Using AI libraries for incompressible Computational Fluid Dynamics

Boyang Chen[a], Claire E. Heaney[a,b], Christopher C. Pain[a,b,c,d]

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

[b] *Centre for AI-Physics Modelling, Imperial-X, Imperial College London, W12 7SL*

[c] *Novel Reservoir Modelling and Simulation Group, Department of Earth Science and Engineering, Imperial College London, London, UK*

[d] *Data Assimilation Laboratory, Data Science Institute, Imperial College London, SW7 2AZ, UK*

Abstract

Recently, there has been a huge effort to develop highly efficient libraries to perform Artificial Intelligence (AI) related computation on different computer architectures (for example, CPU, GPU and AI computers) and in the majority of the cases these libraries are open source. This has not only made the algorithms based on these libraries highly efficient and portable between different architectures, but also has substantially simplified the entry barrier to develop methods using AI. In this way, prototyping and developing highly efficient and architecture-agnostic algorithms has become the norm in the field due to the ubiquity of these libraries. Here, we present a novel methodology to bring the power of AI software and hardware into the field of numerical modelling by repurposing AI methods, such as Convolutional Neural Networks (CNNs), for the standard operations required in the field of the numerical solution of Partial Differential Equations (PDEs). The aim of this work is to bring the rapid prototyping, high performance and architecture agnosticism into the field of solving numerically PDEs. We present a methodology to solve the advection-diffusion equation, the non-linear Burgers equation incompressible flow past a bluff body. For some of these problems we use a convolutional neural network as a multigrid solver. We show that the presented methodology can solve all those problems using repurposed AI libraries in an efficient way, and presents a new avenue to explore in the development of methods to solve PDEs and CFD problems.

Keywords: Numerical solutions of PDEs, Multigrid, Discretisation, Convolutional Neural Network, Convolutional Autoencoder, Computational Fluid Dynamics

1. Introduction

In recent years, Artificial intelligence (AI) has provided the ability to solve problems that previously could not be solved by classical means. AI algorithms have been able to solve problems such as image identification, spam control, predictive writing, self-driving cars [1]. This huge success has encouraged not only the development of better and more sophisticated methods within the field of AI, but also the development of open source libraries devoted to simplify the entrance to and usage of AI. Some of the most popular libraries, PyTorch [2], TensorFlow [3], scikit-learn [4], and XGBoost [5], are maintained by large companies such as Google and Facebook, as well as contributed to by the external community. These machine learning (ML) libraries are highly optimised, easy to use, well documented and already deployed on many different hardware architectures. Moreover, to access these libraries, the language of choice is Python. Since Python is a scripting language, as long as the interpreter is compatible with the

operating system, the resulting code does not need to be rewritten. Therefore, programming using the aforementioned AI libraries means that the Python code can easily be deployed on different hardware and software without any rewriting required. In this way, these libraries are effectively the modern BLAS [6] and LAPACK [7] libraries for machine learning. Moreover, specific CPU architectures optimised for AI operations are becoming more and more common, namely the Tensor Processing Unit (TPU), providing a set of CPUs that are faster than conventional CPUs or GPUs for solving problems using AI. A noteworthy recent development is the "AI computer" called Cerebras CS-2 [8], which has nearly 1 million cores on a single chip and is potentially faster than even GPUs as well as being memory and energy efficient. The latter is particularly important for green computing as we may not be able to afford to run large petascale or exascale simulations that use the energy resources required by current CPUs due to our need to reduce our energy usage and the associated impact on our planet. The approach proposed here, of solving numerically Partial Differential Equations (PDEs) using AI software, therefore makes the use of these new AI computers potentially more accessible and the solution of the large systems of equations formed by discretisation of PDEs, more environmentally friendly.

The development of CFD software [9–11] is a challenging task as the developer needs to understand the mathematics, the physics and write optimised code for serial and for many cores, normally using a distributed memory approach based on OpenMPI libraries. In addition to this, normally further effort is required to ensure that the resulting code can be run on different operating systems (e.g. Windows and RHEL) and on different processor architectures (x86, ARM, or even GPUs). There have been attempts in the past to use Python to simplify this process [12], but a scripting language (Python in this case) is not as efficient as low level languages such as C, C++ or Fortran and therefore these attempts have not received sufficient attention to be considered a good alternative.

Here, we propose a way of developing CFD software based on AI libraries, which can simplify the development of CFD software by repurposing the AI libraries to perform the same operations that CFD software would normally do. In this way, for example, a convolutional neural network (CNN) can be repurposed to become a structured multigrid solver, where the convolution operator is equivalent to the restriction operator and the smoothing steps are performed using a neural network. By repurposing readily available software, we expect that the development of CFD can benefit in at least two areas: (i) deployment to different computer architectures (such as TPUs and new AI computers with distributed memory) will be more efficient, and the resulting code will be specifically optimised for that architecture; and (ii) reduction of the implementation time and lowering of the entry barrier by making the code more "high-level" than current CFD software implementations.

2. Results

In this section we present results for a one-step multigrid solver applied to the advection-diffusion equation; an explicit time discretisation of the advection-diffusion equation; a solution of the nonlinear Burgers equation and finally results are presented for incompressible flow past a bluff body by solving the Navier-Stokes equations combining the methods used in solving the three previous problems. Throughout this section, lengths are in metres, times are in seconds and velocities are in metres per second.

2.1. Multigrid solver for the advection-diffusion equation

Here we show results for the solution of the advection-diffusion equation:

$$\frac{\partial T}{\partial t} + \boldsymbol{q} \cdot \nabla T - \nu \nabla \cdot \nabla T = s \tag{1}$$

by a multigrid method implemented using a CNN. In Equation (1), T is the scalar solution field, $\boldsymbol{q} = (1,\ 1)^T$ is the advection velocity, s the source strength and ν the constant diffusivity. For a domain measuring 128 by 128, we advect an initial condition of Gaussian distribution centred at $(x_0, y_0) = (0, 0)$ and the coefficient representing the width of the distribution is $\gamma = 10$ (see Equation (2)). The velocity field is uniform at $(u, v) = (1, 1)$ and the uniform diffusion coefficient is $k = 2\,\mathrm{m^2 s^{-1}}$. The finite difference cell sizes in the x and y directions are constant at $\Delta x = \Delta y = 1$. The Courant number associated with the grid is $C = 50$ and the time step size is $\Delta t = 50$. We investigate one time-step solution, starting with an initial guess of $T^{1,0} = T^0 = 0$, using the multigrid with upwind differencing in space for advection and central differencing for diffusion, see Section 4.2. The Jacobi iterative method is embedded in the proposed neural network representing the multigrid method and a skip connection (or skip layer) is used to transfer the residuals between the convolutional layers, see Section 4.1. The numerical methods used to generate these results are implemented by initialising the weights of the kernel of a convolutional layer. No training of the network is performed. The results of applying this network to the initial condition are shown in Figure 1 and indicate the converged numerical prediction by CNN-based multigrid framework. In this figure we show how the solution converges with iteration number. We also show the difference iterations which shows how the multi-grid corrects the solution to obtain the final converged solution. We also show this in graph form and show the residuals at different multi-grid levels which shows how the solution is corrected on each grid level. This shows that the course grids levels seem to do much of the work in representing the solution correction.

2.2. Advection-diffusion equation

We apply the proposed CNN to solve the 2D time-dependent advection-diffusion equation in a domain measuring 300 by 300 with the origin located at the centre of the domain. A Gaussian distribution centred in the middle of the domain at $(x_0, y_0) = (0, 0)$ and with a width of $\gamma = 10$ (see Equation (2)) is taken as the initial condition. The advection velocity is $(u, v) = (0, 0)$ and the diffusion coefficient k has value $2\,\mathrm{m^2 s^{-1}}$. The model is compared with a traditional PDE solver and tested on different grids, showing that the results are identical to within round-off error (see the line plots in Figure 2). For the same domain, we advect an initial condition with a uniform velocity field $(u, v) = (1, 1)$ and a uniform diffusion coefficient $k = 2\,\mathrm{m^2 s^{-1}}$. The finite difference cell sizes in the x and y directions are constant at $\Delta x = \Delta y = 1$. The grid Courant number $C = 0.1$ and the time-step size is $\Delta t = 0.1$. We investigate the advection and diffusion of a combination of a Gaussian distribution and a square. This combination can be written as

$$T^0(x, y) = \exp\left(-\left(\frac{(x - x_0)^2}{2\gamma^2} + \frac{(y - y_0)^2}{2\gamma^2}\right)\right) \qquad \textit{(Gaussian distribution)} \qquad (2)$$

$$+ \begin{cases} 1, & x \in \text{[-25,25]},\ y \in \text{[-75,-125]} \\ 0, & \text{otherwise} \end{cases} \qquad \textit{(Uniform distribution)} \qquad (3)$$

in which T^0 represents the initial condition of the scalar field T, $x, y \in [-150, 150]$, the centre of the Gaussian distribution is at $(x_0, y_0) = (-50, 0)$, and the coefficient representing the width of the distribution is $\gamma = 40$. The square distribution is centred at $(x_0, y_0) = (0, -100)$. We investigate two discretisation methods, both with a predictor-corrector time discretisation that is second order accurate and with Crank-Nicolson time stepping: (i) a first order upwind differencing for advection and central differencing for diffusion; (ii) a second order central differencing for both advection and diffusion operators. Both discretisation methods are implemented by specifying the weights of the kernel of a convolutional layer, see Section 4.2 (and by not performing any training). From Figure 2, we see that both methods produce similar results, and, for longer times, the effect of diffusion becomes more pronounced as expected.

2.3. Burgers equation solution

For a domain measuring 300 by 300 with the origin at the centre of this, we validate the CNN which has been initialised in such a way that it can solve the Burgers equation on different grids and compare the results with a PDE solver. The results from the CNN and the traditional PDE solver were found to be identical up to round-off errors. Subsequently, two tests were performed. The first with a v velocity component of zero (which remains at zero throughout) and a u velocity component that is initialised using the Gaussian distribution, (see the first term in Equation (2)). The finite difference cell sizes in the increasing x and y directions are constant at $\Delta x = \Delta y = 1$. For the first example, the Gaussian distribution is centred at $(x_0, y_0) = (-100, 0)$ with a width defined by $\gamma = 15$. For the second example, the Gaussian (associated with $u^0(x, y)$ and $v^0(x, y)$) is centred at $(x_0, y_0) = (0, 0)$ with a width of $\gamma = 40\,\mathrm{m}$. In these examples, the viscosity, absorption and source terms of the Burgers equation are set to zero. We run both simulations with central differencing in space for diffusion and a predictor-corrector time discretisation scheme. First-order upwind differencing and central differencing schemes in space are used for the advection. The Courant number is set to be $C = 0.1$ using $\Delta t = 0.1$. To solve the Burgers equation, the discretisation schemes are coded as proposed, by initialising the weights of the kernel of a convolutional layer, but also by modifying the activation function so it is able to capture the non-linearity of the Burgers equation, see Section 4.3. The results can be seen in Figure 3. We see that the method resolves the non-linearity of Burgers equation resulting in the expected shock formation.

2.4. Flow past a bluff body - Navier-Stokes solutions

Here we integrate the proposed CNN's ability to forecast 2D and 3D flow past a bluff body by solving the incompressible Navier-Stokes equations as described in section 4.4. The domain has dimensions 512 by 512 and has a 40 by 40 solid square block centred at $(x_0, y_0) = (128, 256)$. The cell size is uniform in the x and the y directions (Δx and Δy) and set to 1. The initial velocity and pressure fields are set to zero across the whole domain. A zero pressure boundary condition is imposed at the right boundary and other three sides are applied with zero derivative boundary condition for the pressure field. Boundary conditions for the velocity field include the use of the slip boundary condition at the bottom and the top walls. An inflow and an outflow velocity in the x-direction of 1 are imposed at the left and the right boundaries and a velocity in the y-direction of 0 is imposed on these boundaries. The Reynolds number used here is 200 and is specified through the viscosity of the flow, assuming a unity density. We run the simulation with central differencing in space for the viscous and advection terms, while a second order accurate predictor-corrector scheme is applied to discretise the time derivative, see Section 4.2. A fixed time step $\Delta = 0.1$ is used. To solve the Navier-Stokes equations, the discretisation scheme is coded by initialising the weights of the kernel of a convolutional layer. The two components of the velocity field u, v for 2D and three velocity components u, v, w, in 3D, are treated separately as inputs of the neural network. The pressure field is obtained by applying multigrid networks to solve the resulting Poisson equation, see Equation (28). The activation function is modified to capture the non-linearity of the advection terms. The results are shown in Figure. 4. We see that the method resolves the flow structures around bluff body, including the unsteady separation of flow and vortex shedding. In the 2D AI simulation, the non-dimensional frequency of vortex shedding or Strouhal number is calculated to be 0.142 from our results for $Re = 200$, which is very close to the database value of 0.147 established by [13]. We further explore the vortex structure in 3D flow past a bluff body by extending the proposed network into 3D. For this case, the dimension of the domain is 128 by 128 by 128 with a 10 by 10 by 10 solid square block centred at $(x_0, y_0, z_0) = (32, 64, 64)$ and in which the solution domain is defined by $x \in [0, 128],\ y \in [0, 128], z \in [0, 128]$. The results are shown in Figure 4(a)-(c). The length of the re-circulation bubble l_1 is 22 m and the distance between the centre of the circulation bubble

where there is no velocity and the cube l_2 is 10 m. The ratio of l_1 to l_2 can be computed as 2.2, showing a close agreement with the value calculated in [14].

3. Discussion

We have demonstrated how neural networks can be used to solve differential equations, and have presented some simple applications and theory to underpin the approach. Implementing discretisations of PDEs using AI libraries with the method proposed here, has been found to produce identical results, to within round-off error, to PDE solvers written in more conventional programming languages e.g. modern Fortran. This, and similar work that is underway, is showing how the vast quantity of highly optimised AI software can be leveraged to solve differential equations across disciplines. For example, in fluids, solids, electromagnetism, radiation, economics or finance.

Programming PDE solutions using AI software is potentially very important because it enables interoperability between GPUs, CPUs and AI computers, which, in turn, enables exploitation of AI community software. Furthermore, energy shortfalls and climate change mean that the use of energy efficient computers will become increasingly desirable. Running software on any new architecture can be difficult, however, as these machines are designed to run AI software, this problem will have already been solved. Finally, it is also important because it can help to combine AI-based Reduced-Order Modelling (referred to here as AI-Physics modelling) and discretisation of the differential equations, facilitating a number of approaches including AI-based Subgrid-Scale (SGS) models and physics-informed methods [15]. It is now accepted that future SGS methods will be increasingly based on AI and that these often require implicit coupling to CFD: AI programming is the only satisfactory implementation approach.

The key advantages of programming numerical discretisations of PDEs using AI software include: (a) relatively easy implementation of multigrid methods and preconditioners; (b) model hierarchies can be readily formed ranging from high fidelity CFD to both fine and coarse AI-Physics models to make the methods much more computationally efficient; (c) CFD and AI-Physics modelling can be embedded in a single neural network, which is necessary for the implicit treatment within CFD of sophisticated SGS models based on AI and Physics-Informed methods; (d) automatic adjoint generation for digital twins that are able to assimilate data, perform uncertainty quantification and optimisation (see [16, 17]) using the optimisation engines embedded in all AI software, see [16]; (e) the realisation of new features in modelling without writing large quantities of code (e.g. mixed arithmetic precision or coupling different physics); (f) using AI software would make model development more accessible than programming within existing CFD codes, which can be highly complex; (g) the code generated by programming in AI software would be automatically optimised and be interoperable between computer architectures: CPU, GPU and AI computers, including exascale computers; (h) long term sustainability as code is based on long standing, community supported, AI software. For unstructured meshes and parallel computing one can generalise the above approach. For example, model parallelisation can be implemented, using MPI message passing to update the halos on the perimeters of the partitioned subdomains. In addition, this approach also allows blocks of different patterns to be used which enables semi-structured or block structured grids to be used and merges the use of parallelisation and unstructured meshes: more than one subdomain can be used on a given core using the MPI approach. In addition, the stencils or discretisations associated with fully unstructured meshes can be held on graph neural networks (see [18, 19]) in which there is no structured stencil. Instead, there is a sparse graph just as with an unstructured finite element method. On this graph we can define the discretisations as we have done before, except there may be a need to define every weight individually associated with the edges of the graph rather than using convolutional filters.

To form the multigrid method we can take the approach of coarsening the graph as often done in graph CNNs, see [20]. However, one can use mappings to structured 1D grids as used in the Space Filling Curve (SFC) CNN approach, see [21]. This is particularly efficient as the 1D grid structure can map highly efficiently onto the memory of the computer and with no indirect addressing.

For unstructured meshes, one can generalise the above approach by using graph neural networks (GNNs) [18, 19]. When using multigrid methods, the graph can be coarsened, as often done in GNN approaches [20] or one could use space-filling curves to generate mapping to structured 1D grids [21]. The latter could be highly efficacious as the 1D grid structure can map efficiently onto the memory of the computer, with no indirect addressing. To implement the parallelism, message passing (MPI) can be used to update the halos on the perimeters of the partitioned subdomains. We intend to explore these possibilities in the future.

4. Methods

4.1. Multigrid methods based on CNNs

For incompressible flow CFD simulators, the most computationally expensive part of the simulation is the resolution of the linear system of equations. multigrid are the fastest solvers for solving these large systems of equations. The similarities between CNN and multigrid solvers are striking and work is already underway to use CNNs to solve systems of descrete equations, from CFD, using multigrid methods, see [22].

A Geometric Multigrid method works by creating a hierarchy of nested meshes and using the coarser meshes to correct the solution of the finer meshes. Multigrid methods consist of three basic operations: (i) Smoothing: 1, 2 or 3 iterations of a linear solver such as Jacobi or Gauss-Seidel are performed; (ii) Restriction: interpolation of the residual from a given fine mesh to its lower coarse mesh; (iii) Prolongation: extrapolation of the error correction from a coarse mesh to the finer mesh. A multigrid loop consists of performing the following steps:

- Smooth (η) times.
- Compute residual and restrict to a coarser mesh
- Smooth (η) times.
- Repeat steps 2 and 3 until reaching the coarsest mesh desired.
- Solve exactly the system.
- Prolongate from coarse mesh and correct the solution from the finer mesh
- Smooth (ϕ) times.
- Repeat steps 6 and 7 until reaching the finest mesh.

In this work we use the simplest multigrid (MG) method similar to an F-cycle [23] but going from coarse resolution to fine resolution each multigrid iteration, see Figure 5.

Here we effectively, send the equation residuals onto the different grid levels, starting from the coarsest grid level we perform alternating Jacobi relaxation of the solution followed by prolongation of the resulting solution to the next grid level up in the hierarchy. The approach uses the coarsening levels of the CNN and in this case a single convolutional filter is used to represent the matrix equation, see Figure 5 (top). The approach is outlined in 1D and for 4 cells

in Figure 5 (middle row of pictures). Here we have labelled on the left the weights of the neural network which can be effected by a single filter between the layers. The filters change between the layers, when we apply Jacobi relaxation, because the grid spacing Δx changes between the layers.

We have indicated, only on the output layer, how biases (middle row left picture of Figure 5) can be applied to the neurons to effect the source problem associated with the Jacobi relaxation. However, biases are not consistent with the feed-forward propagation of the neural network information i.e. placing residuals from previous convolutional layers into the biases. Thus we use in our implementation layer skipping to send these residuals from the coarsening action of the convolutional neural network in a method akin to the use of the U-net (a commonly used neural network, see [24]), see Figure 5 bottom row left diagram. This figure shows the structure of the neural network for one multigrid cycle (we use 1MG to denote the resulting neural network). Several 1MG networks, or multigrid cycles, can be strung together to form an overall multigrid (MG) method, see figure bottom row left diagram Figure 5.

Assuming we are solving $Ax = s$, the Jacobi smoother or relaxation method can be defined as a Jacobi iteration:

$$x_i^{new} = \frac{1}{a_{ii}} \left(\sum_j a_{ij} x_j^{old} \quad + s_i \right). \tag{4}$$

Now applying relaxation $x_i = \alpha x_i^{new} + (1-\alpha) x_i^{old}$ results in:

$$x_i = \alpha x_i^{new} + (1-\alpha) \frac{1}{a_{ii}} \left(\sum_j a_{ij} x_j^{old} \quad + s_i \right). \tag{5}$$

Here, we use $\alpha = 1$ for the relaxation coefficient, although, in general, $\alpha \in (0, 2]$. Thus

$$x_i = \sum_j b_{ij} x_j^{old} \quad + c_i \tag{6}$$

in which

$$b_{ij} = \alpha \frac{1}{a_{ii}} a_{ij} \quad \text{if} \quad i \neq j, \tag{7}$$

$$b_{ii} = 1, \tag{8}$$

$$\text{and } c_i = \alpha \left(\frac{1}{a_{ii}} \right) s_i, \tag{9}$$

where c_i is the bias, realised by CNN layer skipping, see Figure 5, bottom left diagram. Thus $w_{ij} = b_{ij}$ are the weights of the convolutional filter.

4.2. Advection-diffusion solution

The advection-diffusion equation solved here is given by:

$$\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + \sigma T - \nu \nabla^2 T = s \,, \tag{10}$$

in which T is a scalar concentration field, (u, v) are the advection velocities, σ is an absorption term, ν the constant diffusivity and s the source. In the advection-diffusion simulations presented here, $\sigma = s = 0$ although none zero values of σ will be used for the momentum equations, Equations 25. The discretisation of Equations (10) are written out in full to enable easier comparison with the discretisations that follow.

For simplicity we write out the discretisation only for a simple central difference advection scheme, second-order diffusion scheme and second order in time. We also use a first order upwind advection scheme in some of the applications and more complex central difference discretisations of advection and diffusion for the flow past a bluff body, see supplementary notes. For a regular grid ($\Delta x = \Delta y$) and an advection velocity of $(u, v) = (1, 1)$, the spatial derivatives can be written as

$$\left(\frac{\partial T}{\partial x}\right)_{\ell,\,m} \approx \frac{T_{\ell+1,\,m} - T_{\ell-1,\,m}}{2\Delta x}, \tag{11}$$

$$\left(\frac{\partial T}{\partial y}\right)_{\ell,\,m+1} \approx \frac{T_{\ell,\,m} - T_{\ell,\,m-1}}{2\Delta x}, \tag{12}$$

$$\left(\frac{\partial^2 T}{\partial x^2}\right)_{\ell,\,m} \approx \frac{-T_{\ell+1,\,m} + 2T_{\ell,\,m} - T_{\ell-1,\,m}}{\Delta x^2}, \tag{13}$$

$$\left(\frac{\partial^2 T}{\partial y^2}\right)_{\ell,\,m} \approx \frac{-T_{\ell,\,m+1} + 2T_{\ell,\,m} - T_{\ell,\,m-1}}{\Delta x^2}, \tag{14}$$

in which the solution field (e.g. temperature, concentration, velocity component) at the ℓth node in the increasing x direction and the mth node in the y-direction is written as $T_{\ell,\,m}$. To implement a predictor-corrector scheme, we define the following

$$\left(\frac{\partial T}{\partial x}\right)^{n+1/2}_{\ell,\,m} \approx \frac{(\widetilde{T}^{n+1}_{\ell,\,m} + T^{n}_{\ell,\,m}) - (\widetilde{T}^{n+1}_{\ell-1,\,m} + T^{n}_{\ell-1,\,m})}{4\Delta x}, \tag{15a}$$

$$\left(\frac{\partial T}{\partial y}\right)^{n+1/2}_{\ell,\,m} \approx \frac{(\widetilde{T}^{n+1}_{\ell,\,m} + T^{n}_{\ell,\,m}) - (\widetilde{T}^{n+1}_{\ell,\,m-1} + T^{n}_{\ell,\,m-1})}{4\Delta x}, \tag{15b}$$

$$\left(\frac{\partial^2 T}{\partial x^2}\right)^{n+1/2}_{\ell,\,m} \approx \frac{-(\widetilde{T}^{n+1}_{\ell+1,\,m} + T^{n}_{\ell+1,\,m}) + 4(\widetilde{T}^{n+1}_{\ell+1,\,m} + T^{n}_{\ell,\,m}) - (\widetilde{T}^{n+1}_{\ell-1,\,m} + T^{n}_{\ell-1,\,m})}{2\Delta x^2}, \tag{15c}$$

$$\left(\frac{\partial^2 T}{\partial y^2}\right)^{n+1/2}_{\ell,\,m} \approx \frac{-(\widetilde{T}^{n+1}_{\ell,\,m+1} + T^{n}_{\ell,\,m+1}) + 4(\widetilde{T}^{n+1}_{\ell,\,m} + T^{n}_{\ell,\,m}) - (\widetilde{T}^{n+1}_{\ell,\,m-1} + T^{n}_{\ell,\,m-1})}{2\Delta x^2}, \tag{15d}$$

in which $\widetilde{T}^{n+1}_{\ell,\,m}$ represents the best guess for the solution at node ℓ, m at time level $n + 1$. Initially, the best guess for $\widetilde{T}^{n+1}_{\ell,\,m}$ is $T^{n}_{\ell,\,m}$. A prediction is calculated by using this best guess, Equations (15) and the following

$$T^{n}_{\ell,\,m} + \Delta t \left(\left(\frac{\partial T}{\partial x}\right)^{n+1/2}_{\ell,\,m} + \left(\frac{\partial T}{\partial y}\right)^{n+1/2}_{\ell,\,m} + \nu \left(\frac{\partial^2 T}{\partial x^2}\right)^{n+1/2}_{\ell,\,m} + \nu \left(\frac{\partial^2 T}{\partial y^2}\right)^{n+1/2}_{\ell,\,m} \right), \tag{16}$$

in which Δt is the time step size. Using Equation (16) to update the best guess, this is substituted back into the equations in system (15) and the result substituted into Equation (16) to give the corrected approximation to the solution at node ℓ, m at time level $n + 1$. Defining a 3 by 3 matrix of values on the grid as

$$[T]^{n}_{\ell,\,m} := \begin{pmatrix} T^{n}_{\ell-1,\,m+1} & T^{n}_{\ell,\,m+1} & T^{n}_{\ell+1,\,m+1} \\ T^{n}_{\ell-1,\,m} & T^{n}_{\ell,\,m} & T^{n}_{\ell+1,\,m} \\ T^{n}_{\ell-1,\,m-1} & T^{n}_{\ell,\,m-1} & T^{n}_{\ell+1,\,m-1} \end{pmatrix}. \tag{17}$$

We can rewrite the predictor corrector scheme from Equations (15) and (16) as

$$\widetilde{T}^{n+1}_{\ell,m} = T^n_{\ell,m} + \text{sum}\left\{ \left(\frac{\Delta t}{4\Delta x} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} + \frac{\nu \Delta t}{2\Delta x^2} \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \right) \odot \left([\widetilde{T}]^{n+1}_{\ell,m} + [T]^n_{\ell,m} \right) \right\}, \tag{18}$$

where the Hadamard product is represented by the symbol $\odot$ and the sum function sums all the entries of the matrix to give a scalar value (sometimes denoted as trace). Writing the predictor-corrector scheme in this way reveals the similarity between this discretisation and a convolutional layer which has with linear activation functions, a 3 by 3 kernel or filter with weights of

$$[A] := \frac{\Delta t}{4\Delta x} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} + \frac{\nu \Delta t}{2\Delta x^2} \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}, \tag{19}$$

operating on a grid of values $[\widetilde{T}]^{n+1}_{\ell,m} + [T]^n_{\ell,m}$ for all values of ℓ, m and in which we define the filter as $[A]$. Thus, the time stepping simply becomes:

$$\widetilde{T}^{n+1}_{\ell,m} = T^n_{\ell,m} + \text{sum}\left\{ [A] \odot \left([\widetilde{T}]^{n+1}_{\ell,m} + [T]^n_{\ell,m} \right) \right\}, \tag{20}$$

which can then be implemented by CNNs. For generality of the approach it will be useful to write matrices in index notation and in this case Equation (20) becomes:

$$\widetilde{T}^{n+1}_i = T^n_i + \sum_j a_{ij} \left(\widetilde{T}^{n+1}_j + T^n_j \right), \tag{21}$$

in which the matrix components a_{ij} are the result of the applying the filter $[A]$ as described above and cell in 2D indexes l, m correspond to cell i e.g. $T^n_i = T^n_{\ell,m}$.

The boundary conditions here are enforced by using the zero padding option which enforces a zero solution on the boundaries and is also applied to the Burgers equation solution.

4.3. Burgers equation solution

The Burgers equation can be written as

$$\frac{\partial \boldsymbol{q}}{\partial t} + u\frac{\partial \boldsymbol{q}}{\partial x} + v\frac{\partial \boldsymbol{q}}{\partial y} + \sigma \boldsymbol{q} - \nu \nabla \cdot \nabla \boldsymbol{q} = \boldsymbol{s}, \tag{22}$$

in which $\boldsymbol{q} = (u \;\; v)^T$ are the velocities in 2D, $\boldsymbol{s} = (s_x \;\; s_y)^T$ is a source (taken as zero), σ is an absorption term (taken as zero) and ν is the viscosity coefficient. This equation is discretised in a similar way to the advection-diffusion equation described in the previous section.

The implementation we have used for this problem addresses the need to allow for non-uniform advection velocity fields. This either means we have to define a different stencil for each cell of the mesh or we include the non-uniform velocity field in the neural network itself by re-defining the activation functions used. We have taken the latter approach, because, for non-linear problems, this allows the system of discrete equations to be differentiated and also allows non-linear multigrid methods to be used. We have implemented specifically designed activation functions To effectively implement the multiplication needed to discretise the non-linear terms we have used activation functions that perform multiplication within tensor flow. However, one could for, more complex non-linearity's, used specially designed activation functions.

We still use the fixed 3×3 filters on a 2D regular grid and use the same discretisation approach in space (either central or upwind differencing in space) and two step second order accurate in

time time-stepping method. We discretise each velocity component in $\boldsymbol{q} = (u\ v)^T$ and thus to maintain consistency with the previous section of advection-diffusion equation solutions we can denote a velocity component (either u or v) as T and solve for each one separately (first $T \equiv u$, then $T \equiv v$), see Figure 6 top diagrams. The discretisation results in a matrix system expressed with index notation (implemented through individual CNN filters as described in the previous section):

$$\left(\frac{T_i^{n+1}-T_i^n}{\Delta t}\right) + \sigma_i T_i^{n+1} \quad + \quad u_i \sum_j \tilde{a}_{ij}(1/2)(\tilde{T}_j^{n+1} + T_j^n) \\ + \quad v_i \sum_j \beta_{ij} \frac{1}{2}(\tilde{T}_j^{n+1} + T_j^n) + \sum_j \mu_{ij}(\tilde{T}_j^{n+1} + T_j^n) = s_i, \tag{23}$$

in which $a_{ij} = \frac{1}{2}\tilde{a}_{ij}\Delta t$, $b_{ij} = \frac{1}{2}\beta_{ij}\Delta t$ and $k_{ij} = \frac{\frac{1}{2}\mu_{ij}\Delta t}{(1+\Delta t\sigma_i)}$. Then,

$$T_i^{n+1} = \frac{1}{(1+\Delta t\sigma_i)} T_i^n + s_i \quad + \quad \frac{u_i}{(1+\Delta t\sigma_i)} \sum_j a_{ij}(\tilde{T}_j^{n+1} + T_j^n) \\ + \quad \frac{v_i}{(1+\Delta t\sigma_i)} \sum_j b_{ij}(\tilde{T}_j^{n+1} + T_j^n) + \sum_j k_{ij}(\tilde{T}_j^{n+1} + T_j^n)\,. \tag{24}$$

The three filters used are defined by the matrices a_{ij}, b_{ij} and k_{ij} which help discretise the velocity in the u and v and the diffusivitiy terms respectively in Equation (10). The half in the below is used to obtain second order accuracy in time, $\tilde{T}_i^{n+1}$ is the best guess for T_i^{n+1} which might be T_i^n. Δt = time step size. $\tilde{a}$, β are advection/diffusion stencils/discretisations. See Figure 6 for the diagram (top right left) showing the corresponding CNN with the filters indicated. Here we use non-linear activation functions defined by Equation (24) after operating using filters a and b. We need to use this CNN twice to achieve second order accuracy in time: once for the predictor step and once for the corrector.

The approach used for the Burgers equation is the same as that used for the momentum equation except we use a viscosity (diffusion) coefficient and central difference for the advection term. For the Burgers equation is discretised based on both first order upwind and central difference advection discretisation. Since we initialise the velocity based on a Gaussian function (2) for the u velocity component and the initial v velocity is zero. Thus only one component of velocity is non-zero and these are the results we present in Figure 3.

4.4. Navier-Stokes solution

For the momentum equation we need to solve for both the velocity components and we use two separate networks with the same architecture as shown top left but with different inputs as shown in figure top middle row. These two networks are combined to form a single network shown top right of figure 6. This is the network used shown for the velocity increment in time in the overall Navier-Stokes solution CNN outlines in figure 6 bottom. We use central schemes for the advection as well as diffusion operations. We use a FEM discretisation based on a bilinear rectangular (in 2D) and quadrilateral (in 3D) finite element for the advection terms. We also use the 27 point stencil for the diffusion operation. Both the advection and diffusion operators are described in the supplementary documentation.

The Navier-Stokes equations can be written as

$$\frac{\partial \boldsymbol{q}}{\partial t} + u\frac{\partial \boldsymbol{q}}{\partial x} + v\frac{\partial \boldsymbol{q}}{\partial y} + \sigma \boldsymbol{q} - \nu \nabla \cdot \nabla \boldsymbol{q} = -\nabla p\,, \tag{25}$$

$$\nabla \cdot \boldsymbol{q} = \boldsymbol{0}\,, \tag{26}$$

in which $\boldsymbol{q} = (u\ v)^T$ in 2D and $\boldsymbol{q} = (u\ v\ w)^T$ in 3D, p is the pressure, σ is an absorption term (taken as zero) and ν is the viscosity coefficient. This equation is discretised similarly to the Burgers equation (Equation (22)), but with a source of $\boldsymbol{s} = -\nabla p$.

The boundary conditions here are enforced by using a method similar to zero padding but populating the padding with a value if we have a Dirichlet boundary condition that is not zero and if we have a no-normal derivative boundary condition populating the corresponding field with the values next to the padding region.

Here we use a projection based solution method formed by manipulating the discretised equations which results in (see Figure 6 for a summary - bottom diagram):

Algorithm 1 Projection-based solution method for velocity and pressure (see top diagram of Figure 6 for a summary).

1: Solve for $\boldsymbol{q}^{n+1}$ using the two-step approach outlined for the Burgers and advection-diffusion equation but treating the term involving σ fully implicitly:

$$\frac{\boldsymbol{q}^{n+1} - \boldsymbol{q}^{n}}{\Delta t} + u^n \frac{\partial \boldsymbol{q}^{n+\frac{1}{2}}}{\partial x} + v^n \frac{\partial \boldsymbol{q}^{n+\frac{1}{2}}}{\partial y} + \sigma \boldsymbol{q}^{n+1} - \nu \nabla \cdot \nabla \boldsymbol{q}^{n+\frac{1}{2}} = -\nabla p^n\,. \tag{27}$$

2: Solve for pressure correction Δp:

$$\nabla^2 \Delta p = -\frac{1}{\Delta t} \nabla \cdot \boldsymbol{q}^{n+1}. \tag{28}$$

3: Solve for the velocity correction $\Delta \boldsymbol{q}$ using the multigrid solver (Figure 5):

$$\Delta \boldsymbol{q} = -\Delta t \nabla \Delta p. \tag{29}$$

4: Update pressure solution: $p^{n+1} = p^n + \Delta p$

5: Update velocity solution: $\boldsymbol{q}^{n+1} \leftarrow \boldsymbol{q}^{n+1} + \Delta \boldsymbol{q}$.

Acknowledgements

The authors would like to acknowledge useful discussions on the idea presented here with Dr Pablo Salinas and Dr Arash Hamzehloo. We would also like to acknowledge the following EPSRC grants: **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); the **PREMIERE** programme grant (EP/T000414/1); **MUFFINS**, MUltiphase Flow-induced Fluid-flexible structure InteractioN in Subsea applications (EP/P033180/1); and **MAGIC**, Managing Air for Green Inner Cities (EP/N010221/1).

Supplementary documentation

2D and 3D Convolutional Filters for Diffusion and Advection Discretization

The 3 by 3 filters implemented in the proposed 2D and 3D convolutional neural networks are shown below based on $\Delta t = 0.1s$, $\Delta x = 1$ m and $u = 1$ ms^{-1}. 2D 5-point stencil (diffusion

operator, advection operator in x and advection operator in y in which u and v are the advection velocity components in the increasing $x-$ and $y-$directions)

$$\nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & -4.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & -0.5 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, v}{\Delta x}\begin{pmatrix} 0.0 & -0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 \end{pmatrix}.$$

2D 9-point stencil (diffusion operator, advection operator in x and advection operator in y)

$$\nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & -3.0 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{pmatrix}, \frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.08 & 0.0 & -0.08 \\ 0.33 & 0.0 & -0.33 \\ 0.08 & 0.0 & -0.08 \end{pmatrix}, \frac{\Delta t\, v}{\Delta x}\begin{pmatrix} -0.08 & -0.33 & -0.08 \\ 0.0 & 0.0 & 0.0 \\ 0.08 & 0.33 & 0.08 \end{pmatrix}.$$

3D 7-point stencil diffusion operator (1st layer, 2nd layer and 3rd layer)

$$\nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & -6.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \end{pmatrix}, \nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

3D 7-point stencil advection operator in x (1st layer, 2nd layer and 3rd layer)

$$\frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & -0.5 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

3D 7-point stencil advection operator in y (1st layer, 2nd layer and 3rd layer)

$$\frac{\Delta t\, v}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, v}{\Delta x}\begin{pmatrix} 0.0 & -0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 \end{pmatrix}, \frac{\Delta t\, v}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

3D 7-point stencil advection operator in z (1st layer, 2nd layer and 3rd layer, w is the velocity component in the $z-$direction)

$$\frac{\Delta t\, w}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, w}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, w}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & -0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

3D 27-point stencil diffusion operator (1st layer, 2nd layer and 3rd layer)

$$\nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.077 & 0.115 & 0.077 \\ 0.115 & 0.231 & 0.115 \\ 0.077 & 0.115 & 0.077 \end{pmatrix}, \nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.115 & 0.231 & 0.115 \\ 0.231 & -3.385 & 0.231 \\ 0.115 & 0.231 & 0.115 \end{pmatrix},$$

$$\nu\frac{\Delta t}{\Delta x^2}\begin{pmatrix} 0.077 & 0.115 & 0.077 \\ 0.115 & 0.231 & 0.115 \\ 0.077 & 0.115 & 0.077 \end{pmatrix}.$$

3D 27-point stencil advection operator in x (1st layer, 2nd layer and 3rd layer)

$$\frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.014 & 0.0 & -0.014 \\ 0.056 & 0.0 & -0.056 \\ 0.014 & 0.0 & -0.014 \end{pmatrix}, \frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.056 & 0.0 & -0.056 \\ 0.22 & 0.0 & -0.22 \\ 0.056 & 0.0 & -0.056 \end{pmatrix}, \frac{\Delta t\, u}{\Delta x}\begin{pmatrix} 0.014 & 0.0 & -0.014 \\ 0.056 & 0.0 & -0.056 \\ 0.014 & 0.0 & -0.014 \end{pmatrix}.$$

3D 27-point stencil advection operator in y (1st layer, 2nd layer and 3rd layer)

$$\frac{\Delta t\, v}{\Delta x}\begin{pmatrix} -0.014 & -0.056 & -0.014 \\ 0.0 & 0.0 & 0.0 \\ 0.014 & 0.056 & -0.014 \end{pmatrix}, \frac{\Delta t\, v}{\Delta x}\begin{pmatrix} -0.056 & -0.22 & -0.056 \\ 0.0 & 0.0 & 0.0 \\ 0.056 & 0.22 & 0.056 \end{pmatrix},$$

$$\frac{\Delta t\, v}{\Delta x}\begin{pmatrix} -0.014 & -0.056 & -0.014 \\ 0.0 & 0.0 & 0.0 \\ 0.014 & 0.056 & -0.014 \end{pmatrix}.$$

3D 27-point stencil advection operator in z (1st layer, 2nd layer and 3rd layer)

$$\frac{\Delta t\, w}{\Delta x}\begin{pmatrix} 0.014 & 0.056 & 0.014 \\ 0.056 & 0.22 & 0.056 \\ 0.014 & 0.056 & 0.014 \end{pmatrix}, \frac{\Delta t\, w}{\Delta x}\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \frac{\Delta t\, w}{\Delta x}\begin{pmatrix} -0.014 & -0.056 & -0.014 \\ -0.056 & -0.22 & -0.056 \\ -0.014 & -0.056 & -0.014 \end{pmatrix}.$$

References

[1] D. Dhall, R. Kaur, M. Juneja, Machine learning: A review of the algorithms and its applications, in: P. K. Singh, A. K. Kar, Y. Singh, M. H. Kolekar, S. Tanwar (Eds.), Proceedings of ICRIC 2019, Springer International Publishing, Cham, 2020, pp. 47–63.

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

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.
URL https://dl.acm.org/doi/10.5555/3026877.3026899

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
URL http://jmlr.org/papers/v12/pedregosa11a.html

[5] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2016, pp. 785–794. doi:10.1145/2939672.2939785.

[6] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al., An updated set of basic linear algebra subprograms (BLAS), ACM Transactions on Mathematical Software 28 (2) (2002) 135–151. doi:10.1145/567806.567807.

[7] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, 3rd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[8] CS-2: A Revolution in AI Infrastructure, https://www.cerebras.net/product-system/, accessed: 19-09-2022.

[9] C. A. J. Fletcher, Computational Techniques for Fluid Dynamics: Fundamental and General Techniques, 2nd Edition, Vol. Scientific Computation, Springer, 1998.

[10] C. A. J. Fletcher, Computational Techniques for Fluid Dynamics: Specific Techniques for Different Flow Categories, Vol. Scientific Computation, Springer, 1991. doi:10.1007/978-3-642-58239-4.

[11] The OpenFOAM Foundation, OpenFOAM v9 User Guide.
URL https://cfd.direct/openfoam/user-guide

[12] A. V. Mohanan, C. Bonamy, M. Linares, P. Augier, FluidSim: Modular, Object-Oriented Python Package for High-Performance CFD Simulations, Journal of Open Research Software 7 (04 2019). doi:10.5334/jors.239.

[13] W. Commons, File:strouhal de corps 2d, blevins et autres.svg — wikimedia commons, the free media repository, [Online; accessed 30-June-2022] (2020).
URL https://commons.wikimedia.org/w/index.php?title=File:Strouhal_de_corps_2D,_Blevins_et_autres.svg&oldid=505620977

[14] Q. Meng, H. An, L. Cheng, M. Kimiaei, Wake transitions behind a cube at low and moderate reynolds numbers, Journal of Fluid Mechanics 919 (2021).

[15] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707. doi:10.1016/j.jcp.2018.10.045.

[16] C. M. Bishop, Pattern recognition and machine learning, Information science and statistics, Springer, New York, NY, 2006.

[17] D. G. Cacuci, M. Ionescu-Bujor, I. M. Navon, Sensitivity and Uncertainty Analysis: Applications to Large-Scale Systems, CRC Press, 2005.

[18] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, D. Cohen-Or, MeshCNN: A Network with an Edge, ACM Transactions on Graphics 38 (4) (2019). doi:10.1145/3306346.3322959.

[19] J. Tencer, K. Potter, A tailored convolutional neural network for nonlinear manifold learning of computational physics data using unstructured spatial discretizations, SIAM Journal on Scientific Computing 43 (4) (2021) A2581–A2613. doi:10.1137/20m1344263.

[20] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. Yu, A Comprehensive Survey on Graph Neural Networks, IEEE Transactions on Neural Networks and Learning Systems 32 (1) (2020) 1–21. doi:10.1109/TNNLS.2020.2978386.

[21] C. E. Heaney, Y. Li, O. K. Matar, C. C. Pain, Applying Convolutional Neural Networks to Data on Unstructured Meshes with Space-Filling Curves, arXiv preprint 2011.14820 (2020).
URL https://arxiv.org/abs/2011.14820

[22] N. Margenberg, D. Hartmann, C. Lessig, T. Richter, A neural network multigrid solver for the Navier-Stokes equations, Journal of Computational Physics 460 (2022) 110983. doi:https://doi.org/10.1016/j.jcp.2022.110983.
URL https://www.sciencedirect.com/science/article/pii/S0021999122000456

[23] Wikipedia contributors, Multigrid method — Wikipedia, the free encyclopedia, [Online; accessed 2-July-2022] (2022).
URL https://en.wikipedia.org/w/index.php?title=Multigrid_method&oldid=1095163500

[24] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: N. Navab, J. Hornegger, W. M. Wells, A. F. Frangi (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, Springer International Publishing, Cham, 2015, pp. 234–241.

[25] T. Ke, M. Maire, S. X. Yu, Neural multigrid, CoRR abs/1611.07661 (2016). arXiv:1611.07661.
URL http://arxiv.org/abs/1611.07661

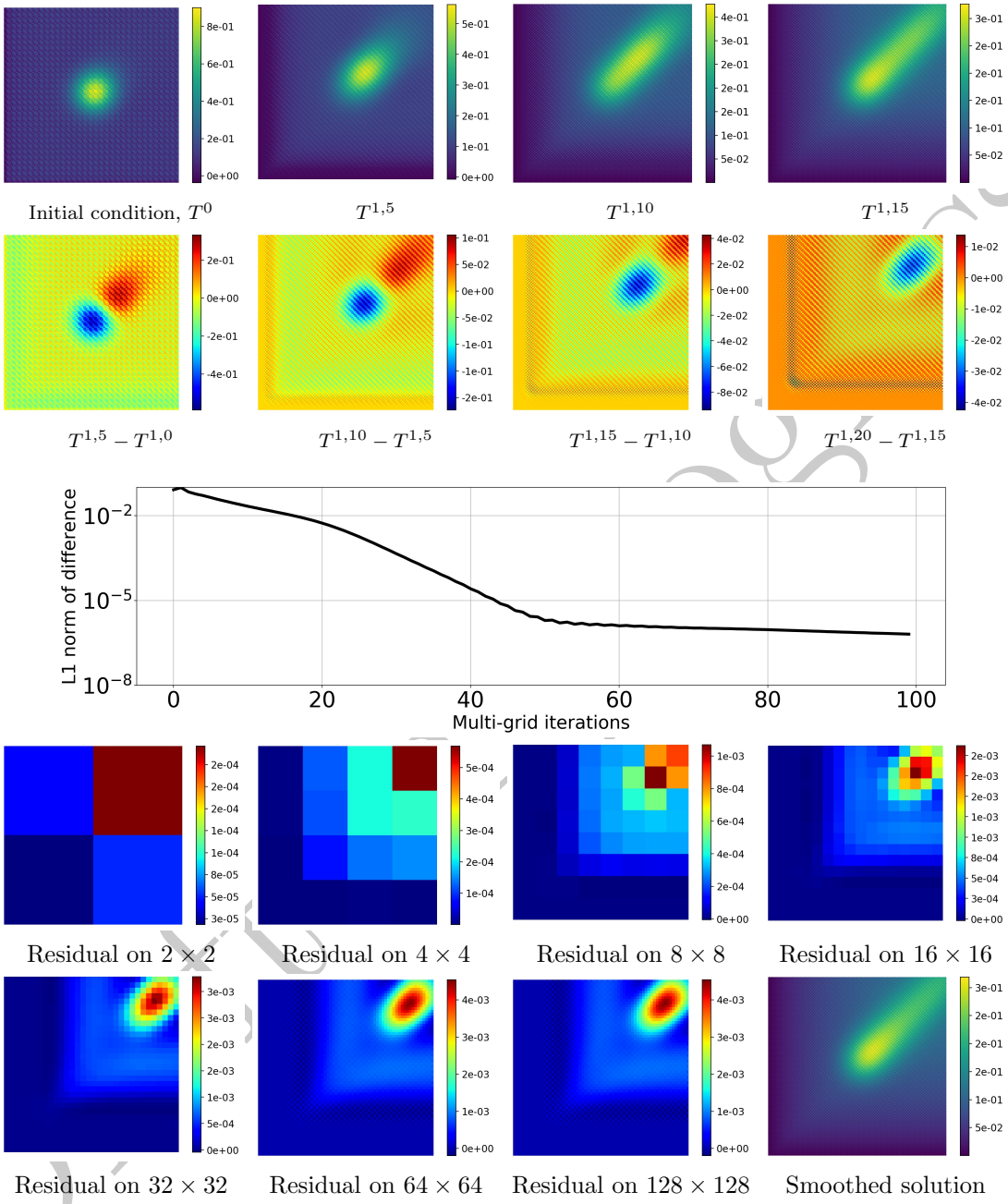

Figure 1: Multigrid solution of advection-diffusion equation by a CNN. The first row, at the top, shows the solution with an initial condition of a Gaussian distribution centred at $(x_0, y_0) = (0, 0)$ after 5, 10 and 15 multigrid iterations ($T^{1,i}$ where i represents the iteration number). The second row shows the difference between the solutions after two particular numbers of iterations. The middle line plot represents the L_1 norm of difference between the solutions at the current and previous iterations. The last of the bottom 8 plots represents the final solution, and the remaining plots represent the spatial evolution of residuals on different grids within one multigrid iteration.

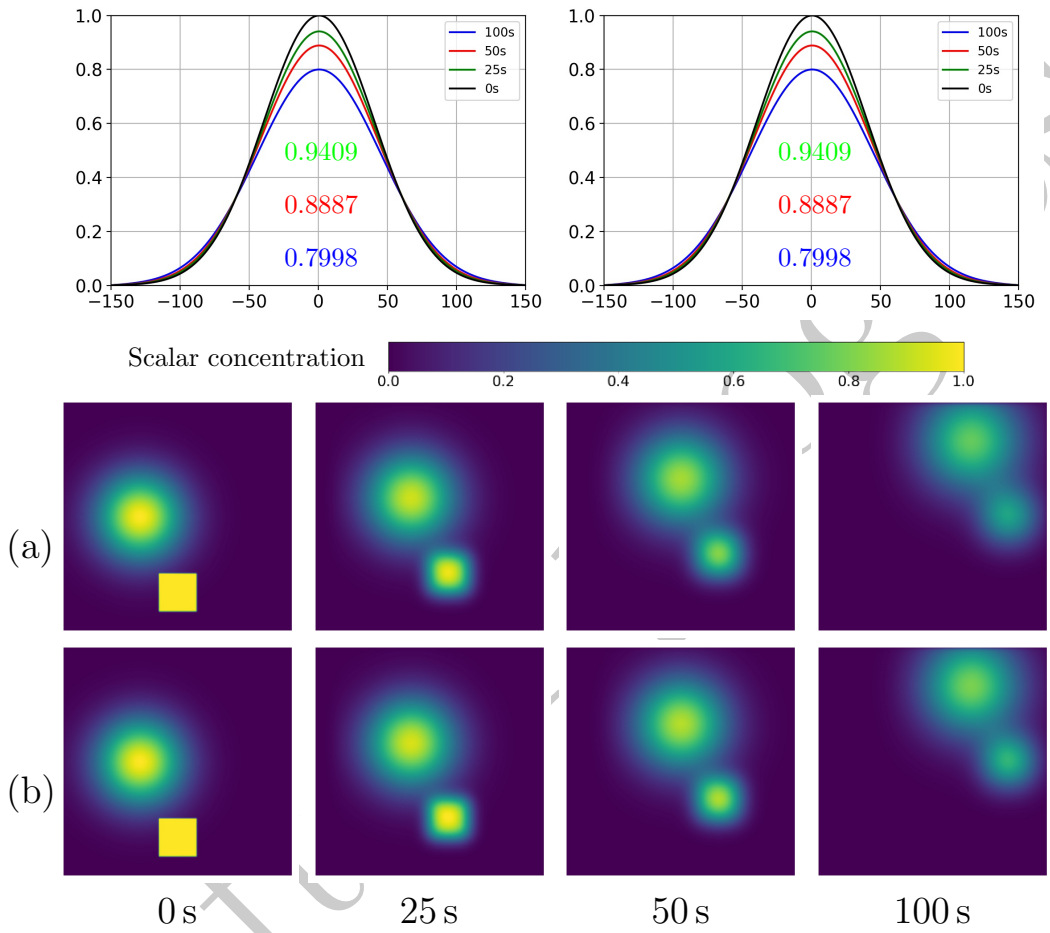

Figure 2: Numerical solution of the advection-diffusion equation by a CNN at 0, 25, 50 and 100. The top two line plots compare the solution of the advection-diffusion equation (with no advection) for an initial condition of a Gaussian distribution by a traditional fluid dynamics code (left) and using the approach described here based on CNNs (right). The three values shown in each graph correspond to the maximum values of the solution, occurring at x=0. The eight contour plots relate to the solution of the advection-diffusion equation for an initial condition of the combination of a square and a Gaussian centred at $(x_0, y_0) = (0, -100)$ and $(x_0, y_0) = (-50, 0)$ respectively. Both sets of results have a second-order time discretisation scheme with central differencing in space for the diffusion operator. The plots on row (a) have upwind differencing for the advection operator; and on row (b) have central differencing for the advection operator.

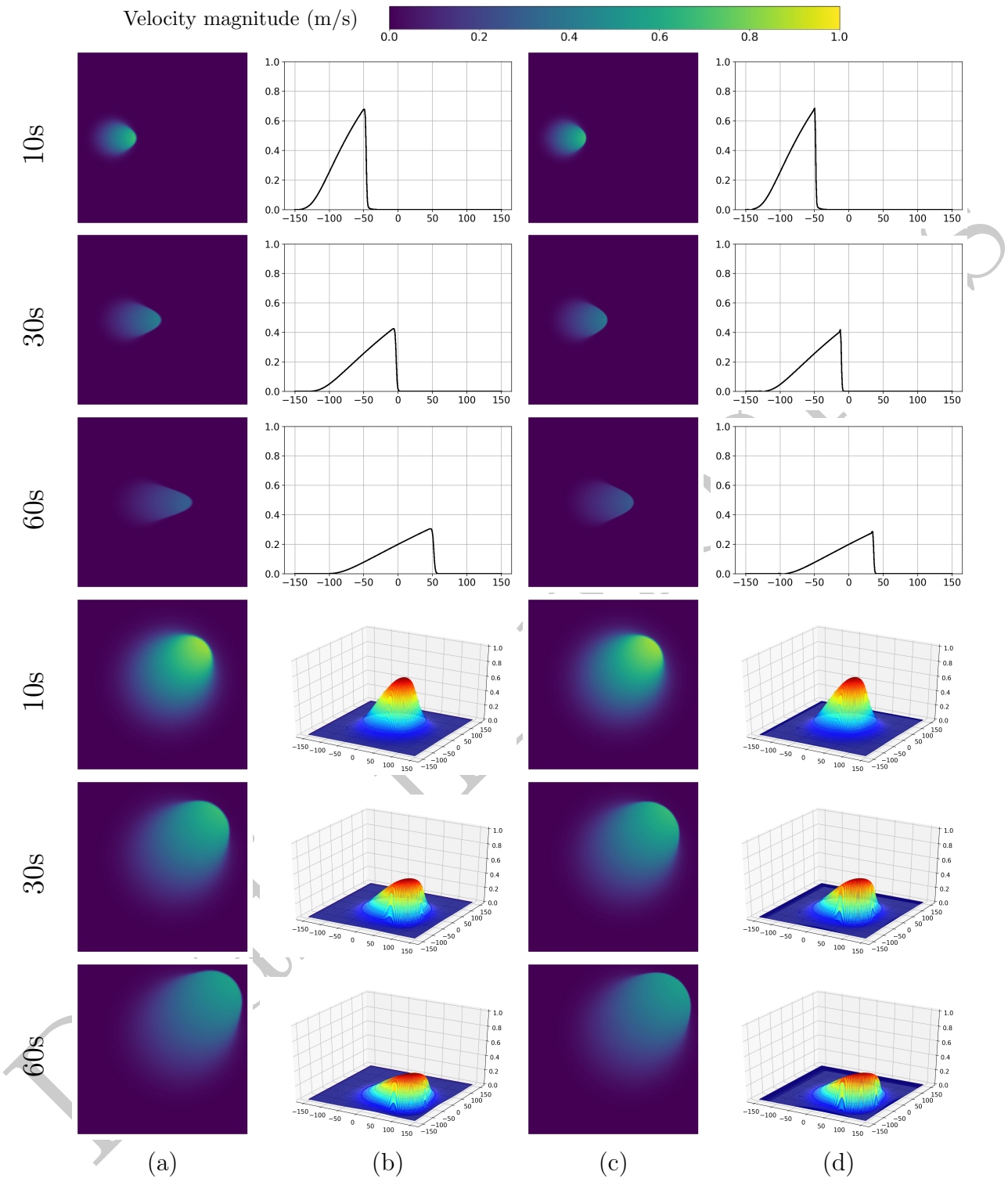

Figure 3: Numerical prediction of 2D nonlinear Burgers equation by a CNN. The top 12 plots relate to the flow field with only the u component and a Gaussian-shaped wave initially centred at $(x_0, y_0) = (-100, 0)$, at 10, 30 and 60. The bottom 12 plots relate to the flow field with the u and v components and a Gaussian-shaped wave initially centred at $(x_0, y_0) = (0, 0)$, at 10, 30 and 60. Columns represent second-order time stepping scheme: (a) and (b) upwind differencing for advection operator and central differencing for diffusion operator in space; (c) and (d) upwind differencing for advection operator and second-order central differencing for diffsuion operator in space.

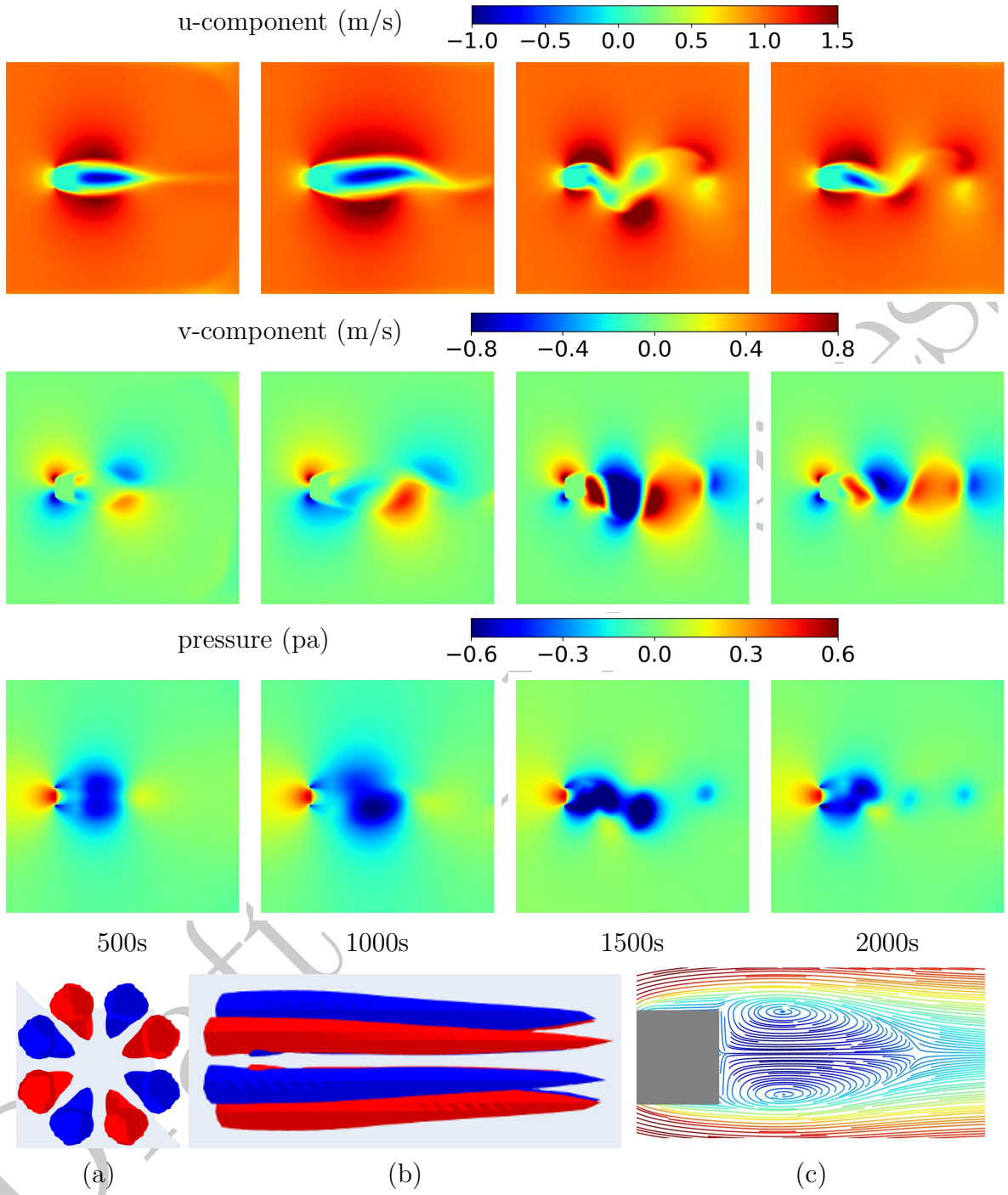

Figure 4: Numerical prediction of 2D and 3D flow past a bluff body by a CNN. The 12 plots relate to the two velocities and the pressure of the 2D flow field, at 500 s, 1000 s, 1500 s and 2000 s. The dimension of 2D computational domain is 512 m by 512 m, where a solid square body is modelled as 40 m by 40 m and centred at $(x_0, y_0) = (128\,\text{m}, 256\,\text{m})$. The bottom 3 plots relate to the steady vortex structure in the 3D flow field at 1000 s: (a) and (b) are the rear and side views of the isosurface representing $\omega_x = \pm 0.01$ with red and blue for positive and negative; (c) is 2D steady streamlines coloured by the u-component velocity. The dimension of 3D computational domain is 128 m by 128 m by 128 m, where a solid square body is modelled as 10 m by 10 m by 10 m and centred at $(x_0, y_0, z_0) = (32\,\text{m}, 64\,\text{m}, 64\,\text{m})$. The Reynolds number of the two case is set by 200. Both sets of results have a second-order time discretisation scheme with central differencing in space for the diffusion and the advection.

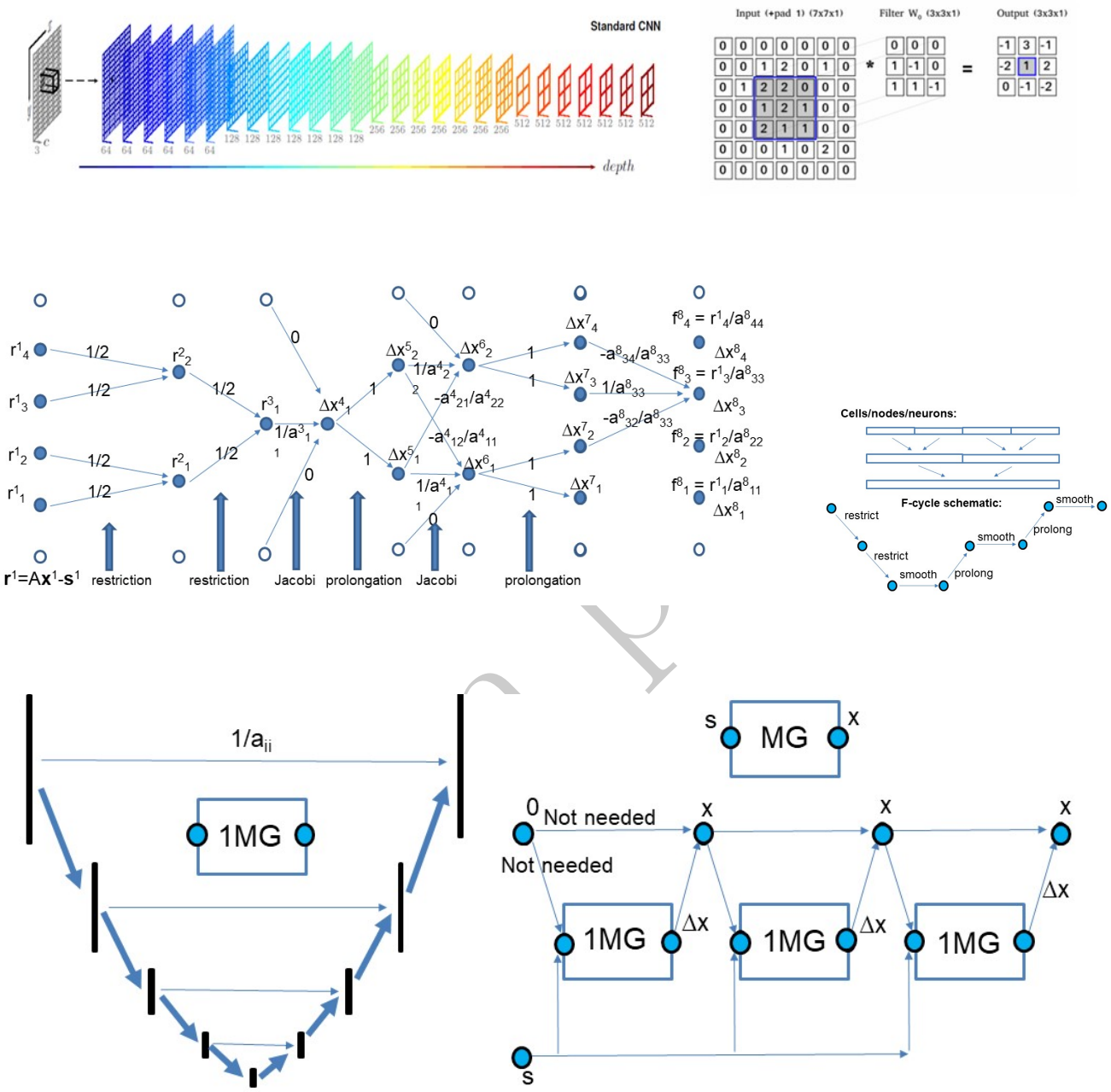

Figure 5: Schematic showing the how the CNNs are repurposed to produce a multigrid (MG) method. Top row: The CNN with different levels and an example of a filter thats passed over the convolutional layers, see [25]. Middle row left: How a 1D CNN multigrid method works with biases f. Middle right: The four 1D cells that have two levels of coarsening and the schematic showing how this multigrid method works. Bottom row left: Schematic showing how one multigrid (MG) cycle works using layer skipping to pass the residiual between the layers akin the U-net - this is a better alternative to using biases for the residuals. Bottom row right: Schematic showing how three multigrid cycles are brought together to form an overall matrix solution method.

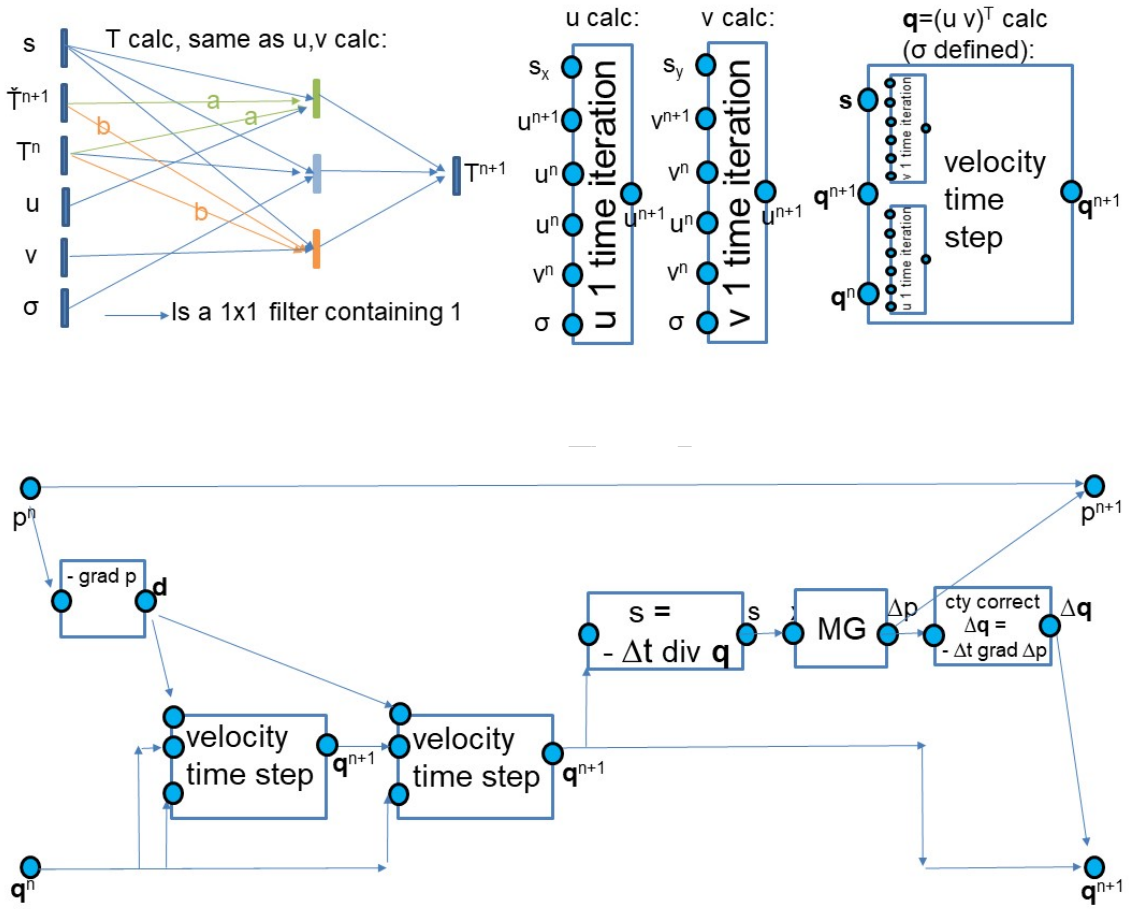

Figure 6: Schematic showing the how the neural networks works for time stepping. Top: For a scalar T and velocity components. The green, orange and purple coloured filters are associated with advection in u and v directions and diffusivity respectively. Bottom: A single time step for the Navier-Stokes solution.