Solving the discretised multiphase flow equations with interface capturing on structured grids using machine learning libraries

Boyang Chen [a], Claire E. Heaney [a,b], Jefferson L.M.A. Gomes [c,*], Omar K. Matar [b,d], Christopher C. Pain [a,b,e]

[a] *Applied Modelling and Computation Group, Department of Earth Science and Engineering, Imperial College London, London, SW7 2AZ, UK*
[b] *Centre for AI-Physics Modelling, Imperial-X, White City Campus, Imperial College London, London, W12 7SL, UK*
[c] *Fluid Mechanics Research Group, School of Engineering, University of Aberdeen, Aberdeen, AB24 3FX, UK*
[d] *Department of Chemical Engineering, Imperial College London, London, SW7 2AZ, UK*
[e] *Data Assimilation Laboratory, Data Science Institute, Imperial College London, London, SW7 2AZ, UK*

ARTICLE INFO

ABSTRACT

This paper solves the discretised multiphase flow equations using tools and methods from machine-learning libraries. The idea comes from the observation that convolutional layers can be used to express a discretisation as a neural network whose weights are determined by the numerical method, rather than by training, and hence, we refer to this approach as Neural Networks for PDEs (NN4PDEs). To solve the discretised multiphase flow equations, a multigrid solver is implemented through a convolutional neural network with a U-Net architecture. Immiscible two-phase flow is modelled by the 3D incompressible Navier–Stokes equations with surface tension and advection of a volume fraction field, which describes the interface between the fluids. A new compressive algebraic volume-of-fluids method is introduced, based on a residual formulation using Petrov–Galerkin for accuracy and designed with NN4PDEs in mind. High-order finite-element based schemes are chosen to model a collapsing water column and a rising bubble. Results compare well with experimental data and other numerical results from the literature, demonstrating that, for the first time, finite element discretisations of multiphase flows can be solved using an approach based on (untrained) convolutional neural networks. A benefit of expressing numerical discretisations as neural networks is that the code can run, without modification, on CPUs, GPUs or the latest accelerators designed especially to run AI codes.

1. Introduction

1.1. Motivation

Detailed numerical flow simulations provide significant insight into a wide range of sectors, including the environment [1], energy [2] and food [3] sectors. Over the past two decades, computational fluid dynamics (CFD) models, including multiphase flow models, have become critical for predicting environmental [4] and industrial [5] flow behaviour; conducting safety [6] and environmental [7] assessments; studying the effect of cloud formation on the climate [8]; analysing slugging in fluidised bed reactors [9] and pipelines [10]; modelling food processing to reduce waste [11]; as well as being used to investigate fundamental

---

* Corresponding author.
E-mail address: jefferson.gomes@abdn.ac.uk (J.L.M.A. Gomes).

multiphase turbulence [12]. Our aim is to develop a method that can be used to simulate complex environmental and industrial processes involving immiscible multiphase flows and that has the potential to run on the latest GPUs and AI processors. The finite element discretisations and Petrov–Galerkin terms are all written as a neural network and solved with a neural network, giving the same result as if these methods had been programmed in a traditional way. We benchmark the proposed method with a collapsing water column problem and a rising bubble problem, running the code on a GPU.

1.2. Background

The so-called one-fluid approach has been widely used to simulate the flow of two or more immiscible fluids or phases [13]. In this approach, a single model describes the flow throughout the domain and an interface marks the boundary between the different fluids or phases, across which, surface tension forces can be applied. Schemes for modelling the interface can be broadly classified as interface tracking or interface capturing. Although interface tracking methods maintain a sharp interface and enable physical processes to be incorporated at the interface, these methods suffer from mass conservation issues, and struggle to model break-up and coalescence [14–17]. Combining these methods with level-set methods has resulted in an improvement in their ability to conserve mass [18,19]. Interface-capturing methods treat interfaces as discontinuities in the material properties which are advected by a volume fraction field. In sharp-interface approaches, such as Volume-of-Fluid (VoF) methods [20,21] or enriched element methods (including XFEM [22] and Cut-Cell methods [23,24]), values of density and viscosity exhibit a discontinuity or jump at the interface, whereas for diffuse–interface approaches, such as the phase field method [25], values of density and viscosity are functions of the volume fraction field. VoF methods have good mass conservation properties, are able to maintain a sharp interface [14,26] (provided there is sufficient resolution), and can handle coalescence and break-up [27]. Phase-field methods diffuse the interface over a number of cells [28], which improves stability but loses the sharpness of the interface. Level set methods can be considered as sharp [15] or diffuse [23], and, although the volume fraction field is smooth, conservation of mass can be problematic [14,16]. For more details on many of the available approaches to modelling interfaces, see [15,29,30], and for some recent work on the combination of interface tracking and interface-capturing methods, see [31]. In this paper, our goal is to model the break-up of immiscible two-phase flows, and we choose a VoF type approach for its ability in this regard. Following Pavlidis et al. [32,33] and Obeysekara et al. [34], additional diffusion terms are included to keep the interface between the two fluids sharp: positive diffusion is added if oscillations are detected in the solution and negative diffusion is added if the solution is smooth. However, in this paper, oscillations are detected in a different manner to the methods presented previously [32–34] and the amount of diffusion is calculated more conservatively to increase accuracy.

The accuracy of CFD models strongly relies on the resolution of the mesh or grid, which becomes particularly challenging for multiphase flow problems due to the enormous range of length scales within the governing mechanisms [1,3,17]. As a consequence, computational costs are extremely high when simulating these processes. The use of high performance computing (HPC) resources can be beneficial, although, to obtain good performance, an efficient implementation of models, methods and algorithms is needed [35]. Originally, HPC clusters consisted predominantly of central processing units (CPUs), however, Graphics Processing Units (GPUs), containing hundreds to thousands of processing cores, are now also available in such clusters [36,37]. In order to make use of GPUs, however, existing CFD models either need to be re-implemented in terms of graphical primitives, using specialist programming platforms such as CUDA, OpenCL and OpenACC [38,39], or require implementation of communication protocols between program units across computational architectures [40,41].

Appleyard and Drikakis [42] presented an early comparison between the performance of a single CPU and a single GPU when solving the level set equations describing an interface, and reported a difference of two orders of magnitude in favour of the GPU implementation. Following this, a number of researchers developed hybrid CPU/GPU implementations of multiphase flow solvers [14,43,44]. In all three cases, the computationally expensive pressure equation was solved on the GPUs and the other equations were solved on CPUs. For a variety of problems with up to 27 million grid points, speed-ups of 50 to 100 were obtained. Bryngelson et al. [45] introduced the MFC code, a parallel multi-component, multiphase and multiscale flow solver that was extended by Radhakrishnan et al. [46] to run on GPU systems. The solver uses high-order interface-capturing with diffuse–interface models to describe the interface between fluids. They ran their code on up to 576 GPUs to solve shock-induced collapse of air bubbles in the vicinity of a kidney stone with half a million grid points and on 960 GPUs to solve atomising droplets with 2 billion (10^9) grid points. Crialesi-Esposito et al. [17] presented FluTAS, a solver for incompressible multiphase flows with heat transfer that can be run on multi-CPU and multi-GPU architectures. A direct method based on the Fast Fourier Transform is used to solve the pressure equation and the interface is represented by a volume-of-fluid method. Demonstrating their code on two-layer Rayleigh–Bénard convection and emulsions in homogeneous isotropic turbulence using up to 1 billion (10^9) grid points with up to 128 GPUs, they identify a communication burden which increased with the number of GPUs.

Not only is there significant work to be done in getting a code to run on a GPU system relating to re-implementing algorithms with graphical primitives, in addition, many of the afore-mentioned papers fine-tune their codes to obtain good performance, by assigning particular tasks to CPUs and others to GPUs. The level of computational expertise required can pose a barrier for many, however, a recent AI-based approach offers a way of circumventing this issue. Instead of using AI technologies to *approximate* CFD models, as is often done, this recent approach solves discretised fluid dynamics systems *exactly* (to within solver tolerances) using the functionality found within ML libraries [47–51]. This is because numerical discretisations can be expressed as discrete convolutions between a solution field and a stencil, the weights of which are determined explicitly by the numerical discretisation [52,53]. Exactly the same operation is performed in the convolutional layer of a convolutional neural network (CNN) [54–56] by a filter when it is convolved with an image. The weights of the filters of CNNs are usually determined by training, however, in this case, no

training is needed as the weights are determined by the numerical discretisation. Hence, numerical discretisations can be expressed as CNNs using functionality from machine learning libraries. To solve the resulting systems, Jacobi methods can be implemented directly or multigrid methods can be implemented through a special neural network architecture known as a U-Net [57]. Previous examples of solving discretised governing equations using machine learning techniques include Zhao et al. [47], who solved a finite difference discretisation of the Navier–Stokes equations using a neural network with pre-defined weights adopting an explicit solver for the pressure equation based on preconditioned conjugate gradients. The solutions of the neural network implementation were in good agreement with CFD benchmarks. Wang et al. [48] developed a similar approach, and obtained linear weak scaling and a superlinear strong scaling for 2048 cores on TPUs (Google's accelerated tensor processing units) for CFD problems with up to a billion (10^9) computational cells. Woo et al. [58] developed a particle-in-cell multiphase solver to model particles immersed in a gas, porting parts of the MFiX code (Multiphase Flow with Interphase eXchanges) to TensorFlow. Some operations, such as particle initialisation, were handled by CPUs through Fortran code, but the computationally demanding calculations were handled by GPUs through python code and TensorFlow functions. Chen et al. [49] implemented both finite difference and finite element discretisations through convolutional neural networks with a sawtooth multigrid method implemented within a U-Net architecture, to solve benchmark single-phase CFD problems. This method was extended to solve the neutron diffusion equations [50] and radiation transport problems [51]. In the latter, a new convolutional finite element method (ConvFEM) was proposed, which was designed to simplify the discrete convolutions of higher-order Finite Element Methods (FEMs), so that they can be easily represented by convolutional layers in neural networks. Although applied to structured grids in this paper, by using graph neural networks, the NN4PDEs approach has been extended to unstructured meshes, where Li et al. [59] apply the method to a Discontinuous Galerkin formulation for solving diffusion equations.

1.3. Contribution

In this paper, we start from the NN4PDEs approach proposed by Chen et al. [49], which includes a single-phase CFD solver written as a neural network, and extend it to model immiscible two-phase flows. We solve the 3D incompressible Navier–Stokes equations with surface tension on structured grids and develop a type of compressive algebraic VOF method to capture the interface. A segregated pressure and velocity time-stepping method is used with explicit time-stepping schemes for volume fraction and velocity, and an implicit pressure update is used to enforce the incompressibility constraint. The latter is solved using a multigrid method with Jacobi smoothing through a U-Net architecture, as described in Phillips et al. [50]. Using convolutional neural networks, we solve the discretised system of equations and obtain the same answer (to within solver tolerance) as if the numerical methods had been coded in Fortran or C++. The first advantage of this AI-based framework is that the code can be run on CPU- or GPU-based systems without having to make any changes or re-implement code using CUDA, OpenCL or OpenACC. The writers of ML libraries such as TensorFlow [60], PyTorch [61] and JAX [62] have abstracted away platform-related code, meaning that it is very simple for the user to run their code on CPUs or GPUs. Furthermore, as the code uses functionality found in machine learning libraries, it can also be deployed on the latest energy-efficient AI accelerators such as those developed by GraphCore [63] and Cerebras [64], which have the potential for even greater computational speeds than GPUs. The C++ library Kokkos [65] operates in a similar vein to ML libraries, abstracting away code relating to the low-level memory access needs for different GPU architectures, enabling users to run their C++ codes on CPUs and GPUs without writing in CUDA, OpenCL or OpenACC themselves. Kokkos has been used to simulate boiling in two-phase flow [66] and two-phase-flow in a fluidised bed [67]. A second advantage of expressing the discretised governing equations as neural networks is the potential for seamless integration with machine learning workflows such as surrogate models or digital twins. For example, having a CFD-based solver written as a neural network will streamline physics-informed approaches, which minimise the residual of the governing equations whilst training a neural network [68,69]. In addition, the backpropagation functionality of machine learning libraries will allow the CFD model to be differentiated [70,71], facilitating data assimilation, control and inversion. The NN4PDEs approach has already been successfully applied to discover subsurface geology by inverting measurements of electrical resistivity [72].

The contribution of this paper is to extend, to multiphase flow problems, the NN4PDEs approach of solving discretised PDEs by capitalising on the functionality found within machine-learning libraries. The numerical schemes used should all be suitable for implementation in the AI-based framework proposed here, so a compressive algebraic VoF method is developed which maintains a sharp interface between the two fluids. The remaining text is divided as follows: the methodology is described in Section 2; results for a series of collapsing water column problems and rising bubble problems are presented in Section 3; and conclusions are drawn in Section 4.

2. Methodology

In Section 2.1, we outline the governing equations: the advection equation for the interface between the fluids, and the equations conserving mass and momentum. The idea underlying the NN4PDEs approach is introduced in Section 2.2 and demonstrated for some simple cases. Here, we also explain the notation used to express discretisations through filters of convolutional layers which is fundamental to the NN4PDEs approach. In Section 2.3, we describe the application of NN4PDEs to the equations governing multiphase flow, and give an outline of the interface-capturing scheme that detects oscillatory behaviour and the non-linear Petrov–Galerkin method that attempts to diffuse away oscillations. The final solution algorithm is presented in Section 2.4.

2.1. Governing equations

2.1.1. Advection equation for the volume fraction field

The advection equation that governs the scalar volume fraction field C (or indicator field) is given by:

$$\frac{\partial C}{\partial t} + \boldsymbol{q} \cdot \nabla C = 0, \tag{1}$$

where $\boldsymbol{q} = (u, v, w)^T$ represents the advection velocity. Here, two-phase flow is modelled, where a value of $C = 1$ indicates pure liquid and a value of $C = 0$ indicates a pure gas. The overall density, ρ, is:

$$\rho = C\rho_l + (1 - C)\rho_g, \tag{2}$$

with densities of the liquid and gas denoted by ρ_l and ρ_g respectively.

2.1.2. Navier–Stokes equations

The incompressible Navier–Stokes equations can be written as

$$\rho \left(\frac{\partial \boldsymbol{q}}{\partial t} + \boldsymbol{q} \cdot \nabla \boldsymbol{q} \right) + \sigma \boldsymbol{q} - \nabla \cdot (\mu \nabla \boldsymbol{q}) = -\nabla p + \boldsymbol{s}_q + \boldsymbol{s}_t, \tag{3a}$$

$$\nabla \cdot \boldsymbol{q} = 0, \tag{3b}$$

in which p denotes the pressure, σ is an absorption coefficient and μ is the dynamic viscosity. Buoyancy is represented by the term $\boldsymbol{s}_q = -\rho g \boldsymbol{e}_z$ where g is the acceleration due to gravity and $\boldsymbol{e}_z$ is a unit vector in the upwards vertical direction. The effects of surface tension are represented through a source term, $\boldsymbol{s}_t$, given by

$$\boldsymbol{s}_t = \sigma_t \kappa \nabla C, \tag{4}$$

where σ_t is the surface tension coefficient and the curvature is represented by κ. The curvature is a continuous volumetric field which represents the effective normal to the liquid–gas interface and is defined as follows:

$$\kappa = \nabla \cdot \boldsymbol{n}_t \quad \text{where} \quad \boldsymbol{n}_t = \frac{\nabla C}{||\nabla C||_2}. \tag{5}$$

This formulation is sometimes referred to as continuous surface force [15]. A consequence of Eqs. (1), (2) and (3b), is that the mass conservation equation holds:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{q}) = 0. \tag{6}$$

Dirichlet or Neumann boundary conditions are applied as needed, depending on the example being investigated.

2.2. Discretisation using convolutional layers

The NN4PDEs approach [49] is based on the idea that the matrix–vector multiplications associated with numerical discretisations can be written as discrete convolutions. Such operations are commonly used in the convolutional layers of neural networks, thus many numerical discretisations can be expressed using functions within machine learning libraries. See [54–56] for more details about CNNs.

2.2.1. Equivalence between numerical discretisations and discrete convolutions

Although finite element discretisations are used in this paper, finite difference discretisation schemes have a simpler mathematical form, so an example of writing finite difference discretisations as discrete convolutions is presented here. For the discretised isotropic diffusion operator (∇^2) in 3D, a second-order finite difference scheme has the following representation as a $3 \times 3 \times 3$ convolutional filter:

$$\mathbf{w}_{\nabla^2 \cdot, \cdot, -1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{-1}{(\Delta z)^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{w}_{\nabla^2 \cdot, \cdot, 0} = \begin{bmatrix} 0 & \frac{-1}{(\Delta y)^2} & 0 \\ \frac{-1}{(\Delta x)^2} & \frac{2}{(\Delta x)^2} + \frac{2}{(\Delta y)^2} + \frac{2}{(\Delta z)^2} & \frac{-1}{(\Delta x)^2} \\ 0 & \frac{-1}{(\Delta y)^2} & 0 \end{bmatrix}, \quad \mathbf{w}_{\nabla^2 \cdot, \cdot, 1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{-1}{(\Delta z)^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{7}$$

where Δx, Δy and Δz denote the grid spacing between the nodes in the x, y and z directions. The rank 3 tensor is shown here in three 'slices', labelled as -1, 0 and 1. For more details on finite difference discretisations, the reader is referred to Fletcher [73] and Linge and Langtangen [74]. Considering this diffusion operator in 2D for a uniform grid ($\Delta x = 1 = \Delta y$), Fig. 1 shows a stencil representing the discretised operator acting on a part of the solution field which calculates the value for cell Y_{33}. This application of the finite difference stencil is exactly equivalent to passing a filter, with weights as specified in Fig. 1, over an image, as done in the convolutional layer of a convolutional neural network. In the left of this figure, a halo region can be seen, which is made up of ghost cells that are represented by the white cells (cells ij where $i \in \{0, 6\}$, $j \in \{0, 1, \ldots, 6\}$ and $i \in \{0, 1, \ldots, 6\}$, $j \in \{0, 6\}$), through which boundary conditions can be applied. In machine learning terminology, this is known as padding, and is used to obtain a feature map of the same dimension as that of the image to which the filter is applied. In this way, numerical discretisations can be represented as convolutional layers of neural networks. Systems arising from this can be solved directly with a Jacobi method or implicitly using a multigrid solver implemented through a U-Net architecture [49,50].
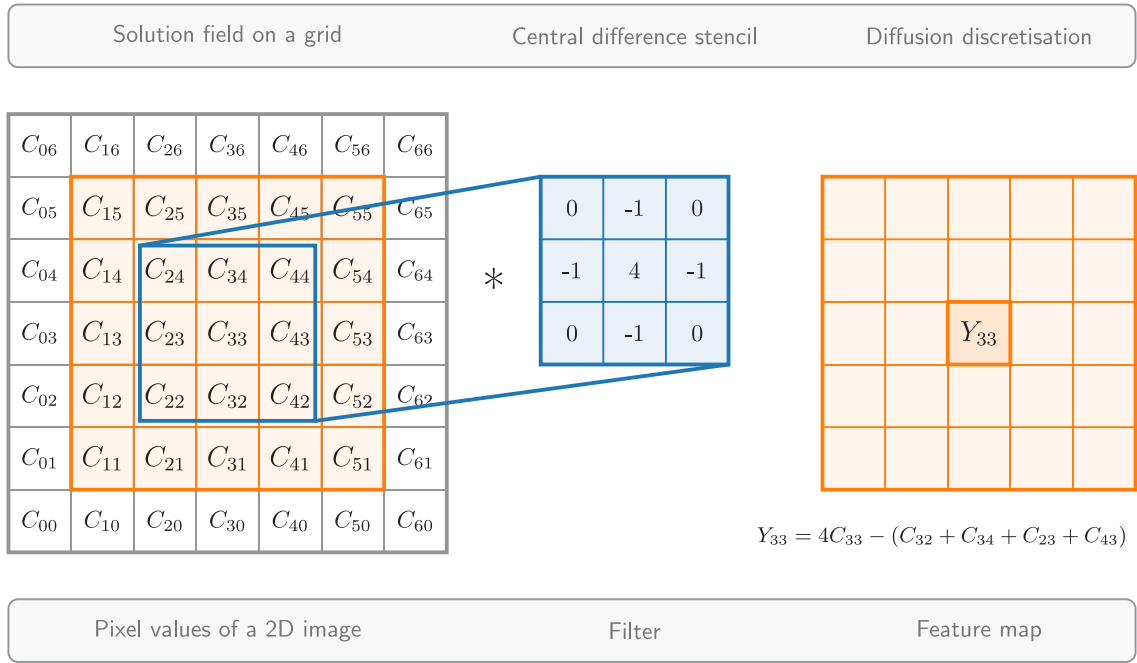
Fig. 1. On the left (in orange) is a 5 by 5 image with pixel values C_{ij} ($i, j \in \{1, 2, \ldots, 5\}$). A filter (in blue) of a convolutional layer, with weights as shown, is applied to the 3 by 3 part of the image centred around pixel C_{33}. The sum of the products of the weights and the pixel values determines the central value of the resulting feature map Y_{33} (in orange). Equivalently, on the left (in orange) is a 5 by 5 solution field C_{ij} ($i, j \in \{1, 2, \ldots, 5\}$) to which is applied the stencil of the diffusion operator discretised by finite differences (in blue). This results in the value Y_{33} (in orange). (The finite difference nodes are located at the centre of the cells.) Forming a halo around the inner domain are the white cells or ghost cells, through which boundary conditions can be applied. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2.2.2. Notation

The action of a discretisation stencil on a 3D tensor, C^n, can be written as follows:

$$f(C^n; \mathbf{w})\big|_{i,j,k} = \sum_{ii=-\ell}^{\ell} \sum_{jj=-\ell}^{\ell} \sum_{kk=-\ell}^{\ell} \mathbf{w}_{ii,jj,kk}\, C^n_{i+ii,j+jj,k+kk} \tag{8}$$

where $\mathbf{w}$ is of dimension $2\ell + 1$ by $2\ell + 1$ by $2\ell + 1$ and represents the coefficients associated with the numerical discretisation. This expression is equivalent to convolving a filter of weights $\mathbf{w}$ with an image, as is done in a convolutional layer, see page 128 in chapter 3 of [75] and also [53,76]. This notation is used throughout to represent the application of a filter with weights $\mathbf{w}$ to a tensor C^n which holds the values of a solution field at the nodes or grid points. The discretised differential operator is expressed through the weights and is applied to the discretised field. Generally, the tensor will have a padding of ℓ cells (through which boundary conditions are applied) and be of dimension $N_x + 2\ell$ by $N_y + 2\ell$ by $N_z + 2\ell$, but the output of the function can be written without the padding (N_x by N_y by N_z). The value of ℓ depends on the discretisation applied, see Section 2.3.3.

2.2.3. Finite element discretisation of first-order derivatives

Filters can also be used to represent finite element discretisations. The weights of the filter are calculated from integrals of products of basis functions (N_a) and their derivatives in the usual manner, *e.g.*,

$$\int_V N_a \frac{\partial N_b}{\partial x} dV, \quad \forall b, \qquad \text{or} \qquad \int_V N_{i,j,k} \frac{\partial N_{i',j',k'}}{\partial x} dV, \quad \forall \left(i', j', k'\right). \tag{9}$$

in which the finite element nodes a and b are replaced with the equivalent tensor indices (i, j, k) and $\left(i', j', k'\right)$ respectively. For every node away from the boundary and, with every element being of quadrilateral shape and of the same size (*i.e.*, regular structured quadrilateral elements), these matrix entries are the same for each node and written as the weights of a filter of a convolutional layer:

$$\mathbf{w}_{\mathbf{x}\,ii,jj,kk} = \int_V N_{i,j,k} \frac{\partial N_{i',j',k'}}{\partial x} dV \quad \text{with} \quad ii = i' - i, \; jj = j' - j, \; kk = k' - k. \tag{10}$$

Due to the compact support of the FEM basis functions, these matrix entries are only non-zero for indices $ii \in \{-\ell, \ldots, \ell\}$, $jj \in \{-\ell, \ldots, \ell\}$, $kk \in \{-\ell, \ldots, \ell\}$ and they can be replaced by a $2\ell + 1$ by $2\ell + 1$ by $2\ell + 1$ tensor $\mathbf{w_x}$ or $\mathbf{w}_{\mathbf{x}\,ii,jj,kk}$. For linear

elements $\ell = 1$, quadratic elements $\ell = 2$ and cubic elements $\ell = 3$. Thus, the weights of a linear FEM discretisation of the derivative with respect to x, $\mathbf{w_x}$, are defined as

$$\mathbf{w}_{\mathbf{x}\cdot,\cdot,-1} = \frac{m_l}{72\Delta x}\begin{bmatrix} -1 & 0 & 1 \\ -4 & 0 & 4 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathbf{w}_{\mathbf{x}\cdot,\cdot,0} = \frac{m_l}{72\Delta x}\begin{bmatrix} -4 & 0 & 4 \\ -16 & 0 & 16 \\ -4 & 0 & 4 \end{bmatrix}, \quad \mathbf{w}_{\mathbf{x}\cdot,\cdot,1} = \frac{m_l}{72\Delta x}\begin{bmatrix} -1 & 0 & 1 \\ -4 & 0 & 4 \\ -1 & 0 & 1 \end{bmatrix}, \tag{11}$$

where $m_l = \Delta x \Delta y \Delta z$.

A challenge arising during the design of quadratic or higher order FE methods within the NN4PDEs approach, is that the coefficients of the stencil are different at each node. For convolutional layers, the same filter is applied to each pixel in the image. In order to address this, a new FEM has been devised known as the Convolutional Finite Element Method (ConvFEM), which has the same coefficients at each node in the mesh or grid. This greatly simplifies the implementation of higher-order FE discretisations using the filters of convolutional layers on structured grids. The basic idea in forming the ConvFEM filters is to sum coefficients of the stencils and take the average (see Phillips et al. [51] for further details). A number of FE discretisations are applied in this paper, thus filters are tensors of shape of either $3 \times 3 \times 3$ for linear FE, $5 \times 5 \times 5$ for quadratic ConvFEM elements, or $7 \times 7 \times 7$ for cubic ConvFEM elements. These filters are listed in the GitHub repository by Phillips [77], along with the computer code that automatically generates them.

2.2.4. Finite-element based discretisation of second-order derivatives

Second-order derivatives can be implemented in the same way as first-order derivatives, with the coefficients corresponding to the numerical discretisation making up the weights of the filters, $\mathbf{w_{xx}}$, $\mathbf{w_{yy}}$ and $\mathbf{w_{zz}}$. In this work, any stabilising terms (such as those derived from Petrov–Galerkin methods) also involve second-order spatial derivatives. Considering the second derivative in the x direction of field C^n with diffusion coefficient k_C^n, its typical form can be expanded as three terms, all involving the Laplacian operator,

$$
\begin{aligned}
-\frac{\partial}{\partial x}\left(k_C^n \frac{\partial C^n}{\partial x} \right) &= -\frac{1}{2}\left(\frac{\partial^2 (k_C^n C^n)}{\partial x^2} + k_C^n \frac{\partial^2 C^n}{\partial x^2} - C^n \frac{\partial^2 k_C^n}{\partial x^2} \right) &&\text{(Continuum Form)} \\
&\sim \frac{1}{2}\left(f(k_C^n \odot C^n; \mathbf{w_{xx}}) + k_C^n \odot f(C^n; \mathbf{w_{xx}}) - C^n \odot f(k_C^n; \mathbf{w_{xx}}) \right) \\
&=: f^{\nabla^2}(C^n, k_C^n; \mathbf{w_{xx}}), &&\text{(Discretised Form)}
\end{aligned}
\tag{12}
$$

where $\mathbf{w_{xx}}$ contains weights associated with a particular discretisation of the second derivative (as yet unspecified), $k_C{}^n$ is the tensor containing the diffusion coefficients at each node of the grid at time level n, and $\odot$ is the Hadamard product (also known as Schur product) which represents entry-wise multiplication. Eq. (12) presents a very convenient way of writing terms involving second derivatives of the form seen in the left-hand side of Eq. (12) for implementation using convolutional layers, as it only requires the use of a discretised Laplacian (or associated filter) to form the diffusion operator in the case of varying diffusion coefficients. For isotropic diffusion (in three directions) this equation becomes:

$$
\begin{aligned}
-\nabla \cdot \left(k_C^n \nabla C^n \right) &= -\frac{1}{2}\left(\nabla^2(k_C^n C^n) + k_C^n \nabla^2 C^n - C^n \nabla^2 k_C^n \right) &&\text{(Continuum Form)} \\
&\sim \frac{1}{2}\left(f(k_C^n \odot C^n; \mathbf{w}_{\nabla^2}) + k_C^n \odot f(C^n; \mathbf{w}_{\nabla^2}) - C^n \odot f(k_C^n; \mathbf{w}_{\nabla^2}) \right) \\
&= f^{\nabla^2}(C^n, k_C^n; \mathbf{w}_{\nabla^2}), &&\text{(Discretised Form)}
\end{aligned}
\tag{13}
$$

in which the filter $\mathbf{w}_{\nabla^2}$ contains weights associated with a particular discretisation of isotropic diffusion. These expressions will be used, in the following sections, to represent viscous and stabilisation terms that are determined from a non-linear Petrov–Galerkin discretisation.

2.3. Discretisation of the governing equations using convolutional layers

We now discretise multiphase flow equations in space and time with FEM, and express these discretisations as convolutions using the notation introduced in Section 2.2.2.

2.3.1. Advection equation for the volume fraction field

The equation governing the volume fraction field, Eq. (1), is discretised in time using forward Euler time-stepping:

$$\frac{C^{n+1} - C^n}{\Delta t} + q^{n+1} \cdot \nabla C^n = 0, \tag{14}$$

where C^n is the volume fraction field at time level n, q^n is the velocity field at time level n and Δt is the time step. The forward Euler time-stepping method has been chosen because it has a negative diffusion coefficient in the truncation error which compresses the solution slightly [32,33]. Using the filters derived in Section 2.2.4, Eq. (14) can be discretised in space as

$$r_C^{n+\frac{1}{2}} = f\left(\frac{C^{n+1} - C^n}{\Delta t}; \mathbf{w_{ml}} \right) + \tilde{s}_C^{n+\frac{1}{2}}, \tag{15}$$

with $\quad \tilde{s}_C^{n+\frac{1}{2}} = u^{n+1} \odot f\left(C^n; \mathbf{w_x} \right) + v^{n+1} \odot f\left(C^n; \mathbf{w_y} \right) + w^{n+1} \odot f\left(C^n; \mathbf{w_z} \right)$

$$+ f^{\nabla^2}\left(C^n; k_C^{x\,n}, \mathbf{w_{xx}^{PG}}\right) + f^{\nabla^2}\left(C^n; k_C^{y\,n}, \mathbf{w_{yy}^{PG}}\right) + f^{\nabla^2}\left(C^n; k_C^{z\,n}, \mathbf{w_{zz}^{PG}}\right),$$

(16)

where f denotes the convolution of a filter with the first argument of the function; $\mathbf{w_{ml}}$ represents a lumped mass tensor whose only non-zero entry corresponds to $i = 0$, $j = 0$, $k = 0$ and has value $\Delta x \Delta y \Delta z$; u^{n+1}, v^{n+1} and w^{n+1} are tensors which represent the three velocity components on the structured grid; the first-order discretised differential operators are represented by a convolutional layer with weights $\mathbf{w_x}$, $\mathbf{w_y}$ and $\mathbf{w_z}$. Finally, filters with weights from a linear $3 \times 3 \times 3$ FEM discretisation for calculating the second-order derivatives are represented by $\mathbf{w_{xx}^{PG}}$, $\mathbf{w_{yy}^{PG}}$, and $\mathbf{w_{zz}^{PG}}$. These second derivative terms provide diffusion (based on the Petrov–Galerkin method) and are included to reduce the oscillations that can occur when central-difference or FEM-based discretisations are applied to an advection term (see Sections 2.3.4 and 2.3.5).

2.3.2. Navier–Stokes equations

The momentum equation, Eq. (3a), is discretised in time using a two-level central difference scheme similar to the Crank–Nicolson time-stepping method, however σ and the pressure terms are discretised using backward Euler time-stepping, resulting in:

$$\rho^n\left(\frac{\tilde{q}^{n+1} - q^n}{\Delta t} + u^{n+\frac{1}{2}}\frac{\partial q^{n+\frac{1}{2}}}{\partial x} + v^{n+\frac{1}{2}}\frac{\partial q^{n+\frac{1}{2}}}{\partial y} + w^{n+\frac{1}{2}}\frac{\partial q^{n+\frac{1}{2}}}{\partial z}\right) + \sigma\tilde{q}^{n+1} - \nabla\cdot(\mu\nabla q^{n+\frac{1}{2}})$$

$$= -\nabla p_{hs}^{n+1} - \nabla\tilde{p}_{nh}^{n+1} + s_q^n + s_t^n.$$

(17)

The buoyancy and surface tension forces are represented by s_q^n and s_t^n respectively, and the pressure has been split into two terms: a term which balances hydrostatic and surface tension forces (p_{hs}^{n+1}), and a non-hydrostatic non-surface-tension force component $(\tilde{p}_{nh}^{n+1})$. The tilde sign above a variable indicates that we will use the best approximation currently available for that variable. Eq. (17) can then be discretised in space using a Petrov–Galerkin method [78], with mass lumping for the advection, time and absorption terms. Using the filter notation of Section 2.2.2, this can be expressed as:

$$r_u^{n+\frac{1}{2}} = f\left(\rho^n \odot \left(\frac{\tilde{u}^{n+1} - u^n}{\Delta t}\right) + \sigma \odot \tilde{u}^{n+1}; \mathbf{w_{ml}}\right) + \tilde{s}_u^{n+\frac{1}{2}} \quad \text{with}$$

$$\tilde{s}_u^{n+\frac{1}{2}} = \rho^n \odot \left(u^{n+\frac{1}{2}} \odot f(u^{n+\frac{1}{2}}; \mathbf{w_x}) + v^{n+\frac{1}{2}} \odot f(u^{n+\frac{1}{2}}; \mathbf{w_y}) + w^{n+\frac{1}{2}} \odot f(u^{n+\frac{1}{2}}; \mathbf{w_z})\right)$$

(18a)

$$+ f^{\nabla^2}(u^{n+\frac{1}{2}}, k_u^{n+\frac{1}{2}}; \mathbf{w_{\nabla^2}}) + f(p_{hs}^{n+1} + \tilde{p}_{nh}^{n+1}; \mathbf{w_x}) - f(s_u^n + s_{tu}^n; \mathbf{w_m}),$$

$$r_v^{n+\frac{1}{2}} = f\left(\rho^n \odot \left(\frac{v^{n+1} - v^n}{\Delta t}\right) + \sigma \odot \tilde{v}^{n+1}; \mathbf{w_{ml}}\right) + \tilde{s}_v^{n+\frac{1}{2}} \quad \text{with}$$

$$\tilde{s}_v^{n+\frac{1}{2}} = \rho^n \odot \left(u^{n+\frac{1}{2}} \odot f(v^{n+\frac{1}{2}}; \mathbf{w_x}) + v^{n+\frac{1}{2}} \odot f(v^{n+\frac{1}{2}}; \mathbf{w_y}) + w^{n+\frac{1}{2}} \odot f(v^{n+\frac{1}{2}}; \mathbf{w_z})\right)$$

(18b)

$$+ f^{\nabla^2}(v^{n+\frac{1}{2}}, k_v^{n+\frac{1}{2}}; \mathbf{w_{\nabla^2}}) + f(p_{hs}^{n+1} + \tilde{p}_{nh}^{n+1}; \mathbf{w_y}) - f(s_v^n + s_{tv}^n; \mathbf{w_m}),$$

$$r_w^{n+\frac{1}{2}} = f\left(\rho^n \odot \left(\frac{\tilde{w}^{n+1} - w^n}{\Delta t}\right) + \sigma \odot \tilde{w}^{n+1}; \mathbf{w_{ml}}\right) + \tilde{s}_u^{n+\frac{1}{2}} \quad \text{with}$$

$$\tilde{s}_w^{n+\frac{1}{2}} = \rho^n \odot \left(u^{n+\frac{1}{2}} \odot f(w^{n+\frac{1}{2}}; \mathbf{w_x}) + v^{n+\frac{1}{2}} \odot f(w^{n+\frac{1}{2}}; \mathbf{w_y}) + w^{n+\frac{1}{2}} \odot f(w^{n+\frac{1}{2}}; \mathbf{w_z})\right)$$

(18c)

$$+ f^{\nabla^2}(w^{n+\frac{1}{2}}, k_w^{n+\frac{1}{2}}; \mathbf{w_{\nabla^2}}) + f(p_{hs}^{n+1} + \tilde{p}_{nh}^{n+1}; \mathbf{w_z}) - f(s_w^n + s_{tw}^n; \mathbf{w_m}),$$

in which $\mathbf{w_m}$ is the consistent mass filter. When solving the discretised Navier–Stokes equations the aim is to minimise the residual, that is, to obtain: $r_u^{n+\frac{1}{2}} = 0$, $r_v^{n+\frac{1}{2}} = 0$, $r_w^{n+\frac{1}{2}} = 0$. These equations are solved using a two-step predictor–corrector time level approach, in order to obtain better estimates of the velocity variables at time level $n + \frac{1}{2}$. In Eqs. (18), the density at time level n is given by:

$$\rho^n = \rho_l C^n + \rho_g(1 - C^n),$$

(19)

where $\mathbf{1}$ is a tensor in which each entry is unity $(\text{that is } \mathbf{1}|_{i,j,k} = 1)$ and similarly, $\mathbf{0}$ represents the zero tensor (every entry is 0). There are three buoyancy tensors, one of which is non-zero, as determined by the direction of gravity

$$s_u^n = \mathbf{0}, \qquad s_v^n = \mathbf{0}, \qquad s_w^n = -g\,\rho^n.$$

(20)

On discretising the surface tension given in Eq. (4) we obtain:

$$s_{tu}^n = \sigma_t \kappa^n \odot C_x^n, \qquad s_{tv}^n = \sigma_t \kappa^n \odot C_y^n, \qquad s_{tw}^n = \sigma_t \kappa^n \odot C_z^n, \tag{21}$$

where the curvature tensor is represented by κ^n. In order to calculate the curvature, we take the gradient of the normal to the interface using filters as follows

$$\kappa^n = f\left(f(n_{tu}^n; \mathbf{w_x}) + f(n_{tv}^n; \mathbf{w_y}) + f(n_{tw}^n; \mathbf{w_v}); \mathbf{w_{ml}^{-1}}\right), \tag{22}$$

in which $\mathbf{w_{ml}^{-1}}$ is the inverse of the lumped mass filter and where the tensors representing the normal are given by

$$
\begin{aligned}
n_{tu}^n &= C_x^n \oslash (\epsilon_t \mathbf{1} + l^n), \\
n_{tv}^n &= C_y^n \oslash (\epsilon_t \mathbf{1} + l^n), \quad \text{with} \quad l_{i,j,k}^n = \sqrt{(C_{xi,j,k}^n)^2 + (C_{yi,j,k}^n)^2 + (C_{zi,j,k}^n)^2} \quad \text{and} \quad \epsilon_t = \frac{10^{-7}}{h}, \\
n_{tw}^n &= C_z^n \oslash (\epsilon_t \mathbf{1} + l^n),
\end{aligned}
\tag{23}
$$

where h is the typical grid spacing and $\oslash$ is the Hadamard entry-wise division of equally sized tensors, *i.e.*,

$$\hat{c} = \hat{a} \oslash \hat{b} \iff \hat{c}_{i,j,k} = \frac{\hat{a}_{i,j,k}}{\hat{b}_{i,j,k}}.$$

Spurious currents can be generated when the capillary number is low [79], which is mitigated by introducing dissipation to the volume fraction field, done here through a Petrov–Galerkin formulation (see Section 2.3.4).

2.3.3. Application of boundary conditions

The boundary conditions are applied through the halo nodes using padding (see Fig. 1). This means that the discretisation scheme is the same wherever it is applied, whether this is next to the boundary or within the domain, thereby simplifying the implementation of the method. These halos are one node thick for linear elements, two nodes thick for quadratic elements and three nodes thick for cubic elements. Dirichlet boundary conditions are applied by setting the values of the halo nodes to equal the desired value. When applying a specified normal derivative, the value at the halo nodes is calculated by extrapolating from the nearest node within the domain using the specified gradient. Most commonly, a zero derivative is applied, in which case, this simplifies to copying the nearest nodal value within the domain to the halo nodes at least for linear ConvFEM. See Fig. 2 for further details of how the boundary conditions are applied.

Thus, for incoming flows one might specify the values of $C_{i,j,k}^n$ at the halo nodes based on the desired Dirichlet boundary conditions and for boundaries with outgoing flows one might apply a zero derivative boundary condition to $C_{i,j,k}^n$. For incoming velocities, the Dirichlet velocity components are specified at the halo nodes and for outgoing flows (outlet boundary), a zero normal-derivative boundary condition is applied by copying the nodal values at the boundary to the halo nodes for the velocity component normal to the boundary. For non-hydrostatic non-surface-tension pressure, a zero derivative boundary condition is applied on all boundaries other than the outlet boundary, at which a zero value is specified. For hydrostatic and surface-tension pressure, p_{hs}, a zero normal-derivative boundary condition is applied on the lateral (side) boundaries and a zero value is applied at the top boundary of the domain. For the bottom boundary, the pressure gradient $\frac{\partial p_{hs}}{\partial z} = -\rho g$ is enforced, in which the density ρ from the nearest node within the domain is used. For the hydrostatic and surface-tension pressure, and the non-hydrostatic non-surface-tension pressure correction steps, the boundary conditions just described are applied at each of the levels within the multigrid method, see Phillips et al. [51].

2.3.4. Isotropic non-linear Petrov–Galerkin method

We apply an isotropic non-linear Petrov–Galerkin method to both the velocity components and the volume fraction field in order to reduce the oscillations that would otherwise occur, as a result of applying a high-order discretisation, see Godunov's theorem [80]. Introducing non-linearity enables us to control the oscillations. Based on Donéa and Huerta [78, page 185], the approach outlined in this section will calculate diffusion coefficients to be used for volume fraction and velocities that are calculated in Eqs. (15), (18a), (18b) and (18c). Using three methods, we calculate three diffusion coefficients, and select the one with the lowest value at each grid point. Suppose that Ψ^n is either the volume fraction field C^n or a velocity component (either u^n, v^n or w^n). Then, according to Codina [81], if the discretised derivatives of Ψ^n with respect to x, y and z are written as $\Psi_x^n = f(\Psi^n; \mathbf{w_x})$, $\Psi_y^n = f(\Psi^n; \mathbf{w_y})$ and $\Psi_z^n = f(\Psi^n; \mathbf{w_z})$, the first residual-based diffusion coefficient is given by:

$$k_{abs}^n = \alpha_{kabs} |a_\Psi^n| \odot h_\Psi^n \oslash (\epsilon_k \mathbf{1} + \tfrac{1}{3}(|\Psi_x^n| + |\Psi_y^n| + |\Psi_z^n|)), \tag{24}$$

where ϵ_k is a small positive number added to the term in the divisor, so that the diffusion coefficients do not become too large (in this paper, 10^{-7} is used); and a_Ψ^n represents the residual of the advection part of the governing equation. For the general case where the grid spacing is different in different directions, the lengthscale associated with the diffusion is given by

$$h_\Psi^n = \left(\Delta x |\Psi_x^n| + \Delta y |\Psi_y^n| + \Delta z |\Psi_z^n|\right) \oslash \left(\epsilon_k \mathbf{1} + |\Psi_x^n| + |\Psi_y^n| + |\Psi_z^n|\right), \tag{25}$$

and the expression for the first diffusion coefficient becomes

$$k_{abs}^n = \alpha_{kabs} |a_\Psi^n| \odot (\Delta x |\Psi_x^n| + \Delta y |\Psi_y^n| + \Delta z |\Psi_z^n|) \oslash (\epsilon_k \mathbf{1} + \tfrac{1}{3}(|\Psi_x^n| + |\Psi_y^n| + |\Psi_z^n|)^2). \tag{26}$$

(a) Symmetry boundary conditions

(b) Combined symmetry and wall boundary conditions (zero velocity and Dirichlet $C = B$, for volume fraction C)

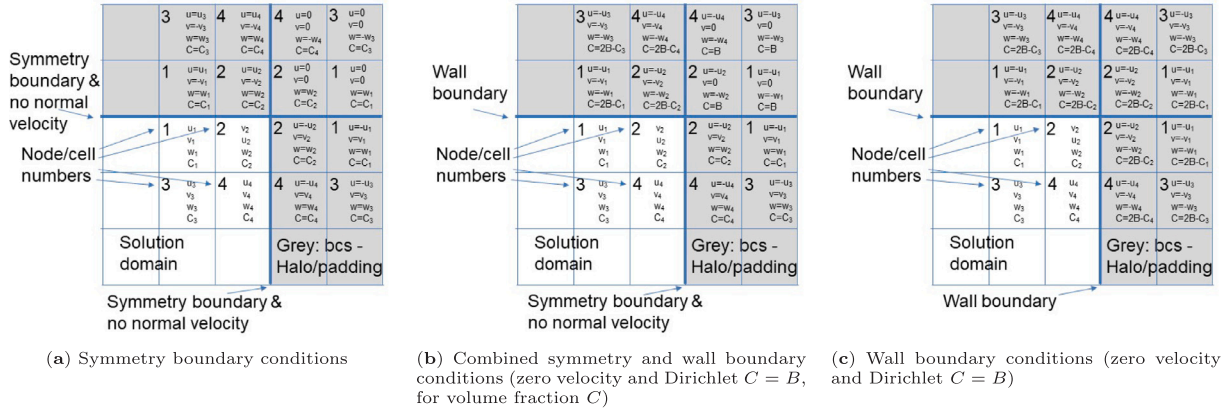(c) Wall boundary conditions (zero velocity and Dirichlet $C = B$)

Fig. 2. Diagrams showing how boundary conditions are applied through halo values when using the quadratic ConvFEM. Linear ConvFEM can be realised simply by ignoring the second layer of halo values. The indices 1, 2, 3 and 4 indicate the cell or node number within the solution domain. These same indices are used in the halo nodes to indicate where the information for forming the halo values is obtained. The basic idea behind the boundary conditions is to specify the value of the halo values such that when one interpolates these with the values inside the domain then one obtains the desired boundary conditions along the boundaries (indicated here by thick blue lines). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

For the case when the grid spacing is the same in all three directions, Eq. (25) simplifies to $h_{\Psi}^{n} = \Delta x \mathbf{1}$ resulting in

$$
k_{abs}^{n} = \alpha_{kabs} \Delta x |a_{\Psi}^{n}| \oslash \left(\epsilon_k \mathbf{1} + \frac{1}{3}(|\Psi_x^{n}| + |\Psi_y^{n}| + |\Psi_z^{n}|) \right). \tag{27}
$$

Taking the expression from Hansbo and Johnson [82], the second residual-based diffusion coefficient is given by:

$$
\begin{aligned}
k_{square}^{n} &= \alpha_{ksquare} (a_{\Psi}^{n})^2 \odot h_{\Psi}^{n} \\
&\quad \oslash \left(\epsilon_k \mathbf{1} + \frac{1}{3}(|u^{n} \odot \Psi_x^{n}| + |v^{n} \odot \Psi_y^{n}| + |w^{n} \odot \Psi_z^{n}|)((\Psi_x^{n})^2 + (\Psi_y^{n})^2 + (\Psi_z^{n})^2) \right),
\end{aligned} \tag{28}
$$

where $(a_{\Psi}^{n})^2 = a_{\Psi}^{n} \odot a_{\Psi}^{n}$ and $|\Psi_x^{n}|$ is a tensor in which the absolute value of each entry of Ψ_x^{n} is taken. The coefficients α_{kabs}, $\alpha_{ksquare}$ have been determined by trial and error, and are defined as:

$$
\alpha_{kabs} = \frac{1}{8 \times 2^{p}} \qquad \text{and} \qquad \alpha_{ksquare} = \frac{8}{2^{p}},
$$

where p is the polynomial order of the finite element expansion, *i.e.*, p = 1 for linear $3 \times 3 \times 3$ filters, p = 2 for quadratic $5 \times 5 \times 5$ filters and p = 3 for cubic filters.

To calculate the advection residual, a_{Ψ}^{n}, which appears in Eqs. (24) and (28), a simple mixed mass approach is used

$$
a_{\Psi}^{n} = \alpha_a f \left(u^{n} \odot f(\Psi^{n}; \mathbf{w}_x) + v^{n} \odot f(\Psi^{n}; \mathbf{w}_y) + w^{n} \odot f(\Psi^{n}; \mathbf{w}_z); \frac{1}{m_l} \mathbf{w_m} - \mathbf{E} \right), \tag{29}
$$

where $m_l = \Delta x \Delta y \Delta z$ is the lumped mass term, $\mathbf{w_m}$ filter representing the consistent mass matrix and $\mathbf{E}$ is the "identity filter" (on convolving this with a tensor produces the same tensor). A scalar coefficient, α_a, is introduced, for which a value of $\alpha_a = 2$ has been found effective. The underlying assumption, in using Eq. (29) for calculating the residual tensor, is that most of the errors in the discretisation are within the spatial gradients associated with the transport terms, which, given the dominance of these terms in the applications presented here, is a reasonable assumption.

Finally, the rate of diffusion should not exceed the rate of advection, so we introduce a maximum diffusion coefficient calculated as follows:

$$
k_{max}^{n} = \Delta x |u^{n}| + \Delta y |v^{n}| + \Delta z |w^{n}|. \tag{30}
$$

Assuming that all the diffusion coefficients given by Eqs. (24), (28), (30) are conservatively large, one can determine the diffusion coefficient using:

$$
k_{\Psi}^{n} = \rho_{\Psi}^{n} \odot \min\{k_{max}^{n}, k_{abs}^{n}, k_{square}^{n}\} + \mu \mathbf{1}, \tag{31}
$$

$$
\text{or} \quad k_{\Psi i,j,k}^{n} = \rho_{\Psi i,j,k}^{n} \min\{k_{max\,i,j,k}^{n}, k_{abs\,i,j,k}^{n}, k_{square\,i,j,k}^{n}\} + \mu. \tag{32}
$$

At each grid point (*i.e.*, for each i, j, k) the minimum of the three diffusion coefficients is taken. When solving for the velocity components $\rho_{\Psi}^{n} = \rho^{n}$ and when solving for the volume fraction field, $\rho_{\Psi}^{n} = 1$. Also, if Ψ^{n} is a velocity component, μ is the dynamic viscosity and if Ψ^{n} is the volume fraction field, $\mu = 0$ is the diffusivity. By taking the minimum value of these diffusion coefficients (Eq. (31)), it is hoped that the scheme will minimally intervene in the discretisation.

2.3.5. Compressive advection for interface-capturing with Petrov–Galerkin and extrema detecting for the volume fraction field

The positive Petrov–Galerkin diffusion can result in a smeared interface, so to counteract this, we introduce negative diffusion through another Petrov–Galerkin-based formulation. We use the approach outlined in Pavlidis et al. [33] (Equation (33) in that paper) to sharpen up interfaces by adding negative diffusion to the advection equation for the volume fraction when this field has no extrema in a particular direction, which is determined by an oscillatory detecting variable.

We now outline the new approach used to detect oscillations. Variables detecting oscillations in each direction with increasing x, y and z, are calculated for linear $3 \times 3 \times 3$ filters $S^{x\,n}_{i,j,k}$, $S^{y\,n}_{i,j,k}$, $S^{z\,n}_{i,j,k}$ as follows:

$$S^{x\,n}_{i,j,k} = (C^n_{i,j,k} - C^n_{i-1,j,k})(C^n_{i+1,j,k} - C^n_{i,j,k}), \tag{33a}$$

$$S^{y\,n}_{i,j,k} = (C^n_{i,j,k} - C^n_{i,j-1,k})(C^n_{i,j+1,k} - C^n_{i,j,k}), \tag{33b}$$

$$S^{z\,n}_{i,j,k} = (C^n_{i,j,k} - C^n_{i,j,k-1})(C^n_{i,j,k+1} - C^n_{i,j,k}). \tag{33c}$$

When quadratic and higher-order filters are used, the following oscillatory-detecting variables are calculated:

$$S^{x\,n}_{i,j,k} = \min\{ \quad (C^n_{i-1,j,k} - C^n_{i-2,j,k})(C^n_{i,j,k} - C^n_{i-1,j,k}),$$
$$(C^n_{i,j,k} - C^n_{i-1,j,k})(C^n_{i+1,j,k} - C^n_{i,j,k}), \tag{34a}$$
$$(C^n_{i+1,j,k} - C^n_{i,j,k})(C^n_{i+2,j,k} - C^n_{i+1,j,k}) \quad \},$$

$$S^{y\,n}_{i,j,k} = \min\{ \quad (C^n_{i,j-1,k} - C^n_{i,j-2,k})(C^n_{i,j,k} - C^n_{i,j-1,k}),$$
$$(C^n_{i,j,k} - C^n_{i,j-1,k})(C^n_{i,j+1,k} - C^n_{i,j,k}), \tag{34b}$$
$$(C^n_{i,j+1,k} - C^n_{i,j,k})(C^n_{i,j+2,k} - C^n_{i,j+1,k}) \quad \},$$

$$S^{z\,n}_{i,j,k} = \min\{ \quad (C^n_{i,j,k-1} - C^n_{i,j,k-2})(C^n_{i,j,k} - C^n_{i,j,k-1}),$$
$$(C^n_{i,j,k} - C^n_{i,j,k-1})(C^n_{i,j,k+1} - C^n_{i,j,k}), \tag{34c}$$
$$(C^n_{i,j,k+1} - C^n_{i,j,k})(C^n_{i,j,k+2} - C^n_{i,j,k+1}) \quad \}.$$

The condition $S^{x\,n}_{i,j,k} \geqslant 0$ indicates that there is no change of sign in gradient and thus no oscillation; $S^{x\,n}_{i,j,k} < 0$ indicates that there is an oscillation. Similar statements can be made for $S^{y\,n}_{i,j,k}$ and $S^{z\,n}_{i,j,k}$. Eqs. (34a), (34b) and (34c) provide a much more rigorous approach to detecting an oscillation, locally, in a particular direction than the approach used in Eqs. (33a), (33b) and (33c). In fact, the use of the latter can lead to significant oscillations when diffusion is introduced. In addition to using $S^{x\,n}_{i,j,k} < 0$ for detecting an oscillation (with similar terms for y and z), if $C^n_{i,j,k} > 0.95$ or $C^n_{i,j,k} < 0.05$, then an oscillation is also deemed to have been detected and thus the algorithm will apply positive Petrov–Galerkin diffusion and does this by setting $S^{x\,n}_{i,j,k} = -1$ $S^{y\,n}_{i,j,k} = -1$, $S^{z\,n}_{i,j,k} = -1$. These modifications to $S^{x\,n}_{i,j,k}$ $S^{y\,n}_{i,j,k}$, $S^{z\,n}_{i,j,k}$ reduce the likelihood of the volume fraction field becoming greater than one or less than zero. The weights that determine how much positive (*e.g.*, $w^{x\,n+}_{i,j,k}$) and how much negative (*e.g.*, $w^{x\,n-}_{i,j,k}$) diffusion is applied, are calculated from:

$$w^{x\,n+}_{i,j,k} = -\max\{-1, \min\{0, \mathcal{K}^+ S^{x\,n}_{i,j,k}\}\}, \qquad\qquad w^{x\,n-}_{i,j,k} = \min\{1, \max\{0, \mathcal{K}^- S^{x\,n}_{i,j,k}\}\}, \tag{35a}$$

$$w^{y\,n+}_{i,j,k} = -\max\{-1, \min\{0, \mathcal{K}^+ S^{y\,n}_{i,j,k}\}\}, \qquad\qquad w^{y\,n-}_{i,j,k} = \min\{1, \max\{0, \mathcal{K}^- S^{y\,n}_{i,j,k}\}\}, \tag{35b}$$

$$w^{z\,n+}_{i,j,k} = -\max\{-1, \min\{0, \mathcal{K}^+ S^{z\,n}_{i,j,k}\}\}, \qquad\qquad w^{z\,n-}_{i,j,k} = \min\{1, \max\{0, \mathcal{K}^- S^{z\,n}_{i,j,k}\}\}. \tag{35c}$$

The values of $\mathcal{K}^+ = 300$ and $\mathcal{K}^- = 300$ are used here which tend to mean that these weightings produce their extreme values of 0 and 1 most of the time. The negative diffusion coefficient used to maintain a sharp interface [32,33] is:

$$k^{n-}_{i,j,k} = -\beta \frac{u^2_{i,j,k} + v^2_{i,j,k} + w^2_{i,j,k}}{\epsilon_x + ((C^n_{x\,i,j,k})^2 + (C^n_{y\,i,j,k})^2 + (C^n_{z\,i,j,k})^2)(|u^{n*}_{i,j,k}| + |v^{n*}_{i,j,k}| + |w^{n*}_{i,j,k}|)/3}, \tag{36}$$

where $\beta = 0.1$; $\epsilon_x = 10^{-4}$; $C^n_{x\,i,j,k}$, $C^n_{y\,i,j,k}$ and $C^n_{z\,i,j,k}$ represent the gradient of volume fraction field in each direction at time level n (*e.g.*, $C^n_{x\,i,j,k} = C^n_{\boldsymbol{x}}|_{i,j,k}$); and the $*$ superscript denotes the normalisation operator for each velocity component which is defined by

$$(u^{n*}_{i,j,k}, v^{n*}_{i,j,k,j,k}, w^{n*}_{i,j,k}) = (C^n_{x\,i,j,k}, C^n_{y\,i,j,k}, C^n_{z\,i,j,k}) \left(\frac{u^n_{i,j,k} C^n_{x\,i,j,k} + v^n_{i,j,k} C^n_{y\,i,j,k} + w^n_{i,j,k} C^n_{z\,i,j,k}}{\epsilon_x + (C^n_{x\,i,j,k})^2 + (C^n_{y\,i,j,k})^2 + (C^n_{z\,i,j,k})^2} \right). \tag{37}$$

This negative diffusion coefficient is used to generate negative diffusion coefficients in x, y and z as follows:

$$k^{x\,n-}_{i,j,k} = \frac{k^{n-}_{i,j,k}}{\varDelta x}, \qquad k^{y\,n-}_{i,j,k} = \frac{k^{n-}_{i,j,k}}{\varDelta y}, \qquad k^{z\,n-}_{i,j,k} = \frac{k^{n-}_{i,j,k}}{\varDelta z}. \tag{38}$$

An isotropic positive diffusion coefficient is also applied to reduce oscillations. To do this, Eq. (26) is used to obtain $k^n_{abs\,i,j,k}$ which is multiplied by $\mathcal{K}$ to form:

$$k^{x\,n+}_{i,j,k} = k^{y\,n+}_{i,j,k} = k^{z\,n+}_{i,j,k} = \mathcal{K} k^n_{abs\,i,j,k}, \tag{39}$$
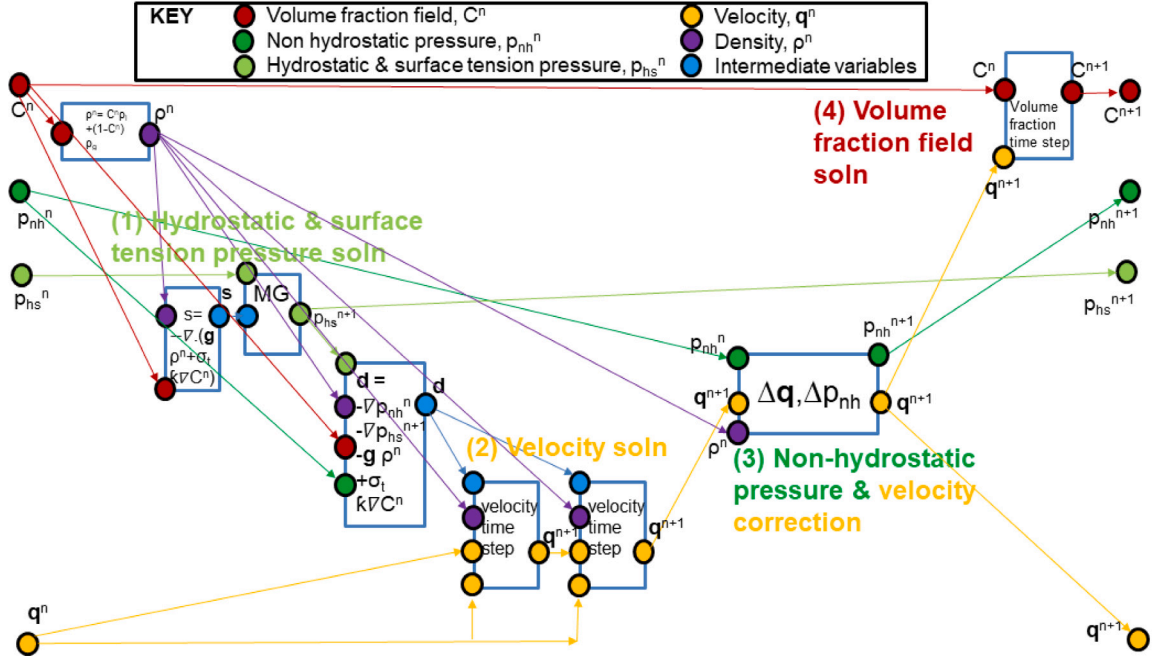
Fig. 3. Schematic of the multiphase flow solver as a neural network containing the four stages of the overall solution method. See Fig. 4 for a schematic of the velocity and the non-hydrostatic pressure correction. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

in which $\mathcal{K} = 3$ (determined by trial and error) artificially elevates the Petrov–Galerkin diffusivity in order to control oscillations in the volume fraction field generated by negative diffusion, Eq. (36). The diffusion coefficient for interface-capturing becomes:

$$k_{i,j,k}^{x\,n} = w_{i,j,k}^{x\,n+}k_{i,j,k}^{x\,n+} + w_{i,j,k}^{x\,n-}k_{i,j,k}^{x\,n-}, \tag{40}$$

and similarly for $k_{i,j,k}^{y\,n}$, $k_{i,j,k}^{z\,n}$. After stabilising the overall diffusion coefficient, the final anisotropic diffusion coefficients become:

$$k_{i,j,k}^{x\,n} \leftarrow \max\{-k_{min}, \min\{k_{max}, k_{i,j,k}^{x\,n}\}\}, \tag{41}$$

where k_{min} and k_{max} are determined based on the stability condition of explicit time-stepping schemes:

$$k_{min} = \frac{\alpha_{min}}{\Delta t}\max\{\Delta x, \Delta y, \Delta z\}^2, \qquad k_{max} = \frac{\alpha_{max}}{\Delta t}\min\{\Delta x, \Delta y, \Delta z\}^2, \tag{42}$$

in which $\alpha_{min} = 0.0001$, $\alpha_{max} = 0.05$ has been determined by trial and error. If there are issues with dissipation of the interface (not the case within this paper), we have seen that reducing α_{max} to 0.005 will improve the results. Expressions similar to Eq. (42) are developed for $k_{i,j,k}^{y\,n}$ and $k_{i,j,k}^{z\,n}$.

2.4. Solution method for the multiphase flow equations

This section describes the overall solution method for solving the Navier–Stokes and volume fraction field equations. Although these equations are highly coupled, a segregated solution method is applied in four stages. First, the hydrostatic and surface-tension pressure is found; second, a velocity solution is calculated; third, non-hydrostatic and non-surface-tension pressure is calculated and the velocity solution is corrected; and finally, an advection equation is solved for the volume fraction. Fig. 3 shows a diagrammatic representation of the multiphase solution method implemented as a neural network, where the four stages in the solution process are identified as detailed below.

Stage 1: Hydrostatic and Surface-Tension Pressure Solution. By taking the divergence of the terms on the right-hand side of Eq. (17) (excluding the term involving p_{nh}) and setting this to zero, we obtain an equation for the hydrostatic and surface-tension pressure as follows:

$$-\nabla \cdot \nabla p_{hs}^{n+1} + \nabla \cdot (s_q^n + s_t^n) = 0, \tag{43}$$

where the buoyancy term is given in Eq. (20) and the surface tension term is given by Eq. (21). After discretising Eq. (43) in space, we obtain an expression for the residual of the pressure

$$r_{hs}^{n+\frac{1}{2}} = -f(p_{hs}^{n+1}; \mathbf{w}_{\nabla^2}) + f(s_{tu}^n; \mathbf{w_x}) + f(s_{tv}^n; \mathbf{w_y}) + f(s_{tw}^n + g\rho^n; \mathbf{w_z}), \tag{44}$$
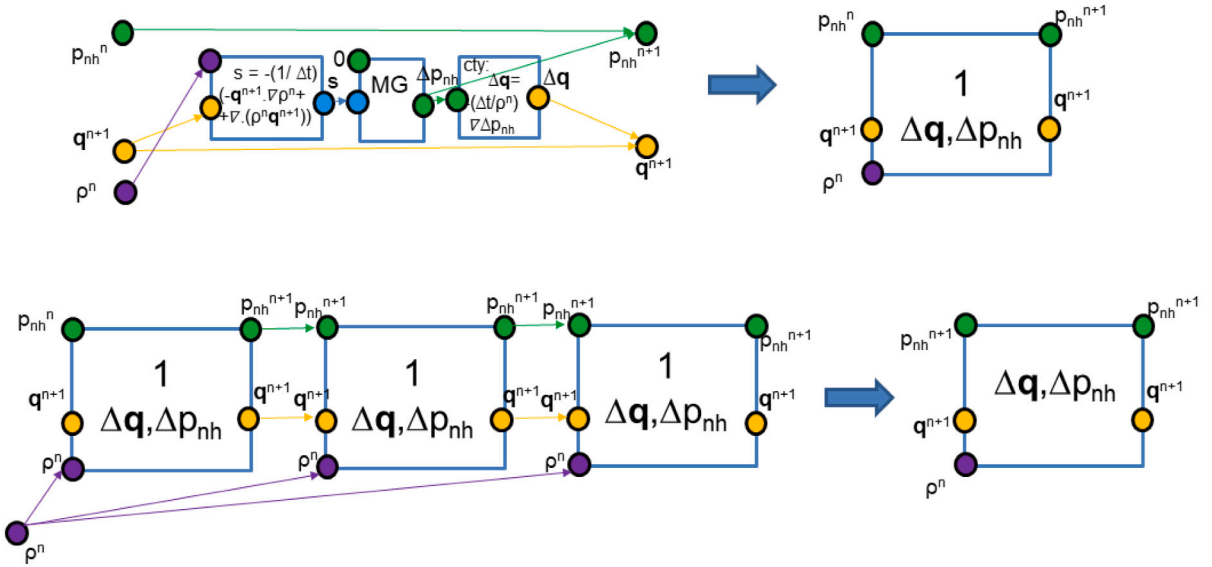
Fig. 4. Schematic of the velocity and the non-hydrostatic pressure correction necessary to satisfy the continuity equation. Details of a single iteration of the pressure and velocity correction are shown in top left and top right, respectively. Three of these iterations are combined to form an overall pressure and velocity correction to satisfy the continuity equation, the schematic of which is shown bottom right. (See Fig. 3 for a key of the colours).
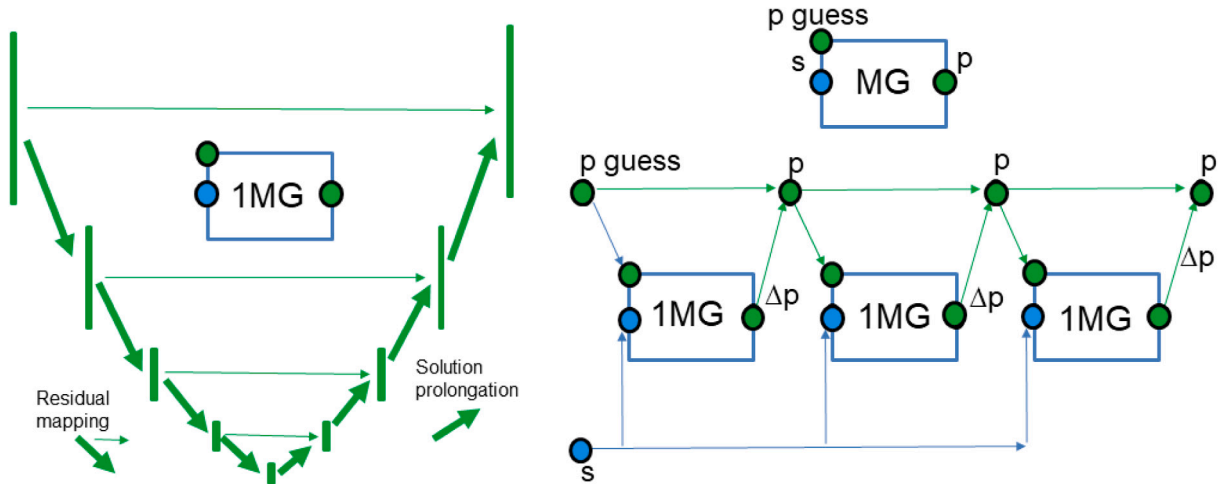

Fig. 5. Saw-tooth multigrid method, based on a U-Net architecture, used to solve for the hydrostatic and surface-tension pressure and the non-hydrostatic non-surface-tension pressure correction.

which is forced to zero using the U-Net multigrid solver in order to calculate the hydrostatic pressure tensor p_{hs}^{n+1}. Fig. 5 shows a schematic diagram that describes how this and the (non-hydrostatic non-surface-tension) pressure correction equation are solved using a saw tooth multigrid method using the U-Net architecture, see [51]. A fixed number of 20 multigrid cycles for each pressure equation solution is used in this paper, except where otherwise stated.

Stage 2: Velocity Solution. With a surface tension coefficient for water of $\sigma_t = 7.28 \times 10^{-2}\,\mathrm{N\,m^{-1}}$, the tensors representing surface tension, see Eq. (21), are substituted in the right-hand side of the discrete residuals of the velocities (see Eqs. (18a), (18b) and (18c)), together with the pressure gradients and buoyancy source. The solution for the velocities requires a two-step approach using the discrete sources $\bar{s}_u^{n+\frac{1}{2}}$, $\bar{s}_v^{n+\frac{1}{2}}$, $\bar{s}_w^{n+\frac{1}{2}}$ (see Eqs. (18)) and starting with the best approximation to $u^{n+\frac{1}{2}}$, $v^{n+\frac{1}{2}}$, $w^{n+\frac{1}{2}}$, that is $u^{n+\frac{1}{2}} = u^n$, $v^{n+\frac{1}{2}} = v^n$, $w^{n+\frac{1}{2}} = w^n$. By setting the residual to zero, applying the inverse identity filter and applying Hadamard entry-wise division, the approximate velocities at $n + 1$ are given by:

$$\tilde{u}^{n+1} = \left(-f(\bar{s}_u^{n+\frac{1}{2}}; \mathbf{w}_{\mathbf{ml}}^{-1}) + \frac{1}{\Delta t} \rho^n \odot u^n \right) \oslash \left(\frac{1}{\Delta t} \rho^n + \sigma \right),$$

$$\tilde{\boldsymbol{v}}^{n+1} = \left(-\boldsymbol{f}(\tilde{\boldsymbol{s}}_{\boldsymbol{v}}^{n+\frac{1}{2}}; \mathbf{w_{ml}^{-1}}) + \frac{1}{\Delta t}\rho^n \odot \boldsymbol{v}^n\right) \oslash \left(\frac{1}{\Delta t}\rho^n + \boldsymbol{\sigma}\right),$$

$$\tilde{\boldsymbol{w}}^{n+1} = \left(-\boldsymbol{f}(\tilde{\boldsymbol{s}}_{\boldsymbol{w}}^{n+\frac{1}{2}}; \mathbf{w_{ml}^{-1}}) + \frac{1}{\Delta t}\rho^n \odot \boldsymbol{w}^n\right) \oslash \left(\frac{1}{\Delta t}\rho^n + \boldsymbol{\sigma}\right). \tag{45}$$

The velocities at time level $n+\frac{1}{2}$ are updated:

$$\boldsymbol{u}^{n+\frac{1}{2}} = \frac{1}{2}(\boldsymbol{u}^n + \tilde{\boldsymbol{u}}^{n+1}), \quad \boldsymbol{v}^{n+\frac{1}{2}} = \frac{1}{2}(\boldsymbol{v}^n + \tilde{\boldsymbol{v}}^{n+1}), \quad \boldsymbol{w}^{n+\frac{1}{2}} = \frac{1}{2}(\boldsymbol{w}^n + \tilde{\boldsymbol{w}}^{n+1}). \tag{46}$$

A second iteration can be performed by recalculating the sources $\tilde{\boldsymbol{s}}_{\boldsymbol{u}}^{n+\frac{1}{2}}$, $\tilde{\boldsymbol{s}}_{\boldsymbol{v}}^{n+\frac{1}{2}}$, $\tilde{\boldsymbol{s}}_{\boldsymbol{w}}^{n+\frac{1}{2}}$ (Eq. (18)) with the latest solutions at time levels $n+1$ and $n+\frac{1}{2}$ and using Eq. (45) again to calculate an improved estimate of the solutions at time level $n+1$. In this work we use two iterations, although more could be done if desired.

Stage 3: Non-Hydrostatic Pressure and Velocity Correction. In the following, the non-hydrostatic pressure and velocity correction equations are formed, see Fig. 4. To do this, Eq. (17) is modified by replacing $\tilde{\boldsymbol{q}}^{n+1}$ and $\tilde{p}_{nh}^n$ with the variables that satisfy mass conservation at time level $n+1$ ($\boldsymbol{q}^{n+1}$ and p_{nh}^{n+1}) resulting in:

$$\rho^n \left(\frac{\boldsymbol{q}^{n+1} - \boldsymbol{q}^n}{\Delta t} + u^{n+\frac{1}{2}}\frac{\partial \boldsymbol{q}^{n+\frac{1}{2}}}{\partial x} + v^{n+\frac{1}{2}}\frac{\partial \boldsymbol{q}^{n+\frac{1}{2}}}{\partial y} + w^{n+\frac{1}{2}}\frac{\partial \boldsymbol{q}^{n+\frac{1}{2}}}{\partial z}\right) + \sigma\tilde{\boldsymbol{q}}^{n+1} - \nabla \cdot (\mu\nabla\boldsymbol{q}^{n+\frac{1}{2}})$$
$$= -\nabla p_{hs}^{n+1} - \nabla p_{nh}^{n+1} + s_q^n + s_t^n. \tag{47}$$

On subtracting Eq. (17) from Eq. (47), the following velocity correction equation is formed:

$$\frac{\rho^n}{\Delta t}\Delta \boldsymbol{q} = -\nabla\Delta p_{nh}, \tag{48}$$

in which $\Delta \boldsymbol{q} = \boldsymbol{q}^{n+1} - \tilde{\boldsymbol{q}}^{n+1}$ and $\Delta p_{nh} = p_{nh}^{n+1} - \tilde{p}_{nh}^{n+1}$. By combining the equation for the volume fraction discretised in time (Eq. (14)) with the expression for density (Eq. (2)), we obtain an equation for conservation of mass:

$$\frac{\rho^{n+1} - \rho^n}{\Delta t} + \nabla \cdot \left(\boldsymbol{q}^{n+1}\rho^n\right) = \frac{\rho^{n+1} - \rho^n}{\Delta t} + \boldsymbol{q}^{n+1} \cdot \nabla\rho^n = 0. \tag{49}$$

Taking the divergence of Eq. (48) and substituting in Eq. (49) enables us to form the correction for non-hydrostatic non-surface-tension pressure (see Pavlidis et al. [32,33]):

$$-\nabla \cdot \nabla\Delta p_{nh} = \frac{1}{\Delta t}\nabla \cdot (\rho^n\boldsymbol{q}^{n+1} - \rho^n\tilde{\boldsymbol{q}}^{n+1}) = -\frac{1}{\Delta t}\left(\frac{\rho^{n+1} - \rho^n}{\Delta t} + \nabla \cdot (\rho^n\tilde{\boldsymbol{q}}^{n+1})\right). \tag{50}$$

Approximating the mass conservation equation by using the best estimate of velocity ($\tilde{\boldsymbol{q}}^{n+1}$),

$$\frac{\rho^{n+1} - \rho^n}{\Delta t} = -\tilde{\boldsymbol{q}}^{n+1} \cdot \nabla\rho^n, \tag{51}$$

means that we can replace the unknown density ρ^{n+1} in Eq. (50), resulting in the non-hydrostatic non-surface-tension pressure correction of

$$-\nabla \cdot \left(\nabla\Delta p_{nh}\right) = -\frac{1}{\Delta t}\left(\nabla \cdot (\rho^n\tilde{\boldsymbol{q}}^{n+1}) - \tilde{\boldsymbol{q}}^{n+1} \cdot \nabla\rho^n\right). \tag{52}$$

The fully discretised form of Eq. (52) is:

$$\boldsymbol{r}_{nh}^{n+\frac{1}{2}} = -\boldsymbol{f}(\Delta p_{nh}; \mathbf{w_{\nabla^2}})$$
$$- \frac{1}{\Delta t}\left(\boldsymbol{f}(\rho^n \odot \tilde{\boldsymbol{u}}^{n+1}; \mathbf{w_x}) + \boldsymbol{f}(\rho^n \odot \tilde{\boldsymbol{v}}^{n+1}; \mathbf{w_y}) + \boldsymbol{f}(\rho^n \odot \tilde{\boldsymbol{w}}^{n+1}; \mathbf{w_z})\right)$$
$$+ \frac{1}{\Delta t}\left(\tilde{\boldsymbol{u}}^{n+1} \odot \boldsymbol{f}(\rho^n; \mathbf{w_x}) + \tilde{\boldsymbol{v}}^{n+1} \odot \boldsymbol{f}(\rho^n; \mathbf{w_y}) + \tilde{\boldsymbol{w}}^{n+1} \odot \boldsymbol{f}(\rho^n; \mathbf{w_z})\right), \tag{53}$$

in which $\tilde{\boldsymbol{u}}^{n+1}$, $\tilde{\boldsymbol{v}}^{n+1}$, $\tilde{\boldsymbol{w}}^{n+1}$ are the estimates for $\boldsymbol{u}^{n+1}$, $\boldsymbol{v}^{n+1}$, $\boldsymbol{w}^{n+1}$ obtained at the end of Stage 2. The residual in Eq. (53) is forced to zero using the U-Net multigrid solver to calculate the non-hydrostatic pressure correction tensor Δp_{nh}. Although the right-hand sides of Eqs. (50) and (52) are identical in the continuum, at the discrete level, they are very different. For instance, the sign of the implied diffusion (realised through the Petrov–Galerkin diffusion introduced into the equations for $\boldsymbol{C}^{n+1}$) is reversed.

Eq. (48) is then discretised in space to form an equation for the velocity corrections:

$$\Delta\boldsymbol{u} = -\Delta t\left(\boldsymbol{f}\left(\boldsymbol{f}(\Delta p_{nh}; \mathbf{w_x}); \mathbf{w_{ml}^{-1}}\right) \oslash \rho^n\right),$$
$$\Delta\boldsymbol{v} = -\Delta t\left(\boldsymbol{f}\left(\boldsymbol{f}(\Delta p_{nh}; \mathbf{w_y}); \mathbf{w_{ml}^{-1}}\right) \oslash \rho^n\right),$$
$$\Delta\boldsymbol{w} = -\Delta t\left(\boldsymbol{f}\left(\boldsymbol{f}(\Delta p_{nh}; \mathbf{w_z}); \mathbf{w_{ml}^{-1}}\right) \oslash \rho^n\right). \tag{54}$$

Finally, the non-hydrostatic non-surface-tension pressure and velocities are updated:

$$p_{nh}^{n+1} = p_{nh}^n + \Delta p_{nh} \quad \text{and} \quad \begin{array}{ccc} \boldsymbol{u}^{n+1} & \leftarrow & \boldsymbol{u}^{n+1} + \Delta\boldsymbol{u}, \\ \boldsymbol{v}^{n+1} & \leftarrow & \boldsymbol{v}^{n+1} + \Delta\boldsymbol{v}, \\ \boldsymbol{w}^{n+1} & \leftarrow & \boldsymbol{w}^{n+1} + \Delta\boldsymbol{w}. \end{array}$$

Table 1

Comparison of the computational speed of numerical models found in the literature with the estimated computational speed of NN4PDEs. One single CPU is used in [83–85]; 8 A100 GPUs and 512 Intel Xeon Gold 6130 CPUs are tested in [17]; 100 AMD Opteron CPUs are used in [45]; and 8 V100 GPUs are used in [46]. The wall time for each reference refers to the overall timing for complete architectures (*e.g.,* the simulation on 8 GPUs takes 0.191 s, and the equivalent time that this is estimated to take on a single GPU is 8×0.191 s $= 1.528$ s). The wall time for NN4PDEs to solve the same problem (with one time step) on a single GPU is estimated using Eq. (56).

Computer architectures	Number of nodes/cells	Wall time per time step		Estimated wall time per time step for NN4PDEs
		As quoted	Estimate for single CPU/GPU	
$1 \times$ CPU [83]	16 384	10.50 s	10.50 s	0.609 ms
$1 \times$ CPU [84]	108 960	11.12 s	11.12 s	4.02 ms
$1 \times$ CPU [85]	180 000	14.19 s	14.19 s	6.71 ms
$8 \times$ GPU [17]	512^3	0.191 s	1.528 s	1.625 s
$512 \times$ CPU [17]	512^3	1.075 s	550.4 s	1.625 s
$100 \times$ CPU [45]	500^3	220 s	22 000 s	1.514 s
$8 \times$ GPU [46]	64 000 000	0.273 s	2.184 s	0.775 s

To obtain improved mass conservation, one can repeat this process of solving Eqs. (53) and (54), and updating the non-hydrostatic non-surface-tension pressure and velocities at time level $n + 1$. Here this process is carried out just twice, see Fig. 18.

Stage 4: Volume Fraction Solution. Solve for volume fraction field based on interface tracking with compressive advection. Setting the residual in Eq. (15) to zero and rearranging, gives

$$C^{n+1} = -\Delta t f(\bar{\mathfrak{s}}_C^{n+\frac{1}{2}}; \mathbf{w_{ml}^{-1}}) + C^n,$$

(55)

in which $\mathbf{w_{ml}} = m_l \mathbf{E}$ and $\mathbf{w_{ml}^{-1}} = m_l^{-1} \mathbf{E}$, where $\mathbf{E}$ has values $\mathrm{E}_{i,j,k} = 1$ when $i = j = k = 0$, zero otherwise.

2.5. Computational speed of the approach

The following equation for the time it takes to run the multiphase code for a given simulation seems to hold (to within 10%) for the simulations performed here:

$$\text{Wall clock time of simulation in seconds} = p\mathcal{T}\mathcal{N}\left[M_s + (N_h + N_I N_{nh})H_s\right],$$

(56)

in which $\mathcal{N} = N_x \times N_y \times N_z$ is the number of nodes or grid points, $\mathcal{T}$ is the number of time-steps in the simulation, p is the ConvFEM polynomial order, N_h is the number of multigrid iterations for the hydrostatic pressure, N_{nh} is the number of multigrid iterations for the non-hydrostatic pressure, and N_I is the number of pressure–velocity correction iterations. The coefficients M_s and H_s depend on the computer being used: M_s is associated with the cost of solving for momentum and volume fraction, and H_s is associated with the cost of a multigrid solve. For the NVIDIA A10 Tensor Core GPU and NVIDIA RTX A5000, the value of these coefficients are $M_s = 1.2 \times 10^{-8}$ s and $H_s = 1.1 \times 10^{-10}$ s, and $M_s = 1.37 \times 10^{-8}$ s and $H_s = 1.3 \times 10^{-10}$ s, respectively. Also when the surface tension calculation is used, M_s decreases by 1.12% and H_s increases by 14%. For example, model 5 from Table 2 uses quadratic elements (p = 2), $\mathcal{N} = 67 \times 10^6$ nodes, $\mathcal{T} = 4 \times 10^4$ time-steps, $N_h = N_{nh} = 20$ multigrid iterations, $N_I = 2$ correction iterations and, thus, according to Eq. (56), will take 24.42 h on the NVIDIA A10 Tensor Core GPU, which compares well with the actual run time for this simulation of 23.04 h.

Table 1 reports computational speeds (wall time per time step) of multiphase flow codes quoted in the literature for a range of computer systems and compares this with estimated timings of NN4PDEs when running on one NVIDIA A10 Tensor Core GPU. The reference and computer architecture that was used is given in the first column (where $8 \times GPU$ indicates that 8 GPUs were used). The second column gives the number of nodes in the simulations carried out in each reference. The wall time obtained from each reference is shown in the third column. In the fourth column we adjust the wall times to take account of when the references used more than one CPU or GPU. We assume perfect scaling and multiply the quoted time by 512 in the case where 512 CPUs were used, for example. In order to compare the speed of our code with these references without setting up each of their cases and methods, we use Eq. (56) to adjust our computational speed to the size of problem (number of nodes) given in each reference. In this calculation, we assume linear elements (p = 1) and $N_h = N_{nh} = 20$ multigrid iterations. Hence, we have a way of estimating how long our code would take to run the problems in the cited references. In comparison to the reference studies [83–85] (running on one CPU) and [45] (running on 100 CPUs), our code demonstrates a significant speed-up when executed on one GPU, running at least 2000 times faster than the other codes. Moreover, we notice that NN4PDEs has a comparable performance (1.625 s) to the reference work [17] (1.528 s), despite using a less powerful GPU (the A10) than their GPU (the A100). It has been reported that the A100 is at least twice as capable as the A10 for inference tasks [86].

3. Results

In this section, the proposed multiphase flow solver is validated against a number of classic CFD problems: collapsing water columns (in 2D and 3D), and a rising water bubble. All the models, presented here, were run on two types of single GPU: NVIDIA A10 Tensor Core GPU and NVIDIA RTX A5000.

Table 2
Details of eight different collapsing water column models simulated by the proposed NN4PDEs approach. The numbers of grid nodes listed do not include the halo nodes through which the boundary conditions were applied. Table 3 explains the mixed formulations (*e.g.,* linear/quadratic, linear/cubic etc.) that are listed here in the "ConvFEM discretisation" column. The computational domain can be calculated by multiplying the number of nodes in each direction by the grid size, or looking at Table 4 and Fig. 7.

Model	Number of grid points		Grid size	Time step	Obstacle	Discretisation	Figure
	$N_x \times N_y \times N_z$	Nodes	(mm)	(ms)			
1	512×512	262,144	1.0	0.5	No	Linear	Fig. 6
2	$1024 \times 64 \times 1024$	67,108,864	0.5	0.025	No	Linear	Figs. 8(a), 9(a)
3	$1024 \times 64 \times 1024$	67,108,864	0.5	0.025	Yes	Linear	Fig. 11
4	$1024 \times 64 \times 1024$	67,108,864	0.5	0.025	No	Quadratic	Figs. 8(b), 9(b)
5	$1024 \times 64 \times 1024$	67,108,864	0.5	0.025	No	Linear/quadratic	Fig. 8(c), 9(c)
6	$1024 \times 64 \times 1024$	67,108,864	0.5	0.025	No	Linear/cubic	Figs. 8(d), 8(d)
7	$1024 \times 256 \times 1024$	268,435,456	0.5	0.025	No	Linear/quadratic	Fig. 10
8	$512 \times 512 \times 512$	134,217,728	1.0	0.1	No	Linear/quadratic	Figs. 12, 13

Table 3
Discretisations used in the various models given in Table 2.

Description	Volume fraction (diffusion term)	All other terms
Linear	Linear	Linear
Quadratic	Quadratic	Quadratic
Linear/quadratic	Linear	Quadratic
Linear/cubic	Linear	Cubic

3.1. Collapsing water column

A column of water is subjected to gravity ($g = 9.81 \, \text{m/s}^2$), leading to the collapse of the column towards a flat floor. Two fluids (water and air) are considered with different densities, $\rho_{\text{water}} = 1000 \, \text{kg/m}^3$ and $\rho_{\text{air}} = 1 \, \text{kg/m}^3$. The problem is initialised by setting the volume fraction of the fluid to be zero for water and one for air. Free-slip boundary conditions are imposed on the sides and bottom of the computational domain. A zero pressure boundary condition is applied to the hydrostatic and non-hydrostatic pressure at the top of the domain. Neumann boundary conditions (*i.e.,* zero derivative) are applied for both volume fraction and density fields on all surfaces of the domain.

Eight different numerical model with various grid sizes and discretisation orders were applied to this problem in both 2D (Model 1) and 3D (Models 2–8), as summarised in Table 2. For all models, a uniform grid spacing is used to generate structured grids, see grid size in Table 2. Various orders of discretisation are used here (linear, quadratic and cubic ConvFEM) as shown in Table 3. Model 7 is similar to Models 2, 4, 5 and 6 as regards grid size (width and height) and time step, but has a depth of the domain that is 4 times larger, in order to investigate 3D behaviour of the collapsing column. To ensure that the Courant number is below 0.1, a fixed time step is applied of $0.5 \, \text{ms}$ (Model 1), $0.025 \, \text{ms}$ (Models 2–7) and $0.1 \, \text{ms}$ (Model 8).

3.1.1. 2D collapsing water column

In order to validate the accuracy of our simulator when predicting the evolution of an initial column of water in 2D, Model 1 is adopted (see Table 2). The results are compared with the experimental measurements by Yeoh and Barber [87], see also Martin et al. [88]. The dimensions of the 2D domain are $0.512 \, \text{m} \times 0.512 \, \text{m}$ in the x and y directions, respectively. A column of water is initialised with width of $w_0 = 0.064 \, \text{m}$ (x direction) and height of $h_0 = 0.128 \, \text{m}$ (y direction). As the water column collapses due to gravity (imposed in the negative y-direction), the water front advances over the flat floor and the height of the column decreases in time. The initial column width (w_0) and height (h_0) are applied to normalise distances in the x and y directions ($x^* = w/w_0$ and $y^* = h/h_0$). The evolution of the initial stages of the water column collapse is shown by plotting the progress of the normalised front (x^*) and height (y^*) of the column through time (t^*) in Fig. 6, demonstrating excellent agreement with experimental data (taken from Figure 7 within Yeoh and Barber [87]). In Fig. 6 we also plot numerical results from a Discontinuous Galerkin (DG) formulation [33] solved on a 2D structured mesh. The normalised distance x^* predicted by NN4PDEs is closer to the experimental data than the DG model. The predicted normalised height of the water column, y^*, is similar for both numerical models. For the column front (Fig. 6(a)), time is normalised by $\sqrt{2g/w_0}$ and for the column height (Fig. 6(b)), time is normalised by $\sqrt{2g/h_0}$, as done by Yeoh and Barber [87].

3.1.2. 3D collapsing water column

In this section, three collapsing water column scenarios are investigated, for which the dimensions of both the computational domain and the water column vary (see Table 4 and Fig. 7). The force due to gravity is imposed in the negative z direction (vertical), the x direction is horizontal and the y direction is into the page.

Figs. 8 and 9 show transient results in the range $0 \, \text{s} \leqslant t \leqslant 1 \, \text{s}$ for a grid of 67 million ConvFEM nodes from Models 2, 4, 5 and 6, which have the same domain and grid size, but different orders and combinations of FE discretisations (see Table 2 for details). Fig. 8 shows the water-air interface, chosen here to be where the volume fraction field has a value of 0.5, and Fig. 9 shows the
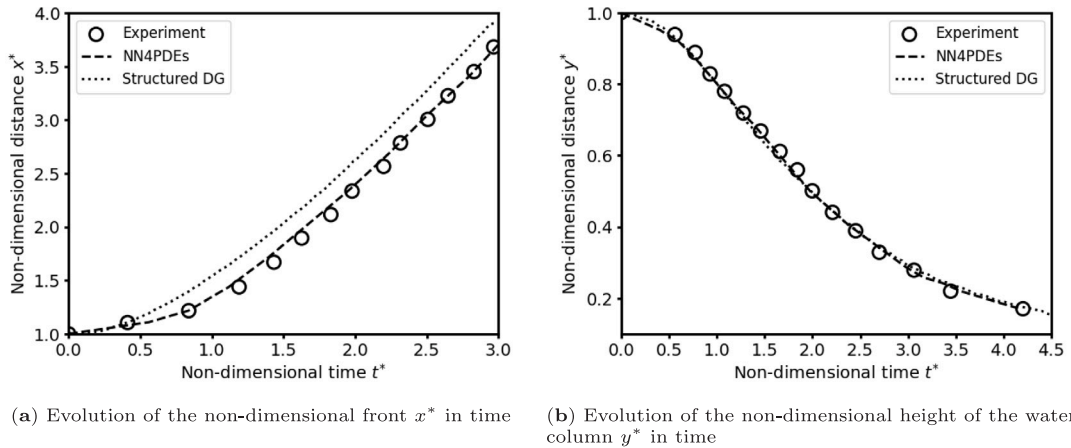
(**a**) Evolution of the non-dimensional front x^* in time

(**b**) Evolution of the non-dimensional height of the water column y^* in time

Fig. 6. Results for the collapsing water column from NN4PDEs predicted using Model 1 (see Table 2). For comparison, experimental data [87] and numerical results [33] for a 2D version of this test case are also shown.

Table 4

Collapsing water column: size of computational domains and water columns used in three scenarios of the 3D collapsing water columns. See Fig. 7.

Test case	Fig. 7(a)	Fig. 7(b)	Fig. 7(c)
Domain	$0.512\,\text{m} \times 0.032\,\text{m} \times 0.512\,\text{m}$	$0.512\,\text{m} \times 0.032\,\text{m} \times 0.512\,\text{m}$	$0.512\,\text{m} \times 0.512\,\text{m} \times 0.512\,\text{m}$
Water	$0.128\,\text{m} \times 0.032\,\text{m} \times 0.256\,\text{m}$	$0.128\,\text{m} \times 0.032\,\text{m} \times 0.256\,\text{m}$	$0.128\,\text{m} \times 0.128\,\text{m} \times 0.384\,\text{m}$
Obstacle	–	$0.024\,\text{m} \times 0.032\,\text{m} \times 0.048\,\text{m}$	–

(**a**) rectangular domain without obstacle   (**b**) rectangular domain with obstacle   (**c**) cubic domain without obstacle
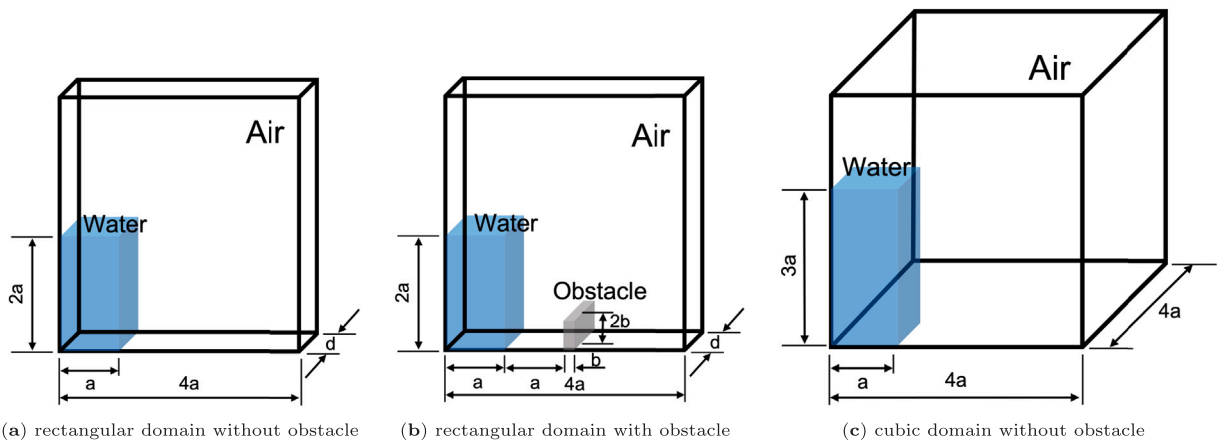
Fig. 7. Schematic diagrams of three different scenarios used in the collapsing water column tests. The values of a, b and d are $0.128\,\text{m}$, $0.024\,\text{m}$ and $0.032\,\text{m}$, respectively. For all models, the height of the domains is 4a. For Model 7, not shown here, the length of the edges in x and y are 4a and 4d. See also Table 4.

volume fraction field, which ranges from 0 to 1. Overall, numerical results indicate similar time-dependent spatial distributions of the water volume fraction field to the numerical and experimental investigations of Yeoh and Barber [87]. The evolution of the water columns is shown through the volume fraction field in Figs. 8 and 9, and the results display good qualitative agreement with those shown experimentally and computationally by Yeoh and Barber [87] (see their Figs. 2, and 4 and 5 respectively).

Comparing the approaches implemented in these four models, a better performance in terms of interface sharpness between phases was obtained from Models 5 and 6 (as demonstrated in Figs. 9(c) and 9(d)) due to the compressive-advection interface-capturing formulation (see Section 2.3.5). The reason is that higher-order (quadratic and cubic FE) diffusion schemes introduce more oscillations than lower-order (linear FE) schemes, which leads to a more diffusive volume fraction field as shown in Fig. 9(b). In order to mitigate the additional diffusion of the volume fraction field caused by higher-order discretisations, Models 5 and 6 combine advantages of lower- and higher-order FE discretisations: a lower-order FE discretisation (linear ConvFEM) is used to discretise the diffusion of the volume fraction field and all other terms are discretised using higher-order FE schemes (quadratic or cubic ConvFEM).
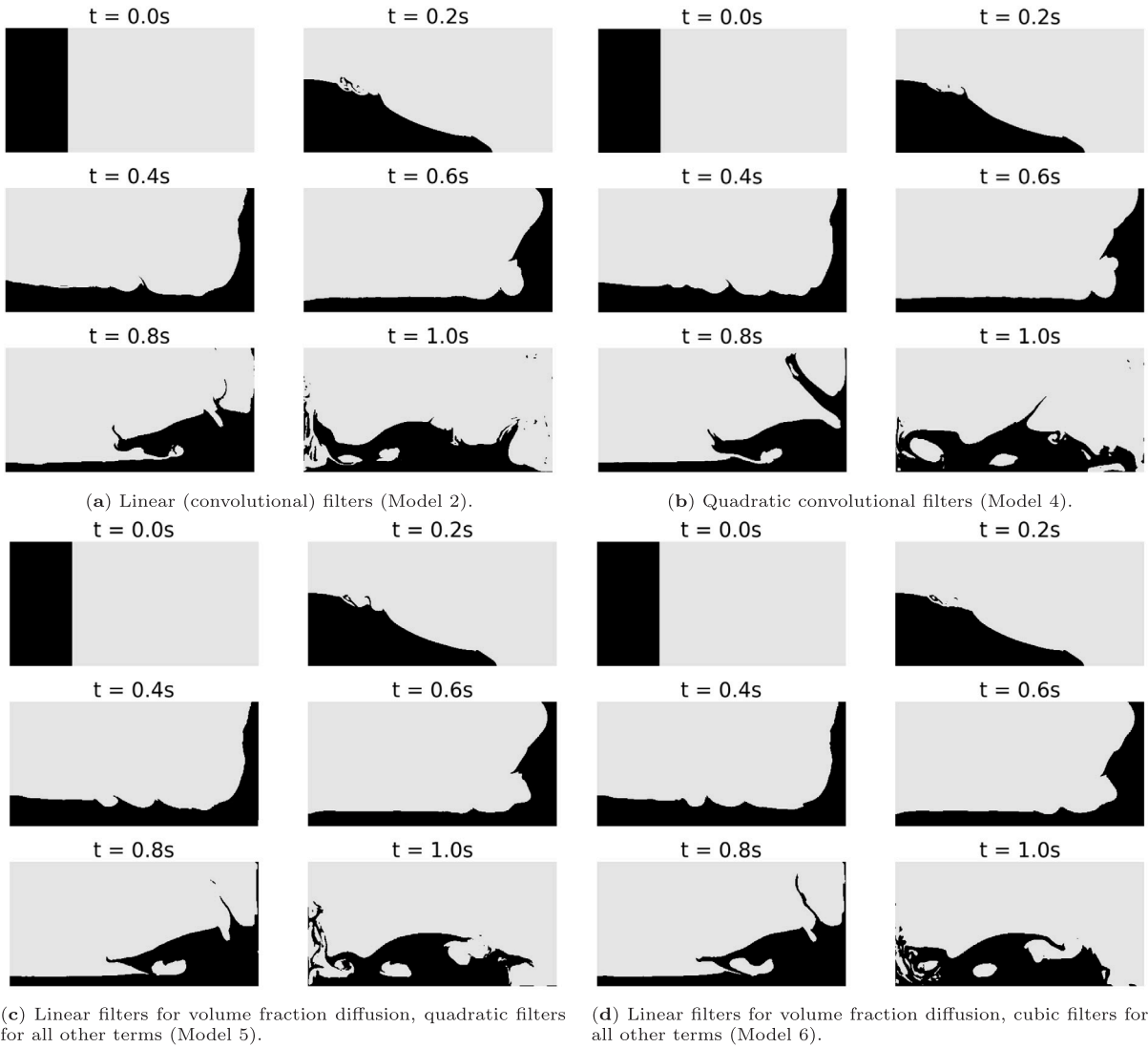
(a) Linear (convolutional) filters (Model 2).

(b) Quadratic convolutional filters (Model 4).

(c) Linear filters for volume fraction diffusion, quadratic filters for all other terms (Model 5).

(d) Linear filters for volume fraction diffusion, cubic filters for all other terms (Model 6).

Fig. 8. Snapshots of water-air interface for the 3D collapsing water column without an obstacle from $t = 0$ s to $t = 1.0$ s with 0.2 s intervals and solved on a grid of $1024 \times 64 \times 1024$ nodes. Volume fraction values greater than 0.5 are shown in black and where less than 0.5 is coloured grey. The resulting fields are shown on an xz plane at the midpoint of the extent of the domain in the y direction. See Table 2 for details.

Fig. 10 shows results for the water-air interface and volume fraction field for Model 7, which has the same time step and uses the same filters (linear filters for the diffusion term of the volume fraction and quadratic filters for all other terms) as the results shown in Figs. 8(c) and 9(c) for Model 5. The computational domain used to generate the results shown in Fig. 10 has a depth that is 4 times larger than that of Model 5, using a total of 256 million nodes (the largest number of nodes used here). The motivation for running this simulation is to excite fully three-dimensional behaviour (due to the extended domain in the y direction) and also explore the limit of the size of the problem that can be simulated on our single GPU. Comparing Figs. 8(c) and 9(c) with Figs. 10(a) and 10(b), a similar dispersion and advection of the collapsing water column can be seen in both cases. Figs. 10(c)–10(f) show the water-air interface (an isosurface at a value of volume fraction of 0.5) at four time levels from 0.4 s to 1 s. They show the initial water column descending due to gravity, and forming waves on the interface between the falling water and the surrounding air in the early stages (Fig. 10(c)). As waves propagate throughout the domain, they breakup upon impact with the boundary, as shown in Figs. 10(e) and 10(f) and become highly three dimensional, aligning with the results shown in Figure 3 of Yeoh and Barber [87].

Fig. 11 shows the interaction between the collapsing water and an obstacle (see Table 4 for location of the obstacle). Figs. 11(a) and 11(b) show the water-air interface and volume fraction, respectively. The results indicate that the solution obtained from the NN4PDEs approach captures instabilities induced by the obstacle, including the generation of waves and ripples due to the sudden change in the momentum of the water. Qualitative comparisons can be made between the evolution of the volume fraction field shown here, in Fig. 11(b), with the results shown in Yeoh and Barber [87], experimentally in their Fig. 3 and computationally in
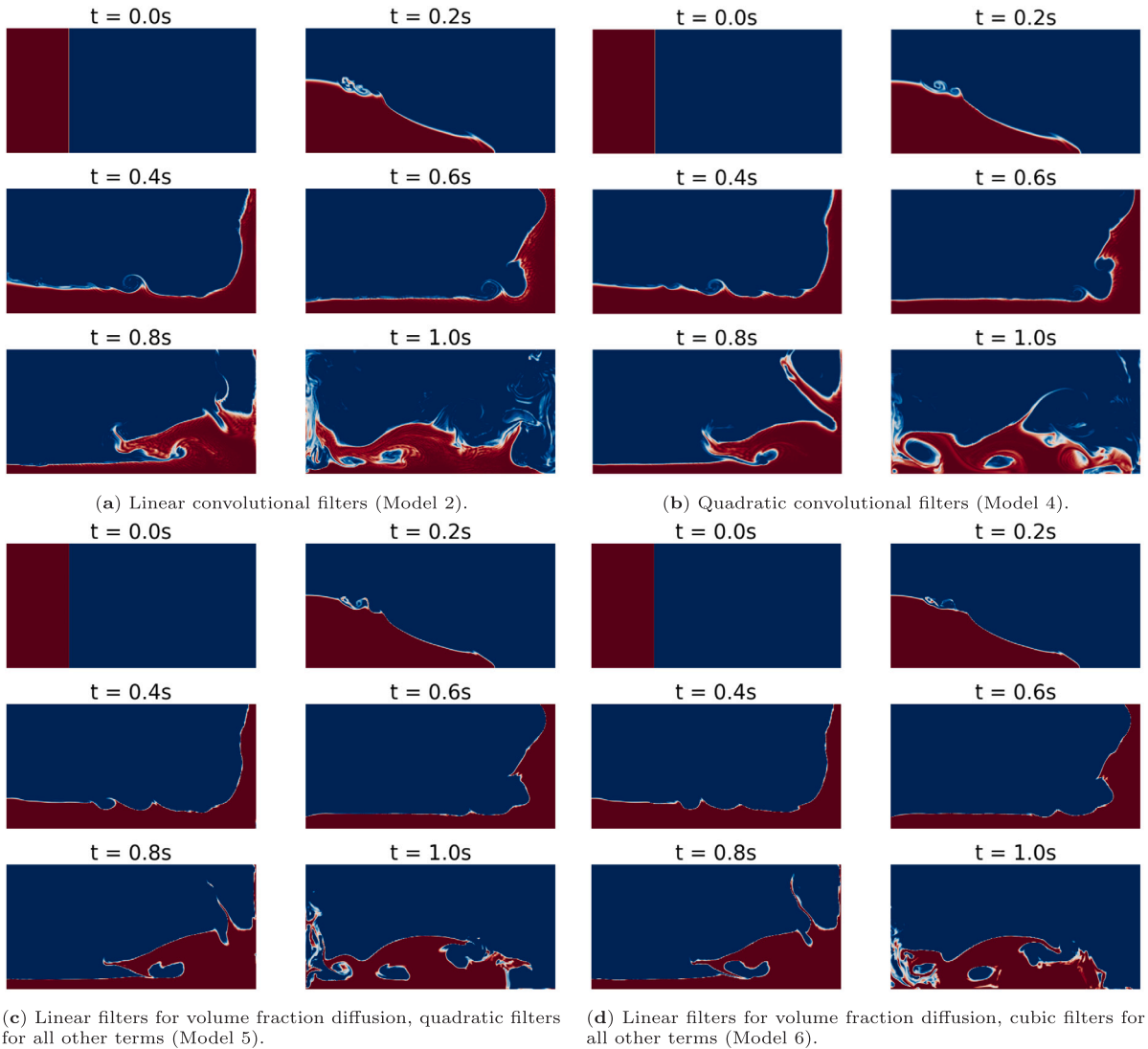
(a) Linear convolutional filters (Model 2).

(b) Quadratic convolutional filters (Model 4).

(c) Linear filters for volume fraction diffusion, quadratic filters for all other terms (Model 5).

(d) Linear filters for volume fraction diffusion, cubic filters for all other terms (Model 6).

Fig. 9. Snapshots of volume fraction field for the 3D collapsing water column without an obstacle from $t = 0\,\mathrm{s}$ to $t = 1.0\,\mathrm{s}$ at intervals of $0.2\,\mathrm{s}$, solved on a grid of $1024 \times 64 \times 1024$ nodes. The volume fraction is shown on an xz plane at the midpoint of the extent of the domain in the y direction.

their Figs. 8 and 9. A close match in terms of wave propagation and interaction between collapsing water and obstacle is seen, with our results and those of Yeoh and Barber [87] producing similar spatial variation of the water field at each time level.

The collapsing water column in a cube-shaped computational domain is shown in Fig. 12. The cube-shaped domain is used here, as it results in highly 3D behaviour (at both large and smaller scales) unlike the previous collapsing water column problems that are solved within a highly anisotropic domain, constrained to have 2D large scale behaviour in which the water column fills the extent of the domain in the y direction (see Models 2–6, and Figs. 7(a) and 7(b)). The simulation was conducted with Model 8 and the results plotted at eight time levels from $0.15\,\mathrm{s}$ to $1.2\,\mathrm{s}$ with an interval of $0.15\,\mathrm{s}$. This figure shows the isosurfaces of the volume fraction field, taken at a value of 0.5 to enable visualisation of the interface between the descending water and surrounding air. At $t = 0.15\,\mathrm{s}$, $0.30\,\mathrm{s}$ and $0.45\,\mathrm{s}$, the interface shows that the waves are symmetric diagonally across the cube-shaped domain. Following the initial impact with the opposing wall, the water field undergoes a transformation into an asymmetrical shape, leading to intensified splashing and spraying, that is particularly noticeable at $t = 0.60\,\mathrm{s}$, $0.75\,\mathrm{s}$ and $0.90\,\mathrm{s}$. Gravity causes the water column to drop and spread across the flat floor. This behaviour is most pronounced around $t = 1.20\,\mathrm{s}$, showcasing the ongoing dynamic evolution of the water field. Fig. 13 shows results of volume fraction for Model 8 on a grid of size $512 \times 512 \times 512$, plotted on a diagonal plane at eight times from $0.1\,\mathrm{s}$ to $0.8\,\mathrm{s}$ at intervals of $0.1\,\mathrm{s}$.

(a) Water-air interface.

(b) Volume fraction field.

(c) Isosurface at $t = 0.4\,\mathrm{s}$.

(d) Isosurface at $t = 0.6\,\mathrm{s}$.

(e) Isosurface at $t = 0.8\,\mathrm{s}$.

(f) Isosurface at $t = 1.0\,\mathrm{s}$.

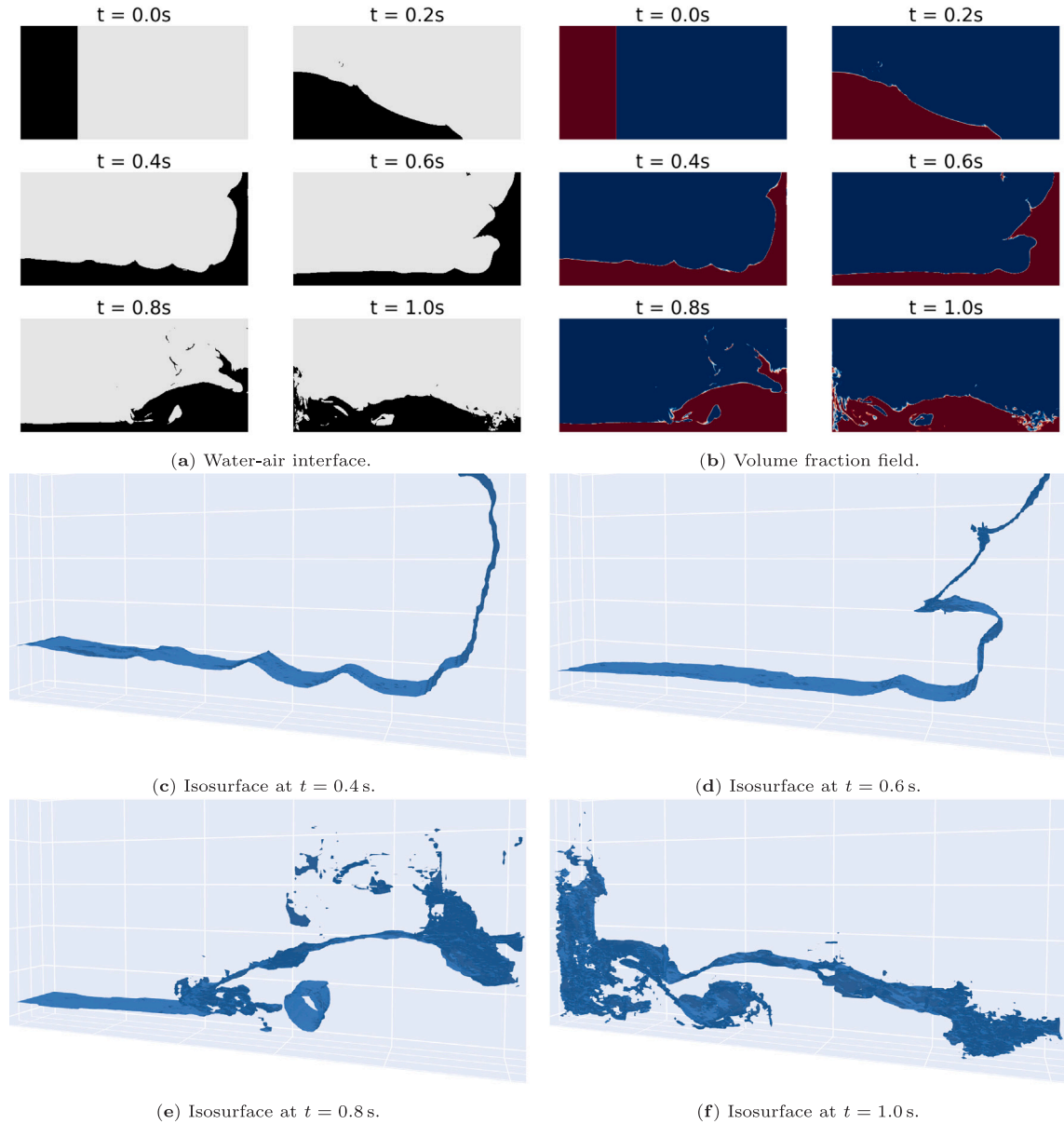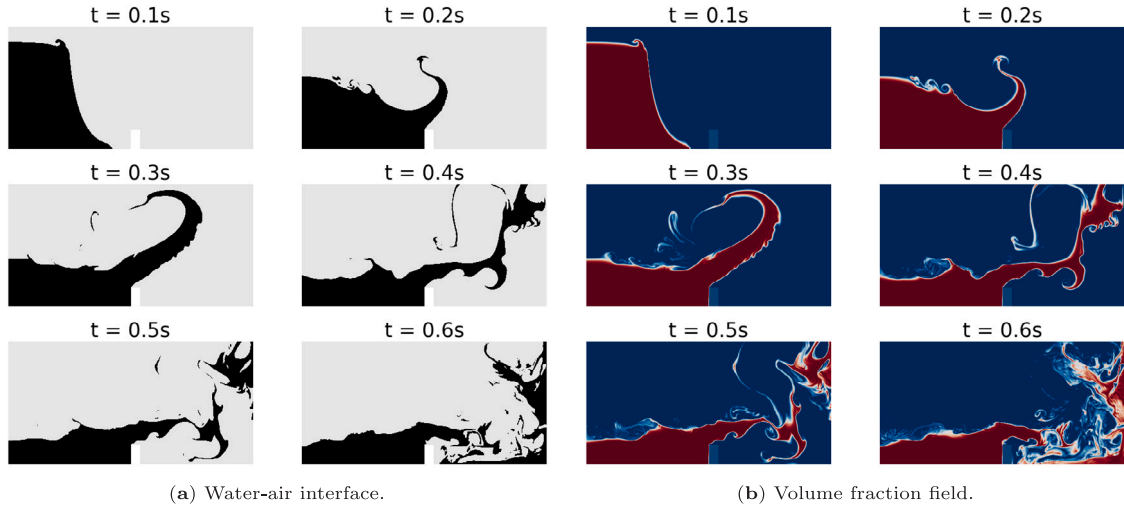**Fig. 10.** Snapshots of three-dimensional numerical predictions of collapsing water column without an obstacle from $t = 0\,\mathrm{s}$ to $t = 1.0\,\mathrm{s}$ solved on a grid of $1024 \times 256 \times 1024$ nodes. The results are obtained by Model 7 using linear filters for the volume fraction diffusion and quadratic convolutional filters for all other terms. Plots (a) and (b) are shown on an xz plane located at the midpoint of the domain on the y-axis. Plots (c) to (f) are isosurfaces showing the water-air interface.

3.2. Rising bubble in water

The NN4PDEs approach is used to simulate a single, rising 3D air bubble in water. The computational domain (Fig. 14(a)) has a cuboidal shape with dimensions of $3H \times 3H \times 6H$ (length in x direction, width in y direction and height in z direction), where H is set to $2\,\mathrm{mm}$. An air bubble is centred at $(1.5H, 1.5H, H)$. The boundary conditions used in the simulations within this section are the same as those described for the collapsing water column test-cases (Section 3.1.2). One of the best performing discretisations from the previous experiments is used here, namely, linear filters (*i.e.,* filters that represent a linear FEM discretisation) for the diffusion term of the volume fraction and quadratic filters (*i.e.,* filters that represent a quadratic ConvFEM discretisation) for all other terms. The surface tension model described in Eqs. (21) and (22) is incorporated for this test case in order to model the rising bubble.

To investigate the ability of the proposed NN4PDEs approach to forecast the dynamics of a rising bubble, a uniform grid is used with 67 million ($256 \times 256 \times 512$) structured nodes and fixed time step ($\Delta t = 0.1\,\mathrm{\mu s}$). A single, initially spherical, bubble

(a) Water-air interface.   (b) Volume fraction field.

Fig. 11. Snapshots of three-dimensional numerical predictions of the collapsing water column with an obstacle from $t = 0.1$ s to $t = 0.6$ s at an interval of 0.1 s. The fields are shown on an xz plane located at the midpoint of the domain on the y-axis. The results are obtained using Model 3 (Table 2) solved on a grid of $1024 \times 64 \times 1024$ nodes using linear convolutional filters.

with equivalent diameter of $D_b = H = 2$ mm is modelled in this problem. To obtain the results shown, two iterative corrections to velocity and non-hydrostatic pressure were made, and 20 multigrid iterations were used to solve the pressure equations. Fig. 14(b) shows the evolution of the rising bubble through an isosurface, which represents the interface between the bubble and water. The interface is taken to be a volume fraction value of 0.5. The isosurface is plotted on the central plane, and the red circle represents the initial location and shape of the air bubble. Fig. 15 shows a 3D visualisation of the shape of the bubble using an isosurface at four different time levels. The shape of the rising bubble simulated with NN4PDEs matches that reported by Crialesi-Esposito et al. [17] for a density ratio ($\lambda_\rho = \rho_l/\rho_g$) of 10. Fig. 16 illustrates the transient volume fraction field (0 s $\leqslant t \leqslant 0.044$ s) of the bubble on the central (diagonal) plane. This plot displays the shape of the bubble more precisely than Fig. 15, revealing that the lower part of the interface of the bubble is more diffuse than the upper part.

Fig. 17 helps validate the proposed solver by comparing the values relating to bubble shape and terminal velocity from the NN4PDEs approach with experimental measurements and other numerical results. Fig. 17(a) compares the shape of a single rising bubble with an initial diameter of $D_b = 1.92$ mm, where the red line represents the numerical results and the black line denotes the experimental data of Duineveld [89]. The results indicate that both the bubble shape and aspect ratio (χ) predicted by NN4PDEs match, closely, the experimental data. The terminal velocities of different bubble sizes are also compared with experimental results from Clift et al. [90] in Fig. 17(b). Seven distinct bubble sizes are modelled with initial diameters ranging from 0.4 mm to 1.0 mm. The comparison shows that the velocities of the bubble computed by NN4PDEs always fall within the range of accuracy of the experimental results and are consistently good across the range of diameter sizes. Fig. 17(c) compares the time evolution of bubble rising velocity with numerical results from Crialesi-Esposito et al. [17]. The velocity and time axes are non-dimensionalised using the reference velocity $u_r = \sqrt{gd_0}$ and reference time $t_r = \sqrt{d_0/g}$ (d_0 refers to the diameter of bubble). The results demonstrate the capability of the solver within the NN4PDEs approach to forecast accurately the water-air interface.

Fig. 18 examines the impact of the iterative approaches used for velocity and non-hydrostatic pressure correction (see Fig. 4) on the prediction of a rising bubble, by examining how well mass is conserved. The results are collected over 1000 time-steps and investigate the temporal evolution of normalised volume fraction from the initial time step by five iterative strategies, which vary the number of corrections to velocity and non-hydrostatic pressure, and the number of multigrid iterations used for solving the non-hydrostatic pressure correction equation. The iterative strategies are labelled i-j, where i represents the number of pressure–velocity corrections/iterations and j represents the number of multigrid iterations for pressure correction. The results suggest that employing more iterations to solve the pressure and velocity corrections (see Fig. 4 for a schematic of three iterations) yields a significant improvement in performance as regards mass conservation, as evidenced by a comparison of by cases 1–10 and 2–10 as well as 1–20 and 2–20. However, increasing the number of multigrid iterations leads to a smaller enhancement in performance in terms of conserving mass, as evidenced by a comparison of cases 1–10 and 1–20, with 2–10 and 2–20. Thus, we adopted 2–20 as default iterative approach to obtain the results in Section 3.2. It should also be pointed out that further increasing the number of pressure and velocity correction iterations beyond two significantly reduces the stability of the method and is thus not done here. This seems to be because, in this case, one also needs to iterate over the momentum equations, otherwise the approach tends to 'forget' about the momentum equations in favour of forcing down the residual of the continuity equation.

We remark that the only reason why mass is not exactly conserved is that the volume fraction field is solved in non-conservative form (Eq. (14)). Simply switching to a conservative method — by taking the discretised divergence of the convolution of velocity and volume fraction field — would always conserve mass, but results in small sources and sinks of mass across the domain. Thus,
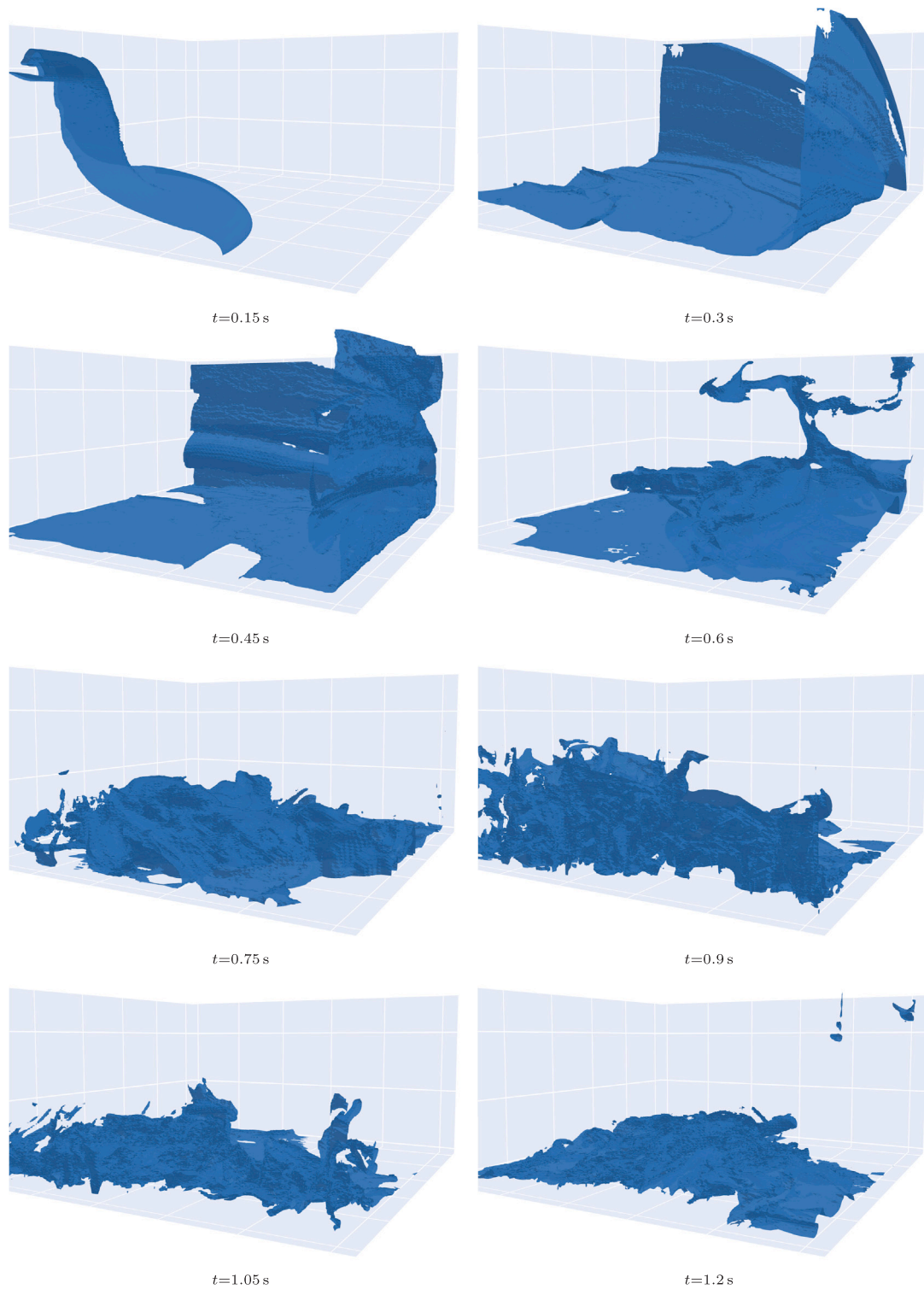
Fig. 12. Snapshots of the volume fraction isosurface drawn at a value of 0.5 from $t = 0.15\,\text{s}$ to $t = 1.2\,\text{s}$ at an interval of $0.15\,\text{s}$. The results are obtained using Model 8 shown in Table 2 and solved on a grid of $512 \times 512 \times 512$ nodes and using linear filters for the volume fraction diffusion and quadratic convolutional filters for all other terms.

the pressure/velocity correction step is used to enforce mass conservation and force the right-hand side (the discretised divergence of velocity) of the pressure correction step (Eq. (53)) to zero.
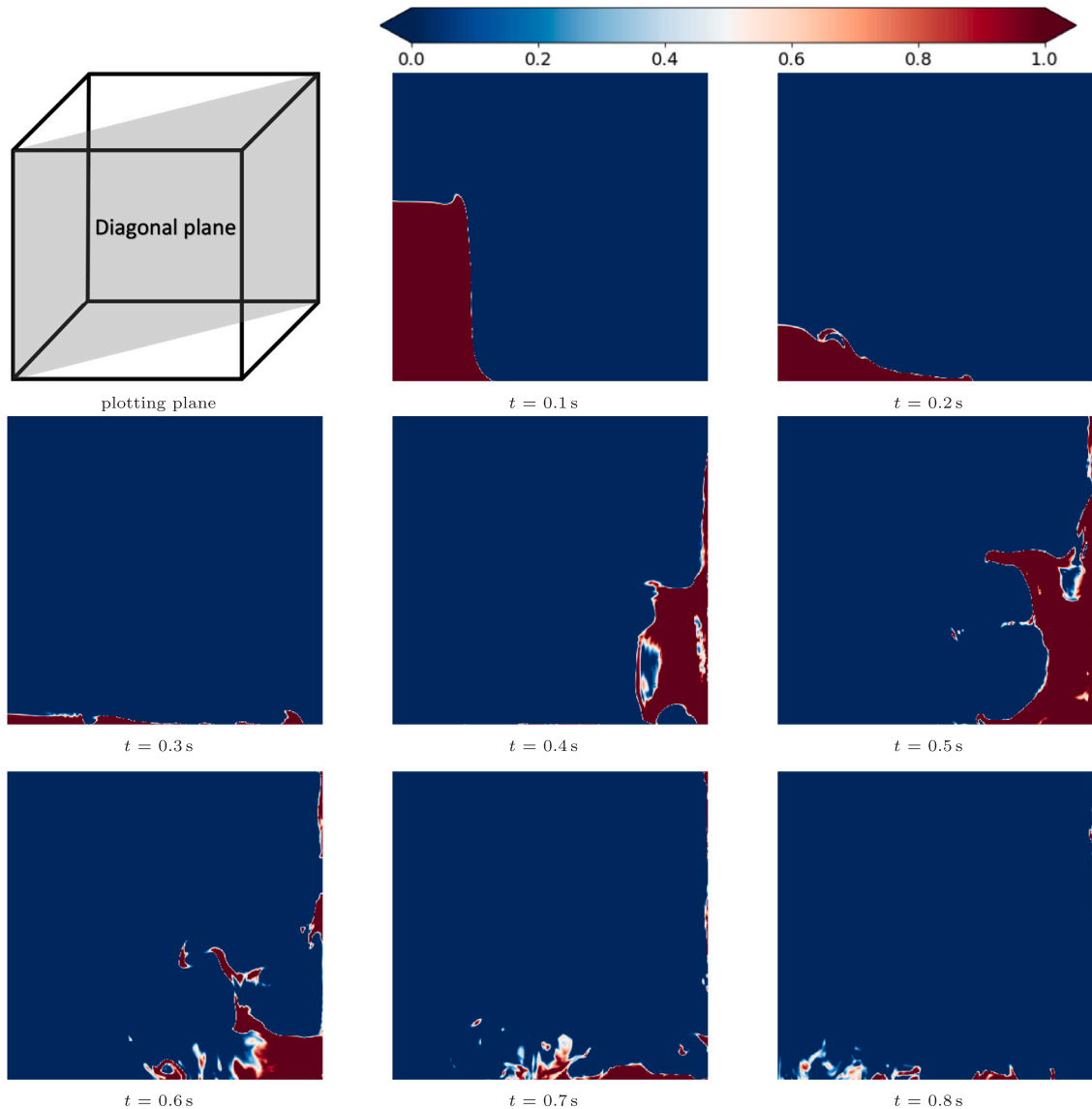
Fig. 13. Snapshots of volume fraction field on a diagonal plane $x = y$ (top left) from $t = 0.1\,\mathrm{s}$ to $t = 0.8\,\mathrm{s}$ with an interval of $0.1\,\mathrm{s}$ between the snapshots. The results are obtained from Model 8 (for details see Table 2) and solved on a grid of $512 \times 512 \times 512$ nodes using linear convolutional filters for volume fraction diffusion and quadratic convolutional filters for all other terms.

4. Discussion and conclusions

This paper further extends a new approach that was initially developed for single-phase fluids. By using tools within Artificial Intelligence (AI) software libraries, the process of solving partial differential equations (PDEs) that have been discretised through standard or other numerical methods can be replicated. Written as a convolutional neural network, this solver (referred to as NN4PDEs) gives the same results as if the numerical methods had been implemented in Fortran or C++ (to within solver tolerances). Whilst applicable to PDEs in general, this article has focused, for the first time, on applying NN4PDEs to multiphase flows with interface capturing. Furthermore, whilst the underlying discretisation methods can be arbitrary, our demonstration focuses on the use of linear, quadratic and cubic convolutional finite elements (ConvFEM), implemented exactly through convolutional neural networks on structured grids. A segregated velocity-pressure solution method is used here to enforce the incompressibility constraint. To solve the resulting matrix equation for pressure, a sawtooth multigrid method was used and implemented using a type of convolutional neural network known as a U-Net. This solution method proved to be highly efficient needing only a few iterations, with the use of standard neural network architectures such as the U-Net greatly speeding up the model development. For the important task of modelling the interface, a new compressive advection scheme was proposed. Designed to fit within the neural network framework, it

(a) Schematic of the computational domain on the central plane.

(b) Temporal evolution of bubble along a central plane.
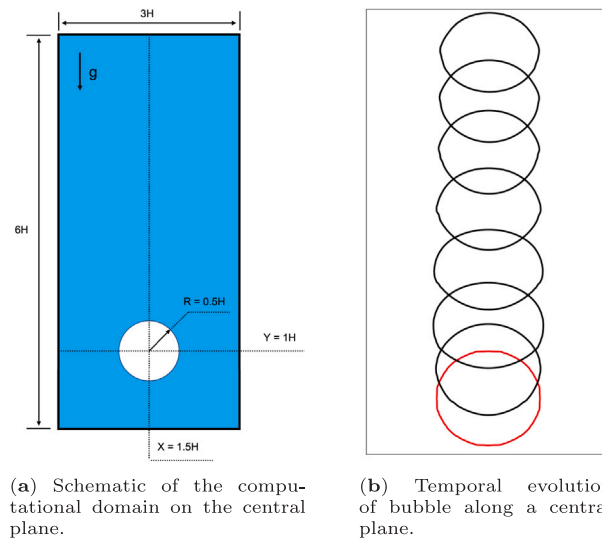
Fig. 14. Numerical prediction of a single air bubble rising in water with equivalent diameter $D_b = 2\,\text{mm}$ using NN4PDEs. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)
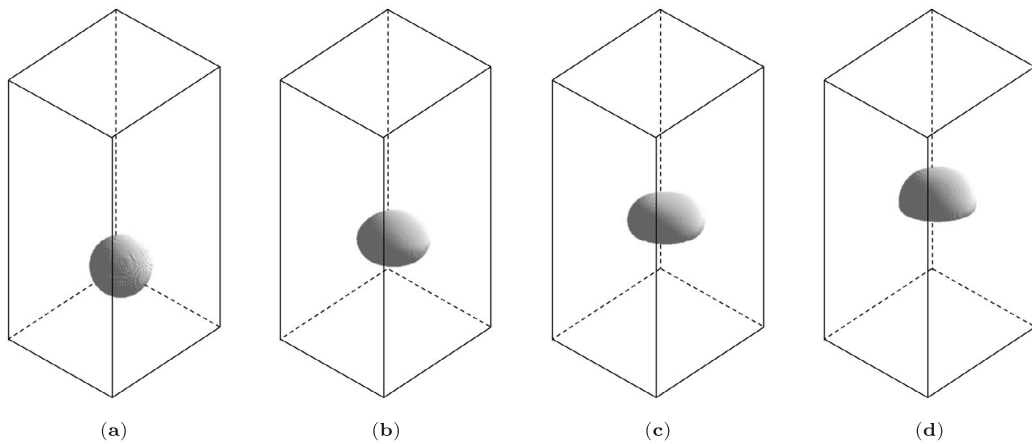

(a)   (b)   (c)   (d)

Fig. 15. Numerical prediction of a single air bubble rising in water with $D_b = 2\,\text{mm}$ using NN4PDEs. Temporal evolution of bubble visualised using an isosurface for a value of 0.5 of the volume fraction field at four dimensionless time levels, from 0 to 3 with an interval of 1. (a), 0, (b), 1, (c), 2, (d), 3.
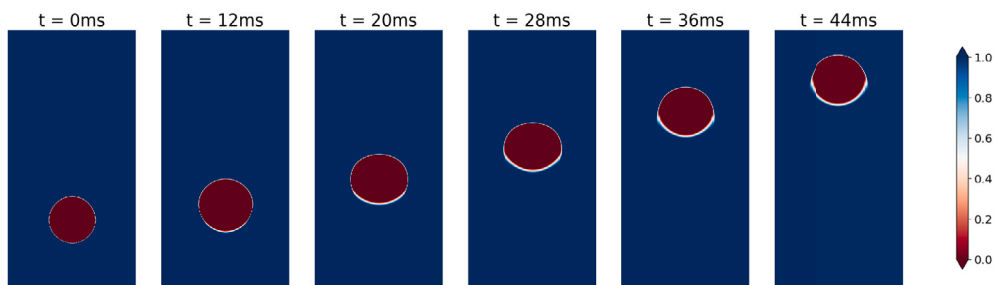
Fig. 16. Snapshots of the volume fraction field for a single rising bubble in water, plotted along the central plane in middle of the domain with equivalent diameter $D_b = 2\,\text{mm}$ and at times from $t = 0\,\text{s}$ to $t = 0.044\,\text{s}$.

was derived using a Petrov–Galerkin approach for robustness and is able to introduce diffusion smoothly in response to the variation of the solution variables.

(**a**) Bubble shape with an equivalent diameter of 1.92 mm.

(**b**) Terminal velocities with different bubble sizes.

(**c**) Bubble rising velocity as a function of time.
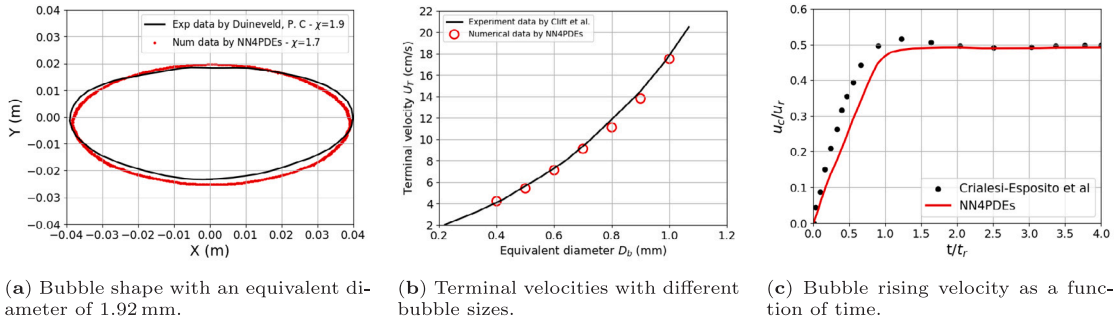
Fig. 17. Comparison of experimental data from Duineveld [89] and numerical data from Crialesi-Esposito et al. [17] with three-dimensional numerical results from NN4PDEs for a single bubble rising in water. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)
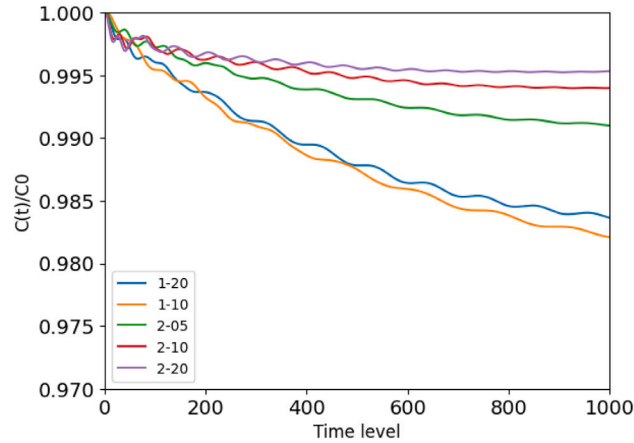
Fig. 18. Mass conservation analysis over 1000 time-steps for rising bubble case comparing different iterative approaches for solving non-hydrostatic pressure and velocity corrections. $C(t)/C0$ represents normalised volume fraction integrated over the domain, relative to the initial value $C0 = C(t = 0)$, where $C(t = n\Delta t) = \int_V C^n dV$. In the legend i-j is shown, in which i represents the number of iterations for the pressure and velocity correction within a time step and j represents the number of multigrid iterations used to solve for the non-hydrostatic pressure correction.

Two classes of test cases are used to demonstrate this approach, collapsing water columns and rising bubble problems. For the collapsing water column problems, excellent agreement is seen between the solutions from the proposed solver and both experimental data and other numerical solutions from the literature. These test cases are also used to demonstrate how linear, quadratic or cubic finite element discretisations may be used in convolutional layers to form multiphase discretisation and solution methods. It is suggested that the quadratic or cubic ConvFEM discretisations provide a good compromise between computational speed and accuracy. It is also suggested that the use of linear ConvFEM filters for the diffusion term of the volume fraction field is important, in order to reduce the tendency to produce oscillations seen when using high-order (ConvFEM) discretisations.

While it is difficult to make a rigorous comparison between the computational speed of NN4PDEs and other codes reported in the literature, we have been able to estimate how NN4PDEs compares with selected codes. What we have found is that the speed of the fastest GPU-based codes are comparable to our implementation of NN4PDEs. This seems to suggest that the current NN4PDEs implementation is reasonably well optimised even though the approach has not taken considerable time to develop. This does highlight the advantages of using neural networks to help quickly develop rapid running models. NN4PDEs (running on a single GPU) is at least 2000 faster than codes that run on CPUs, broadly in line with the expectation that code can run faster on GPUs than CPUs.

Expressing PDE solvers as neural networks marks a significant step in forming a bridge between physics-based modelling and AI, bringing to standard numerical methods and models some of the developments made in AI. One benefit of using the NN4PDEs approach is that it enables the power of AI libraries and their built-in technologies to be exploited. For example, the codes described here are deployed on an NVIDIA GPU but could have been run, with no modification, on CPUs, TPUs (tensor processing units) or the latest energy-efficient AI accelerators thanks to the writers of the machine learning libraries. Another benefit of the NN4PDEs approach is that the flow solver can be fully integrated into machine learning workflows such as AI-based surrogate models or digital twins. For example, the automatic differentiation procedures found within machine learning libraries can be applied to physics models enabling easier implementation of optimisation processes such as data assimilation, control and inversion.

CRediT authorship contribution statement

Boyang Chen: Writing – review & editing, Writing – original draft, Software, Methodology. **Claire E. Heaney:** Writing – review & editing, Writing – original draft, Supervision, Methodology. **Jefferson L.M.A. Gomes:** Writing – review & editing, Writing – original draft, Methodology. **Omar K. Matar:** Writing – review & editing, Funding acquisition. **Christopher C. Pain:** Writing – original draft, Supervision, Software, Methodology, Funding acquisition, Conceptualization, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Omar K Matar and Christopher C Pain report financial support was provided by Engineering and Physical Sciences Research Council. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code used to generate these results can be found at the following github repository: https://github.com/bc1chen/AI4PDE.

References

[1] T. Dauxois, T. Peacock, P. Bauer, C.P. Caulfield, C. Cenedese, C. Gorlé, G. Haller, G.N. Ivey, P.F. Linden, E. Meiburg, N. Pinardi, N.M. Vriend, A.W. Woods, Confronting grand challenges in environmental fluid mechanics, Phys. Rev. Fluids 6 (2021) 020501, http://dx.doi.org/10.1103/PhysRevFluids.6.020501.

[2] G.F. Hewitt, Multiphase flow in the energy industries, J. Eng. Thermophys. 17 (1) (2008) 12–23, http://dx.doi.org/10.1007/s11823-008-1002-4, URL: https://link.springer.com/article/10.1007/s11823-008-1002-4.

[3] I.D. Wilson, Y.M.J. Chew, Fluid mechanics in food engineering, Curr. Opin. Food Sci. 51 (2023) 101038, http://dx.doi.org/10.1016/j.cofs.2023.101038.

[4] H. Woodward, A.K. Schroeder, C.M.A. Le Cornec, M.E.J. Stettler, H. ApSimon, A. Robins, C. Pain, P.F. Linden, High resolution modelling of traffic emissions using the large eddy simulation code fluidity, Atmosphere 13 (8) (2022) http://dx.doi.org/10.3390/atmos13081203.

[5] J. Xiang, B. Chen, J.-P. Latham, C.C. Pain, Numerical simulation of rock erosion performance of a high-speed water jet using an immersed-body method, Int. J. Rock Mech. Min. Sci. 158 (2022) 105179, http://dx.doi.org/10.1016/j.ijrmms.2022.105179.

[6] A. Venkateshwaran, M. Kumar, M.B.S. Kumar, J.D.D. Jebaseelan, S. Ramasami, A. Joshi, C.C. Pain, Numerical study of the effect of geometry on the behaviour of internally heated melt pools for in-vessel melt retention, Prog. Nucl. Energy 156 (2023) 104555, http://dx.doi.org/10.1016/j.pnucene.2022.104555.

[7] H. Woodward, A. Schroeder, A. de Nazelle, C.C. Pain, M.E.J. Stettler, H. ApSimon, A. Robins, P.F. Linden, Do we need high temporal resolution modelling of exposure in urban areas? A test case, Sci. Total Environ. 885 (2023) 163711, http://dx.doi.org/10.1016/j.scitotenv.2023.163711.

[8] W.W. Grabowski, L.-P. Wang, Growth of cloud droplets in a turbulent environment, Annu. Rev. Fluid Mech. 45 (1) (2013) 293–324, http://dx.doi.org/10.1146/annurev-fluid-011212-140750.

[9] E. Ramirez, C.E.A. Finney, S. Pannala, C.S. Daw, J. Halow, Q. Xiong, Computational study of the bubbling-to-slugging transition in a laboratory-scale fluidized bed, Chem. Eng. J. 308 (2017) 544–556, http://dx.doi.org/10.1016/j.cej.2016.08.113.

[10] O.S. Osundare, A. Elliott, G. Falcone, L. Lao, Gas-liquid flow regime maps for horizontal pipelines: Predicting flow regimes using dimensionless parameter groups, Multiph. Sci. Technol. 34 (4) (2022) 75–99, http://dx.doi.org/10.1615/MultScienTechn.2022043690.

[11] M.I.H. Khan, M.U.H. Joardder, C. Kumar, M.A. Karim, Multiphase porous media modelling: A novel approach to predicting food processing performance, Crit. Rev. Food Sci. Nutr. 58 (4) (2018) 528–546, http://dx.doi.org/10.1080/10408398.2016.1197881.

[12] Y. Ling, D. Fuster, G. Tryggvason, S. Zaleski, A two-phase mixing layer between parallel gas and liquid streams: multiphase turbulence statistics and influence of interfacial instability, J. Fluid Mech. 859 (2018) 268–307, http://dx.doi.org/10.1017/jfm.2018.825.

[13] G. Tryggvason, M. Sussman, M.Y. Hussaini, Immersed boundary methods for fluid interfaces, in: A. Prosperetti, G. Tryggvason (Eds.), Computational Methods for Multiphase Flow, Cambridge University Press, 2007, pp. 37–77, http://dx.doi.org/10.1017/CBO9780511607486.004.

[14] R. Reddy, R. Banerjee, GPU accelerated VOF based multiphase flow solver and its application to sprays, Comput. & Fluids 117 (2015) 287–303, http://dx.doi.org/10.1016/j.compfluid.2015.05.013.

[15] S. Mirjalili, S.S. Jain, M.S. Dodd, Interface-capturing methods for two-phase flows: an overview and recent developments, in: Insights into Imaging Center for Turbulence Research: Annual Research Briefs, 2017, pp. 117–135, URL: https://ctr.stanford.edu/publications/annual-research-briefs/annual-research-briefs-2017.

[16] M.F.P. ten Eikelder, I. Akkerman, A novel diffuse-interface model and a fully-discrete maximum-principle-preserving energy-stable method for two-phase flow with surface tension and non-matching densities, Comput. Methods Appl. Mech. Engrg. 379 (2021) 113751, http://dx.doi.org/10.1016/j.cma.2021.113751.

[17] M. Crialesi-Esposito, N. Scapin, A.D. Demou, M.E. Rosti, P. Costa, F. Spiga, L. Brandt, FluTAS: A GPU-accelerated finite difference code for multiphase flows, Comput. Phys. Comm. 284 (2023) 108602, http://dx.doi.org/10.1016/j.cpc.2022.108602.

[18] S. Shin, J. Chergui, D. Juric, A solver for massively parallel direct numerical simulation of three-dimensional multiphase flows, J. Mech. Sci. Technol. 31 (2017) 1739–1751, http://dx.doi.org/10.1007/s12206-017-0322-y.

[19] S. Shin, J. Chergui, D. Juric, L. Kahouadji, O.K. Matar, R.V. Craster, A hybrid interface tracking — level set technique for multiphase flow with soluble surfactant, J. Comput. Phys. 359 (2018) 409–435, http://dx.doi.org/10.1016/j.jcp.2018.01.010.

[20] W.J. Rider, D.B. Kothe, Reconstructing volume tracking, J. Comput. Phys. 141 (2) (1998) 112–152, http://dx.doi.org/10.1006/jcph.1998.5906.

[21] R. Scardovelli, S. Zaleski, Direct numerical simulation of free-surface and interfacial flow, Annu. Rev. Fluid Mech. 31 (1) (1999) 567–603, http://dx.doi.org/10.1146/annurev.fluid.31.1.567.

[22] J. Chessa, T. Belytschko, An enriched finite element method and level sets for axisymmetric two-phase flow with surface tension, Internat. J. Numer. Methods Engrg. 58 (13) (2003) 2041–2064, http://dx.doi.org/10.1002/nme.946.

[23] S. Claus, P. Kerfriden, A CutFEM method for two-phase flow problems, Comput. Methods Appl. Mech. Engrg. 348 (2019) 185–206, http://dx.doi.org/10.1016/j.cma.2019.01.009.

[24] Z. Xie, T. Stoesser, S. Yan, Q. Ma, P. Lin, A Cartesian cut-cell based multiphase flow model for large-eddy simulation of three-dimensional wave-structure interaction, Comput. & Fluids 213 (2020) 104747, http://dx.doi.org/10.1016/j.compfluid.2020.104747.

[25] D.M. Anderson, G.B. McFadden, A.A. Wheeler, Diffuse-interface methods in fluid mechanics, Annu. Rev. Fluid Mech. 30 (1) (1998) 139–165, http://dx.doi.org/10.1146/annurev.fluid.30.1.139.

[26] H. Montazeri, S.H. Zandavi, A. Bazylak, Sharp interface models for two-phase flows: Insights towards new approaches, Comput. Methods Appl. Mech. Engrg. 322 (2017) 238–261, http://dx.doi.org/10.1016/j.cma.2017.04.022.

[27] L. Via-Estrem, P. Salinas, Z. Xie, J. Xiang, J.-P. Latham, S. Douglas, I. Nistor, C.C. Pain, Robust control volume finite element methods for numerical wave tanks using extreme adaptive anisotropic meshes, Internat. J. Numer. Methods Fluids 92 (12) (2020) 1707–1722, http://dx.doi.org/10.1002/fld.4845.

[28] R.H. Nochetto, A.J. Salgado, I. Tomas, A diffuse interface model for two-phase ferrofluid flows, Comput. Methods Appl. Mech. Engrg. 309 (2016) 497–531, http://dx.doi.org/10.1016/j.cma.2016.06.011.

[29] A. Prosperetti, G. Tryggvason (Eds.), Computational Methods for Multiphase Flow, Cambridge University Press, 2007, http://dx.doi.org/10.1017/CBO9780511607486.

[30] S. Elgeti, H. Sauerland, Deforming fluid domains within the finite element method: Five mesh-based tracking methods in comparison, Arch. Comput. Methods Eng. 23 (2016) 323–361, http://dx.doi.org/10.1007/s11831-015-9143-2.

[31] G. Tryggvason, R. Scardovelli, S. Zaleski, Direct Numerical Simulations of Gas-Liquid Multiphase Flows, Cambridge University Press, 2011, http://dx.doi.org/10.1017/CBO9780511975264.

[32] D. Pavlidis, Z. Xie, J.R. Percival, J.L.M.A. Gomes, C.C. Pain, O.K. Matar, Two- and three-phase horizontal slug flow simulations using an interface-capturing compositional approach, Int. J. Multiph. Flow 67 (2014) 85–91, http://dx.doi.org/10.1016/j.ijmultiphaseflow.2014.07.007.

[33] D. Pavlidis, J.L.M.A. Gomes, Z. Xie, J.R. Percival, C.C. Pain, O.K. Matar, Compressive advection and multi-component methods for interface-capturing, Internat. J. Numer. Methods Fluids 80 (4) (2016) 256–282, http://dx.doi.org/10.1002/fld.4078.

[34] A. Obeysekara, P. Salinas, C.E. Heaney, L. Kahouadji, L. Via-Estrem, J. Xiang, N. Srinil, A. Nicolle, O.K. Matar, C.C. Pain, Prediction of multiphase flows with sharp interfaces using anisotropic mesh optimisation, Adv. Eng. Softw. 160 (2021) 103044, http://dx.doi.org/10.1016/j.advengsoft.2021.103044.

[35] F. Banchelli, M. Garcia-Gasulla, G. Houzeaux, F. Mantovani, Benchmarking of state-of-the-art HPC clusters with a production CFD code, in: PASC 20: Proceedings of the Platform for Advanced Scientific Computing Conference, 2020, pp. 88–92, http://dx.doi.org/10.1145/3394277.3401847.

[36] K.E. Niemeyer, C.J. Sung, Recent progress and challenges in exploiting graphics processors in computational fluid dynamics, J. Supercomput. 67 (2014) 528–564, http://dx.doi.org/10.1007/s11227-013-1015-7.

[37] A. Afzal, Z. Ansari, A.R. Faizabadi, M.K. Ramis, Parallelization strategies for computational fluid dynamics software: State of the art review, Arch. Comput. Methods Eng. 24 (2017) 337–363, http://dx.doi.org/10.1007/s11831-016-9165-4.

[38] S. Memeti, L. Li, S. Pllana, J. Kołodziej, C. Kessler, Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming productivity, performance, and energy consumption, in: Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, in: ARMS-CC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1–6, http://dx.doi.org/10.1145/3110355.3110356.

[39] G. Araujo, D. Griebler, D.A. Rockenbach, M. Danelutto, L.G. Fernandes, NAS parallel benchmarks with CUDA and beyond, Softw. - Pract. Exp. 53 (1) (2023) 53–80, http://dx.doi.org/10.1002/spe.3056.

[40] J. Lai, H. Yu, Z. Tian, H. Li, A.J. Peña, Hybrid MPI and CUDA parallelization for CFD applications on multi-GPU HPC clusters, Sci. Program. 2020 (2020) http://dx.doi.org/10.1155/2020/8862123.

[41] A. Zhu, Q. Chang, J. Xu, W. Ge, A dynamic load balancing algorithm for CFD–DEM simulation with CPU-GPU heterogeneous computing, Powder Technol. 428 (2023) 118782, http://dx.doi.org/10.1016/j.powtec.2023.118782.

[42] J. Appleyard, D. Drikakis, Higher-order CFD and interface tracking methods on highly-parallel MPI and GPU systems, Comput. Fluids 46 (2011) 101–105, http://dx.doi.org/10.1016/j.compfluid.2010.10.019.

[43] S.R. Codyer, M. Raessi, G. Khanna, Using graphics processing units to accelerate numerical simulations of interfacial incompressible flows, in: ASME 2012 Fluids Engineering Division Summer Meeting, in: Fluids Engineering Division Summer Meeting, 1: Symposia, Parts A and B, 2012, pp. 625–634, http://dx.doi.org/10.1115/FEDSM2012-72176.

[44] M. Griebel, P. Zaspel, A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations, Comput. Sci. Res. Dev. 25 (2010) 65–73, http://dx.doi.org/10.1007/s00450-010-0111-7.

[45] S.H. Bryngelson, K. Schmidmayer, V. Coralic, J.C. Meng, K. Maeda, T. Colonius, MFC: An open-source high-order multi-component, multi-phase, and multi-scale compressible flow solver, Comput. Phys. Comm. 266 (2021) 107396, http://dx.doi.org/10.1016/j.cpc.2020.107396.

[46] A. Radhakrishnan, H.L. Berre, B. Wilfong, J.-S. Spratt, M. Rodriguez Jr., T. Colonius, S.H. Bryngelson, Method for portable, scalable, and performant GPU-accelerated simulation of multiphase compressible flow, 2023, http://dx.doi.org/10.48550/arXiv.2305.09163, arXiv preprint arXiv:2305.09163.

[47] X.-Z. Zhao, T.-Y. Xu, Z.-T. Ye, W.-J. Liu, A TensorFlow-based new high-performance computational framework for CFD, J. Hydrodyn. 32 (4) (2020) 735–746, http://dx.doi.org/10.1007/s42241-020-0050-0.

[48] Q. Wang, M. Ihme, Y.-F. Chen, J. Anderson, A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units, Comput. Phys. Comm. 274 (2022) 108292, http://dx.doi.org/10.1016/j.cpc.2022.108292.

[49] B. Chen, C.E. Heaney, C.C. Pain, Using AI libraries for incompressible computational fluid dynamics, 2024, http://dx.doi.org/10.48550/arXiv.2402.17913, arXiv preprint.

[50] T.R.F. Phillips, C.E. Heaney, B. Chen, A.G. Buchan, C.C. Pain, Solving the discretised neutron diffusion equations using neural networks, Internat. J. Numer. Methods Engrg. 124 (21) (2023) 4659–4686, http://dx.doi.org/10.1002/nme.7321.

[51] T.R.F. Phillips, C.E. Heaney, B. Chen, A.G. Buchan, C.C. Pain, Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport, 2023, http://dx.doi.org/10.48550/arXiv.2301.09991, arXiv preprint arXiv:2301.09991.

[52] B. Dong, Q. Jiang, Z. Shen, Image restoration: Wavelet frame shrinkage, nonlinear evolution PDEs, and beyond, Multiscale Model. Simul. 15 (1) (2017) 606–660, http://dx.doi.org/10.1137/15M1037457.

[53] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-net: Learning PDEs from data, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 3208–3216.

[54] R. Yamashita, M. Nishio, R.K.G. Do, K. Togashi, Convolutional neural networks: an overview and application in radiology, Insights Imaging 9 (2018) 611–629, http://dx.doi.org/10.1007/s13244-018-0639-9.

[55] S. Indolia, A.K. Goswami, S. Mishra, P. Asopa, Conceptual understanding of convolutional neural network- A deep learning approach, Procedia Comput. Sci. 132 (2018) 679–688, http://dx.doi.org/10.1016/j.procs.2018.05.069.

[56] M. Mishra, Convolutional neural networks explained, 2022, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. Accessed: 14-11-2023.

[57] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention, MICCAI, in: LNCS, vol. 9351, Springer, 2015, pp. 234–241, http://dx.doi.org/10.48550/arXiv.1505.04597.

[58] M. Woo, T. Jordan, T. Nandi, J.F. Dietiker, C. Guenther, D. van Essendelft, Development of an equation-based parallelization method for multiphase particle-in-cell simulations, Eng. Comput. 39 (2023) 3577–3591, http://dx.doi.org/10.1007/s00366-022-01768-6.

[59] L. Li, J. Xiang, B. Chen, C.E. Heaney, S. Dargaville, C.C. Pain, Implementing the discontinuous-Galerkin finite element method using graph neural networks, 2024, http://dx.doi.org/10.2139/ssrn.4698813, SSRN preprint.

[60] M. Abadi, Agarwal, P. Aand Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from www.tensorflow.org.

[61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, 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: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., 2019, URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[62] R. Frostig, M. Johnson, C. Leary, Compiling machine learning programs via high-level tracing, 2018, URL: https://mlsys.org/Conferences/doc/2018/146.pdf.

[63] Graphcore, Intelligence processing units, 2022, https://www.graphcore.ai/products/ipu. Accessed: 30-10-2023.

[64] Cerebras, CS-2: A revolution in AI infrastructure, 2022, https://www.cerebras.net/product-system/. Accessed: 30-10-2023.

[65] H. Carter Edwards, C.R. Trott, D. Sunderland, Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, J. Parallel Distrib. Comput. 74 (12) (2014) 3202–3216, http://dx.doi.org/10.1016/j.jpdc.2014.07.003.

[66] W. Verdier, P. Kestener, A. Cartalade, Performance portability of lattice Boltzmann methods for two-phase flows with phase change, Comput. Methods Appl. Mech. Engrg. 370 (2020) 113266, http://dx.doi.org/10.1016/j.cma.2020.113266.

[67] A. Chattopadhyay, V.M.K. Kotteda, V. Kumar, W. Spotz, Next generation exascale capable mutliphase solver with trilinos, in: Proceedings of the ASME 2016 International Mechanical Engineering Congress & Exposition, Volume 14: Emerging Technologies; Materials: Genetics to Structures; Safety Engineering and Risk Analysis, 2016, V014T07A025, http://dx.doi.org/10.1115/IMECE2016-67962.

[68] 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, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[69] A.B. Buhendwa, S. Adami, N.A. Adams, Inferring incompressible two-phase flow fields from the interface motion using physics-informed neural networks, Mach. Learn. Appl. 4 (2021) 100029, http://dx.doi.org/10.1016/j.mlwa.2021.100029.

[70] B.v. Merriënboer, O. Breuleux, A. Bergeron, P. Lamblin, Automatic differentiation in ML: Where we are and where we should be going, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 8771–8781.

[71] A. Guenes Baydin, B.A. Pearlmutter, A.A. Radul, S.J. M., Automatic differentiation in machine learning: A survey, J. Mach. Learn. Res. 18 (2018) 1–43.

[72] Y. Li, et al., An AI-based integrated framework for anisotropic electrical resistivity imaging, 2024, in preparation.

[73] C.A.J. Fletcher, Computational Techniques for Fluid Dynamics, vol. 1, Springer, 1998, http://dx.doi.org/10.1007/978-3-642-58229-5_7.

[74] S. Linge, H.P. Langtangen, Finite Difference Computing with PDEs: A Modern Software Approach, Springer, 2017, pp. 207–322, http://dx.doi.org/10.1007/978-3-319-55456-3_3.

[75] L.N. Trefethen, Finite difference and spectral methods for ordinary and partial differential equations, 1996, unpublished text, available at http://people.maths.ox.ac.uk/trefethen/pdetext.html. (visited 01-03-2024).

[76] C.M. Bishop, H. Bishop, Deep Learning: Foundations and Concepts, Springer, 2024, pp. 287–324, http://dx.doi.org/10.1007/978-3-031-45468-4.

[77] T.R.F. Phillips, Neural network transport solver, 2022, https://github.com/trfphillips/Neural-Network-Transport-Solver.

[78] J. Donéa, A. Huerta, Finite Element Methods for Flow Problems, John Wiley & Sons, 2003, http://dx.doi.org/10.1002/0470013826.

[79] V. Inguva, E.Y. Kenig, J. Blair Perot, A front-tracking method for two-phase flow simulation with no spurious currents, J. Comput. Phys. 456 (2022) 111006, http://dx.doi.org/10.1016/j.jcp.2022.111006.

[80] P. Wesseling, Principles of Computational Fluid Dynamics, Springer, Berlin, 2001.

[81] R. Codina, A discontinuity-capturing crosswind-dissipation for the finite element solution of the convection-diffusion equation, Comput. Methods Appl. Mech. Engrg. 110 (3) (1993) 325–342, http://dx.doi.org/10.1016/0045-7825(93)90213-H.

[82] P. Hansbo, C. Johnson, Adaptive streamline diffusion methods for compressible flow using conservation variables, Comput. Methods Appl. Mech. Engrg. 87 (2) (1991) 267–280, http://dx.doi.org/10.1016/0045-7825(91)90008-T.

[83] D.M. Greaves, Simulation of viscous water column collapse using adapting hierarchical grids, Internat. J. Numer. Methods Fluids 50 (6) (2006) 693–711.

[84] V.-T. Nguyen, N.T. Nguyen, T.-H. Phan, W.-G. Park, Efficient three-equation two-phase model for free surface and water impact flows on a general curvilinear body-fitted grid, Comput. & Fluids 196 (2020) 104324.

[85] V.-T. Nguyen, V.-D. Thang, W.-G. Park, A novel sharp interface-capturing method for two-and three-phase incompressible flows, Comput. & Fluids 172 (2018) 147–161.

[86] P. Kiely, NVIDIA A10 vs A100 GPUs for LLM and stable diffusion inference, 2023, https://www.baseten.co/blog/nvidia-a10-vs-a100-gpus-for-llm-and-stable-diffusion-inference/. Accessed: 01-03-2024.

[87] G.H. Yeoh, T. Barber, Assessment of interface-capturing methods in computational fluid dynamics (CFD) codes—A case study, J. Comput. Multip. Flows 1 (2) (2009) 201–215, http://dx.doi.org/10.1260/175748209789563946.

[88] J.C. Martin, W.J. Moyce, J. Martin, W. Moyce, W.G. Penney, A. Price, C. Thornhill, Part IV. An experimental study of the collapse of liquid columns on a rigid horizontal plane, Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 244 (882) (1952) 312–324, http://dx.doi.org/10.1098/rsta.1952.0006.

[89] P. Duineveld, The rise velocity and shape of bubbles in pure water at high Reynolds number, J. Fluid Mech. 292 (1995) 325–332, http://dx.doi.org/10.1017/S0022112095001546.

[90] R. Clift, J.R. Grace, M.E. Weber, Bubbles, Drops, and Particles, Courier Corporation, 2005.