Solving the Discretised Neutron Diffusion Equations using Neural Networks

Toby R.F. Phillips[a], Claire E. Heaney[a], Boyang Chen[a], Andrew G. Buchan[b], Christopher C. Pain[a]

[a]*Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, London, SW7 2AZ United Kingdom*

[b]*School of Engineering and Materials Science, Queen Mary University of London, London, E1 4NS United Kingdom*

Abstract

This paper presents a new approach which uses the tools within Artificial Intelligence (AI) software libraries as an alternative way of solving partial differential equations (PDEs) that have been discretised using standard numerical methods. In particular, we describe how to represent numerical discretisations arising from the finite volume and finite element methods by pre-determining the weights of convolutional layers within a neural network. As the weights are defined by the discretisation scheme, no training of the network is required and the solutions obtained are identical (accounting for solver tolerances) to those obtained with standard codes often written in Fortran or C++. We also explain how to implement the Jacobi method and a multigrid solver using the functions available in AI libraries. For the latter, we use a U-Net architecture which is able to represent a sawtooth multigrid method. A benefit of using AI libraries in this way is that one can exploit their power and their built-in technologies. For example, their executions are already optimised for different computer architectures, whether it be CPUs, GPUs or new-generation AI processors.

In this article, we apply the proposed approach to eigenvalue problems in reactor physics where neutron transport is described by diffusion theory. For a fuel assembly benchmark, we demonstrate that the solution obtained from our new approach is the same (accounting for solver tolerances) as that obtained from the same discretisation coded in a standard way using Fortran. We then proceed to solve a reactor core benchmark using the new approach.

Keywords: Numerical solution of partial differential equations; Finite Difference Method; Finite Volume Methods; Convolutional Neural Network; Multigrid Solver; U-Net; Neutron Diffusion Equation; Reactor Physics

1. Introduction

Development of new computational hardware brings with it the challenge of adapting code in order for it to be deployed successfully on these new architectures. In the field of Artificial Intelligence (AI), this

challenge has largely been met by writers of and contributors to widely used AI libraries (for example TensorFlow [1] and PyTorch [2]). In these libraries, code relating to the architecture has been abstracted away so that users can concentrate on the algorithm they wish to implement without having to think about or understand the code relating to the computer architecture. As a result, the user has only to make minimal changes to their code in order to run on Central Processing Units (CPUs) or Graphical Processing Units (GPUs) or even Tensor Processing Units (TPUs). In other fields, such as scientific computation, perhaps because the codes and libraries are less standard and more numerous, users have to expend much more effort to run their codes on new architectures. Porting code to clusters of CPUs is relatively straightforward nowadays, however the computational gains to be had by running on clusters are limited by memory access and data transfer. Although GPUs have demonstrated superior performance to CPUs, instructions for the GPU must be written in languages such as CUDA or OpenCL that are unfamiliar to many working in scientific computation. This additional coding task has hindered the take-up of GPUs, although there are examples of this having been done successfully, for example, in computational fluid dynamics [3], for acoustic waves [4] and, in radiation transport, for a Monte Carlo neutron transport code [5] and for eigenvalue problems [6]. With CUDA and OpenCL, GPUs have been used to accelerate generation of finite element matrices for unstructured meshes [7–10] and for discontinuous Galerkin methods [11]. Recently, new types of processors have been unveiled, which have been designed specifically for tasks associated with AI such as matrix multiplication and vector operations. These processors are therefore also suited to the linear algebra calculations that arise in the area of scientific computation [12]. Furthermore, these new processors are designed to be more energy efficient than CPUs or GPUs, with hundreds of thousands of cores on a single chip, making it ideal hardware for researchers to run computationally demanding problems in an energy-efficient manner. AI libraries are already up-and-running on these so-called AI processors, which include TPUs of Google [12], Intelligence Processing Units (IPUs) of Graphcore [13] and CS-2 of Cerebras [14]. In order to exploit the speed of GPUs or AI processors for scientific computations, this paper outlines a method of formulating numerical discretisations in terms of operations or functions found in AI libraries, such as discrete convolutions. Writing discretisations in this way means that code can be deployed on whichever platform is available, whether it be CPUs, GPUs or the new AI processors, without having to make major modifications to the code.

Previous work that exploits the linear algebra capabilities of AI processors by using AI libraries to solve scientific problems includes applications in distributed Fourier Transforms [15, 16]; Monte Carlo simulations for finance [17]; many-body quantum physics [18]; and density functional theory [19]. We have found four examples of previous work that exploits operations associated with neural networks that can be found

within AI libraries in order to solve scientific problems [20–23]. Zhao et al. [20] were the first to equate a finite difference discretisation of the Navier-Stokes equations with a convolutional neural network in which the weights were determined by the discretisation. For validation, they use a number of benchmark tests including lid-driven cavity flow and flow past a cylinder. Wang et al. [21] present a similar idea to Zhao et al. [20], again using TensorFlow to implement finite difference discretisations of CFD problems, however using TPUs rather than GPUs. They solve the variable-density Navier-Stokes equations and demonstrate good weak and strong scaling. Chen et al. [22] implement both a finite difference and a finite element discretisation through convolutional neural networks in order to solve a number of CFD problems. They develop a method of solving the discretised systems based on a combination of a sawtooth multigrid method and the Jacobi method implemented as a U-Net [24] (a convolutional neural network with a specific architecture). Using convolutional neural networks with pre-determined weights, Phillips et al. [23] implement an upwind finite volume discretisation and several finite element discretisations arising from a new convolutional finite element method (ConvFEM). The application they study, radiation transport, requires development of a 4D multigrid method, again, based on the U-Net. Researchers have previously noted the similarity between the multigrid method and the encoder-decoder type of neural networks, such as the U-Net [24]. Consequently, the use of multigrid-inspired architectures for (trained) neural networks has been explored and been shown to enhance performance relative to conventional CNN architecture for applications in computer vision [25, 26] and in computational fluid dynamics (CFD) [27, 28]. Taking a different approach, Margenberg et al. [29] use a (trained) neural network to produce solutions for the finer levels of a multigrid and standard CFD solvers to produce solutions at the coarser levels. By contrast to the previous examples of integrating the multigrid method with neural networks [25–29], Chen et al. [22] implements a multigrid method using (untrained) neural networks with pre-determined weights to solve the PDEs on the coarse levels, and determine the residuals and provide Jacobi relaxation on finer levels. It is this method that is adopted in our current investigation.

In this paper we describe how to implement a finite volume discretisation of the neutron diffusion equation using a convolutional neural network whose weights are pre-determined by the particlar discretisation scheme. (In this case our finite volume discretisation is equivalent to a finite difference discretisation.) We also use this approach to implement a quadratic finite-element discretisation for the neutron diffusion equation. The Jacobi method and a sawtooth mutligrid method are used as solvers, implemented through standard operations found in AI software libraries. We demonstrate the approach using a fuel assembly benchmark, and compare the solution for the spatial variation of the neutron flux with that obtained from the same discretisation coded in a standard way using Fortran. We then proceed to solve a reactor core benchmark using the proposed method. The approach described in this article is a new and alter-

native way of harnessing AI technologies for forming solutions of governing PDEs. Ultimately solutions obtained through this new approach are identical to those obtained by standard codes, but the advantage of performing all operations through an AI library is that the code will run efficiently on all architectures. Furthermore, through the neural networks, the latest developments can be realised for methods such as sensitivities [30] , uncertainty quantification [31, 32] and data assimilation [33]. Although AI is becoming popular for nuclear engineering, it is often through surrogate modelling, which requires training a neural network. Some examples of current work include using physics-informed neural networks for point kinetics [34] and for non-smooth heterogeneous neutron diffusion problems [35]; surrogate models for transient analysis [36], eigenvalue problems [37] and digital twins [33]; and cross-section generation with neural networks [38]. The approach presented here is fundamentally different, however, providing an alternative way of exactly representing a given discretisation of a system of PDEs, whereas surrogate models provide an approximation of a discretised system of PDEs. Having formulated these discretisations in terms of neural networks, an obvious extension is to combine both untrained networks (i.e. the networks with pre-determined weights as described here) and trained networks to form more efficient and powerful digital twins, as has previously been observed [20, 22, 39].

The sections of this paper are organised as follows. Section 2 describes how a convolutional neural network can be uesd to express a finite volume discretisation, and how a neural network can be used to formulate Jacobi iterations and multigrid methods. Section 3 presents the three numerical examples using the neural network solver to resolve reactor physics eigenvalue problems, and comparisons are drawn against a standard finite volume method. Finally, Section 4 completes the paper with a conclusion of its findings.

2. Methodology

The first part of this section introduces the governing equations, their discretisation with the finite volume or control volume method and the Jacobi method for solving the resulting system. We then explain how the discretisation can be formulated using convolutional layers of a neural network with pre-defined weights. To solve the resulting system, we embed a Jacobi method within a multigrid method. Both Jacobi and multigrid methods are implemented within the neural network, and the latter is based on the U-Net architecture. Finally, an overview of the solution process is given for the case of multiple energy groups, including how the eigenvalue is determined.

4

2.1. Diffusion Equation

The multi-group steady-state diffusion equation for criticality can be written as:

$$-\nabla \cdot (D_g \nabla \phi_g) + \Sigma_g^a \phi_g + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma_{g \to g'}^s \phi_g = \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma_{g' \to g}^s \phi_{g'} + \lambda \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \Sigma_{g'}^f \phi_{g'},$$

$$\forall g \in \{1, 2, \dots, N_g\}, \tag{1}$$

where ϕ_g is the scalar flux of the neutron population, Σ_g^a represents the absorption cross-section, Σ_g^f represents the fission cross-section, ν_g is the average number of neutrons produced per fission event, Σ_g^s represents the scatter cross-section, χ_g is the proportion of neutrons produced for each energy group per fission event and N_g is the number of energy groups used. The subscript g denotes the particular energy group. The diffusion coefficient, D_g, is defined as:

$$D_g = \frac{1}{3(\Sigma_g^a + \Sigma_g^s)} . \tag{2}$$

The eigenvalue, λ, is taken to be the reciprocal of k_{eff} (i.e. $\lambda = 1/k_{\text{eff}}$), where:

$$k_{\text{eff}} = \frac{\text{number of neutrons in one generation}}{\text{number of neutrons in the preceding generation}} . \tag{3}$$

Reflective and vacuum or bare surface boundary conditions can be implemented as follows:

$$D_g \left(\boldsymbol{n} \cdot \nabla \phi_g \right) = 0 \qquad\qquad \textit{(reflective)} \tag{4}$$

$$-D_g \left(\boldsymbol{n} \cdot \nabla \phi_g \right) = \frac{1}{2} \phi_g \qquad\qquad \textit{(vacuum or bare surface)} \tag{5}$$

where $\boldsymbol{n}$ is the outward-pointing normal to the boundary.

2.2. Discretisation

The diffusion equation in 2D can be discretised with finite volumes on a regular mesh of $N_x \times N_y$ cells as follows:

$$-\frac{(D_{i-1,j,g} + D_{i,j,g})}{2\Delta x^2}\phi_{i-1,j,g} - \frac{(D_{i,j,g} + D_{i+1,j,g})}{2\Delta x^2}\phi_{i+1,j,g} - \frac{(D_{i,j-1,g} + D_{i,j,g})}{2\Delta y^2}\phi_{i,j-1,g}$$

$$-\frac{(D_{i,j,g} + D_{i,j+1,g})}{2\Delta y^2}\phi_{i,j+1,g} + \left(\frac{(D_{i-1,j,g} + 2D_{i,j,g} + D_{i+1,j,g})}{2\Delta x^2} + \frac{(D_{i,j-1,g} + 2D_{i,j,g} + D_{i,j+1,g})}{2\Delta y^2}\right)\phi_{i,j,g}$$

$$+\Sigma^a_{i,j,g}\phi_{i,j,g} + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma^s_{i,j,g \to i,j,g'}\phi_{i,j,g} = \sum_{g'=1}^{N_g} \Sigma^s_{i,j,g' \to i,j,g}\phi_{i,j,g'} + \lambda\chi_g \sum_{g'=1}^{N_g} \nu_{g'}\Sigma^f_{i,j,g'}\phi_{i,j,g'}, \tag{6}$$

$$\forall i \in \{2,3,\dots,N_x - 1\}, \qquad \forall j \in \{2,3,\dots,N_y - 1\}, \qquad \forall g \in \{1,2,\dots,N_g\},$$

where Δx and Δy are the uniform cell widths in the x and y directions respectively, N_x and N_y are the numbers of cells in the x and y directions respectively, the subscripts i and j refer to the cells in the x and y directions respectively and $\phi_{i,j,g}$ represents the scalar flux of energy group g in cell i, j. This discretisation is equivalent to a finite difference discretisation. Boundary conditions are applied to the first and last cells in both the x and y directions, so Equation (6) is not solved for these cells. We want to apply the boundary conditions in such a way as to avoid changing the discretisation stencil near the boundaries to maximise the efficiency of the implementation. With this in mind, reflective boundary conditions for the left edge ($i = 1$) can be enforced by the following constraints:

$$\phi_{1,j,g} = 0, \quad D_{1,j,g} = -D_{2,j,g} \quad \forall j \tag{7}$$

and for the right edge ($i = N_x$):

$$\phi_{N_x,j,g} = 0, \quad D_{N_x,j,g} = -D_{N_x-1,j,g} \quad \forall j. \tag{8}$$

Similar constraints can be applied to the top and bottom edges as required. This way of implementing the boundary conditions ensures that there is an average diffusivity of zero at the interface between boundary cells and their neightbours, and this avoids any diffusion occurring across the interface. For bare surface boundary conditions (see Equation (5)), where the normal to the boundary is aligned with the x-direction, the absorption term is modified as follows:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta x}. \tag{9}$$

6

For bare surface boundary conditions where the boundary is aligned with the y direction:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta y}.$$ (10)

For cells that have both boundary conditions the following modification is made:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta x} + \frac{1}{2\Delta y}.$$ (11)

For the 5 point stencil associated with Equation (6), the boundary conditions are implemented through one layer of 'halo cells'. For higher order discretisations, with larger stencils, more layers of cells will be required to serve as halo cells.

Equation (6) and its associated boundary conditions are often written as:

$$\boldsymbol{A}\phi = \lambda \boldsymbol{B}\phi.$$ (12)

where the matrix $\boldsymbol{A}$ contains the absorption, diffusion and scattering terms; matrix $\boldsymbol{B}$ represents the fission terms; and the vector ϕ contains the values of the scalar flux for each cell in every energy group. In the following, we instead keep with the notation used thus far, which stores the unknown scalar flux of each energy group in a 2D array. Although this way of formulating the problem may be less familiar, the motivation will become clear in the following section, when we compare discretisation stencils to convolutional operators. Bearing this in mind, we rewrite the system in Equation (6) as

$$\sum_{u=-l}^{l}\sum_{v=-l}^{l} a^{u,v}_{i,j,g}\, \phi_{i+u,j+v,g} = s_{i,j,g}, \quad \forall i \in \{2,3,\ldots,N_x-1\},\ \forall j \in \{2,3,\ldots,N_y-1\},\ \forall g \in \{1,2,\ldots,N_g\},$$ (13)

where

$$
a_{i,j,g}^{u,v} = \begin{cases}
-\dfrac{(D_{i,j,g} + D_{i+u,j+v,g})}{2\Delta x^2} & \text{for } |u| = 1,\, v = 0 \\[2ex]
-\dfrac{(D_{i,j,g} + D_{i+u,j+v,g})}{2\Delta y^2} & \text{for } u = 0,\, |v| = 1 \\[2ex]
\dfrac{D_{i-1,j,g} + 2D_{i,j,g} + D_{i+1,j,g}}{2\Delta x^2} + \dfrac{D_{i,j-1,g} + 2D_{i,j,g} + D_{i,j+1,g}}{2\Delta y^2} + \Sigma_{i,j,g}^{as} & \text{for } u = 0 = v \\[2ex]
0 & \text{for } |u| = 1 = |v|
\end{cases}
\tag{14}
$$

$$
\Sigma_{i,j,g}^{as} = \Sigma_{i,j,g}^{a} + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma_{i,j,g \to i,j,g'}^{s}
\tag{15}
$$

$$
s_{i,j,g} = \sum_{g'=1}^{g-1} \Sigma_{i,j,g' \to i,j,g}^{s} \phi_{i,j,g'}^{(k+1)} + \sum_{g'=g}^{N_g} \Sigma_{i,j,g' \to i,j,g}^{s} \phi_{i,j,g'}^{(k)} + \lambda \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \Sigma_{i,j,g'}^{f} \phi_{i,j,g'}^{(k)} \, .
\tag{16}
$$

As the stencil used in Equation (6) is a 5 point stencil (which can be written equivalently as a 3 by 3 stencil), the value of l in Equation (13) is 1. The right-hand side of Equation (13) can be determined by using a "best guess" for $\phi_{i,j,g}$. This effectively linearises Equation (13) which can now be solved by the Jacobi method:

$$
\phi_{i,j,g}^{(k+1)} = \frac{1}{a_{i,j,g}^{0,0}} \left(s_{i,j,g} - \sum_{u=-l}^{l} \sum_{v=-l}^{l} a_{i,j,g}^{u,v} \phi_{i+u,j+v,g}^{(k)} + a_{i,j,g}^{0,0} \phi_{i,j,g}^{(k)} \right),
\tag{17}
$$

where $2l + 1$ is the width of the stencil, k is the Jacobi iteration and $\{\{a_{i,j,g}^{u,v}\}_{u=-l}^{l}\}_{v=-l}^{l}$ represents the coefficients of the stencil used to calculate the scalar flux in cell $i,\, j$. The Jacobi method can be used for diagonally dominant systems, and given an initial guess, is solved for each diagonal component in turn. Iteration continues until the system converges [40]. The diagonal terms of the usual matrix-vector form of Equation (17) (seen in Equation (12)) are now denoted by $a_{i,j,g}^{0,0}$ (for cell $i,\, j$ and energy group g), the remaining terms are the non-diagonal terms ($a_{i,j,g}^{u,v}$ $\forall u,\, v$ such that $|u| = 1$ and $v = 0$, and $u = 0$ and $|v| = 1$), which are subtracted from the source term in Equation (17).

2.3. Implementing discretisations with convolutional neural networks

A convolutional layer of a neural network has a filter or kernel associated with it, which is a small grid (typically of dimension 3×3, 5×5 or 7×7 for 2D filters) whose cells have values known as weights associated with them. The filter is applied to part of the input by multiplying the input value by the weight in the overlapping cells. The products are summed to produce the output. This is illustrated in Figure 1, where a filter acting on one part of the input data can be seen. This process of passing the filter over parts of the input data is repeated until the filter has passed over all the input data and all the
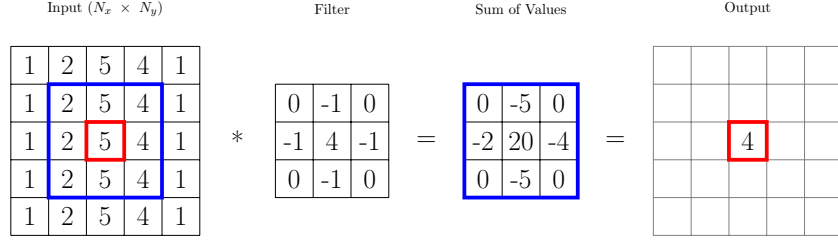
Figure 1: A 3 by 3 convolutional filter which applies the discretised diffusion operator (a five-point finite volume stencil) in 2D to 9 cells. The filter is first applied to all the cells in the blue block on the left; the result of which can be seen in the blue block following the equals sign. The 9 values are then summed to give the value in the red block on the right which is the output value, which represents the value of the diffusion operator acting on the input approximated at the central cell.

output values are known. The action of a 2D convolutional layer on a 2D input can be written as follows

$$x_{i,j}^{(k+1)} = \sum_{u=-l}^{l} \sum_{v=-l}^{l} w^{u,v} x_{i+u,j+v}^{(k)}, \tag{18}$$

where the input and output are 2D grids with components $x_{i,j}^{(k)}$ and $x_{i,j}^{(k+1)}$ respectively. The weights of the filter are represented by $w^{u,v}$ and the size of the filter is $(2l+1) \times (2l+1)$. The example in Figure 1 corresponds to applying a discretised diffusion operator to an input for the case $\Delta x = 1 = \Delta y$ and a constant diffusivity ($\boldsymbol{D}_g$) of 1. For general grid sizes, using the notation in Equation (18) and for a particular set of weights $\boldsymbol{w}$, the discretised diffusion operator applied to the scalar flux of energy group g and cell i, j can be written as

$$-\nabla^2 \boldsymbol{\Phi}_g \Big|_{i,j} = \sum_{u=-l}^{l} \sum_{v=-l}^{l} w^{u,v} \phi_{i+u,j+v,g} = -\left(\frac{\phi_{i-1,j,g} - 2\phi_{i,j,g} + \phi_{i+1,j,g}}{\Delta x^2} + \frac{\phi_{i,j-1,g} - 2\phi_{i,j,g} + \phi_{i,j+1,g}}{\Delta y^2} \right), \tag{19}$$

which is equivalent to

$$\sum_{u=-l}^{l} \sum_{v=-l}^{l} w^{u,v} \phi_{i+u,j+v,g} = \sum_{\text{entries}} \begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix} \odot \begin{bmatrix} \phi_{i-1,j+1,g} & \phi_{i,j+1,g} & \phi_{i+1,j+1,g} \\ \phi_{i-1,j,g} & \phi_{i,j,g} & \phi_{i+1,j,g} \\ \phi_{i-1,j-1,g} & \phi_{i,j-1,g} & \phi_{i+1,j-1,g} \end{bmatrix} =: \boldsymbol{f}(\boldsymbol{\Phi}_g; \boldsymbol{w}) \Big|_{i,j}. \tag{20}$$

where $l = 1$, $\odot$ denotes the Hadamard product which performs entrywise multiplication, the symbol $\sum_{\text{entries}}$ denotes the summation of all the entries of a matrix (see Equation (B.2)) and $\boldsymbol{f}$ represents the discrete convolution applied to the field $\boldsymbol{\Phi}_g$ by a 3×3 filter with weights $\boldsymbol{w}$. The components inside the square brackets used in Equation (20) are ordered as if they are pixels in an image rather than components of a matrix. Equation (19) shows one way of writing a finite volume discretisation of the diffusion operator acting on a field $\boldsymbol{\Phi}_g$ and Equation (20) is exactly the same discretisation written as a convolution (using the Hadamard product). This illustrates how a discretisation scheme can be represented by a convolutional

9

neural network.

By comparing Equations (13) and (18), we can see that the diffusion equation (with a spatially varying $\boldsymbol{D}_g$) cannot yet be written as a convolution layer, as the weights in Equation (13) vary in space (for two cells i, j and i^*, j^*, $a_{i,j}^{u,v} \neq a_{i^*,j^*}^{u,v}$), whereas in Equation (18), the weights do not depend on which part of the input data they are applied to (i.e. $w^{u,v}$ is independent of i, j). By recalling that the diffusion operator in Equation (1) can be written as three terms all of which involve the Laplace operator (see Appendix A), we can therefore write the diffusion operator as three convolutions:

$$-\nabla \cdot (D_g \nabla \phi_g) = \frac{1}{2} \left(-\nabla^2 (D_g \phi_g) - D_g \nabla^2 \phi_g + \phi_g \nabla^2 D_g \right) \qquad \text{analytical form} \qquad (21)$$

$$\boldsymbol{f}^{\text{Diff}}(\boldsymbol{\Phi}_g, \boldsymbol{D}_g; \boldsymbol{w}) = \frac{1}{2} \left(\boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g; \boldsymbol{w}) + \boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g; \boldsymbol{w}) - \boldsymbol{\Phi}_g \odot \boldsymbol{f}(\boldsymbol{D}_g; \boldsymbol{w}) \right) \qquad \text{discretised form} \qquad (22)$$

where $\boldsymbol{\Phi}_g$ is a $N_x \times N_y$ matrix containing all $\phi_{i,j,g}$ components, $\boldsymbol{D}_g$ is a $N_x \times N_y$ matrix containing all $D_{i,j,g}$ components and $\boldsymbol{f}$ represents the application of the convolutional layer with weights $\boldsymbol{w}$. Equation (22) serves as a definition of the diffusion convolution $\boldsymbol{f}^{\text{Diff}}$ and the weights $\boldsymbol{w}$. The equivalence between this formulation of the finite volume discretisation of the diffusion operator and the standard formulation presented in Equation (6) can be seen in Appendix B. The discretised diffusion equation can now be written for energy group g as:

$$\boldsymbol{f}^{\text{Diff}}(\boldsymbol{\Phi}_g, \boldsymbol{D}_g; \boldsymbol{w}) + \left(\boldsymbol{\Sigma}_g^a + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \boldsymbol{\Sigma}_{g \to g'}^s \right) \odot \boldsymbol{\Phi}_g = \boldsymbol{s}_g, \quad \forall g \in \{1, 2, \dots, N_g\}, \qquad (23)$$

in which the source $\boldsymbol{s}_g$ for energy group g also contains coupling terms between the energy groups other than g. The terms $\boldsymbol{\Sigma}_g^a$ and $\boldsymbol{\Sigma}_{g \to g'}^s$ represent matrices which contain the absorbtion and scatter cross-sections for each cell. Equation (23) can be solved with the Jacobi method as before. However, when implementing this, instead of using Equation (23), we rewrite this to use one fewer convolutional operation for efficiency. The term $\sum_{u=-l}^{l} \sum_{v=-l}^{l} a_{i,j,g}^{u,v} \phi_{i+u,j+v,g}^{(k)} - a_{i,j,g}^{0,0} \phi_{i,j,g}^{(k)}$ can be determined using a convolutional filter containing just the off-diagonal terms:

$$\sum_{u=-l}^{l} \sum_{v=-l}^{l} a_{i,j,g}^{u,v} \phi_{i+u,j+v,g}^{(k)} - a_{i,j,g}^{0,0} \phi_{i,j,g}^{(k)} \equiv \frac{1}{2} \left(\boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w}_{\text{od}}) \big|_{i,j,g} + \boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w}_{\text{od}}) \right) \Big|_{i,j,g}, \qquad (24)$$

where $\boldsymbol{f}$ is a convolutional layer with weights $\boldsymbol{w}_{\mathrm{od}}$:

$$
\boldsymbol{w}_{\mathrm{od}} = \begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & 0 & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix}.
\tag{25}
$$

with the central term of $\boldsymbol{w}$ set to zero in order to obtain this. The Jacobi method as written in Equation (17) is therefore equivalent to:

$$
\boldsymbol{\Phi}_g^{(k+1)} = (\boldsymbol{A}_g^{0,0})^{\odot-1} \odot \left(\boldsymbol{s}_g - \frac{1}{2} \left(\boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w}_{\mathrm{od}}) + \boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w}_{\mathrm{od}}) \right) \right),
\tag{26}
$$

where $(\boldsymbol{A}_g^{0,0})^{\odot-1}$ is the Hadamard inverse [41] which is an $N_x \times N_y$ array whose $i^{\mathrm{th}}, j^{\mathrm{th}}$ component is $\frac{1}{a_{i,j,g}^{0,0}}$ for energy group g. This equation can be written as a function J,

$$
\boldsymbol{\Phi}_g^{(k+1)} = \mathrm{J}\left(\boldsymbol{\Phi}_g^{(k)}, (\boldsymbol{A}_g^{0,0})^{\odot-1}, \boldsymbol{s}_g, \boldsymbol{D}_g \right),
\tag{27}
$$

which calculates the updated solution after one Jacobi iteration. Figure 2 shows the architecture of this function, i.e., the neural network that solves one Jacobi iteration of the neutron transport problem as discretised in Equation (26). Green boxes contain the inputs; blue boxes are convolutional layers; orange boxes are mathematical functions as layers; and the grey box is the output of the network. The second line in each box gives the dimension of the output of that box.

So far, we have described how to find the weights for the filters of convolutional layers that correspond to a finite volume discretisation of the neutron diffusion equation, solved with a Jacobi method. The approach described in this paper is not limited to the finite volume method, however, and for comparison, we also use a discretisation based on a new convolutional finite element method (ConvFEM) [23] of the diffusion operator. Using quadratic 9-noded rectangular elements, the 5×5 filter for this discretisation is given by

$$
\boldsymbol{w} = \frac{1}{900} \begin{bmatrix} -5 & 50 & -15 & 50 & -5 \\ 50 & -320 & -660 & -320 & 50 \\ -15 & -660 & 3600 & -660 & -15 \\ 50 & -320 & -660 & -320 & 50 \\ -5 & 50 & -15 & 50 & -5 \end{bmatrix}
\tag{28}
$$

$$
D_{\mathrm{left}} = D_{\mathrm{right}} = D_{\mathrm{top}} = D_{\mathrm{bottom}},
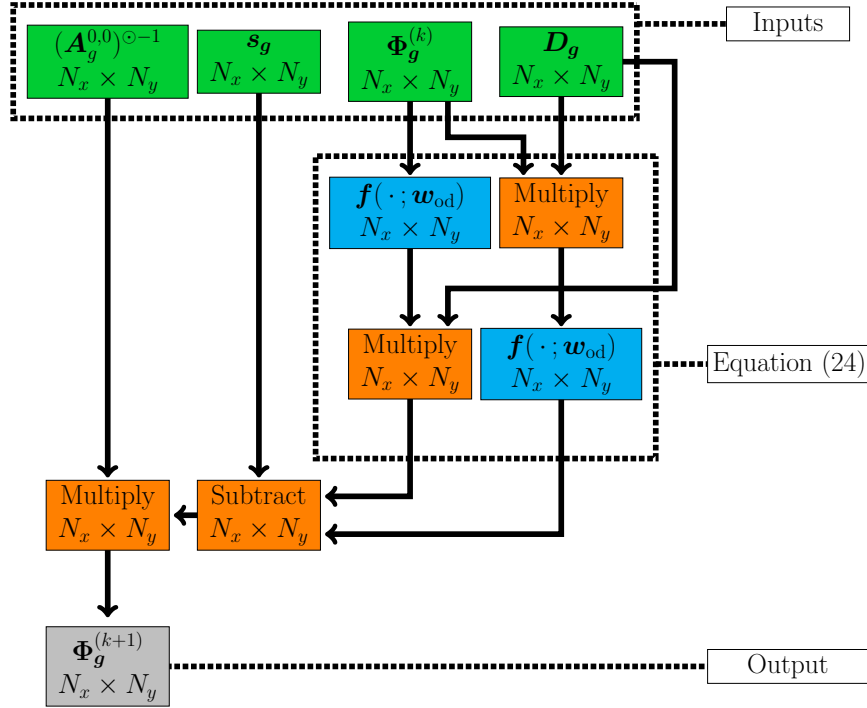\tag{29}
$$

11

Figure 2: Schematic of the neural network used for a single Jacobi iteration written as J($\cdot$) in Equation (26). This network performs a single Jacobi iteration on the flux of a single energy group. The inputs are the flux ($\mathbf{\Phi}_g^{(k)}$), representative source ($\mathbf{s}_g$), diffusion coefficients ($\mathbf{D}_g$) and the strictly diagonal coefficients ($\mathbf{A}_g^{\mathbf{0},\mathbf{0}})^{\odot-1}$) (green boxes). A number of layer operations are performed, mathematical operations are shown in orange and convolutional operations are in cyan. The output is the flux of the next Jacobi iteration ($\mathbf{\Phi}_g^{(k+1)}$). Arrows originate from which layer the data originated and the end of an arrow indicates which layer takes that data as input. The dimensions of the layers are given on the second line of each box.

and for the left side:

$$D_{\text{left}} = D_{i,j,g} \quad \forall i \in \{3,4\} \quad \forall j \in \{3,4,\ldots,N_y-2\}, \tag{30}$$

with corresponding constraints for the right, top and bottom sides. If Equation (29) holds and Equation (30) holds for all sides then the boundary conditions for the left side, $i = 1$, can be implemented with:

$$\phi_{1,j,g} = 0, \quad \phi_{2,j,g} = 0, \quad D_{\text{left}} = -D_{1,j,g} = -D_{2,j,g}, \tag{31}$$

and similar conditions for the other sides. Equation (30) may not hold if two cells next to the boundary do not have the same value and thus this approach might not be used in this situation or one may use some sort of average. An alternative that works for all filter sizes is simply to set the values of the diffusion coefficient and the fluxes to be zero in the halo regions and then no addition to the absorption cross sections for the boundary condition are required. This effectively implements the $2\Delta x$ extrapolation boundary condition obtained using Equations (9), (10) and (11).
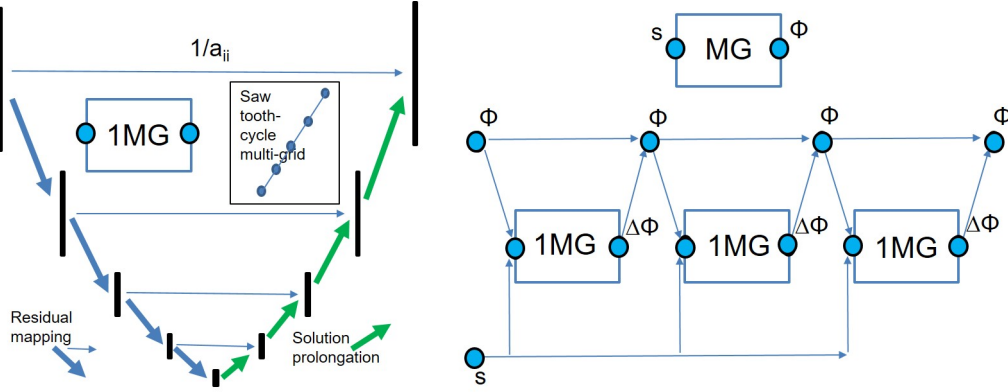
Figure 3: A schematic diagram showing the U-Net architecture (left) that is used to form a single multigrid sawtooth cycle. On the right, we can see how multiple cycles are brought together to form the overall solution method.

In Figure 3 we show how the U-Net [24] architecture has been repurposed to form a sawtooth multigrid method. Figure 4 shows a single multigrid iteration, using the U-Net, with two restrictions. Note that
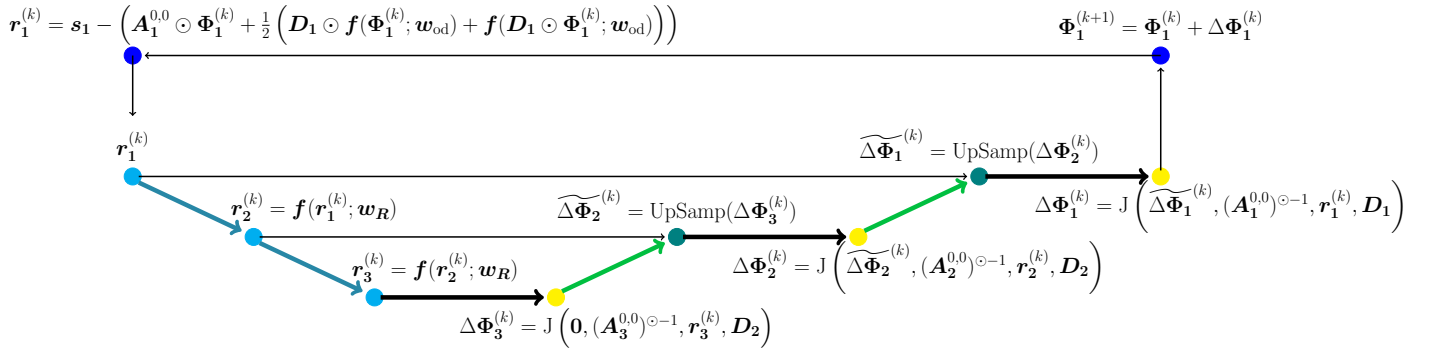

Figure 4: Multigrid iteration, with the subscript indicating the resolution and superscript representing the multigrid iteration and $J(\cdot)$ representing a Jacobi iteration. The residual is calculated and restricted twice, indicated by the cyan nodes. These residuals are used with Jacobi smoothing, indicated by yellow nodes. After smoothing, prolongation is performed using UpSampling layers, indicated by teal nodes. After the finest level is reached, the flux is updated and the process repeats.

the subscript indicating energy group is no longer shown, the bold subscript now indicates the coarseness of the mesh, with 1 being the finest mesh. The residual (r^k) is calculated using:

$$r_1^{(k)} = s_1 - \left(A_1^{0,0} \odot \Phi_1^{(k)} + \frac{1}{2}\left(D_1 \odot f(\Phi_1^{(k)}; w_{\mathrm{od}}) + f(D_1 \odot \Phi_1^{(k)}; w_{\mathrm{od}}) \right) \right) \tag{32}$$

which is then restricted twice to ($r_2^{(k)}$) and ($r_3^{(k)}$). A Jacobi iteration is performed on the coarsest level (bold subscript 3) to determine $\Delta\Phi_3^{(k)}$, starting with an array of zeros. This is prolongated to estimate $\widetilde{\Delta\Phi_2}^{(k)}$ which is then smoothed to $\Delta\Phi_2^{(k)}$ with another Jacobi iteration using the residual of the next highest level. This repeats until the finest level is reached (bold subscript 1), where the flux is updated ($k+1$) and the process is repeated for a number of multigrid iterations. The restriction may be performed

13

with the convolution:

$$r_2^{(k)} = f(r_1^{(k)}; w_R), \tag{33}$$

with filter weights:

$$w_R = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}. \tag{34}$$

Upsampling layers can perform the role of prolongating the solution to a higher level. The upsampling operation simply copies the value from the coarser cell to the associated cells on the finer grid, which increases the dimensions of the data [42] and results in an approximation for the data on a finer mesh:

$$\widetilde{\boldsymbol{\Phi}_1}^{(k)} = \mathrm{UpSamp}(\boldsymbol{\Phi}_2^{(k)}). \tag{35}$$

Figure 5: Multigrid network, MG$(\cdot)$, representing a single multigrid iteration. This network performs a single multigrid iteration on the flux of a single energy group. Takes the flux $(\boldsymbol{\Phi}_1^{(k)})$, representative source $(\boldsymbol{s_1})$, diffusion coefficients $(\boldsymbol{D_1})$ and the strictly diagonal coefficients $(\boldsymbol{A_1^{0,0}})$, along with the coarser resolution coefficients, as inputs (green boxes). A number of layer operations are performed, mathematical operations in orange, convolutional passes in cyan, sub-model operations in yellow and upsampling in teal. The sub-models can be iterated on multiple times. Finally it outputs the flux of the next multigrid iteration flux $(\boldsymbol{\Phi}_1^{(k+1)})$. Arrow origins show which layer the data originated and the end of the arrow shows which layer takes that data as input. Dimensions of layers are given on the second line of each box.

Figure 5 shows how the multigrid method can be represented by a neural network. Green boxes contain

the inputs, blue boxes are convolutional layers, orange boxes are mathematical functions as layers, yellow boxes are sub-networks, teal boxes are upsampling layers and the grey box is the output of the network. The second line in each box is the dimension of the output. This can be written as:

$$\mathbf{\Phi}_1^{(k+1)} = \mathrm{MG}\left(\mathbf{\Phi}_1^{(k)}, \mathbf{s}_1, (\mathbf{A}_1^{0,0}), (\mathbf{A}_1^{0,0})^{\odot-1}, (\mathbf{A}_2^{0,0})^{\odot-1}, (\mathbf{A}_3^{0,0})^{\odot-1}, \mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3\right), \tag{36}$$

and for a single energy group g:

$$\mathbf{\Phi}_{1g}^{(k+1)} = \mathrm{MG}_g\left(\mathbf{\Phi}_{1g}^{(k)}, \mathbf{s}_{1g}, (\mathbf{A}_{1g}^{0,0}), (\mathbf{A}_{1g}^{0,0})^{\odot-1}, (\mathbf{A}_{2g}^{0,0})^{\odot-1}, (\mathbf{A}_{3g}^{0,0})^{\odot-1}, \mathbf{D}_{1g}, \mathbf{D}_{2g}, \mathbf{D}_{3g}\right), \tag{37}$$

where $\mathrm{MG}(\cdot)$ is a function that calculates the result of one sawtooth multigrid iteration and many of these iterations are strung together to form the final solution, see Figure 3. $\mathrm{MG}_g(\cdot)$ is the multigrid iteration applied to energy group g, indicated by the subscript.

It should be noted that the diffusion coefficients and other material properties are mapped to a coarser grid using a harmonic average before the discretisations are formed on the coarser grids. The same discretisation is used at each multigrid level but with different cell sizes.

2.5. Multi-group network

The multigrid function and network, given by Equation (36) and Figure 5 respectively, show how a single multigrid iteration may be applied to a single energy group g. Multi-group problems must have balanced scattering terms, achieved through iterating until the terms balance. Equation (16) shows how a block Gauss-Seidel approach is used when constructing $s_{i,j,g}$. The scattering term, Σ^s, is constructed using the most recent flux information, achieved by resolving each energy group sequentially.

Figure 6 shows how energy groups can be resolved using a block Gauss-Seidel approach within a neural network. Green boxes represent inputs, yellow boxes represent sub-networks and grey boxes represent outputs. For clarity, the green outputs are only shown as being linked to the first energy group but would be linked to all subsequent energy groups. $\mathbf{s}_g$ is a vector containing Σ^s, $\mathbf{s}_{\mathrm{fiss}}$ and $\mathbf{\Phi}^{(k)}$ and is formed using Equation (16). $\mathbf{s}_g$ is then used in the MG sub-model to resolve for $\mathbf{\Phi}_g$, repeating for a number of multigrid iterations until:

$$\mathbf{\Phi}_g^{(k+1)} \approx \mathbf{\Phi}_g^{(k)}. \tag{38}$$

$\mathbf{\Phi}_g$ is then used in $\mathbf{s}_{g'}$ where $g' > g$. Once all energy groups have been resolved they can be concatenated to form $\mathbf{\Phi}^{(k+1)}$. This is repeated until:

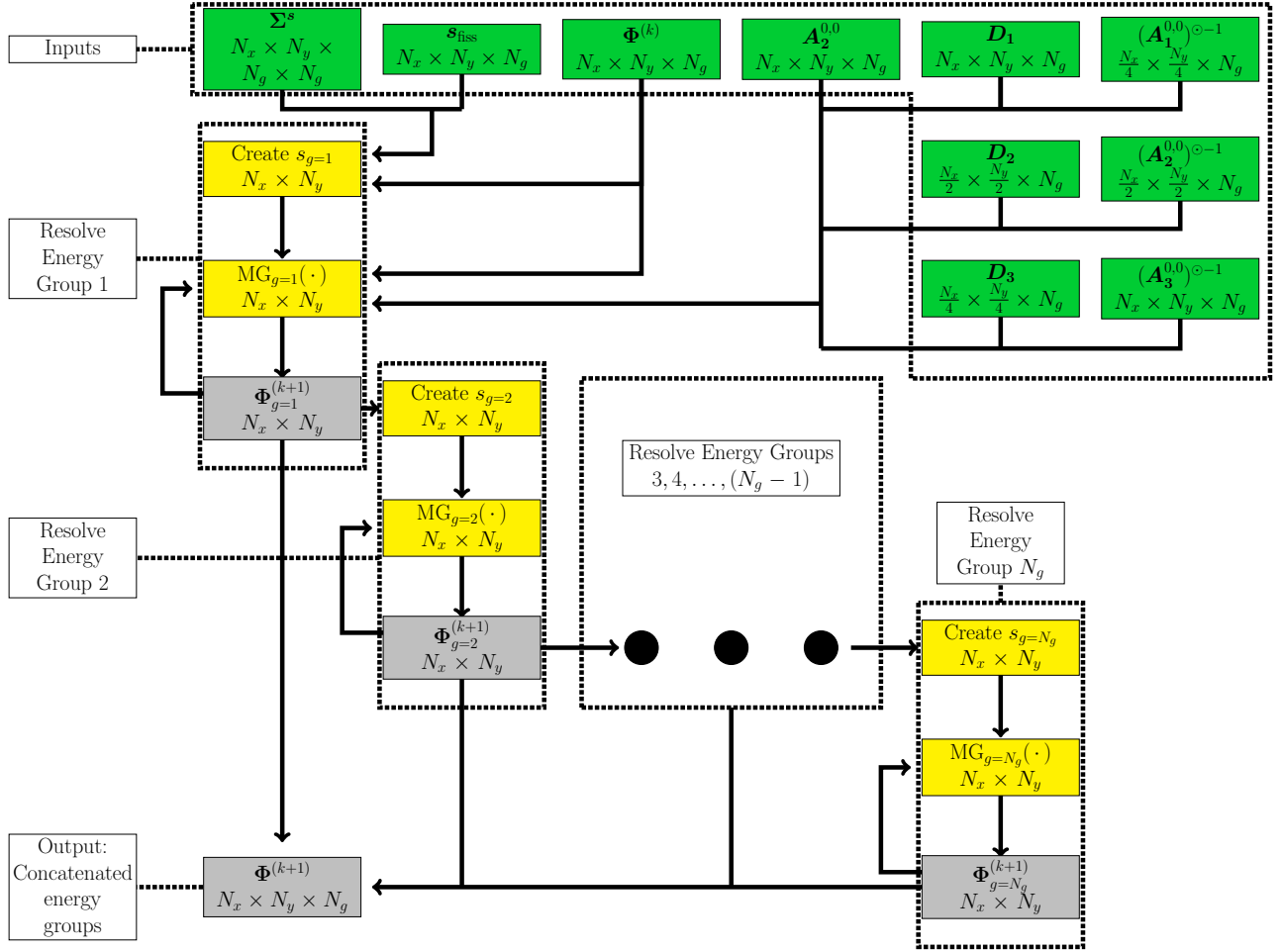$$\mathbf{\Phi}^{(k+1)} \approx \mathbf{\Phi}^{(k)}. \tag{39}$$

Figure 6: Multi-group network representing a single multi-group iteration. This network performs a single multi-group iteration on the flux of all energy groups. Takes the flux ($\mathbf{\Phi}_1^{(k)}$), scattering cross-sections Σ_s, source fission term ($\boldsymbol{s}_{\text{fiss}}$), diffusion coefficients ($\boldsymbol{D}_1$) and the strictly diagonal coefficients ($\boldsymbol{A}_1^{0,0}$), along with the coarser resolution coefficients, as inputs (green boxes). Each energy group is updated sequentially, first through updating the source term for a specific energy group and then performing a number of multigrid sub-model iterations. The updated flux for an energy group is then passed onto subsequent energy groups. Left out of the figure for clarity, the inputs (green boxes) are all connected to subsequent sub-models (yellow boxes). Finally, it outputs the flux of the next multi-group iteration flux ($\mathbf{\Phi}_1^{(k+1)}$). Arrow origins show which layer the data originated and the end of the arrow shows which layer takes that data as input. Dimensions of layers are given on the second line of each box.

An alternative to the Gauss-Seidel approach would be to use the Jacobi approach to resolve all energy groups simultaneously, which could be achieved by using the multigrid network alone, as described in Section 2.4. This is performed by passing all N_g energy groups to the MG network at the same time, only updating $\boldsymbol{s}$ outside of this. The source term in Equation (16) instead changes to:

$$s_{i,j,g} = \sum_{g'=1}^{N_g} \Sigma_{i,j,g' \to i,j,g}^s \phi_{i,j,g'}^{(k)} + \lambda \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \Sigma_{i,j,g'}^f \phi_{i,j,g'}^{(k)} . \tag{40}$$

Equation (12) is an eigenvalue problem so λ needs to be determined. An approximation is used (usually

$\lambda = 1$) and the fission term is passed to the multi-group network where

$$s_{\text{fiss},g} = \lambda \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \boldsymbol{\Sigma}^{\boldsymbol{f}}_{g'} \boldsymbol{\Phi}^{(k)}_{g'}, \tag{41}$$

for each energy group g and $\boldsymbol{s}_{\text{fiss}}$ is an array containing all g of $\boldsymbol{s}_{\text{fiss},g}$. The power method [43] is the method chosen here to determine the dominant eigenvalue for this problem. The implementation of the power method used here is the same as [37] and operates outside of the multi-group network.

3. Results

The approach described in this paper is demonstrated on two test cases: a fuel assembly and a reactor core, both based on the KAIST benchmark [44]. For the fuel assembly, two configurations are investigated (control rods fully withdrawn and fully inserted). Results for a finite volume discretisation of the 2D neutron diffusion equation are generated by a neural network with pre-determined weights and compared with a results from a traditional Fortran implementation. A neural network solution of a discretisation based on the quadratic finite element method, ConvFEM [23], is also presented. For the reactor core, the cross-sections are taken from the KAIST benchmark, and a grid of 3×3 fuel assemblies are used to make up one quarter of the core. Results are presented from a finite volume discretisation of the neutron diffusion equation using a neural network. All the neural networks in this section were implemented in python using Keras [42] with the TensorFlow backend [1].

3.1. Fuel Assembly - Geometry and Configuration

The geometry of the UOX fuel assembly based on the KAIST benchmark [44] can be seen in Figure 7. It consists of a 17×17 lattice containing 264 UOX fuels rods with guide tubes in the remaining 25 lattice-cells which can be filled with either moderator or control rods. We consider two configurations of the assembly. In the first configuration, all 25 of these lattice-cells are filled with moderator, representing a system where the control rods are fully withdrawn. In the second configuration, all 25 of the remaining lattice-cells are control rods, representing a system where the control rods are fully inserted. Two computational grids are used, with either 20×20 cells or 10×10 cells within each lattice-cell. The higher resolution grid is used for the fuel assembly test case and the coarser grid is used when modelling the whole reactor (see Section 3.6). For the 20×20 case, there is a total of $115,600$ computational cells in the lattice with $1,364$ of these forming the boundaries (i.e. as halo cells or ghost cells). The energy is discretised into seven groups, meaning that the fuel assembly has $818,720$ degrees of freedom. Each side of the fuel assembly is

of length $21.42\,\mathrm{cm}$ meaning each computational cell measures $0.063\,\mathrm{cm} \times 0.063\,\mathrm{cm}$. Each side of the fuel assembly has vacuum boundary conditions applied to it.
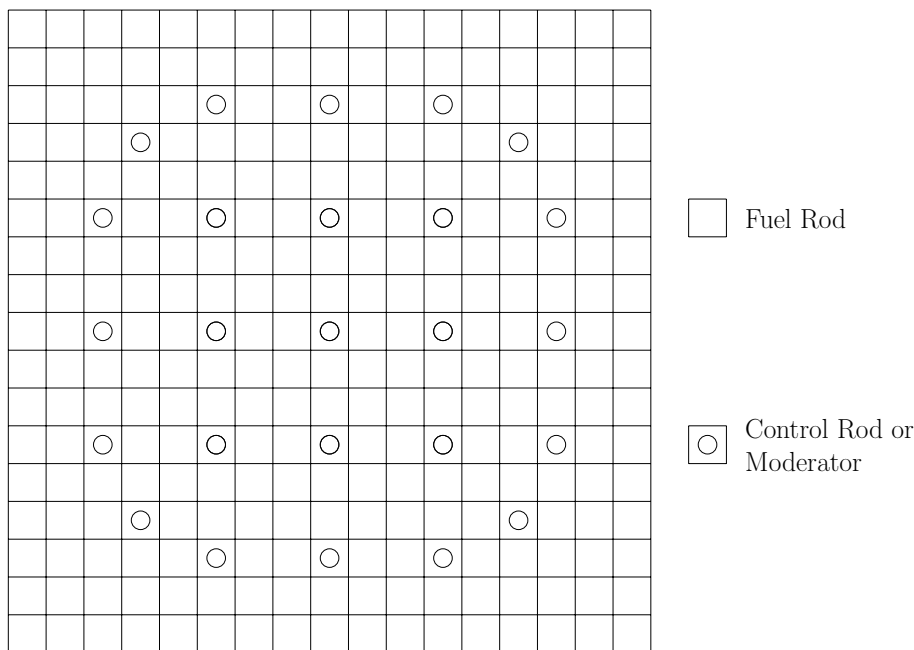

Figure 7: Geometry of UOX fuel assembly with laatice-cells containing fuel rods or guide tubes with either moderator or control rods.
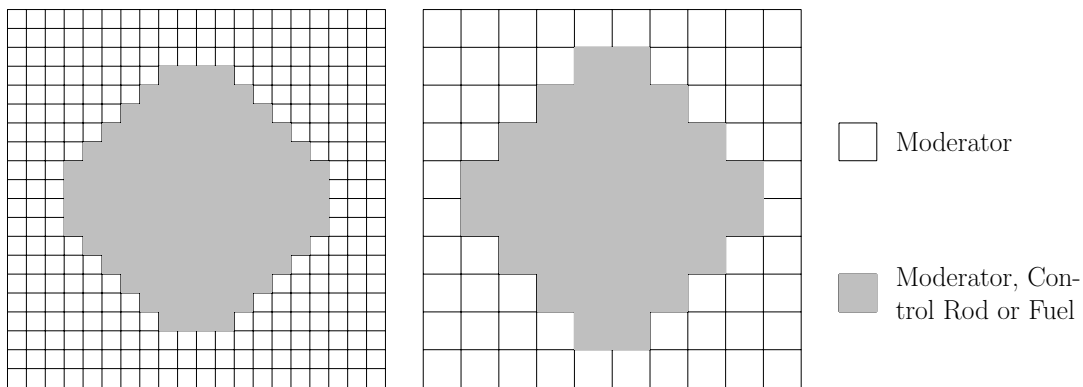

Figure 8: Computational grid shown here for a single lattice-cell for the fine (20×20) grid (used for the assembly calculations in Section 3.2) and the coarse (10×10) grid (used in the reactor core calculations in Section 3.6).

All lattice-cells in the fuel assembly have the same geometry with the moderator occupying the outer region of every lattice-cell and either fuel, a control rod or moderator occupying the inner region. This is shown in Figure 8. The guide tube is not modelled. The material parameters required are UOX cross-sections for the fuel rods, and cross-sections for the control rods and moderator (same as the coolant), all taken from the KAIST benchmark.

3.2. Fuel Assembly - Finite Volume Discretisation

The neutron diffusion equation is solved for a 2D fuel assembly which uses geometry and cross-sections from the KAIST benchmark [44]. We perform two Jacobi iterations, 100 multigrid iterations and 100 multi-group iterations to obtain the solution from the multi-group neural network with weights that are pre-determined by a finite volume discretisation. After the final multi-group iteration, the solution converged to an effective tolerance of 10^{-14}. Solutions obtained from the neural network are compared with solutions from a traditional implementation of the finite volume discretisation in Fortran that uses a Gauss-Seidel iterative method (with a tolerance of 10^{-15}).

Figure 9 contains the flux profiles of three energy groups for a fuel assembly with control rods fully withdrawn. The high values of scalar flux for group 7 indicate the location of the moderator within the guide tubes. It can be observed that the pointwise difference between the neural network solution and the Fortran solution with Gauss-Seidel iteration is small, $\mathcal{O}(10^{-10})$, and within the tolerances set for the solvers.
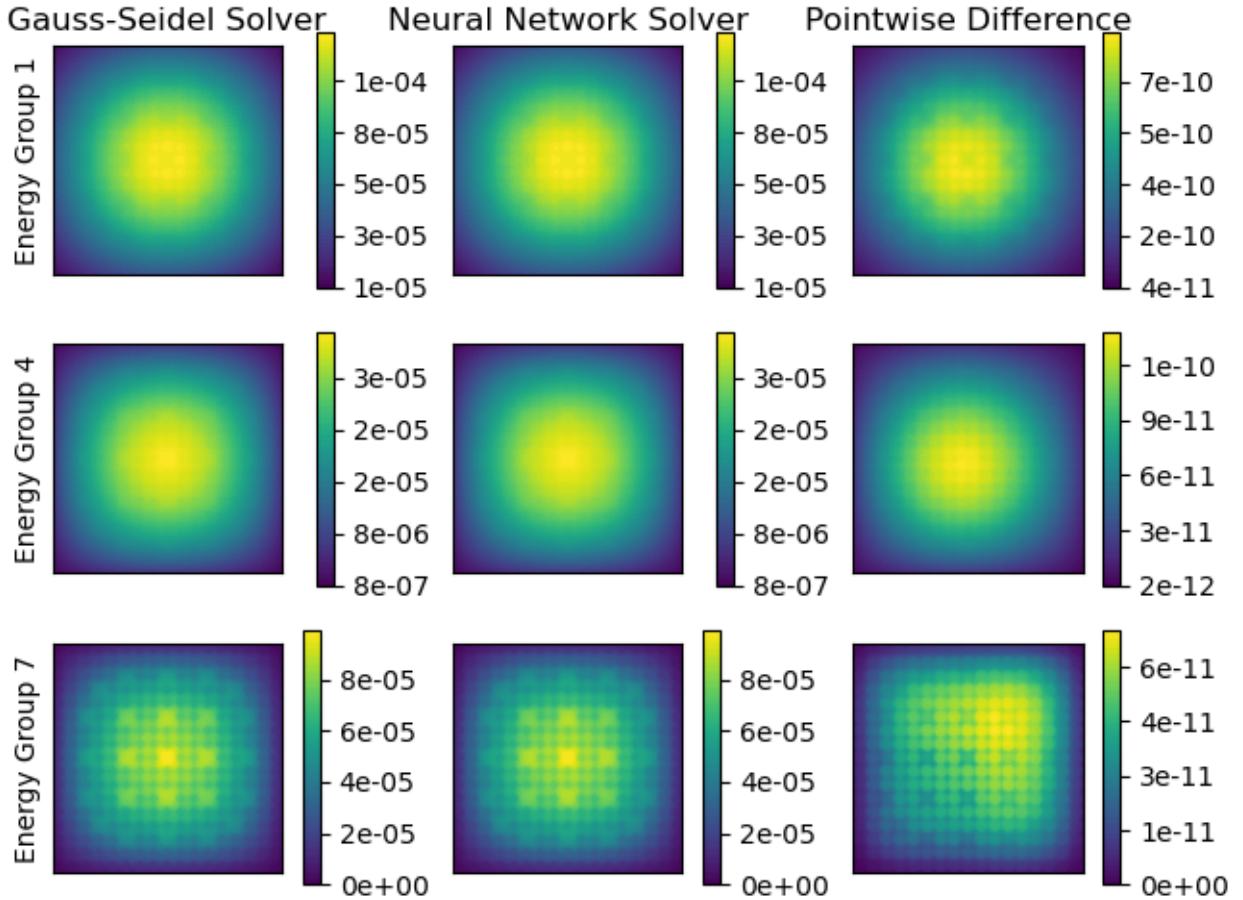

Figure 9: Scalar flux (neutrons cm^{-2} s^{-1}) across the fuel assembly for three energy groups for a fuel assembly with control rods fully withdrawn, generated using the multi-group network.

Figure 10 contains the flux profiles for three energy groups for a fuel assembly with control rods fully inserted. The positions of the control rods can be observed in between the fuel rods, where the flux decreases sharply. It can be observed that the pointwise difference between the neural network solution and the Fortran solution is small, $\mathcal{O}(10^{-10})$, and within the tolerances set for the solvers.
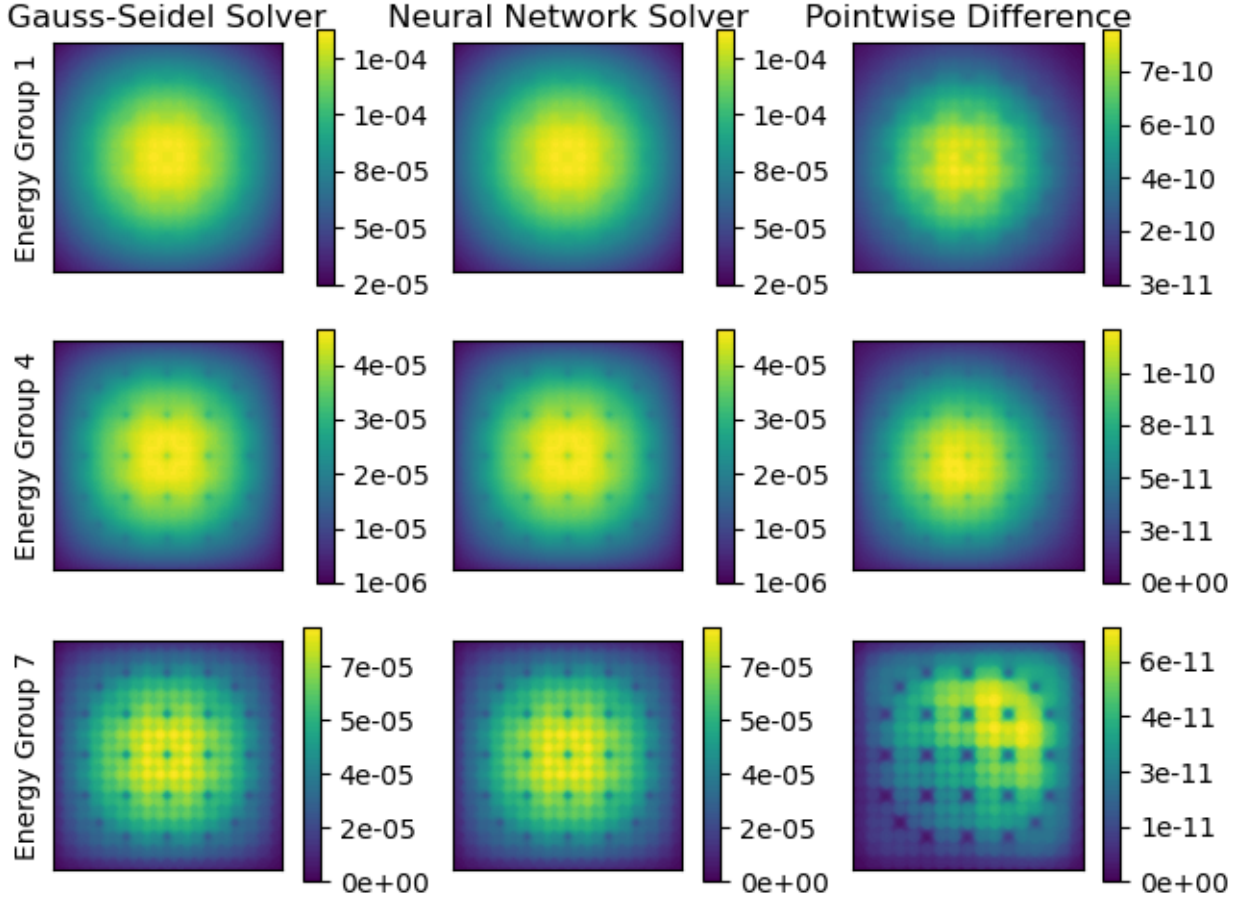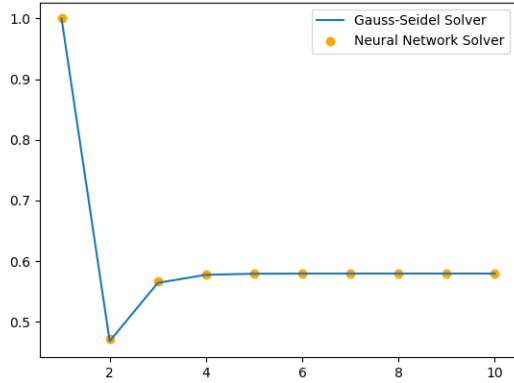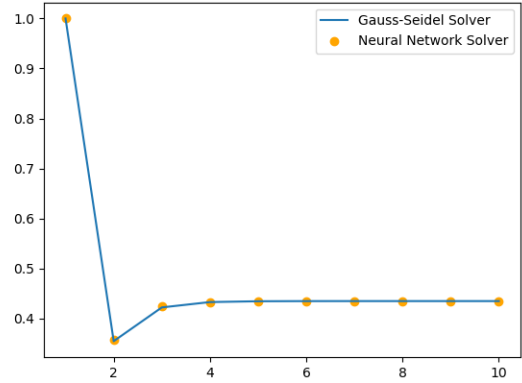

Figure 10: Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for three energy groups for a fuel assembly with control rods fully inserted, generated using the multi-group network.

Figure 11 contains the rate of convergence of k_{eff} for a fuel assembly with control rods fully withdrawn (Figure 11(a)) and fully inserted (Figure 11(b)). It can be observed that k_{eff} is lower when control rods are inserted, as would be expected. The convergence for the solution from the neural network solver and that from the Fortran implementation is identical for both configurations (fully withdrawn and fully inserted control rods). See Table 1 for a comparison of the converged values of k_{eff}.

(a) Control rods fully withdrawn. k_{eff} converging to 0.5797 for the Neural Network Solver and 0.5797 for the Fortran implementation (with the Gauss-Seidel solver).
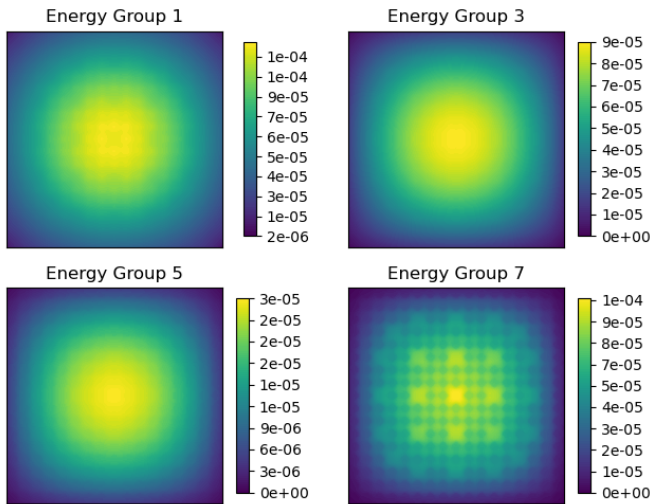

(b) Control rods fully inserted. k_{eff} converging to 0.4347 for the Neural Network Solver and 0.4347 for the Fortran implementation (with the Gauss-Seidel solver).
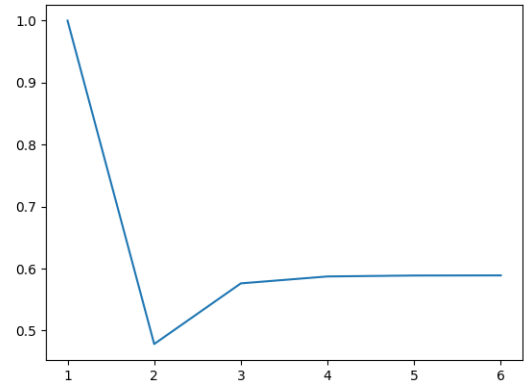
Figure 11: A plot of convergence of k_{eff} against power iteration for the fuel assembly, generated using the multi-group network with the finite volume discretisation.

3.3. Fuel Assembly - Finite Element discretisation

Figures 12 and 13 contain the scalar flux solution for a fuel assembly with control rods fully withdrawn and fully inserted, respectively. Both solutions were generated using quadratic convolutional finite elements (ConvFEM) implemented with a neural network. The weights used in the filters are given in Equation (28). In both cases, the flux profile shows a similar distribution to the solutions generated using the finite volume discretisation in Section 3.2. The converged values of k_{eff} using the quadratic finite elements are both slightly larger than for the finite volume discretisation, see Table 1.
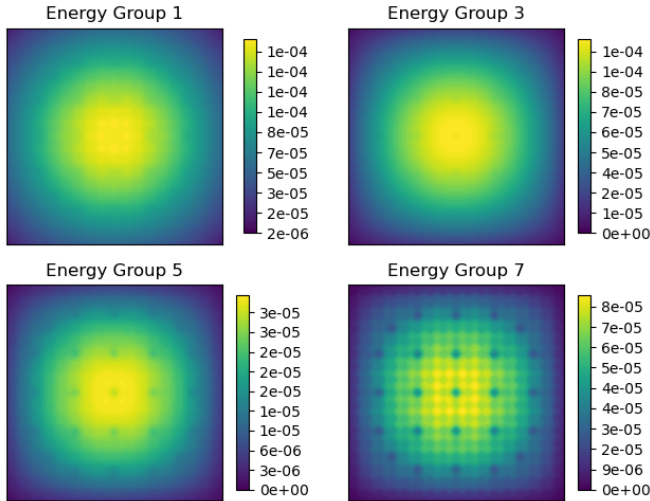

(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the fuel assembly for four energy groups.
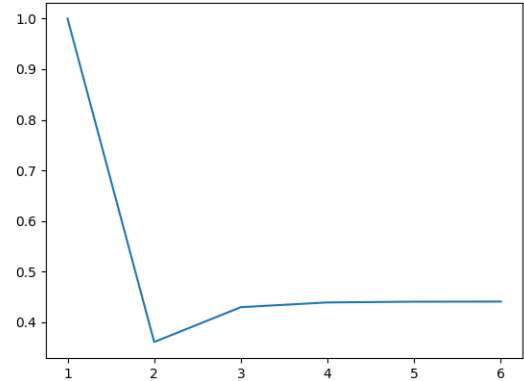

(b) Convergence of k_{eff} against power iteration. k_{eff} converges to 0.5838 compared to 0.5797 for the finite volume discretisation.

Figure 12: Results for a fuel assembly with control rods fully withdrawn, generated using the multi-group network for quadratic convolutional finite elements (using ConvFEM).

(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups.

(b) k_{eff} vs power iteration. k_{eff} converges to 0.4370 compared to 0.4347 for the finite volume discretisation.

Figure 13: Results for a fuel assembly with control rods fully inserted, generated using the multi-group network for quadratic convolutional finite elements (with ConvFEM).

discretisation	implementation	solver	withdrawn	inserted
finite volume	neural network	multigrid with Jacobi iterations	0.5797	0.4347
finite volume	Fortran	Gauss-Seidel	0.5797	0.4347
finite element	neural network	multigrid with Jacobi iterations	0.5838	0.4370

Table 1: Values of k_{eff} for the fuel assembly test cases

3.4. Fuel Assembly - Time comparisons

Table 2 shows the time comparisons for 100 Jacobi iterations performed on the fuel assembly test case. The neural network implementation using the GPU used the multi-group network (see figure 6) but replaced the multigrid network (see figure 5) with the Jacobi network (see figure 2). The equivalent operations were performed in a Fortran code in serial using a CPU. It can be observed that the average time for the neural network solver was less than one-third that of the time for the solver written in Fortran. The neural network solver also shows more consistent timings, with the difference between the minimum and maximum times being 0.0636 seconds. The Fortran solver shows a much greater variance in timings, with the difference between the minimum and maximum times being 1.3400 seconds.

implementation	hardware	max time (s)	min time (s)	average time (s)
neural network	NVIDIA RTX 6000 GPU	1.3412	1.2776	1.2819
Fortran	AMD EPYC 7742 CPU	5.2568	3.9168	4.3681

Table 2: Time comparisons for 100 Jacobi iterations performed on the fuel assembly test case using GPU for the neural network solver and a CPU for the Fortran code. The 100 iterations were performed 400 times so the maximum, minimum and average times from these are shown.

22

3.5. Reactor Core - Geometry and Configuration

We now model a reactor core using the cross-sections from the KAIST benchmark [44]. Unlike the benchmark, our core is a 3×3 grid of fuel assemblies of type UOX only. One quarter of the domain is modelled, using reflective boundary conditions to represent the rest of the core, see Figure 14. The reflector surrounding the fuel assemblies uses the moderator material. Vacuum boundary conditions are applied to the external boundary of the core. The width of the reflector and each of the assemblies is 21.42 cm so each side of the domain shown in Figure 14 measures 85.68 cm. Each lattice-cell of the assemblies has a computational grid of 10×10 cells (see Figure 8). The grid is uniform throughout the domain, meaning that the reflector contains $202,300$ cells, all nine fuel assemblies contain a total of $260,100$ cells and $2,724$ cells are used as halo cells needed to apply the boundary conditions. The energy was again discretised into seven groups resulting in $3,236,800$ degrees of freedom.
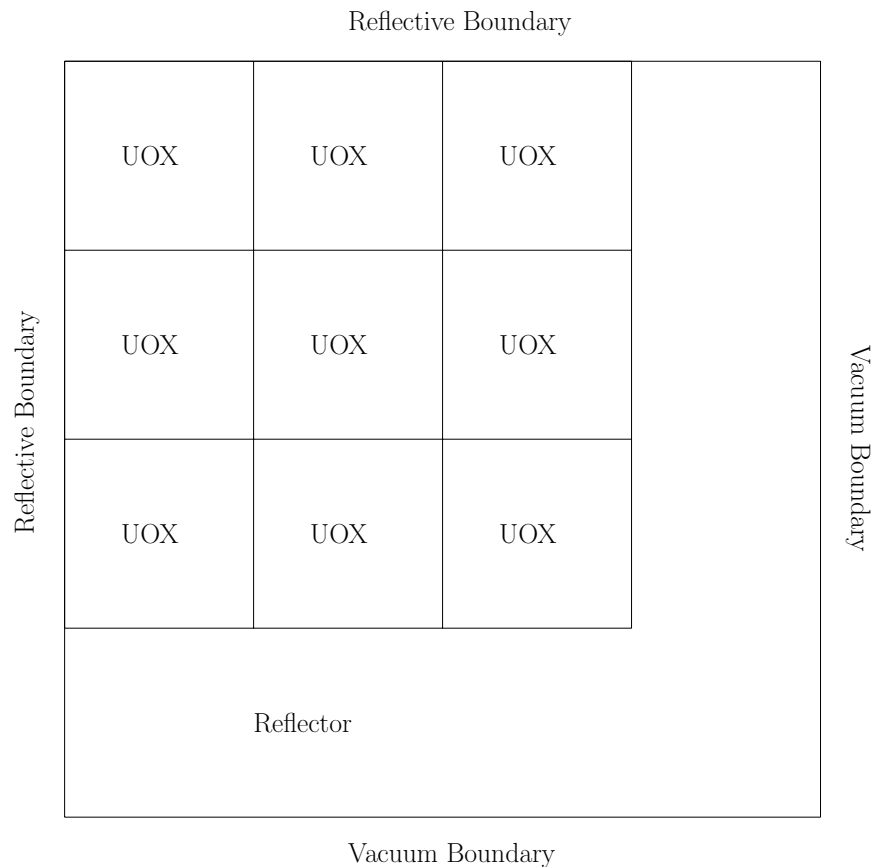
Reflective Boundary

Figure 14: Geometry of Reactor Core for a simplified version of the KAIST benchmark [44].

Each fuel assembly can either have control rods fully inserted or fully withdrawn. The two configurations of the core that are investigated here can be seen in Figure 15. Configuration one has five fuel assemblies with fully withdrawn control rods and four fuel assemblies with control rods fully inserted. Configuration two has six fuel assemblies with fully withdrawn control rods and three fuel assemblies with fully inserted

control rods. These configurations were chosen randomly.

Inserted	Withdrawn	Withdrawn
Withdrawn	Inserted	Withdrawn
Withdrawn	Inserted	Inserted

(**a**) Reactor configuration one.

Withdrawn	Withdrawn	Withdrawn
Withdrawn	Inserted	Withdrawn
Inserted	Inserted	Withdrawn

(**b**) Reactor configuration two.

Figure 15: Reactor core configurations where withdrawn means control rods are fully withdrawn and inserted means control rods are fully inserted.

3.6. Reactor Core - Finite Volume discretisation

A neural network with weights determined by a finite volume discretisation was used to solve the 2D neutron diffusion equation and give solutions for the reactor core described in the previous section. For all the solutions in this section, 5 Jacobi iterations, 100 multigrid iterations and 100 multi-group iterations were performed. Figure 16 contains the flux profiles for four energy groups for reactor configuration one. It can be observed that flux is higher in regions where control rods are fully withdrawn, with a notable drop for the flux of all the energy groups in the upper left corner where they are inserted. In the flux profile of the lowest energy group (group 7), the locations of the control rods and the moderator (within the guide tubes) are clearly picked out with the flux decreasing or increasing sharply respectively.
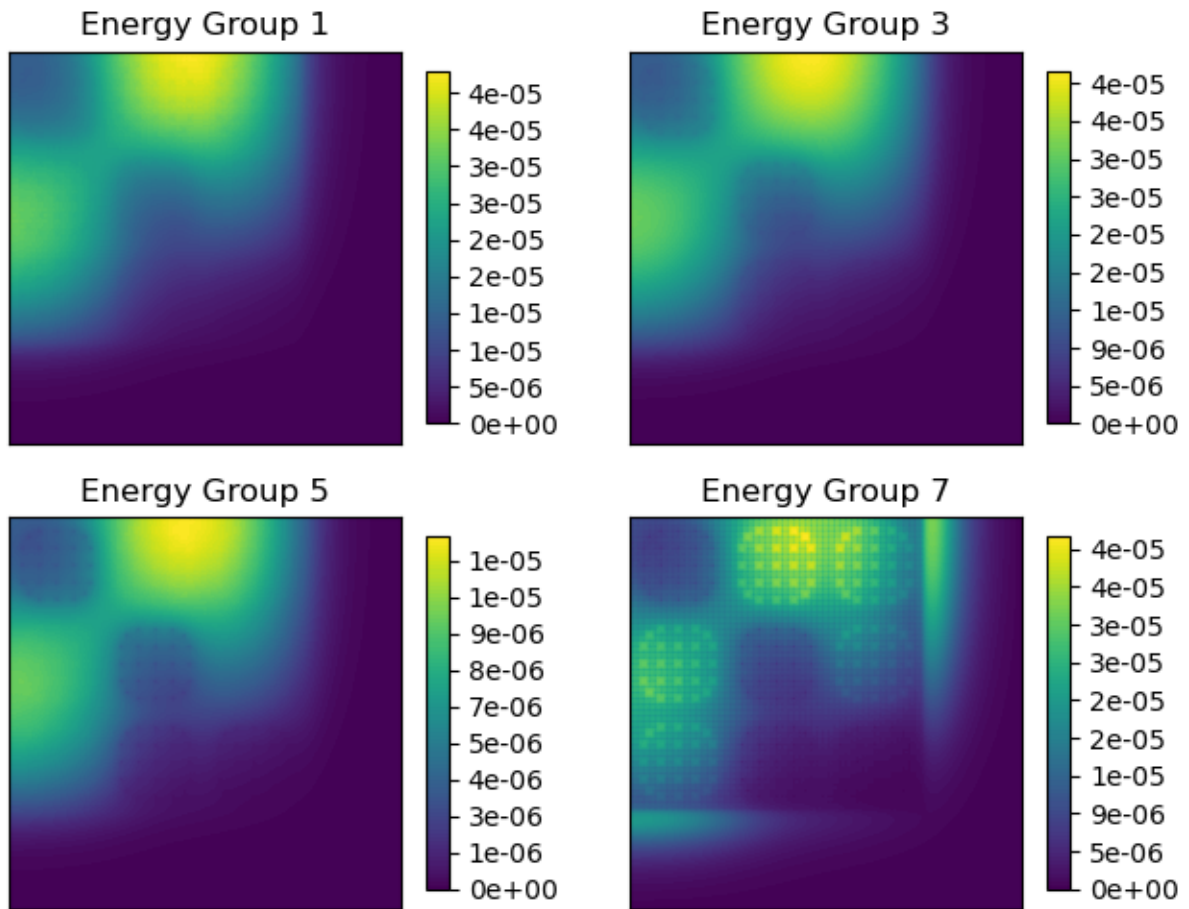
Figure 16: Scalar flux (neutrons cm^{-2} s^{-1}) across the fuel assembly for four energy groups for reactor configuration one, generated using the multi-group network with the finite volume discretisation.

Figure 17 contains the flux profiles for four energy groups for reactor configuration two. Again, the flux drops sharply where control rods are inserted, with the highest flux values occurring in the upper left corner by the reflective boundaries. The locations of both the control rods and moderator within the guide tubes are clearly picked out in the flux profile of the lowest energy group (group 7).
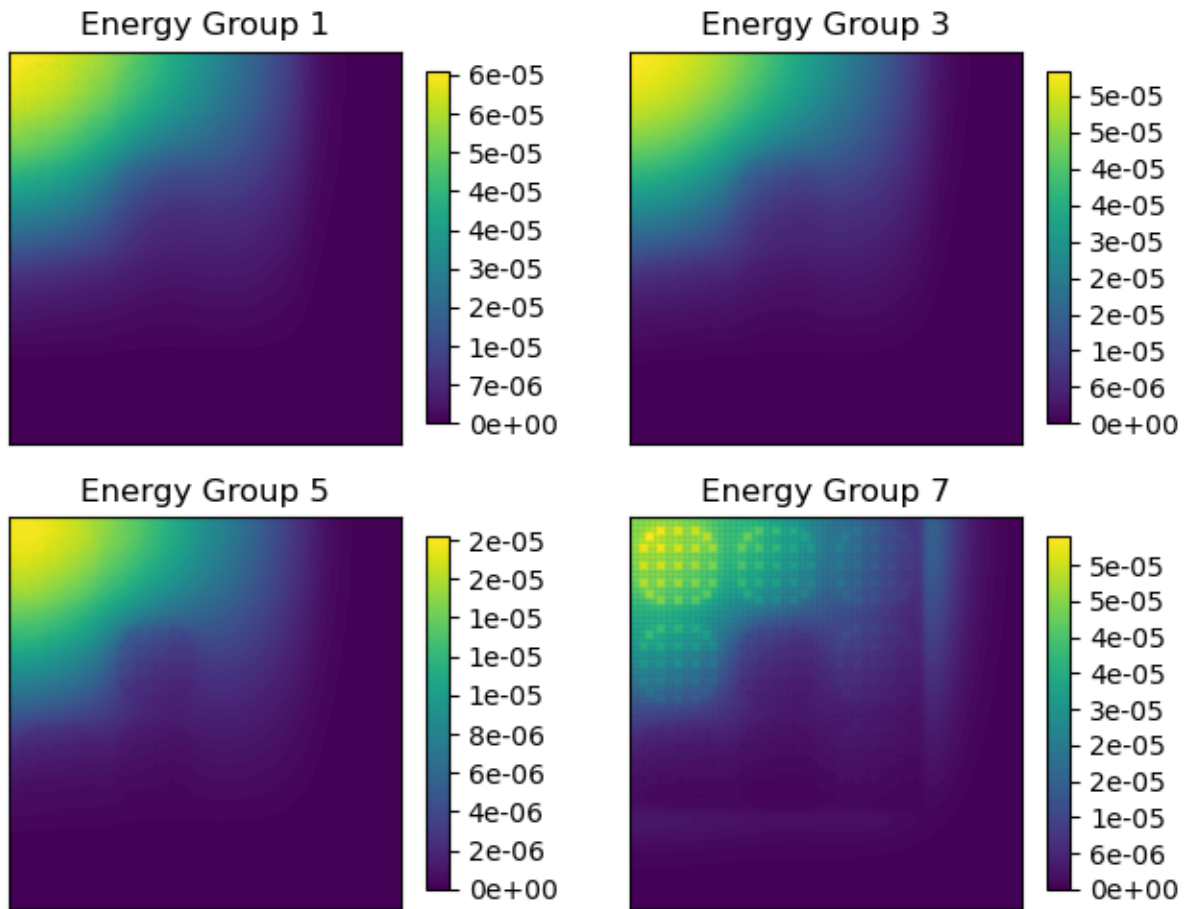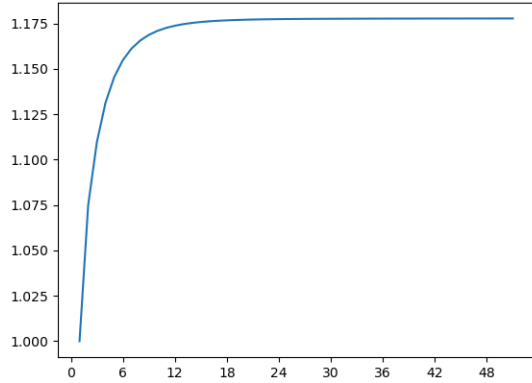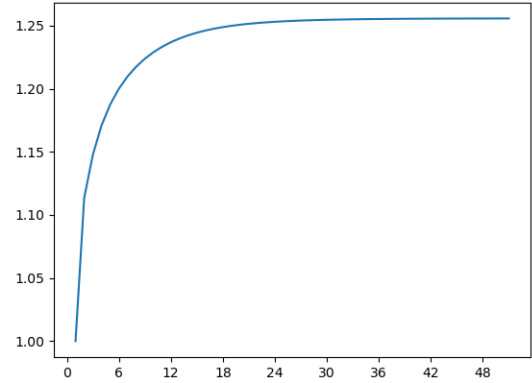
Figure 17: Scalar flux (neutrons cm^{-2} s^{-1}) across the fuel assembly for four energy groups for reactor configuration two, generated using the multi-group network for the finite volume discretisation.

Figure 18 shows the convergence of k_{eff} for both reactor configurations. It can be observed that configuration two has a slightly higher k_{eff} than configuration one, which is expected as configuration two has fewer control rods inserted.

(a) k_{eff} vs power iteration for reactor configuration one, converging to 1.1777.

(b) k_{eff} vs power iteration for reactor configuration two, converging to 1.2557.

Figure 18: Reactor core k_{eff} vs power iteration using the multi-group network with a finite volume discretisation for both configurations.

4. Conclusions and Future work

This paper presents a new approach that uses the tools within Artificial Intelligence (AI) software libraries to replicate the processes of solving partial differential equations that have been discretised through standard numerical method schemes. Whilst applicable to partial differential equations (PDEs) in general, this article has focused on the field of nuclear reactor physics and solves the eigenvalue problem arising from neutron transport, as described through diffusion theory. Furthermore, whilst underlying discretisation methods can be arbitrary, our demonstration focuses on the use of convolutional neural networks to replicate the solution process when using the finite volume method. Instead of training the network, the approach taken here is to define the weights of convolutional neural network in order to reproduce the discretisation exactly. Iterative solvers are also replicated within the network. A sawtooth multigrid method based on the U-Net architecture with an internal Jacobi iteration is investigated here. The multigrid network is then used with another network that acts across all energy groups as a multi-group solver.

Two test cases are used to demonstrate the approach, a fuel assembly and a reactor core. For the fuel assembly test case, the solution from the neural network solution is compared with the same finite volume discretisation solved by a Gauss-Seidel method and implemented in a standard way using Fortran. The absolute pointwise error between the two solutions was $\mathcal{O}(10^{-10})$. The fuel assembly test case demonstrates that the approach produces the identical solution (accounting for solver tolerances) to that obtained through a standard approach, and produces the same rate of convergence for k_{eff}. This test case is also used to demonstrate how a quadratic finite element discretisation may be used in the convolutional

27

layers. The approach is also extended to a more computationally demanding problem, in the form of a reactor core.

A benefit of using such an approach is that it allows one to exploit the power of AI libraries and their built-in technologies. For example, their executions are already optimised for different computer architectures, whether it be CPUs, GPUs or new-generation AI processors. This flexibility brings within easy reach the ability to run code on multiple platforms without the need for modification of the code. A further benefit is that of simplified code development, as the AI libraries abstract away code relating to the platform, leaving the user to concentrate on their programming tasks. As well as exploiting the substantial developments already made in AI libraries, formulating numerical discretisations as convolutional layers in neural networks will mean that these codes are ready to run on the latest AI processors.

Future work will involve including the power eigenvalue iteration within the neural network. This would enable the neural network to calculate sensitivities of the eigenvalue to material properties automatically, using the backpropagation algorithm of the neural network. An important next step would be to optimise the code and methods further (e.g. taking into account the multigrid bottleneck caused by the coarsest grid) so that large problems can be run on GPUs or new AI computers.

CRediT authorship contribution statement

TRFP: methodology, software, writing (original draft, review and editing). **CEH:** methodology, writing (original draft, review and editing), supervision. **BC:** software, writing (review and editing). **AGB:** software, writing (original draft, review and editing). **CCP:** conceptualisation, methodology, software, writing (original draft, review and editing), supervision, funding acquisition.

Acknowledgements

References

[1] M. Abadi, P. Agarwal, Aand Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser,

M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from `www.tensorflow.org`.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[3] B. Vermeire, F. Witherden, P. Vincent, On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools, Journal of Computational Physics 334 (2017) 497–521.

[4] J. Chan, Z. Wang, A. Modave, J.-F. Remacle, T. Warburton, GPU-accelerated discontinuous Galerkin methods on hybrid meshes, Journal of Computational Physics 318 (2016) 142–168.

[5] R. M. Bergmann, K. L. Rowland, N. Radnović, R. N. Slaybaugh, J. L. Vujić, Performance and accuracy of criticality calculations performed using WARP — A framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs, Annals of Nuclear Energy 103 (2017) 334–349.

[6] R. N. Slaybaugh, M. Ramirez-Zweiger, T. Pandya, S. Hamilton, T. M. Evans, Eigenvalue Solvers for Modeling Nuclear Reactors on Leadership Class Machines, Nuclear Science and Engineering 190 (2018) 31–44.

[7] C. Cecka, A. J. Lew, E. Darve, Assembly of finite element methods on graphics processors, International Journal for Numerical Methods in Engineering 85 (2011) 640–669.

[8] F. Mossaiby, R. Rossi, P. Dadvand, S. Idelsohn, OpenCL-based implementation of an unstructured edge-based finite element convection-diffusion solver on graphics hardware, International Journal for Numerical Methods in Engineering 89 (2012) 1635–1651.

[9] A. Dziekonski, P. Sypek, A. Lamecki, M. Mrozowski, Generation of large finite-element matrices on multiple graphics processors, International Journal for Numerical Methods in Engineering 94 (2013) 204–220.

[10] S. Sanfui, D. Sharma, A three-stage graphics processing unit-based finite element analyses matrix generation strategy for unstructured meshes, International Journal for Numerical Methods in Engineering 121 (2020) 3824–3848.

[11] A. Modave, A. Atle, J. Chan, T. Warburton, A GPU-accelerated nodal discontinuous Galerkin method with high-order absorbing boundary conditions and corner/edge compatibility, International Journal for Numerical Methods in Engineering 112 (2017) 1659–1686.

[12] A. G. M. Lewis, J. Beall, M. Ganahl, M. Hauru, S. B. Mallick, G. Vidal, Large-scale distributed linear algebra with tensor processing units, Proceedings of the National Academy of Sciences of the United States of America 119 (2022) e2122762119.

[13] Graphcore, Intelligence Processing Units, `https://www.graphcore.ai/products/ipu`, 2022. Accessed: 16-12-2022.

[14] Cerebras, CS-2: A Revolution in AI Infrastructure, `https://www.cerebras.net/product-system/`, 2022. Accessed: 2022-10-12.

[15] T. Lu, T. Marin, Y. Zhuo, Y.-F. Chen, C. Ma, Accelerating MRI Reconstruction on TPUs, in: 2020 IEEE High Performance Extreme Computing Conference (HPEC), 2020, pp. 1–9. doi:`10.1109/HPEC43674.2020.9286192`.

[16] T. Lu, T. Marin, Y. Zhuo, Y.-F. Chen, C. Ma, Nonuniform Fast Fourier Transform on TPUs, in: 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI), 2021, pp. 783–787. doi:`10.1109/ISBI48211.2021.9434068`.

[17] F. Belletti, D. King, K. Yang, R. Nelet, Y. Shafi, Y.-F. Shen, J. Anderson, Tensor processing units for financial monte carlo, in: Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing, 2020, pp. 12–23. doi:`10.1137/1.9781611976137.2`.

[18] A. Morningstar, M. Hauru, J. Beall, M. Ganahl, A. G. Lewis, V. Khemani, G. Vidal, Simulation of Quantum Many-Body Dynamics with Tensor Processing Units: Floquet Prethermalization, PRX Quantum 3 (2022) 020331.

[19] R. Pederson, J. Kozlowski, R. Song, J. Beall, M. Ganahl, M. Hauru, A. G. M. Lewis, S. B. Mallick, V. Blum, G. Vidal, Tensor Processing Units as Quantum Chemistry Supercomputers, arXiv preprint (2022) 2202.01255.

[20] X.-Z. Zhao, T.-Y. Xu, Z.-T. Ye, W.-J. Liu, A TensorFlow-based new high-performance computational framework for CFD, Journal of Hydrodynamics 32 (2020) 735–746.

[21] Q. Wang, M. Ihme, Y.-F. Chen, J. Anderson, A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units, Computer Physics Communications 274 (2022) 108292.

[22] B. Chen, C. E. Heaney, C. C. Pain, Using AI libraries for Incompressible Computational Fluid Dynamics, in preparation (2023).

[23] T. R. Phillips, C. E. Heaney, B. Chen, A. G. Buchan, C. C. Pain, Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport, in preparation (2023).

[24] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, in: Medical Image Computing and Computer-Assisted Intervention (MICCAI), volume 9351 of *LNCS*, Springer, 2015, pp. 234–241. doi:`10.48550/arXiv.1505.04597`.

[25] T.-W. Ke, M. Maire, S. X. Yu, Multigrid Neural Architectures, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4067–4075. doi:`10.1109/CVPR.2017.433`.

[26] J. He, J. Xu, MgNet: A unified framework of multigrid and convolutional neural network, Science China Mathematics 62 (2019) 1331–1354.

[27] N. Thuerey, K. Weißenow, L. Prantl, X. Hu, Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows, AIAA Journal 58 (2020) 25–36.

[28] Q. T. Le, C. Ooi, Surrogate modeling of fluid dynamics with a multigrid inspired neural network architecture, Machine Learning with Applications 6 (2021) 100176.

[29] N. Margenberg, D. Hartmann, C. Lessig, T. Richter, A neural network multigrid solver for the Navier-Stokes equations, Journal of Computational Physics 460 (2022) 110983.

[30] E. Cervi, X. Lu, A. Cammi, F. Di Maio, E. Zio, Sensitivity-analysis-driven surrogate model for molten salt reactors control, Journal of Nuclear Engineering 3 (2022) 277–294.

[31] H. D. Kabir, A. Khosravi, M. A. Hosen, S. Nahavandi, Neural network-based uncertainty quantification: A survey of methodologies and applications, IEEE Access 6 (2018) 36218–36234.

[32] R. K. Tripathy, I. Bilionis, Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification, Journal of Computational Physics 375 (2018) 565–588.

[33] H. Gong, S. Cheng, Z. Chen, Q. Li, Data-Enabled Physics-Informed Machine Learning for Reduced-Order Modeling Digital Twin: Application to Nuclear Reactor Physics, Nuclear Science and Engineering 196 (2022) 668–693.

[34] E. Schiassi, M. De Florio, B. D. Ganapol, P. Picca, R. Furfaro, Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics, Annals of Nuclear Energy 167 (2022) 108833.

[35] J. Wang, X. Peng, Z. Chen, B. Zhou, Y. Zhou, N. Zhou, Surrogate modeling for neutron diffusion problems based on conservative physics-informed neural networks with boundary conditions enforcement, Annals of Nuclear Energy 176 (2022) 109234.

[36] B. Foad, R. Elzohery, D. R. Novog, Demonstration of combined reduced order model and deep neural network for emulation of a time-dependent reactor transient, Annals of Nuclear Energy 171 (2022) 109017.

[37] T. R. F. Phillips, C. E. Heaney, P. N. Smith, C. C. Pain, An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion, International Journal for Numerical Methods in Engineering 122 (2021) 3780–3811.

[38] S. Qin, Q. Zhang, J. Zhang, L. Liang, Q. Zhao, H. Wu, L. Cao, Application of deep neural network for generating resonance self-shielded cross-section, Annals of Nuclear Energy 149 (2020) 107785.

[39] A. Beck, M. Kurz, A perspective on machine learning methods in turbulence modeling, GAMM-Mitteilungen 44 (2021) e202100002.

[40] F. S. Acton, Numerical methods that usually work, Mathematical Association of America, Washington DC, 1990.

[41] R. Reams, Hadamard inverses, square roots and products of almost semi-definite matrices, Linear Algebra and its Applications 288 (1999) 35–43.

[42] F. Chollet, et al., Keras, 2015. `https://keras.io`.

[43] G. H. Golub, C. F. Loan, Matrix Computations, John Hopkins University Press, 1996.

[44] Z. Cho, Kaist Benchmark Problem 2A : MOX Fuel-Loaded Small PWR Core, `http://nurapt.kaist.ac.kr/benchmark/kaist_ben1a.pdf`, 2000.

Appendix A. Diffusion operator

For two scalars ϕ_g and D_g, the following is true

$$\nabla^2(D_g\phi_g) \ = \ \nabla \cdot \nabla(D_g\phi_g) \tag{A.1}$$

$$= \ \nabla \cdot (\phi_g\nabla D_g + D_g\nabla\phi_g) \tag{A.2}$$

$$= \ \phi_g\nabla^2 D_g + D_g\nabla^2\phi_g + 2\nabla D_g \cdot \nabla\phi_g\,, \tag{A.3}$$

which leads to the following identity

$$2\nabla D_g \cdot \nabla\phi_g = \nabla^2(D_g\phi_g) - \phi_g\nabla^2 D_g - D_g\nabla^2\phi_g\,. \tag{A.4}$$

Expanding out the diffusion term in Equation (1) and then substituting in the expression from Equation (A.4) results in

$$\nabla \cdot (D_g\nabla\phi_g) \ = \ D_g\nabla^2\phi_g + \nabla\phi_g \cdot \nabla D_g \tag{A.5}$$

$$= \ D_g\nabla^2\phi_g + \frac{1}{2}\left(\nabla^2(D_g\phi_g) - \phi_g\nabla^2 D_g - D_g\nabla^2\phi_g\right) \tag{A.6}$$

$$= \ \frac{1}{2}\left(\nabla^2(D_g\phi_g) - \phi_g\nabla^2 D_g + D_g\nabla^2\phi_g\right)\,. \tag{A.7}$$

Equation (A.7) is used in Equation (21).

Appendix B. Equivalence of finite volume discretisation written in standard notation and written as convolutions

First, let us recall that the Hadamard product of two N by M matrices is given by

$$\boldsymbol{A} \odot \boldsymbol{B}\big|_{k\ell} = A_{k\ell}B_{k\ell} \quad \forall k \in \{1, 2, \ldots, N\}, \ell \in \{1, 2, \ldots, M\} \tag{B.1}$$

and the sign $\sum\limits_{\text{entries}}$ sums all the entries of a matrix

$$\sum_{\text{entries}} \boldsymbol{A} \equiv \sum_{k=1}^{M}\sum_{\ell=1}^{N} A_{k\ell}\,. \tag{B.2}$$

In this section, we will show equivalence of the diffusion operator's finite volume discretisation given in Equation (6) (also in Equations (13) and (14)) and the same discretisation formulated as a convolutional

layer with pre-defined weights as described by Equations (20) and (22). Considering each term on the right-hand side of Equation (22), we start with part of the second term and evaluating this in the i, jth cell:

$$\boldsymbol{f}(\boldsymbol{\Phi_g}; \boldsymbol{w})\Big|_{i,j} = \sum_{\text{entries}} \begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix} \odot \begin{bmatrix} \phi_{i-1,j+1,g} & \phi_{i,j+1,g} & \phi_{i+1,j+1,g} \\ \phi_{i-1,j,g} & \phi_{i,j,g} & \phi_{i+1,j,g} \\ \phi_{i-1,j-1,g} & \phi_{i,j-1,g} & \phi_{i+1,j-1,g} \end{bmatrix} \tag{B.3}$$

$$= \frac{-(\phi_{i-1,j,g} + \phi_{i+1,j,g})}{\Delta x^2} + \frac{-(\phi_{i,j-1,g} + \phi_{i,j+1,g})}{\Delta y^2} + \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right)\phi_{i,j,g}. \tag{B.4}$$

Now, considering the second term in its entirety,

$$(\boldsymbol{D_g} \odot \boldsymbol{f}(\boldsymbol{\Phi_g}; \boldsymbol{w}))\Big|_{i,j} = \frac{-D_{i,j,g}(\phi_{i-1,j,g} + \phi_{i+1,j,g})}{\Delta x^2} + \frac{-D_{i,j,g}(\phi_{i,j-1,g} + \phi_{i,j+1,g})}{\Delta y^2}$$
$$+ \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right)D_{i,j,g}\phi_{i,j,g}. \tag{B.5}$$

Similarly, for the third term on the right-hand side of Equation (22)

$$(\boldsymbol{\Phi_g} \odot \boldsymbol{f}(\boldsymbol{D_g}; \boldsymbol{w}))\Big|_{i,j} = \frac{-\phi_{i,j,g}(D_{i-1,j,g} + D_{i+1,j,g})}{\Delta x^2} + \frac{-\phi_{i,j,g}(D_{i,j-1,g} + D_{i,j+1,g})}{\Delta y^2}$$
$$+ \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right)D_{i,j,g}\phi_{i,j,g}. \tag{B.6}$$

The first term on the right-hand side of Equation (22) can be expanded as follows

$$\boldsymbol{f}(\boldsymbol{D_g} \odot \boldsymbol{\Phi_g}; \boldsymbol{w})\Big|_{i,j} = \frac{-(D_{i-1,j,g}\phi_{i-1,j,g} + D_{i+1,j,g}\phi_{i+1,j,g})}{\Delta x^2} + \frac{-(D_{i,j-1,g}\phi_{i,j-1,g} + D_{i,j+1,g}\phi_{i,j+1,g})}{\Delta y^2}$$
$$+ \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right)D_{i,j,g}\phi_{i,j,g}. \tag{B.7}$$

Combining the expressions in Equations (B.5), (B.6) and (B.7) according to the definition of the diffusion

operator from Equation (22) and gathering terms that multiply each scalar flux term gives

$$
\boldsymbol{f}^{\mathrm{Diff}}(\boldsymbol{\Phi}_g, \boldsymbol{D}_g; \boldsymbol{w}) \quad = \quad \frac{1}{2}\left(\boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g; \boldsymbol{w}) + \boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g; \boldsymbol{w}) - \boldsymbol{\Phi}_g \odot \boldsymbol{f}(\boldsymbol{D}_g; \boldsymbol{w})\right) \tag{B.8}
$$

$$
\begin{aligned}
= \quad & \frac{-\left(D_{i-1,j,g}\phi_{i-1,j,g} + D_{i+1,j,g}\phi_{i+1,j,g}\right)}{2\Delta x^2} + \frac{-\left(D_{i,j-1,g}\phi_{i,j-1,g} + D_{i,j+1,g}\phi_{i,j+1,g}\right)}{2\Delta y^2} \\
& + \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}
\end{aligned} \tag{B.9}
$$

$$
\begin{aligned}
& + \frac{-D_{i,j,g}\left(\phi_{i-1,j,g} + \phi_{i+1,j,g}\right)}{2\Delta x^2} + \frac{-D_{i,j,g}\left(\phi_{i,j-1,g} + \phi_{i,j+1,g}\right)}{2\Delta y^2} + \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g} \\
& - \frac{-\phi_{i,j,g}\left(D_{i-1,j,g} + D_{i+1,j,g}\right)}{2\Delta x^2} - \frac{-\phi_{i,j,g}\left(D_{i,j-1,g} + D_{i,j+1,g}\right)}{2\Delta y^2} - \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}
\end{aligned}
$$

$$
\begin{aligned}
= \quad & -\left(\frac{D_{i-1,j,g} + D_{i,j,g}}{2\Delta x^2}\right)\phi_{i-1,j,g} - \left(\frac{D_{i,j,g} + D_{i+1,j,g}}{2\Delta x^2}\right)\phi_{i+1,j,g} \\
& - \left(\frac{D_{i,j-1,g} + D_{i,j,g}}{2\Delta y^2}\right)\phi_{i,j-1,g} - \left(\frac{D_{i,j,g} + D_{i,j+1,g}}{2\Delta y^2}\right)\phi_{i,j+1,g} \\
& + \left(\frac{D_{i-1,j,g} + 2D_{i,j,g} + D_{i+1,j,g}}{2\Delta x^2} + \frac{D_{i,j-1,g} + 2D_{i,j,g} + D_{i,j+1,g}}{2\Delta y^2}\right)\phi_{i,j,g}\,.
\end{aligned} \tag{B.10}
$$

From this, we can see that Equation (B.10) is equivalent to the discretised diffusion operator seen in Equation (6). In other words, this particular finite volume discretisation can be written as a convolutional layer in a neural network with a 3 by 3 kernel or filter with weights

$$
\boldsymbol{w} = \begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix} \tag{B.11}
$$