Check for updates

Received: 12 February 2023 | Revised: 13 June 2023 | Accepted: 25 June 2023

DOI: 10.1002/nme.7321

RESEARCH ARTICLE

WILEY

Solving the discretised neutron diffusion equations using neural networks

Toby R.F. Phillips[1] | **Claire E. Heaney**[1,2] | **Boyang Chen**[1] | **Andrew G. Buchan**[3] | **Christopher C. Pain**[1,2]

[1]Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, London, UK

[2]Centre for AI-Physics Modelling, Imperial-X, White City Campus, Imperial College London, London, UK

[3]School of Engineering and Materials Science, Queen Mary University of London, London, UK

Correspondence
Claire E. Heaney, Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, London, SW7 2AZ, UK.
Email: c.heaney@imperial.ac.uk

Funding information
Engineering and Physical Sciences Research Council, Grant/Award Numbers: EP/L015900/1, EP/T000414/1, EP/V036777/1, EP/W001411/1, EP/T003189/1, EP/N010221/1, EP/P033180/1

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 built-in technologies to enable the same code to run on different computer architectures (such as central processing units, graphics processing units or new-generation AI processors) without any modification. 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. For both benchmarks we give timings for the neural network implementation run on a CPU and a GPU, and a serial Fortran code run on a CPU.

KEYWORDS

convolutional neural network, finite difference method, finite volume method, multigrid solver, neutron diffusion equation, numerical solution of partial differential equations, reactor physics, U-net

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.
© 2023 The Authors. *International Journal for Numerical Methods in Engineering* published by John Wiley & Sons Ltd.

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 (e.g., 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 can run their code on central processing units (CPUs), graphics processing units (GPUs) or AI processors (also known as accelerators) without having to make any modifications to the code. 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. For some applications, GPUs have demonstrated superior performance to CPUs,[3,4] however, instructions for the GPU must be written in languages such as CUDA or OpenCL that are unfamiliar to many working in scientific computation. In addition, in order to realise the potential speed-up of GPUs, the algorithmic implementation often has to be very different from that of CPUs.[5,6] GPUs perform better with algorithms that are able to perform tasks simultaneously rather than sequentially, so Jacobi iteration would be more efficient than sequential Gauss-Seidel iteration, for example. These additional coding tasks and algorithmic developments have hindered the take-up of GPUs, although there are examples of this having been done successfully, for example, in computational fluid dynamics,[3,7] for acoustic waves[6] and, in radiation transport, for a Monte Carlo neutron transport code[4] and for eigenvalue problems.[8] With CUDA and OpenCL, GPUs have been used to accelerate generation of finite element matrices for unstructured meshes[9-12] and for discontinuous Galerkin methods.[13] 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.[14] 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,[14] intelligence processing units (IPUs) of Graphcore[15] and CS-2 of Cerebras.[16] 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 (e.g., CPUs, GPUs or the new AI processors) without having to make any 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;[17,18] Monte Carlo simulations for finance;[19] many-body quantum physics;[20] and density functional theory.[21] We have found four examples of previous work which exploit the operations associated with neural networks that are found within AI libraries in order to solve scientific problems.[22-25] Zhao et al.[22] 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.[23] present a similar idea to Zhao et al.,[22] again using TensorFlow to implement finite difference discretisations of CFD problems, however using tensor processing units (TPUs) rather than GPUs. They solve the variable-density Navier-Stokes equations and demonstrate good weak and strong scaling. Chen et al.[24] 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 (a convolutional neural network with a specific architecture[26]). Using convolutional neural networks with pre-determined weights, Phillips et al.[25] 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 (2D in space, 2D in angle), 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.[26] Consequently, the use of multigrid-inspired architectures for (trained) neural networks has been explored and been shown to enhance performance relative to conventional architectures based on the convolutional neural network (CNN) for applications in computer vision[27,28] as well as in computational fluid dynamics (CFD).[29,30] Taking a different approach, Margenberg et al.[31] 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

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

integrating the multigrid method with (trained) neural networks,[27-31] Chen et al.[24] implement 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 the latter approach[24] that is adopted in our current investigation as we are primarily interested in solving discretisations exactly, instead of using (trained) neural networks to approximate discretised solutions akin to a surrogate modelling approach.

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 particular 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 multigrid 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 alternative 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 on all architectures (that support AI libraries) without the need for modification. Furthermore, through the neural networks, the latest developments can be realised for methods such as sensitivities,[32] uncertainty quantification[33,34] and data assimilation.[35] 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,[36] for non-smooth heterogeneous neutron diffusion problems[37] and for radiation transport in heterogeneous media;[38] surrogate models for transient analysis,[39] eigenvalue problems[40] and digital twins;[35] and cross-section generation with neural networks.[41] The approach presented in our paper 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.[22,24,42]

The sections of this paper are organised as follows. Section 2 describes how a convolutional neural network can be used 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 a 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.

2.1 | Diffusion equation

The multi-group steady-state diffusion equation for criticality[43] [Sect. 7] can be written as:

$$-\nabla\cdot(D_g\nabla\phi_g)+\Sigma^a_g\phi_g+\sum_{\substack{g'=1\\g'\neq g}}^{N_g}\Sigma^s_{g\to g'}\phi_g \;=\; \sum_{\substack{g'=1\\g'\neq g}}^{N_g}\Sigma^s_{g'\to g}\phi_{g'}+\lambda\chi_g\sum_{g'=1}^{N_g}\nu_{g'}\Sigma^f_{g'}\phi_{g'}, \qquad \forall g\in\{1,2,\ldots,N_g\}, \tag{1}$$

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

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 energy group and the notation $g' \to g$ indicates neutrons scattering from group g' to group g. The diffusion coefficient, D_g, is defined as:

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

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:

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

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

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:

$$\begin{aligned}
&-\frac{\left(D_{i-1,j,g}+D_{i,j,g}\right)}{2\Delta x^2}\phi_{i-1,j,g} - \frac{\left(D_{i,j,g}+D_{i+1,j,g}\right)}{2\Delta x^2}\phi_{i+1,j,g} - \frac{\left(D_{i,j-1,g}+D_{i,j,g}\right)}{2\Delta y^2}\phi_{i,j-1,g}\\
&-\frac{\left(D_{i,j,g}+D_{i,j+1,g}\right)}{2\Delta y^2}\phi_{i,j+1,g} + \left(\frac{\left(D_{i-1,j,g}+2D_{i,j,g}+D_{i+1,j,g}\right)}{2\Delta x^2} + \frac{\left(D_{i,j-1,g}+2D_{i,j,g}+D_{i,j+1,g}\right)}{2\Delta y^2}\right)\phi_{i,j,g}\\
&+\Sigma^a{}_{i,j,g}\phi_{i,j,g} + \sum_{\substack{g'=1\\ g'\neq g}}^{N_g}\Sigma^s_{i,j,g\to i,j,g'}\phi_{i,j,g} = \sum_{g'=1}^{N_g}\Sigma^s_{i,j,g'\to i,j,g}\phi_{i,j,g'} + \lambda\chi_g\sum_{g'=1}^{N_g}\nu_{g'}\Sigma^f_{i,j,g'}\phi_{i,j,g'},\\
&\qquad \forall i\in\{2,3,\ldots,N_x-1\}, \qquad \forall j\in\{2,3,\ldots,N_y-1\}, \qquad \forall g\in\{1,2,\ldots,N_g\},
\end{aligned} \tag{6}$$

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 particular finite volume discretisation is equivalent to a finite difference discretisation.[44,45] Details of how to derive Equation (6) are given in Appendix A. 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 neighbours,

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

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:

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

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

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

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

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

For the five 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:

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

where the matrix $\boldsymbol{A}$ contains the absorption, diffusion and scattering terms; matrix $\boldsymbol{B}$ represents the fission terms; and the vector $\boldsymbol{\phi}$ 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

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

where

$$a^{u,v}_{i,j,g} = \begin{cases} -\dfrac{\left(D_{i,j,g}+D_{i+u,j+v,g}\right)}{2\Delta x^2} & \text{for } |u|=1, v=0 \\ -\dfrac{\left(D_{i,j,g}+D_{i+u,j+v,g}\right)}{2\Delta y^2} & \text{for } u=0, |v|=1 \\ \dfrac{D_{i-1,j,g}+2D_{i,j,g}+D_{i+1,j,g}}{2\Delta x^2} + \dfrac{D_{i,j-1,g}+2D_{i,j,g}+D_{i,j+1,g}}{2\Delta y^2} + \Sigma^{as}_{i,j,g} & \text{for } u=0=v \\ 0 & \text{for } |u|=1=|v| \end{cases} \tag{14}$$

$$\Sigma^{as}_{i,j,g} = \Sigma^a_{i,j,g} + \sum_{\substack{g'=1 \\ g'\neq g}}^{N_g} \Sigma^s_{i,j,g\rightarrow i,j,g'} \tag{15}$$

$$s_{i,j,g} = \sum_{g'=1}^{g-1} \Sigma^s_{i,j,g'\rightarrow i,j,g}\phi^{(k+1)}_{i,j,g'} + \sum_{g'=g}^{N_g} \Sigma^s_{i,j,g'\rightarrow i,j,g}\phi^{(k)}_{i,j,g'} + \lambda\chi_g \sum_{g'=1}^{N_g} \nu_{g'}\Sigma^f_{i,j,g'}\phi^{(k)}_{i,j,g'}. \tag{16}$$

As the stencil used in Equation (6) is a five 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:

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

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

where $2l+1$ is the width of the stencil, k is the Jacobi iteration and $\left\{\left\{a_{i,j,g}^{u,v}\right\}_{u=-l}^{l}\right\}_{v=-l}^{l}$ 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.[46] 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 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

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

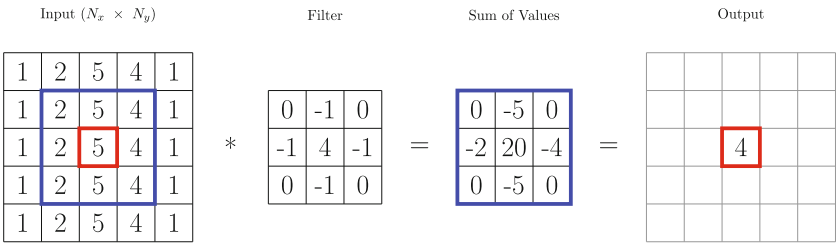


FIGURE 1 A 3 by 3 convolutional filter which applies the discretised diffusion operator (a five-point finite volume stencil) in 2D to 9 cells. The filter is first applied to all the cells in the blue block on the left; the result of which can be seen in the blue block following the equals sign. The 9 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.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

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 C2) 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 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 B), we can therefore write the diffusion operator as three convolutions:

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

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

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 C. The discretised diffusion equation can now be written for energy group g as:

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

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 absorption 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:

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

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

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

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:

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

where $\left(\boldsymbol{A}_g^{0,0}\right)^{\odot -1}$ is the Hadamard inverse[47] which is an $N_x \times N_y$ array whose i^{th},j^{th} component is $\frac{1}{a_{i,j,g}^{0,0}}$ for energy group g. This equation can be written as a function J,

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

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

which calculates the updated solution after one Jacobi iteration. Figure 2 shows the architecture of this function, that is, the neural network that solves one Jacobi iteration of the neutron transport problem as discretised in Equation (26). Green boxes contain the inputs; cyan 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)[25] of the diffusion operator. For quadratic 9-noded rectangular elements, the 5×5 filter of this discretisation is given by

$$\boldsymbol{w} = \frac{1}{900\Delta x^2}\begin{bmatrix} -5 & 50 & -15 & 50 & -5 \\ 50 & -320 & -660 & -320 & 50 \\ -15 & -660 & 3600 & -660 & -15 \\ 50 & -320 & -660 & -320 & 50 \\ -5 & 50 & -15 & 50 & -5 \end{bmatrix}, \tag{28}$$

assuming $\Delta y = \Delta x$. Using a higher-order discretisation results in a larger stencil which requires a wider halo containing more layers of cells. For the quadratic discretisation, this means that Equation (6) is solved for $\forall i \in \{3, 4, \ldots, N_x - 2\}$ and $\forall j \in \{3, 4, \ldots, N_y - 2\}$. The representation of the boundary conditions also changes when using higher-order discretisations. Assuming that the material parameters are homogeneous around the entire boundary for a width of at least two

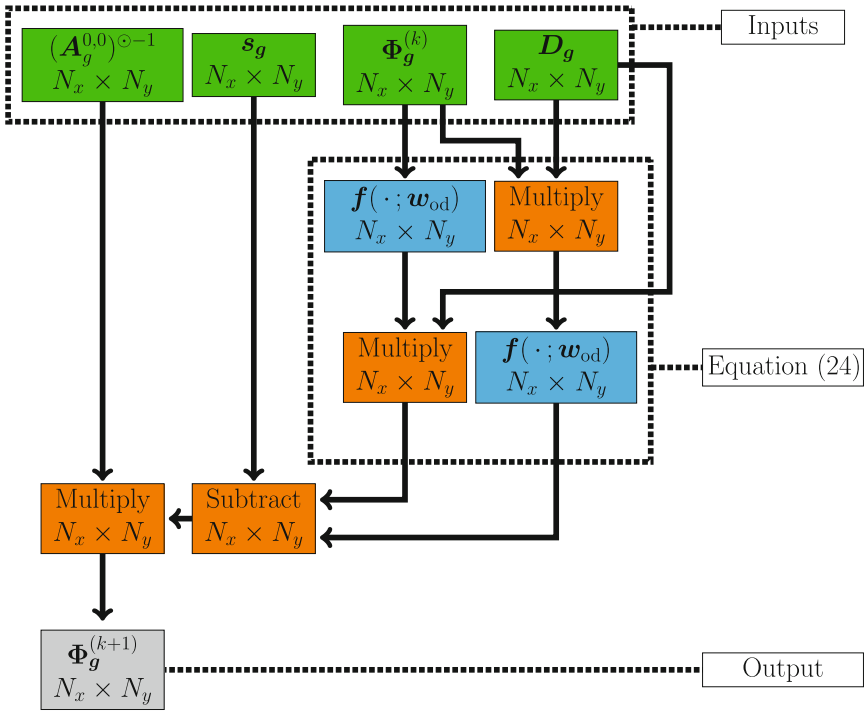


FIGURE 2 Schematic of the neural network used for a single Jacobi iteration written as $\mathrm{J}(\cdot)$ in Equation (26). This network performs a single Jacobi iteration on the flux of a single energy group. The inputs are the flux ($\boldsymbol{\Phi}_g^{(k)}$), representative source ($\boldsymbol{s}_g$), diffusion coefficients ($\boldsymbol{D}_g$) and the strictly diagonal coefficients $\left(\boldsymbol{A}_g^{0,0}\right)^{\odot -1}$ (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 ($\boldsymbol{\Phi}_g^{(k+1)}$). 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.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

cells, then boundary conditions can be enforced easily with the same approach as in Equations (7) and (8). For diffusion coefficients for the left, right, top and bottom sides of D_{left}, D_{right}, D_{top} and D_{bottom} respectively, this can be written as

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

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

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 an 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).

2.4 | Multigrid

In Figure 3, we show how the U-Net[26] 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 the subscript indicating energy group is no longer shown and the bold subscript now indicates the coarseness of the mesh, with 1 being the finest mesh. The residual ($\boldsymbol{r}^{\boldsymbol{k}}$) is calculated using:

$$\boldsymbol{r}_{\mathbf{1}}^{(k)} = \boldsymbol{s}_{\mathbf{1}} - \left(\boldsymbol{A}_{\mathbf{1}}^{0,0} \odot \boldsymbol{\Phi}_{\mathbf{1}}^{(k)} + \frac{1}{2}\left(\boldsymbol{D}_{\mathbf{1}} \odot \boldsymbol{f}\left(\boldsymbol{\Phi}_{\mathbf{1}}^{(k)}; \boldsymbol{w}_{\mathbf{od}}\right) + \boldsymbol{f}\left(\boldsymbol{D}_{\mathbf{1}} \odot \boldsymbol{\Phi}_{\mathbf{1}}^{(k)}; \boldsymbol{w}_{\mathbf{od}}\right)\right)\right) \tag{32}$$

which is then restricted twice to $\left(\boldsymbol{r}_{\mathbf{2}}^{(k)}\right)$ and $\left(\boldsymbol{r}_{\mathbf{3}}^{(k)}\right)$. A Jacobi iteration is performed on the coarsest level (bold subscript 3) to determine $\Delta\boldsymbol{\Phi}_{\mathbf{3}}^{(k)}$, starting with an array of zeros. This is prolongated to estimate $\widetilde{\Delta\boldsymbol{\Phi}_{\boldsymbol{2}}}^{(k)}$ which is then smoothed to $\Delta\boldsymbol{\Phi}_{\mathbf{2}}^{(k)}$ 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 with the convolution:

$$\boldsymbol{r}_{\mathbf{2}}^{(k)} = \boldsymbol{f}(\boldsymbol{r}_{\mathbf{1}}^{(k)}; \boldsymbol{w}_{\boldsymbol{R}}), \tag{33}$$

with filter weights:

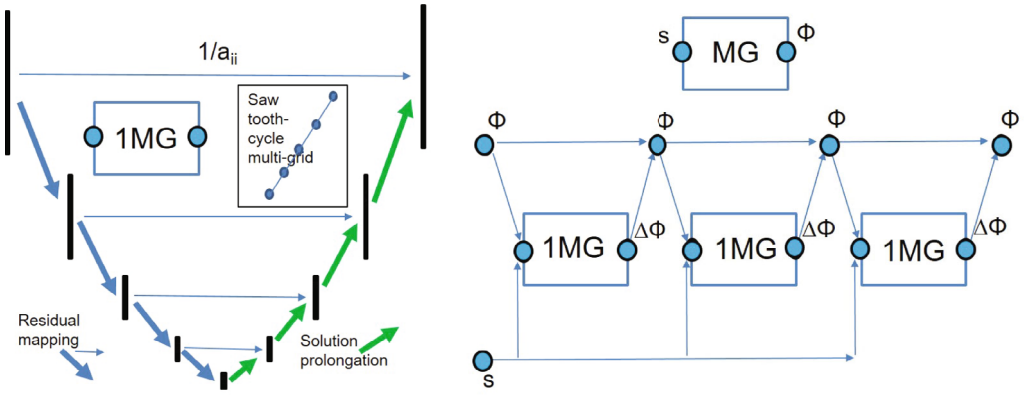


FIGURE 3 A schematic diagram showing the U-net architecture (left) that is used to form a single multigrid sawtooth cycle. On the right, we can see how multiple cycles are brought together to form the overall solution method.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

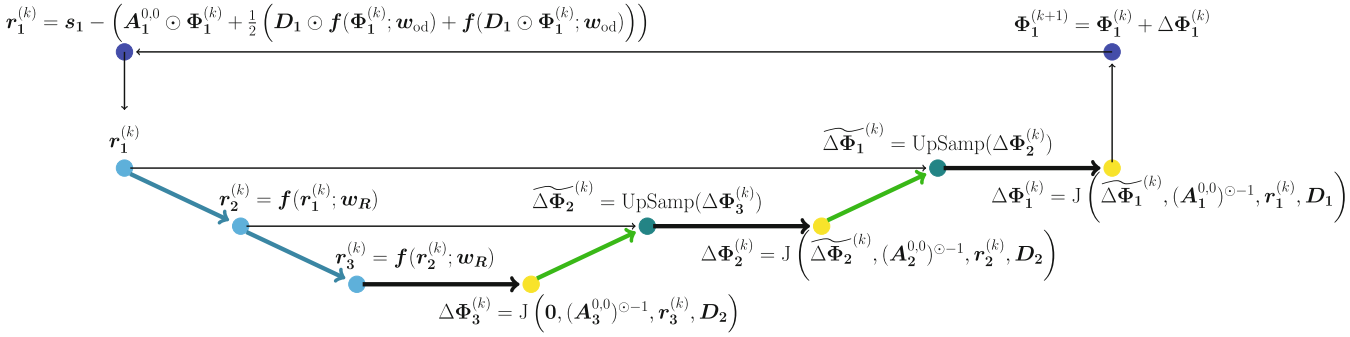


FIGURE 4 Multigrid iteration, with the subscript indicating the resolution and superscript representing the multigrid iteration and $\text{J}(\cdot)$ representing a Jacobi iteration. The residual is calculated and restricted twice, indicated by the cyan nodes. These residuals are used with Jacobi smoothing, 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.

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

The same approach is used to homogenise material parameters when restricting the results from a finer grid to a coarser grid. Replacing $\boldsymbol{r_1^{(k)}}$ in Equation (33) with either $\boldsymbol{\Sigma_f}$, $\boldsymbol{\Sigma_s}$ or $\boldsymbol{\Sigma_a}$ will produce the homogenised material parameters for a coarser level, which is then used to construct $\boldsymbol{A}$ for that level. 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[48] and results in an approximation for the solution on a finer mesh:

$$\widetilde{\boldsymbol{\Phi_1}}^{(k)} = \text{UpSamp}(\boldsymbol{\Phi_2^{(k)}}). \tag{35}$$

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

$$\boldsymbol{\Phi_1^{(k+1)}} = \text{MG}\left(\boldsymbol{\Phi_1^{(k)}}, \boldsymbol{s_1}, \left(\boldsymbol{A_1^{0,0}}\right), \left(\boldsymbol{A_1^{0,0}}\right)^{\odot -1}, \left(\boldsymbol{A_2^{0,0}}\right)^{\odot -1}, \left(\boldsymbol{A_3^{0,0}}\right)^{\odot -1}, \boldsymbol{D_1}, \boldsymbol{D_2}, \boldsymbol{D_3}\right), \tag{36}$$

and for a single energy group g:

$$\boldsymbol{\Phi_{1g}^{(k+1)}} = \text{MG}_g\left(\boldsymbol{\Phi_{1g}^{(k)}}, \boldsymbol{s_{1g}}, \left(\boldsymbol{A_{1g}^{0,0}}\right), \left(\boldsymbol{A_{1g}^{0,0}}\right)^{\odot -1}, \left(\boldsymbol{A_{2g}^{0,0}}\right)^{\odot -1}, \left(\boldsymbol{A_{3g}^{0,0}}\right)^{\odot -1}, \boldsymbol{D_{1g}}, \boldsymbol{D_{2g}}, \boldsymbol{D_{3g}}\right), \tag{37}$$

where $\text{MG}(\cdot)$ 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. $\text{MG}_g(\cdot)$ 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,

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

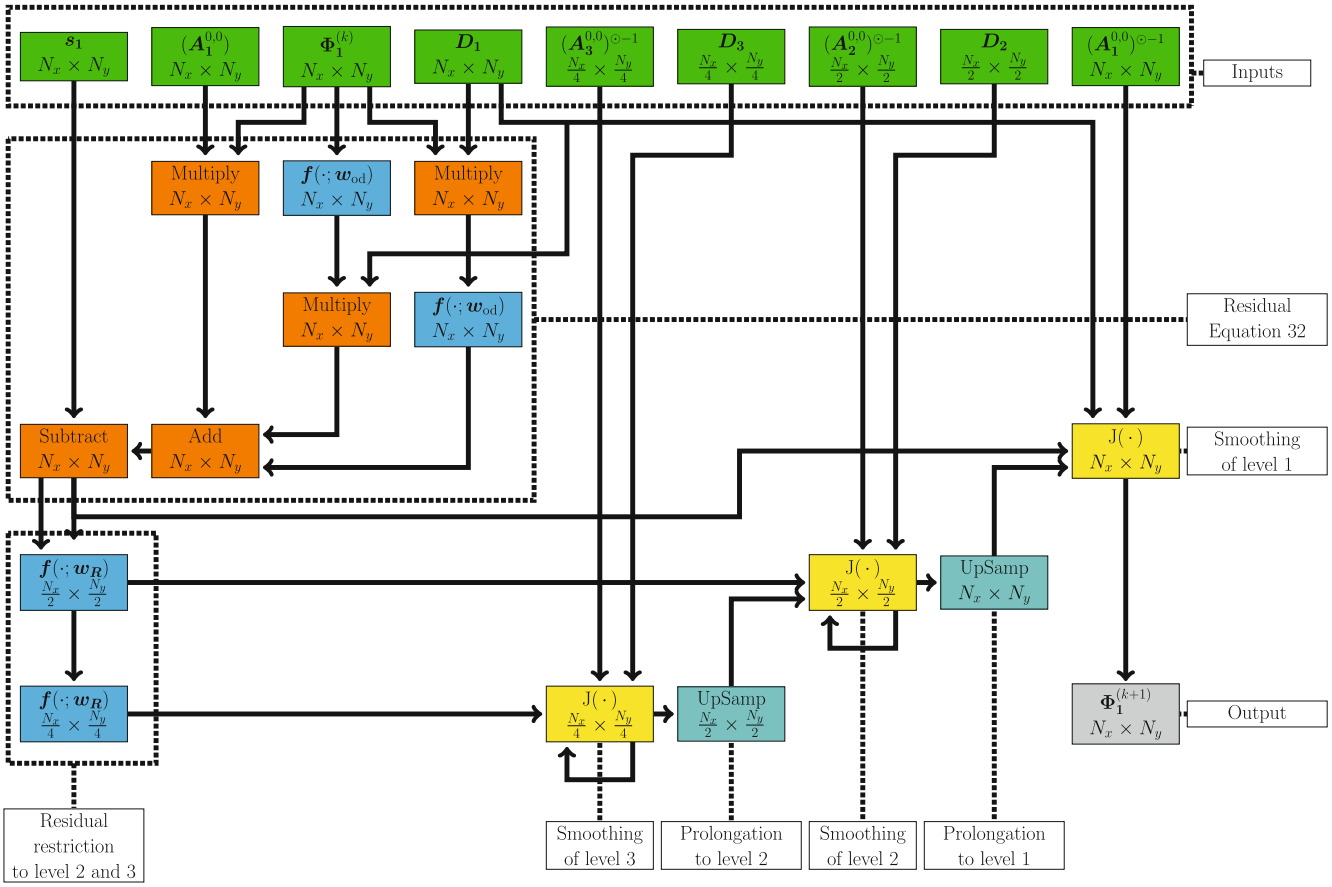


FIGURE 5 Multigrid network, MG(·), representing a single multigrid iteration. This network performs a single multigrid iteration on the flux of a single energy group. As inputs (green boxes), the network takes the flux ($\boldsymbol{\Phi}_1^{(k)}$), representative source ($\boldsymbol{s}_I$), diffusion coefficients ($\boldsymbol{D}_I$) and the strictly diagonal coefficients ($\boldsymbol{A}_1^{0,0}$), along with the coarser resolution coefficients. 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.

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. $\boldsymbol{s}_g$ is a vector containing Σ^s, $\boldsymbol{s}_{\mathbf{fiss}}$ and $\boldsymbol{\Phi}^{(k)}$ and is formed using Equation (16). $\boldsymbol{s}_g$ is then used in the MG sub-model to resolve for $\boldsymbol{\Phi}_g$, repeating for a number of multigrid iterations until:

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

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

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

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:

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

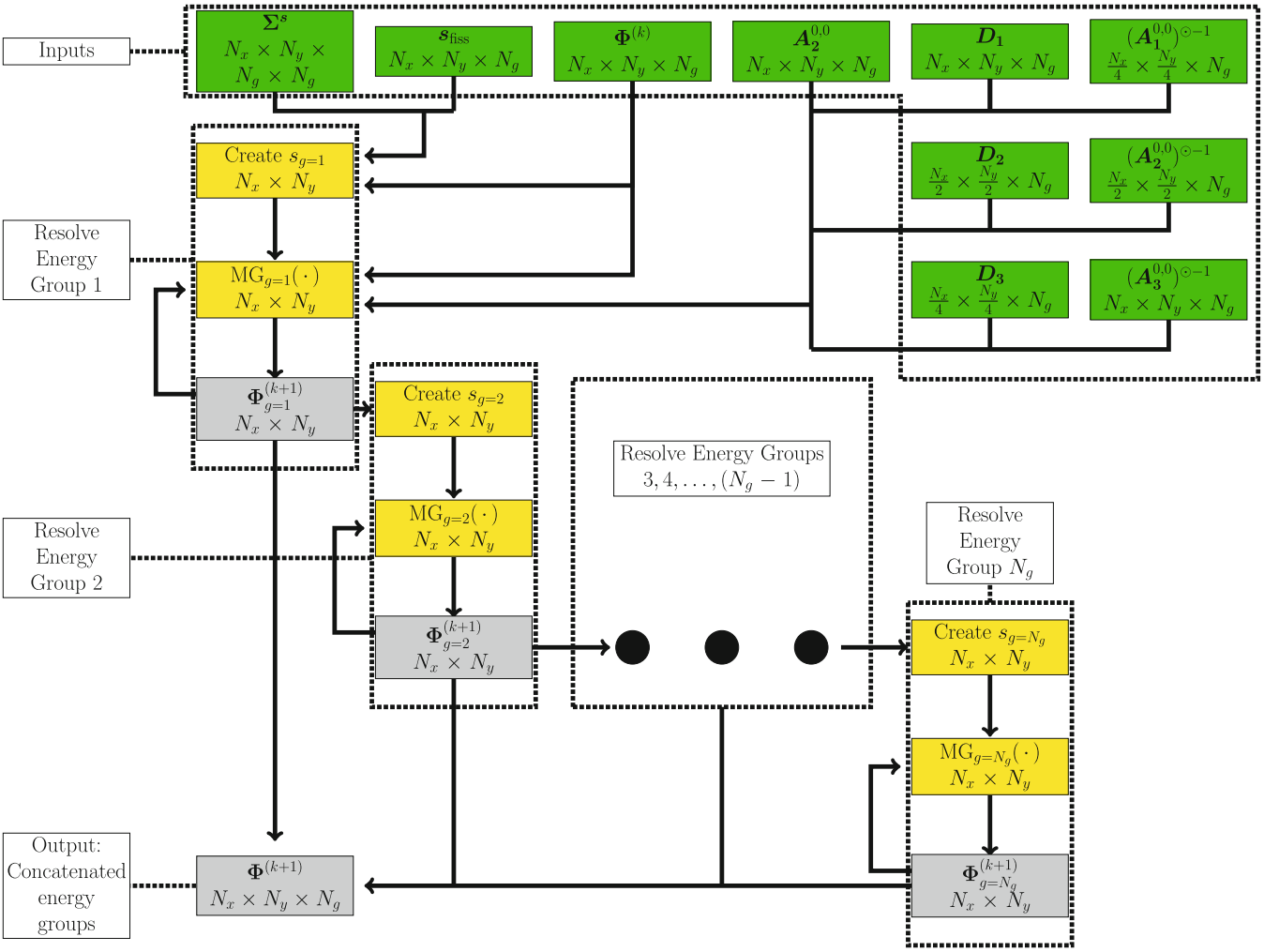


FIGURE 6 Multi-group network representing a single multi-group iteration. This network performs a single multi-group iteration on the flux of all energy groups. Takes the flux ($\boldsymbol{\Phi}_{\boldsymbol{I}}^{(k)}$), scattering cross-sections Σ_s, source fission term ($\boldsymbol{s}_{\text{fiss}}$), diffusion coefficients ($\boldsymbol{D}_{\boldsymbol{I}}$) and the strictly diagonal coefficients ($\boldsymbol{A}_{\mathbf{1}}^{0,0}$), 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 ($\boldsymbol{\Phi}_{\boldsymbol{I}}^{(k+1)}$). 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.

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

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

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

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[49] 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 Reference 40 and operates outside of the multi-group network.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

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.[50] 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. We also present a neural network solution of a discretisation based on the convolutional finite element method, ConvFEM,[25] with quadratic elements. 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[48] with the TensorFlow backend.[1]

3.1 | Fuel assembly–Geometry and configuration

The geometry of the UOX fuel assembly based on the KAIST benchmark[50] can be seen in Figure 7. It consists of a 17×17 lattice containing 264 UOX fuel 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). Each side of the fuel assembly is of length 21.42 cm meaning each computational cell measures $0.063 \times 0.063\,\text{cm}^2$. Each side of the fuel assembly has vacuum boundary conditions applied to it. The energy is discretised into seven groups.

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, B_4C cross-sections for the control rods, and H_2O cross-sections for the moderator and coolant, all taken from the KAIST benchmark.

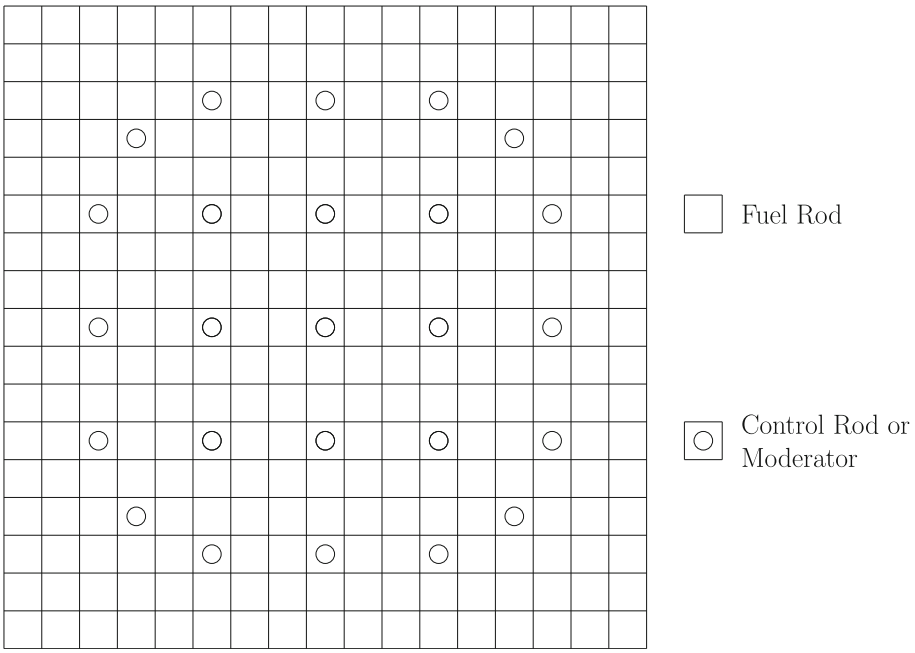


FIGURE 7 Geometry of UOX fuel assembly with laatice-cells containing fuel rods or guide tubes with either moderator or control rods.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

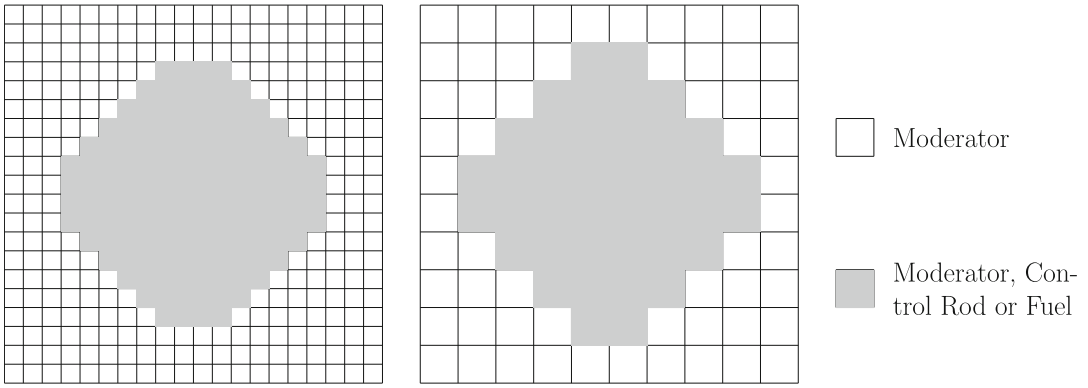


FIGURE 8 Computational grid shown here for a single lattice-cell for the fine (20×20) grid (used for the assembly calculations in Section 3.2) and the coarse (10×10) grid (used in the reactor core calculations in Section 3.6).

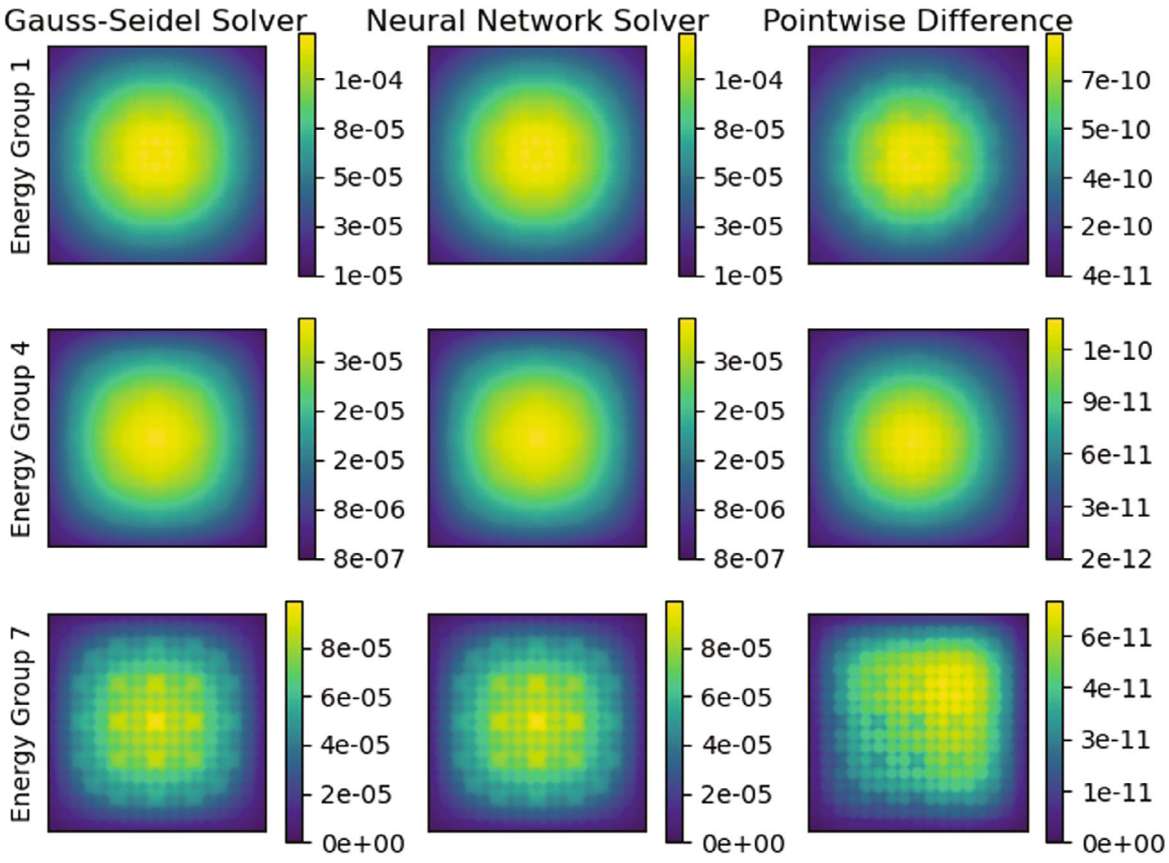


FIGURE 9 Scalar flux (neutrons $\text{cm}^{-2}s^{-1}$) across the fuel assembly for three energy groups for a fuel assembly with control rods fully withdrawn, generated using the multi-group network.

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.[50] 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}. We use a grid of $N_x = N_y = 340$, meaning there are a total of 115,600 computational cells in the lattice, of which 1364 form the boundaries (i.e., as halo

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

cells or ghost cells). This grid is used as level 1 of the multigrid. For level 2, we use $\frac{N_x}{2} = \frac{N_y}{2} = 170$, meaning there are a total of 28,900 computational cells in the lattice, of which 684 form the boundaries. At level 3, $\frac{N_x}{4} = \frac{N_y}{4} = 85$, meaning there are a total of 7225 computational cells in the lattice, of which 344 make up the boundaries. As the energy is discretised into seven groups, the fuel assembly has 818,748 degrees of freedom at level 1, 207,088 degrees of freedom at level 2 and 52,983 degrees of freedom at level 3. Three multigrid levels were used as a fourth would result in a non-integer number of cells $\left(\frac{N_x}{8} = \frac{N_y}{8} = 32.5\right)$. While additional modifications could be made to account for this, it was deemed unnecessary for this problem.

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. We speculate that the lopsided or asymmetrical difference in the group 7 fluxes seen in Figure 9 (and also Figure 10) is due to the sweeping nature of the Gauss-Seidel updating of the linear solution method of the Fortran code. This results in an accumulation of errors as one sweeps across the grid, although the actual error or difference is still small.

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.

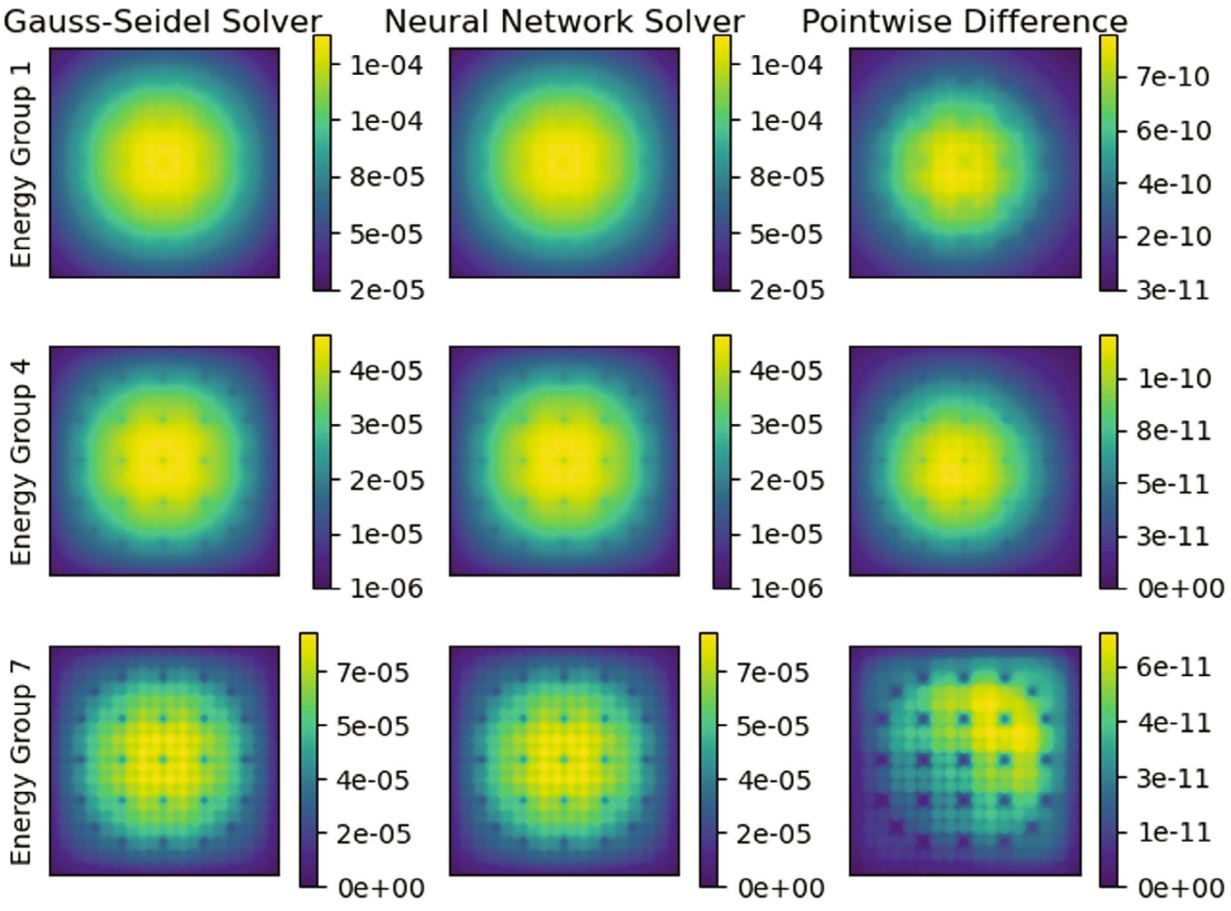


FIGURE 10 Scalar flux (neutrons $cm^{-2}s^{-1}$) across the fuel assembly for three energy groups for a fuel assembly with control rods fully inserted, generated using the multi-group network.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

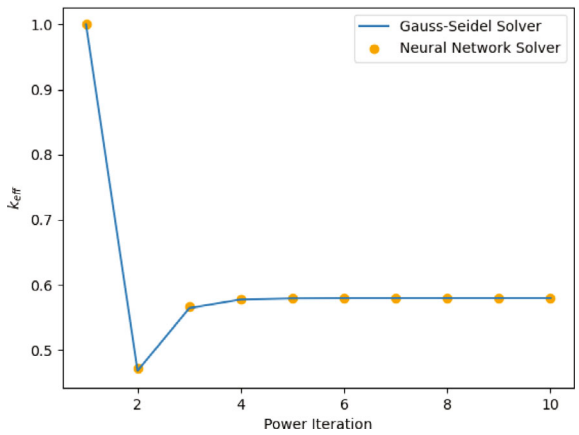
Figure 11 contains the rate of convergence of k_{eff} for a fuel assembly with control rods fully withdrawn (Figure 11A) and fully inserted (Figure 11B). It can be observed that k_{eff} is lower when control rods are inserted, as would be expected. The convergence for the solution from the neural network 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}.

3.3 | Fuel assembly–Finite element discretisation

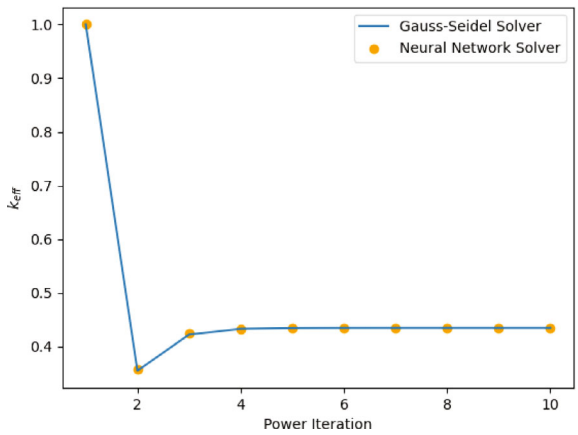
Figures 12 and 13 contain the scalar flux solution for a fuel assembly with control rods fully withdrawn and fully inserted, respectively. Both solutions were generated using quadratic convolutional finite elements (ConvFEM) implemented with a neural network. The weights used in the filters are given 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. Using quadratic ConvFEM for the two control rod configurations, yields converged values of k_{eff} that are both slightly larger than for the finite volume discretisation, see Table 1.

3.4 | Fuel assembly–Time comparisons

Table 2 shows the time comparisons for 100 Jacobi iterations performed for the fuel assembly test case. The 100 iterations were performed 400 times and we calculate the minimum, maximum and average of these times. The neural network implementation used the multi-group network (see Figure 6), but replaced the multigrid network (see Figure 5) with the Jacobi network (see Figure 2) so that it is directly comparable with the Fortran implementation. The neural network implementation was run on a single AMD EPYC 7742 CPU* and a single 24 GB NVIDIA RTX 6000 GPU. The serial



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



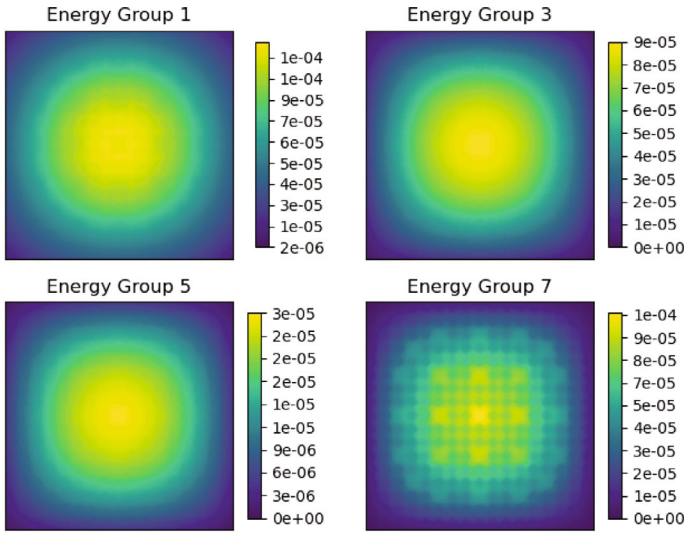
(B) Control rods fully inserted. k_{eff} converges to 0.4347 for the Neural Network Solver and 0.4347 for the Fortran implementation (with the Gauss-Seidel solver).

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.

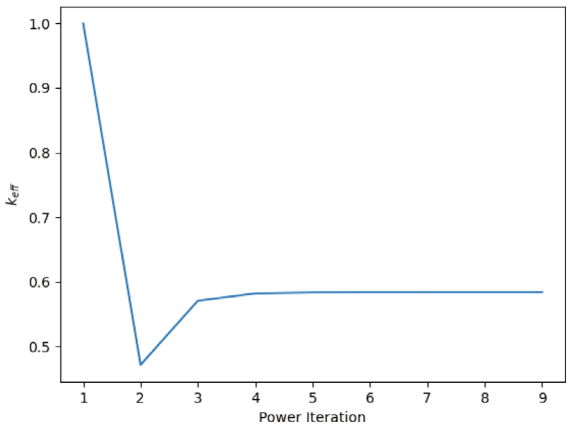
TABLE 1 Values of k_{eff} for the fuel assembly test cases.

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

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

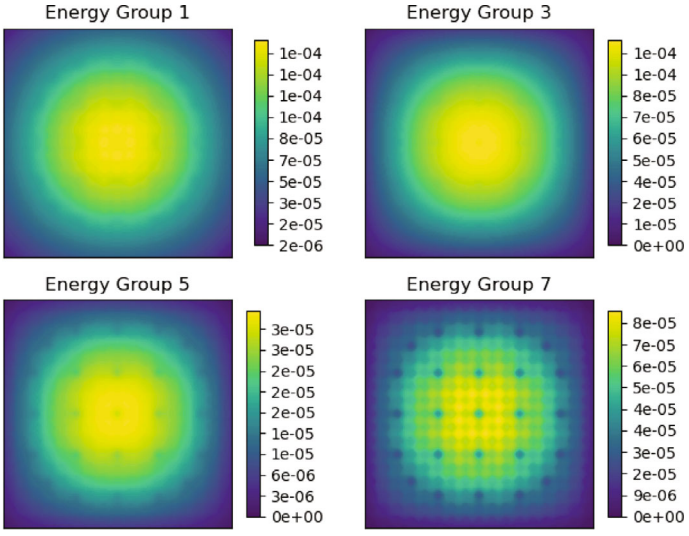


(A) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups.

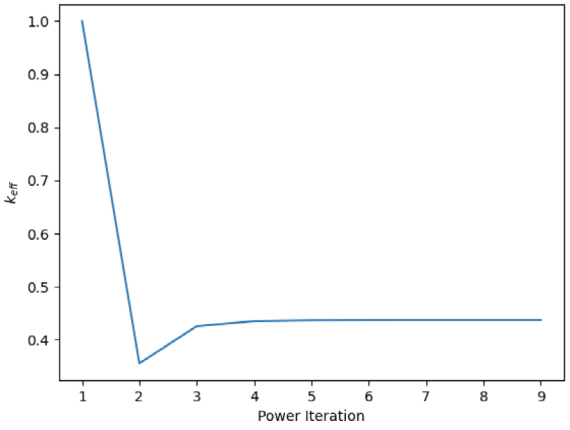


(B) Convergence of k_{eff} against power iteration. k_{eff} converges to 0.5838 compared to 0.5797 for the finite volume discretisation.

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).

Fortran implementation was run on a single CPU of the same type as above. From Table 2, it can be observed that the neural network implementation on the GPU is the fastest, resulting in an average time that is approximately four times faster than the Fortran implementation on the CPU, which is five times faster than the neural network implementation run on a single CPU. Part of the reason as to why the neural network implementation on the CPU is slowest, is that the convolutional filter requires nine operations to be performed when only four are required for the finite volume discretisation (and therefore only four operations are performed in the Fortran code, see Equation 25). Iterations run on the CPU showed a higher variability in their timings than on the GPU, which can be seen in the standard deviations shown in Table 2.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 Time comparisons for 100 Jacobi iterations performed on the fuel assembly test case using a CPU and a GPU for the neural network solver and a CPU for the serial Fortran code.

		Computational times (s)			
Implementation	**Hardware**	**Min**	**Max**	**Average**	**Standard deviation**
Neural network	AMD EPYC 7742 CPU	21.2420	23.0204	22.3475	0.5084
Neural network	NVIDIA RTX 6000 GPU	1.1720	1.3999	1.1847	0.0309
Fortran	AMD EPYC 7742 CPU	4.2615	4.8998	4.5892	0.2748

Note: The 100 iterations were performed 400 times so the minimum, maximum and average times, and standard deviations are shown.

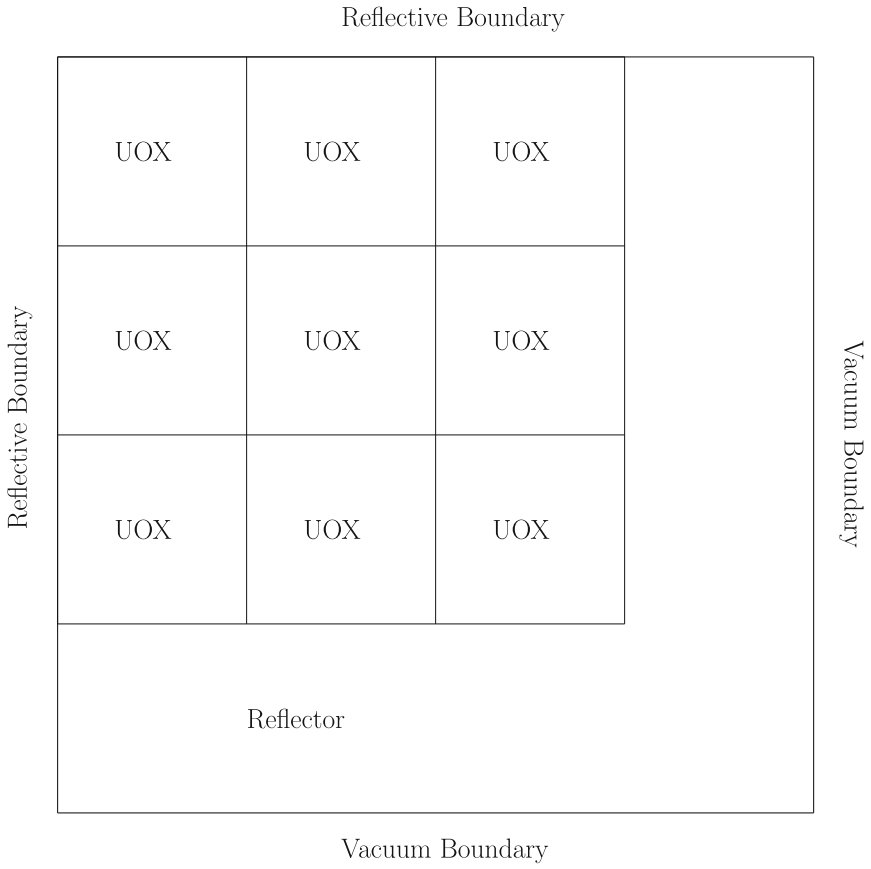


FIGURE 14 Geometry of reactor core for a simplified version of the KAIST benchmark.[50]

3.5 | Reactor core–Geometry and configuration

We now model a reactor core using the cross-sections from the KAIST benchmark.[50] Unlike the benchmark, our core is a 3×3 grid of fuel assemblies of type UOX only. The material parameters required are UOX cross-sections for the fuel rods, B_4C cross-sections for the control rods, and H_2O cross-sections for the moderator, coolant and reflector, all taken from the KAIST benchmark. 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 2724 cells are used as halo cells needed to apply the boundary conditions. Again, the energy was discretised into seven groups.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

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.

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, giving 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. We use the values $N_x = N_y = 680$, meaning there are a total of 462,400 computational cells in the lattice, of which 2724 of these form the boundaries (i.e., as halo cells or ghost cells). For level 2 of the multigrid, $\frac{N_x}{2} = \frac{N_y}{2} = 340$, meaning there are a total of 115,600 computational cells in the lattice, of which 1364 form the boundaries. At level 3, $\frac{N_x}{4} = \frac{N_y}{4} = 170$, meaning there are a total of 28,900 computational cells in the lattice, of which 684 form the boundaries. The energy was again discretised into seven groups resulting in 3,255,868 degrees of freedom at level 1, 818,748 degrees of freedom at level 2, and 207,088 degrees of freedom at level 3.

Figure 16 contains the flux profiles for four energy groups for reactor configuration one. It can be observed that the 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 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 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.

The underlying physics of this reactor core problem is such that the fast neutrons, from the highest energy group (energy group 1), are thermalised as they propagate into the reflector. This results in an attenuation of the fluxes of the highest energy group and a peak in the thermal fluxes (energy group 7) within the reflector and near the fuel assemblies. This effect is seen in particular near the fuel assemblies that have the control rods withdrawn, as neutron production is

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.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

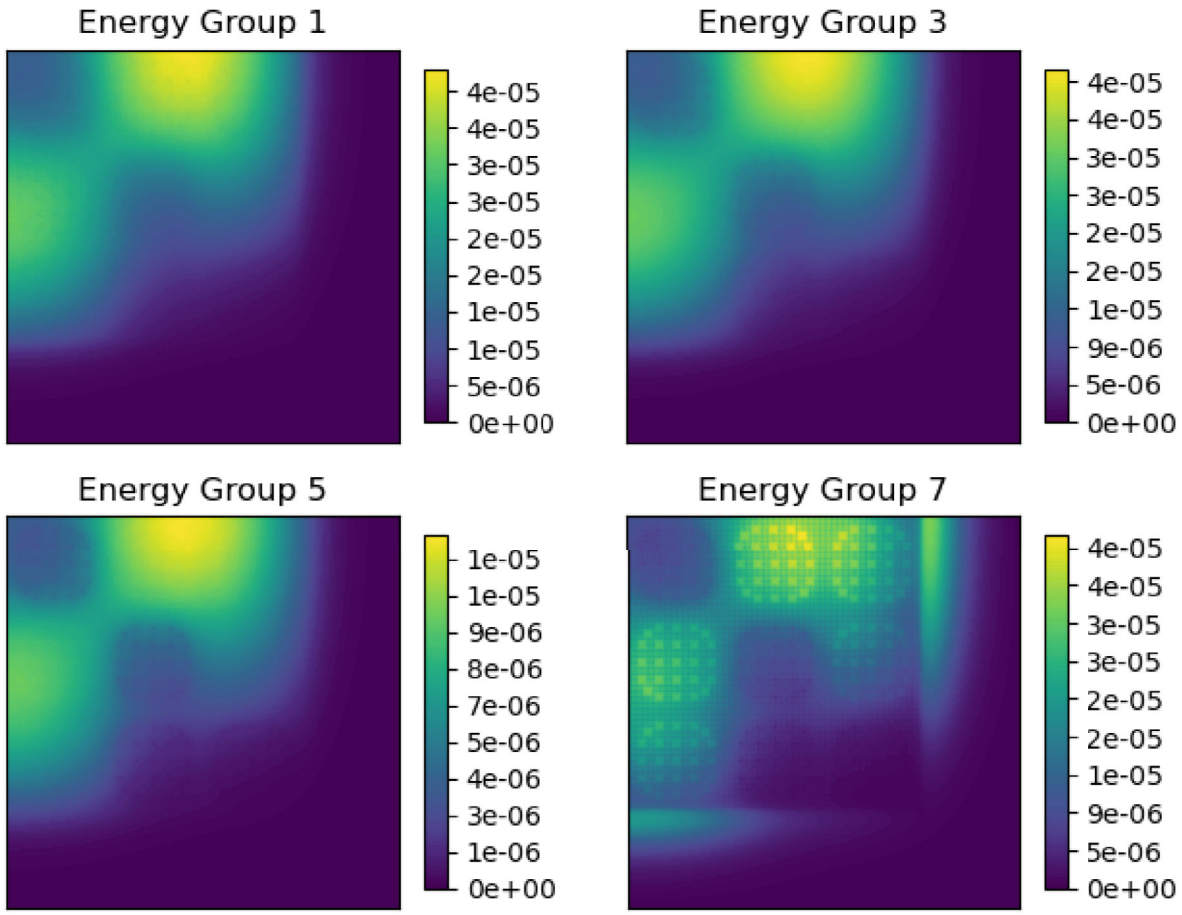


FIGURE 16 Scalar flux (neutrons $cm^{-2}s^{-1}$) across the fuel assembly for four energy groups for reactor configuration one, generated using the multi-group network with the finite volume discretisation.

highest here. It should also be noted that there is a reduction in the fluxes from every energy group within those fuel assemblies that have the fuel rods inserted as could be expected—see fuel rod positions in Figure 15.

We further remark that, for these benchmarks, we observed no negative fluxes (i.e., unphysical solutions). If desired, it would be possible to incorporate a maximum principle to stop the occurrence of any negative fluxes.

More challenging problems than those presented in this section (e.g., with more cells or a higher number of energy groups) will result in more degrees of freedom, which could be an issue for GPUs and AI processors. For very large multi-group diffusion problems, one could simply store a spatial representation of the solution on the coarse grid, which would enable one to form rapidly the fine grid solution as and when needed, after which the solution on the fine grid could be discarded. This is a particularly pressing issue for implementations of the Boltzmann transport equation, which is memory hungry due to the discretisation in angle.

3.7 | Reactor core–Time comparisons

Table 3 shows the time comparisons for 100 Jacobi iterations performed on the reactor core test case. The neural network implementation was tested on the CPU and GPU (details of hardware given in see Section 3.4 and in Table 3), and the serial Fortran implementation was tested on the CPU. The 100 Jacobi iterations were performed 100 times. The reactor core test case has about four times the number of degrees of freedom as for the fuel assembly test case, which is reflected in the timings being four times larger for the reactor core than for the assembly. Again, it can be observed that the neural network implementation on the GPU is the fastest, being about four times faster than the Fortran implementation on the CPU, which is five times faster than the neural network implementation run on a single CPU. The standard deviations for the implementations run on the CPU are higher than those for the GPU, as was also observed for the assembly test case. Although, there is actually very little variation in the multiple realisations of the Fortran code (run on the CPU) and the neural network run on the GPU.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

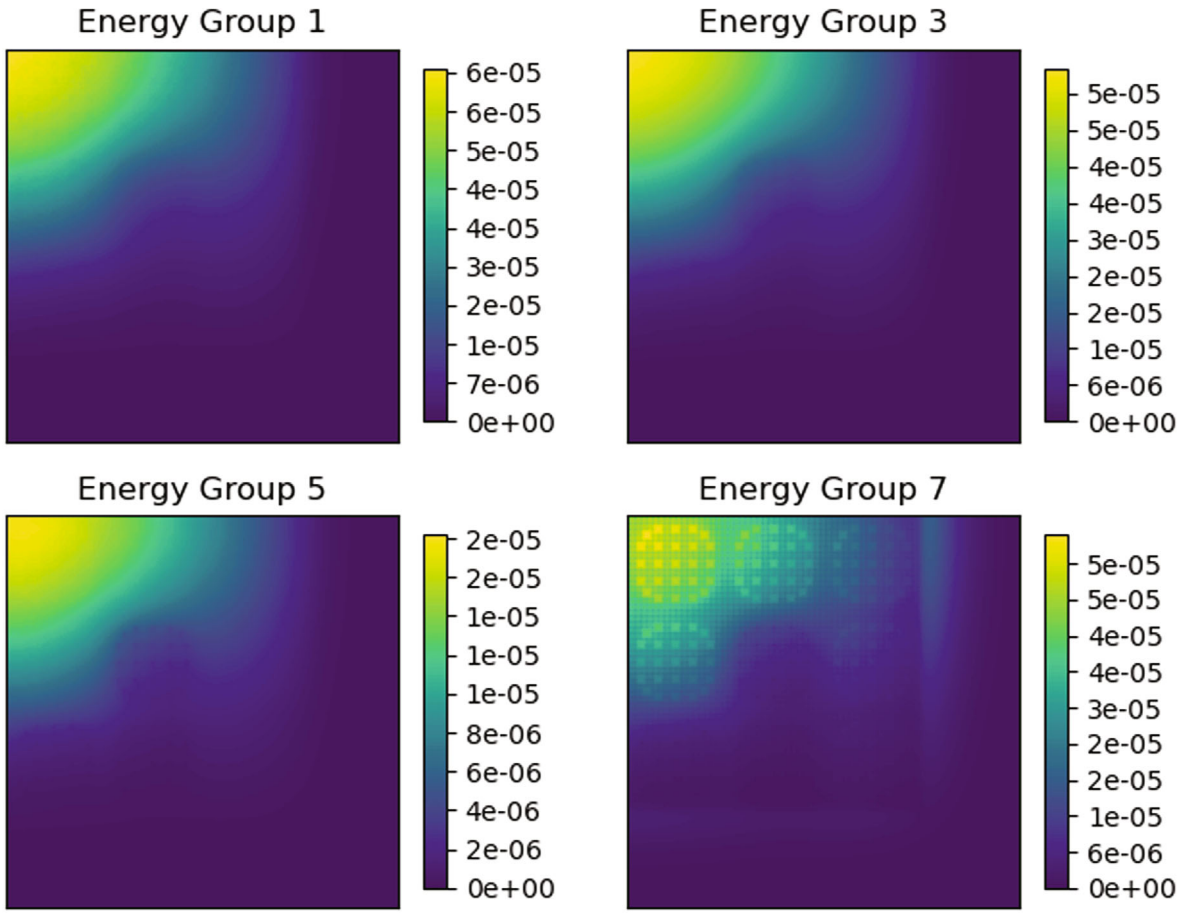
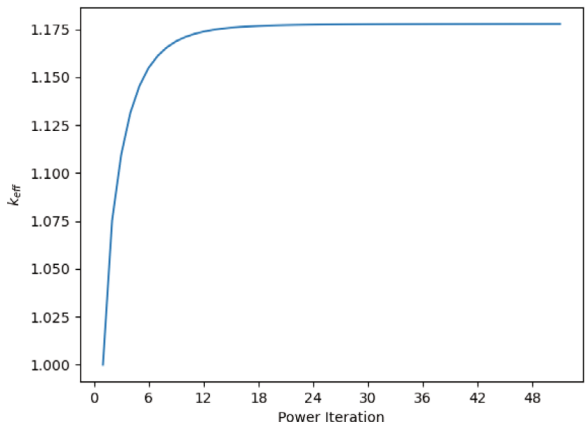
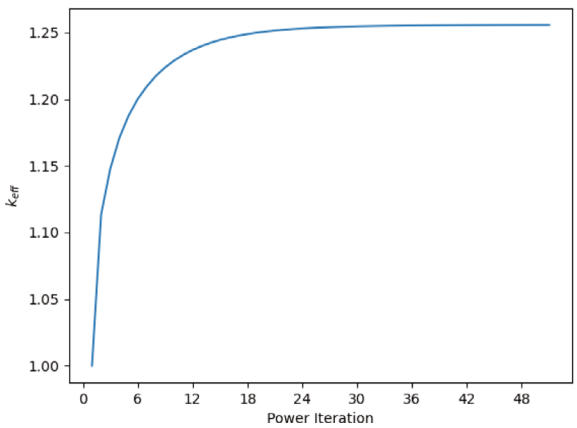


FIGURE 17 Scalar flux (neutrons $cm^{-2}s^{-1}$) across the fuel assembly for four energy groups for reactor configuration two, generated using the multi-group network for the finite volume discretisation.



(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} versus power iteration using the multi-group network with a finite volume discretisation for both configurations.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 3 Time comparisons for 100 Jacobi iterations performed on the reactor core test case using a CPU and a GPU for the neural network solver and a CPU for the serial Fortran code.

		Computational times (s)			
Implementation	**Hardware**	**Min**	**Max**	**Average**	**Standard deviation**
Neural network	AMD EPYC 7742 CPU	88.8473	100.6071	94.5058	5.0606
Neural network	NVIDIA RTX 6000 GPU	4.7403	4.8235	4.7901	0.0167
Fortran	AMD EPYC 7742 CPU	16.5012	16.8404	16.6222	0.0897

Note: The 100 iterations were performed 100 times so the minimum, maximum average times and standard deviations are shown.

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 and a quadratic convolutional finite element 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}. Timings show that, for this particular problem, running the neural network implementation on a GPU is about four times faster than running the Fortran implementation on a CPU, which is about five times faster than running the neural network implementation on a CPU. This test case is also used to demonstrate how a quadratic finite element discretisation may be used in the convolutional layers. The approach is also extended to a more computationally demanding problem, that of a reactor core with about four times the number of degrees of freedom of the fuel assembly benchmark. It is demonstrated that the ratio of the computational timings are similar to those obtained for the first benchmark and the absolute values are about four times higher, as expected.

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 some computer architectures, for example Nvidia GPUs, and in the future there may be a general AI library that is optimised for a majority of commercially available hardware. This flexibility brings within easy reach the ability to run code on multiple platforms without the need for any 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 energy-efficient 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 so that large problems could be run on GPUs and AI processors. Two issues to address would be the bottleneck caused by the coarsest grid level in the multigrid algorithm and the tighter memory restrictions of GPUs compared to CPUs.

AUTHOR CONTRIBUTIONS

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.

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ACKNOWLEDGMENTS

The authors would like to acknowledge the EPSRC Centre for Doctoral Training in Nuclear Energy (Imperial College, Cambridge University and the Open University, EP/L015900/1) and the following EPSRC grants: the PREMIERE programme grant, AI to enhance manufacturing, energy, and healthcare (EP/T000414/1); RELIANT, Risk EvaLuatIon fAst iNtelligent Tool for COVID19 (EP/V036777/1); CO-TRACE, COvid-19 Transmission Risk Assessment Case Studies—Education Establishments (EP/W001411/1); INHALE, Health assessment across biological length scales (EP/T003189/1); MAGIC (EP/N010221/1); and MUFFINS (EP/P033180/1). The authors are grateful for use of the computing resources of Imperial College's Research Computing Service (http://doi.org/10.14469/hpc/2232). We would also like to thank the reviewers for their thorough and insightful comments, which have substantially improved this article.
The authors state that, for the purpose of open access, a Creative Commons Attribution (CC BY) license will be applied to any Author Accepted Manuscript version relating to this article.

DATA AVAILABILITY STATEMENT

The methods described in the current paper have also been applied to problems in radiation transport (without fission).[25] The code pertaining to the latter is available from the following github repository https://github.com/trfphillips/Neural-Network-Transport-Solver.

ENDNOTE

*In order to run on a single CPU, the number of CPUs requested had to be set, both in the instructions to the HPC and in the TensorFlow code.

ORCID

Toby R.F. Phillips https://orcid.org/0000-0003-0347-7114
Claire E. Heaney https://orcid.org/0000-0002-6555-1423
Boyang Chen https://orcid.org/0000-0002-9091-0352
Andrew G. Buchan https://orcid.org/0000-0002-0436-267X
Christopher C. Pain https://orcid.org/0000-0003-4194-2590

REFERENCES

1. Abadi M, Agarwal P, Aand Barham E, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015 Software available from http://www.tensorflow.org
2. Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style. *High-Performance Deep Learning Library, in: Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.; 2019:8024-8035 http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
3. Vermeire B, Witherden F, Vincent P. On the utility of GPU accelerated high-order methods for unsteady flow simulations: a comparison with industry-standard tools. *J Comput Phys*. 2017;334:497-521. doi:10.1016/j.jcp.2016.12.049
4. Bergmann RM, Rowland KL, Radnović N, Slaybaugh RN, Vujić JL. Performance and accuracy of criticality calculations performed using WARP—a framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs. *Ann Nuclear Energy*. 2017;103:334-349. doi:10.1016/j.anucene.2017.01.027
5. Delbosc N, Summers JL, Khan AI, Kapur N, Noakes CJ. Optimized implementation of the lattice Boltzmann method on a graphics processing unit towards real-time fluid simulation. *Comput Math Appl*. 2014;67:462-475. doi:10.1016/j.camwa.2013.10.002
6. Chan J, Wang Z, Modave A, Remacle J-F, Warburton T. GPU-accelerated discontinuous Galerkin methods on hybrid meshes. *J Comput Phys*. 2016;318:142-168. doi:10.1016/j.jcp.2016.04.003
7. King M-F, Khan A, Delbosc N, et al. Modelling urban airflow and natural ventilation using a GPU-based lattice-Boltzmann method. *Build Environ*. 2017;125:273-284. doi:10.1016/j.buildenv.2017.08.048
8. Slaybaugh RN, Ramirez-Zweiger M, Pandya T, Hamilton S, Evans TM. Eigenvalue solvers for modeling nuclear reactors on leadership class machines. *Nuclear Sci Eng*. 2018;190:31-44. doi:10.1080/00295639.2017.1413875
9. Cecka C, Lew AJ, Darve E. Assembly of finite element methods on graphics processors. *Int J Numer Methods Eng*. 2011;85:640-669. doi:10.1002/nme.2989
10. Mossaiby F, Rossi R, Dadvand P, Idelsohn S. OpenCL-based implementation of an unstructured edge-based finite element convection-diffusion solver on graphics hardware. *Int J Numer Methods Eng*. 2012;89:1635-1651. doi:10.1002/nme.3302
11. Dziekonski A, Sypek P, Lamecki A, Mrozowski M. Generation of large finite-element matrices on multiple graphics processors. *Int J Numer Methods Eng*. 2013;94:204-220. doi:10.1002/nme.4452
12. Sanfui S, Sharma D. A three-stage graphics processing unit-based finite element analyses matrix generation strategy for unstructured meshes. *Int J Numer Methods Eng*. 2020;121:3824-3848. doi:10.1002/nme.6383
13. Modave A, Atle A, Chan J, Warburton T. A GPU-accelerated nodal discontinuous Galerkin method with high-order absorbing boundary conditions and corner/edge compatibility. *Int J Numer Methods Eng*. 2017;112:1659-1686. doi:10.1002/nme.5576

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

14. Lewis AGM, Beall J, Ganahl M, Hauru M, Mallick SB, Vidal G. Large-scale distributed linear algebra with tensor processing units. *Proc Natl Acad Sci USA*. 2022;119:e2122762119. doi:10.1073/pnas.2122762119
15. Graphcore. Intelligence Processing Units. 2022. https://www.graphcore.ai/products/ipu
16. Cerebras. CS-2: A Revolution in AI Infrastructure. 2022. https://www.cerebras.net/product-system/
17. Lu T, Marin T, Zhuo Y, Chen Y-F, Ma C. Accelerating MRI reconstruction on TPUs. Paper presented at: IEEE High Performance Extreme Computing Conference (HPEC). 2020;2020:1-9. doi:10.1109/HPEC43674.2020.9286192
18. Lu T, Marin T, Zhuo Y, Chen Y-F, Ma C. Nonuniform fast fourier transform on TPUs. Paper presented at: 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI). 2021:783–787. doi:10.1109/ISBI48211.2021.9434068
19. Belletti F, King D, Yang K, et al. Tensor processing units for financial Monte Carlo. Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing. 2020:12–23. doi:10.1137/1.9781611976137.2
20. Morningstar A, Hauru M, Beall J, et al. Simulation of quantum many-body dynamics with tensor processing units: floquet prethermalization. *PRX Quant*. 2022;3:020331. doi:10.1103/PRXQuantum.3.020331
21. Pederson R, Kozlowski J, Song R, et al. Large scale quantum chemistry with tensor processing units. *J Chem Theory Comput*. 2022;19:25-32. doi:10.1021/acs.jctc.2c00876
22. Zhao X-Z, Xu T-Y, Ye Z-T, Liu W-J. A TensorFlow-based new high-performance computational framework for CFD. *J Hydrodyn*. 2020;32:735-746. doi:10.1007/s42241-020-0050-0
23. Wang Q, Ihme M, Chen Y-F, Anderson J. A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units. *Comput Phys Commun*. 2022;274:108292. doi:10.1016/j.cpc.2022.108292
24. Chen B, Heaney CE, Pain CC. Using AI libraries for Incompressible Computational Fluid Dynamics. 2023.
25. Phillips TRF, Heaney CE, Chen B, Buchan AG, Pain CC. Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport, arXiv preprint 2301.09991. 2023. doi:10.48550/arXiv.2301.09991
26. Ronneberger O, Fischer P, Brox T. U-net: convolutional networks for biomedical image segmentation. Paper presented at: Medical Image Computing and Computer-Assisted Intervention (MICCAI), Volume 9351 of LNCS, Springer. 2015:234–241. doi:10.48550/arXiv.1505.04597
27. Ke T-W, Maire M, Yu SX. Multigrid neural architectures. Paper presented at: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017:4067–4075. doi:10.1109/CVPR.2017.433
28. He J, Xu J. MgNet: a unified framework of multigrid and convolutional neural network. *Sci China Math*. 2019;62:1331-1354. doi:10.1007/s11425-019-9547-2
29. Thuerey N, Weißenow K, Prantl L, Hu X. Deep learning methods for Reynolds-averaged Navier-Stokes simulations of airfoil flows. *AIAA J*. 2020;58:25-36. doi:10.2514/1.J058291
30. Le QT, Ooi C. Surrogate modeling of fluid dynamics with a multigrid inspired neural network architecture. *Mach Learn Appl*. 2021;6:100176. doi:10.1016/j.mlwa.2021.100176
31. Margenberg N, Hartmann D, Lessig C, Richter T. A neural network multigrid solver for the Navier-Stokes equations. *J Comput Phys*. 2022;460:110983. doi:10.1016/j.jcp.2022.110983
32. Cervi E, Lu X, Cammi A, Di Maio F, Zio E. Sensitivity-analysis-driven surrogate model for molten salt reactors control. *J Nuclear Eng*. 2022;3:277-294.
33. Kabir HD, Khosravi A, Hosen MA, Nahavandi S. Neural network-based uncertainty quantification: a survey of methodologies and applications. *IEEE Access*. 2018;6:36218-36234.
34. Tripathy RK, Bilionis I. Deep UQ: learning deep neural network surrogate models for high dimensional uncertainty quantification. *J Comput Phys*. 2018;375:565-588. doi:10.1016/j.jcp.2018.08.036
35. Gong H, Cheng S, Chen Z, Li Q. Data-enabled physics-informed machine learning for reduced-order Modeling digital twin: application to nuclear reactor physics. *Nuclear Sci Eng*. 2022;196:668-693. doi:10.1080/00295639.2021.2014752
36. Schiassi E, De Florio M, Ganapol BD, Picca P, Furfaro R. Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics. *Ann Nuclear Energy*. 2022;167:108833. doi:10.1016/j.anucene.2021.108833
37. Wang J, Peng X, Chen Z, Zhou B, Zhou Y, Zhou N. Surrogate modeling for neutron diffusion problems based on conservative physics-informed neural networks with boundary conditions enforcement. *Ann Nuclear Energy*. 2022;176:109234. doi:10.1016/j.anucene.2022.109234
38. Huhn QA, Tano ME, Ragusa JC. Physics-informed neural network with fourier features for radiation transport in heterogeneous media. *Nuclear Sci Eng*. 2023;1-14. doi:10.1080/00295639.2023.2184194
39. Foad B, Elzohery R, Novog DR. Demonstration of combined reduced order model and deep neural network for emulation of a time-dependent reactor transient. *Ann Nuclear Energy*. 2022;171:109017. doi:10.1016/j.anucene.2022.109017
40. Phillips TRF, Heaney CE, Smith PN, Pain CC. An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. *Int J Numer Methods Eng*. 2021;122:3780-3811. doi:10.1002/nme.6681
41. Qin S, Zhang Q, Zhang J, et al. Application of deep neural network for generating resonance self-shielded cross-section. *Ann Nuclear Energy*. 2020;149:107785. doi:10.1016/j.anucene.2020.107785
42. Beck A, Kurz M. A perspective on machine learning methods in turbulence modeling. *GAMM-Mitteilungen*. 2021;44:e202100002. doi:10.1002/gamm.202100002
43. Hébert A. *Multigroup Neutron Transport and Diffusion Computations*. Springer US; 2010:751-911. doi:10.1007/978-0-387-98149-9_8
44. Eymard R, Gallouët T, Herbin R. Finite volume methods. *Techniques of Scientific Computing (Part 3), Volume 7 of Handbook of Numerical Analysis*. Elsevier; 2000:713-1018. doi:10.1016/S1570-8659(00)07005-8

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

45. Langtangen HP. *Finite Difference Computing with Exponential Decay Models*. SpringerOpen; 2016.
46. Acton FS. *Numerical Methods that Usually Work*. Mathematical Association of America; 1990.
47. Reams R. Hadamard inverses, square roots and products of almost semi-definite matrices. *Linear Algebra Appl*. 1999;288:35-43. doi:10.1016/S0024-3795(98)10162-3
48. Chollet F. Keras. 2015. https://keras.io
49. Golub GH, Loan CF. *Matrix Computations*. John Hopkins University Press; 1996.
50. Cho Z. Kaist Benchmark Problem 2A: MOX Fuel-Loaded Small PWR Core. 2000. http://nurapt.kaist.ac.kr/benchmark/kaist_ben1a.pdf

How to cite this article: Phillips TR, Heaney CE, Chen B, Buchan AG, Pain CC. Solving the discretised neutron diffusion equations using neural networks. *Int J Numer Methods Eng*. 2023;124(21):4659-4686. doi: 10.1002/nme.7321

APPENDIX A. FINITE VOLUME DISCRETISATION OF THE NEUTRON DIFFUSION EQUATION

The multi-group steady-state diffusion equation for criticality[43] can be written as:

$$-\nabla\cdot\left(D_g\nabla\phi_g\right)+\Sigma_g^a\phi_g+\sum_{\substack{g'=1\\g'\neq g}}^{N_g}\Sigma^s_{g\to g'}\phi_g \;=\sum_{\substack{g'=1\\g'\neq g}}^{N_g}\Sigma^s_{g'\to g}\phi_{g'}+\lambda\chi_g\sum_{g'=1}^{N_g}\nu_{g'}\Sigma^f_{g'}\phi_{g'},\qquad \forall g\in\{1,2,\ldots,N_g\}. \tag{A1}$$

To apply the finite volume method[44] to Equation (A1), we multiply by a test or characteristic function $M_{i,j}$ and integrate over the volume associated with cell $V_{i,j}$. The following derivation is for a two-dimensional discretisation, however, the result can be easily extended to the three-dimensional case. Considering the first term in Equation (A1), we apply the divergence theorem to obtain a surface integral:

$$-\int_{V_{i,j}}M_{i,j}\nabla\cdot\left(D_g\nabla\phi_g\right)dV=-\int_{\Gamma_{i,j}}M_{i,j}\,\boldsymbol{n}\cdot\left(D_g\nabla\phi_g\right)d\Gamma \tag{A2}$$

where $\Gamma_{i,j}$ is the surface of cell i,j and $\boldsymbol{n}$ is the outward normal of the surface. The dot product in Equation (A2) can be expanded as

$$-\boldsymbol{n}\cdot\left(D_g\nabla\phi_g\right)=-D_g\begin{pmatrix}n_x\\n_y\end{pmatrix}\cdot\begin{pmatrix}\frac{\partial\phi}{\partial x}\\\frac{\partial\phi}{\partial y}\end{pmatrix}. \tag{A3}$$

By considering the four surfaces that make up $\Gamma_{i,j}=\Gamma_e+\Gamma_n+\Gamma_w+\Gamma_s$, as shown in Figure A1, we can rewrite Equation (A2) as

$$\begin{aligned}-\int_{\Gamma_e}M_{ij}\left(D_g\frac{\partial\phi_g}{\partial x}\right)\Big|_{i+\frac{1}{2},j}d\Gamma-\int_{\Gamma_n}M_{ij}\left(D_g\frac{\partial\phi_g}{\partial y}\right)\Big|_{i,j+\frac{1}{2}}d\Gamma\\+\int_{\Gamma_w}M_{ij}\left(D_g\frac{\partial\phi_g}{\partial x}\right)\Big|_{i-\frac{1}{2},j}d\Gamma+\int_{\Gamma_s}M_{ij}\left(D_g\frac{\partial\phi_g}{\partial y}\right)\Big|_{i,j-\frac{1}{2}}d\Gamma.\end{aligned} \tag{A4}$$

Values on the cell boundaries are written as $i\pm\frac{1}{2},j$ and $i,j\pm\frac{1}{2}$, see Figure A1. We use a first order approximation for the derivatives at the cell boundaries (based on Taylor series) and an average to approximate the diffusion coefficient at cell boundaries, for example:

$$\frac{\partial\phi_g}{\partial x}\Big|_{i+\frac{1}{2},j}\approx\frac{\phi_{i+1,j,g}-\phi_{i,j,g}}{\Delta x},\qquad D_{i+\frac{1}{2},j,g}\approx\frac{D_{i+1,j,g}+D_{i,j,g}}{2}. \tag{A5}$$

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

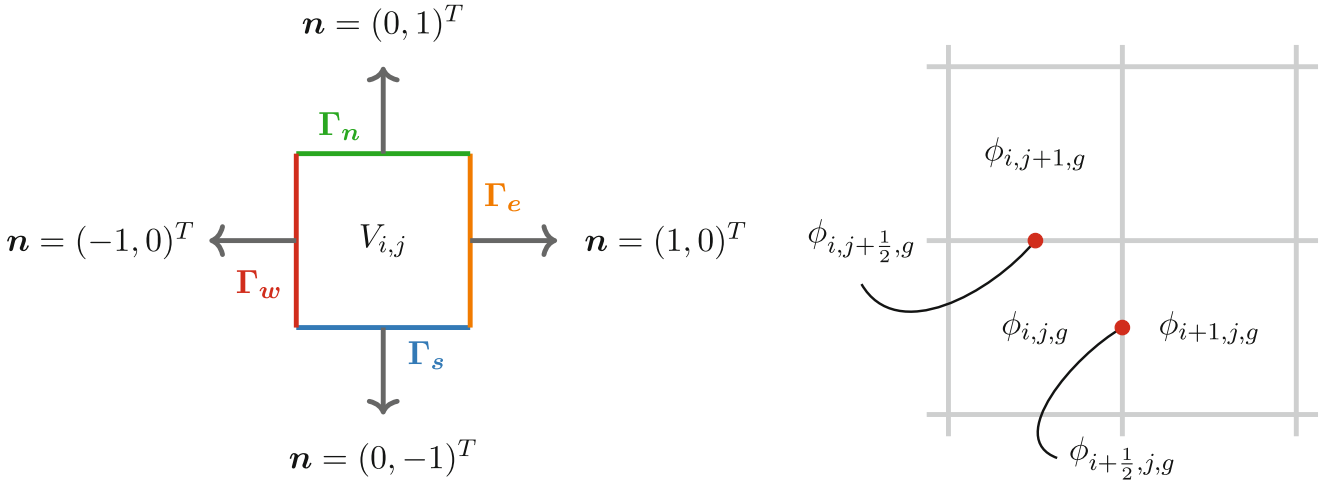


FIGURE A1 Left: The surfaces and outward pointing normals of cell i,j. Right: Four neighbouring cells and two values on the cell boundaries.

For the particular discretisation used in this paper, we take test functions with value one in the cell of interest (and zero elsewhere). Substituting Equations (A5) into (A4), we obtain

$$-\left(\frac{D_{i+1,j,g}+D_{i,j,g}}{2}\right)\left(\frac{\phi_{i+1,j,g}-\phi_{i,j,g}}{\Delta x}\right)\Delta y-\left(\frac{D_{i,j+1,g}+D_{i,j,g}}{2}\right)\left(\frac{\phi_{i,j+1,g}-\phi_{i,j,g}}{\Delta y}\right)\Delta x$$
$$+\left(\frac{D_{i,j,g}+D_{i-1,j,g}}{2}\right)\left(\frac{\phi_{i,j,g}-\phi_{i-1,j,g}}{\Delta x}\right)\Delta y+\left(\frac{D_{i,j,g}+D_{i,j-1,g}}{2}\right)\left(\frac{\phi_{i,j,g}-\phi_{i,j-1,g}}{\Delta y}\right)\Delta x. \tag{A6}$$

Considering the second term in Equation (A1), we again multiply by a test function $M_{i,j}$ and integrate over the volume of the associated cell i,j giving

$$\int_{V_{i,j}} M_{i,j}\left(\Sigma_g^a\phi_g\right)\Big|_{i,j}\,dV=\Sigma_{i,j,g}^a\phi_{i,j,g}\Delta x\Delta y. \tag{A7}$$

The remaining terms in Equation (A1) can be evaluated similarly to the second term, as shown above. Upon rearranging the terms that represent the discretised differential operator, shown in Equation (A6), adding to this the discretisation of the other terms, and factoring out $\Delta x\Delta y$, the discretised neutron diffusion equation is obtained and can be seen in Equation (6).

APPENDIX B. DIFFUSION OPERATOR

For two scalars ϕ_g and D_g, the following is true

$$\nabla^2(D_g\phi_g)=\nabla\cdot\nabla(D_g\phi_g) \tag{B1}$$
$$=\nabla\cdot\left(\phi_g\nabla D_g+D_g\nabla\phi_g\right) \tag{B2}$$
$$=\phi_g\nabla^2D_g+D_g\nabla^2\phi_g+2\nabla D_g\cdot\nabla\phi_g, \tag{B3}$$

which leads to the following identity

$$2\nabla D_g\cdot\nabla\phi_g=\nabla^2(D_g\phi_g)-\phi_g\nabla^2D_g-D_g\nabla^2\phi_g. \tag{B4}$$

Expanding out the diffusion term in Equation (1) and then substituting in the expression from Equation (B4) results in

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

$$\nabla \cdot (D_g \nabla \phi_g) = D_g \nabla^2 \phi_g + \nabla \phi_g \cdot \nabla D_g \tag{B5}$$

$$= D_g \nabla^2 \phi_g + \frac{1}{2}\left(\nabla^2 (D_g \phi_g) - \phi_g \nabla^2 D_g - D_g \nabla^2 \phi_g\right) \tag{B6}$$

$$= \frac{1}{2}\left(\nabla^2 (D_g \phi_g) - \phi_g \nabla^2 D_g + D_g \nabla^2 \phi_g\right). \tag{B7}$$

This alternative way of writing the diffusion operator seen in Equation (B7) is used in Equation (21).

APPENDIX C. 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

$$\boldsymbol{A} \odot \boldsymbol{B}\Big|_{k\ell} = A_{k\ell} B_{k\ell} \quad \forall k \in \{1, 2, \ldots, N\}, \ell \in \{1, 2, \ldots, M\} \tag{C1}$$

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

$$\sum_{\text{entries}} \boldsymbol{A} \equiv \sum_{k=1}^{M} \sum_{\ell=1}^{N} A_{k\ell}. \tag{C2}$$

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:

$$\boldsymbol{f}(\boldsymbol{\Phi_g}; \boldsymbol{w})\Big|_{i,j} = \sum_{\text{entries}} \begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix} \odot \begin{bmatrix} \phi_{i-1,j+1,g} & \phi_{i,j+1,g} & \phi_{i+1,j+1,g} \\ \phi_{i-1,j,g} & \phi_{i,j,g} & \phi_{i+1,j,g} \\ \phi_{i-1,j-1,g} & \phi_{i,j-1,g} & \phi_{i+1,j-1,g} \end{bmatrix} \tag{C3}$$

$$= \frac{-\left(\phi_{i-1,j,g} + \phi_{i+1,j,g}\right)}{\Delta x^2} + \frac{-\left(\phi_{i,j-1,g} + \phi_{i,j+1,g}\right)}{\Delta y^2} + \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right)\phi_{i,j,g}. \tag{C3}$$

Now, considering the second term in its entirety,

$$\left(\boldsymbol{D_g} \odot \boldsymbol{f}(\boldsymbol{\Phi_g}; \boldsymbol{w})\right)\Big|_{i,j} = \frac{-D_{i,j,g}\left(\phi_{i-1,j,g} + \phi_{i+1,j,g}\right)}{\Delta x^2} + \frac{-D_{i,j,g}\left(\phi_{i,j-1,g} + \phi_{i,j+1,g}\right)}{\Delta y^2} + \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}. \tag{C5}$$

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

$$\left(\boldsymbol{\Phi_g} \odot \boldsymbol{f}(\boldsymbol{D_g}; \boldsymbol{w})\right)\Big|_{i,j} = \frac{-\phi_{i,j,g}\left(D_{i-1,j,g} + D_{i+1,j,g}\right)}{\Delta x^2} + \frac{-\phi_{i,j,g}\left(D_{i,j-1,g} + D_{i,j+1,g}\right)}{\Delta y^2} + \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}. \tag{C6}$$

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

$$\boldsymbol{f}\left(\boldsymbol{D_g} \odot \boldsymbol{\Phi_g}; \boldsymbol{w}\right)\Big|_{i,j} = \frac{-\left(D_{i-1,j,g}\phi_{i-1,j,g} + D_{i+1,j,g}\phi_{i+1,j,g}\right)}{\Delta x^2} + \frac{-\left(D_{i,j-1,g}\phi_{i,j-1,g} + D_{i,j+1,g}\phi_{i,j+1,g}\right)}{\Delta y^2} + \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}. \tag{C7}$$

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Combining the expressions in Equations (C5), (C6) and (C7) according to the definition of the diffusion operator from Equation (22) and gathering terms that multiply each scalar flux term gives

$$\boldsymbol{f}^{\text{Diff}}(\boldsymbol{\Phi}_g, \boldsymbol{D}_g; \boldsymbol{w}) = \frac{1}{2}\left(\boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g; \boldsymbol{w}) + \boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g; \boldsymbol{w}) - \boldsymbol{\Phi}_g \odot \boldsymbol{f}(\boldsymbol{D}_g; \boldsymbol{w})\right) \tag{C8}$$

$$= \frac{-\left(D_{i-1,j,g}\phi_{i-1,j,g} + D_{i+1,j,g}\phi_{i+1,j,g}\right)}{2\Delta x^2} + \frac{-\left(D_{i,j-1,g}\phi_{i,j-1,g} + D_{i,j+1,g}\phi_{i,j+1,g}\right)}{2\Delta y^2}$$
$$+ \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g} \tag{C9}$$
$$+ \frac{-D_{i,j,g}\left(\phi_{i-1,j,g} + \phi_{i+1,j,g}\right)}{2\Delta x^2} + \frac{-D_{i,j,g}\left(\phi_{i,j-1,g} + \phi_{i,j+1,g}\right)}{2\Delta y^2} + \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}$$
$$- \frac{-\phi_{i,j,g}\left(D_{i-1,j,g} + D_{i+1,j,g}\right)}{2\Delta x^2} - \frac{-\phi_{i,j,g}\left(D_{i,j-1,g} + D_{i,j+1,g}\right)}{2\Delta y^2} - \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) D_{i,j,g}\phi_{i,j,g}$$
$$= -\left(\frac{D_{i-1,j,g} + D_{i,j,g}}{2\Delta x^2}\right)\phi_{i-1,j,g} - \left(\frac{D_{i,j,g} + D_{i+1,j,g}}{2\Delta x^2}\right)\phi_{i+1,j,g}$$
$$- \left(\frac{D_{i,j-1,g} + D_{i,j,g}}{2\Delta y^2}\right)\phi_{i,j-1,g} - \left(\frac{D_{i,j,g} + D_{i,j+1,g}}{2\Delta y^2}\right)\phi_{i,j+1,g} \tag{C10}$$
$$+ \left(\frac{D_{i-1,j,g} + 2D_{i,j,g} + D_{i+1,j,g}}{2\Delta x^2} + \frac{D_{i,j-1,g} + 2D_{i,j,g} + D_{i,j+1,g}}{2\Delta y^2}\right)\phi_{i,j,g}.$$

From this, we can see that Equation (C10) 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

$$\boldsymbol{w} = \begin{bmatrix} 0 & \frac{-1}{\Delta y^2} & 0 \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} & \frac{-1}{\Delta x^2} \\ 0 & \frac{-1}{\Delta y^2} & 0 \end{bmatrix}. \tag{C11}$$

10970207, 2023, 21, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nme.7321 by Test, Wiley Online Library on [13/10/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License