9c10-Abstract.html

Smith, D. G., & Gray, J. (2018). opt_einsum - A Python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software, 3*(26), 753. Retrieved from https://doi.org/10.21105/joss.00753

Tran, A., Mathews, A., Xie, L., & Ong, C. S. (2021). Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*. Retrieved from https://doi.org/10.48550/arXiv.2111.13802

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*. Retrieved from https://dl.acm.org/doi/10.5555/3295222.3295349