Intrusive deconvolutional neural networks for enhancing PIC/FLIP solutions

Y. van Halder, B.Sanderse
Scientific Computing Group
Centrum Wiskunde & Informatica (CWI)
P.O. Box 94079, 1090 GB, Amsterdam, the Netherlands
`yvanhalder@gmail.com`

B. Koren
Center for Analysis, Scientific Computing and Applications
Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven, the Netherlands

Abstract

Traditional fluid flow predictions require large computational resources. Despite recent progress in parallel and GPU computing, the ability to run fluid flow predictions in real-time is often infeasible. Recently developed machine learning approaches, which are trained on high-fidelity data, perform unsatisfactorily outside the training set and remove the ability of utilising legacy codes after training. We propose a novel methodology that uses a deep learning approach that can be used within a low-fidelity fluid flow solver to significantly increase the accuracy of the low-fidelity simulations. The resulting solver enables accurate while reducing computational times up to 100 times. The deep neural network is trained on a combination of low- and high-fidelity data, and the resulting solver is referred to as a multi-fidelity solver. The proposed methodology is demonstrated by means of enhancing a fluid flow simulator, known as PIC/FLIP, which is a popular fluid flow simulator in the field of computer generated imagery.

1   Introduction

The huge increase in computational power over the last decades has resulted in an increasing interest in real-time computations for physics problems. These real-time computations may be used for monitoring a physical asset or real-time optimisation of a dynamic process. In many industrial processes, such as coastal engineering [1], vehicle design [2], computer generated imagery [3, 4, 5, 6], fluid flow predictions are essential. In fluid dynamics, a distinction can be made between low-fidelity and high-fidelity simulations. Low-fidelity simulations are computationally cheap to perform, but have limited accuracy, as they do not capture all the underlying physics or do not resolve all scales. On the other hand, high-fidelity simulations incorporate all the relevant physical phenomena and corresponding scales, but may require a tremendous amount of computational resources. This makes it difficult to this date to perform accurate fluid simulations in real-time.

In order to address the problem of computational expense, various types of methods have been introduced. A commonly used method is Principal Component Analysis [7, 8], which provides the desired speed-up by transforming the dynamics of the fluid flow simulation to operations on linear combinations of snapshots, therefore restricting the richness of the dynamics. Recently developed data-driven methods use machine learning algorithms [9, 10, 2], such as regression forests and neural networks, that are trained on a large set of high-fidelity fluid flow simulation data. After training, these

approaches are able to simulate fluid flows in real-time. However, they often perform unsatisfactorily outside the training set resulting in unrealistic fluid flow simulations. This extrapolation problem is a weakness of almost all machine learning approaches but can be partially overcome by supplying the machine learning algorithm with more information about the underlying physics problem during training [9]. Furthermore, these recent deep learning methods have not yet been combined with existing fluid flow solvers, which have been developed and validated over many years.

We propose a novel method that *enhances a low-fidelity fluid flow solver by training a deep neural network that maps values from a low-resolution computational grid to a high-resolution computational grid. This neural network is then used intrusively to enhance low-fidelity simulations at each time step*. The input and output of the neural network are the current state of the low-fidelity simulation and the approximate high-fidelity current state, respectively. Compared to data-driven approaches that are purely trained on high-fidelity data, the crucial advantage of our model+data-driven approach is that the low-fidelity simulation is acting as a preconditioner, as it already comprises parts of the physics involved in the problem. Our proposed multi-fidelity approach is able to significantly enhance the accuracy of low-fidelity fluid flow predictions, even outside the training set.

In this work, we focus on enhancing a so-called Particle In Cell/Fluid Implicit Particle (PIC/FLIP) solver [11]. The PIC/FLIP solver is a popular fluid flow solver in the field of computer generated imagery (CGI) [12, 13], because of its straightforward parallel implementation. The solver evolves a set of particles (representing the fluid) over time by computing fluid flow velocities on a staggered Cartesian grid where the fidelity of the simulation is determined by the resolution of the grid and the total number of particles. A parallel implementation of the PIC/FLIP solver is able to simulate millions of particles on a coarse underlying grid in real-time. The capability to simulate a fluid flow in real-time is promising, but as these solutions are in general computed using a coarse grid, it often results in inaccurate fluid flow simulations. Our goal is to *increase the accuracy of 3D low-fidelity PIC/FLIP simulations at each time step, by incorporating into the solver: a deep neural network, which is trained on high-fidelity data*. This allows for accurate simulations in real-time. We focus on enhancing PIC/FLIP sloshing simulations in a rectangular tank, as it is commonly encountered in CGI applications. This test case comprises a wide range of fluid mechanics phenomena.

To see where the neural networks fit into the PIC/FLIP framework, we first discuss the PIC/FLIP solver in detail in section 2. Section 3 discusses our multi-fidelity approach in detail. Lastly, section 4 studies the trained neural network in more detail, shows numerical results of our approach in enhancing a 3D sloshing simulation, and shows how the trained neural network generalises outside the training set.

2   PIC/FLIP for Simulating Fluid Flows in 3D

In this section we discuss the fluid flow solver in more detail. The governing equations for incompressible fluid flow are the Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0 \,, \tag{1a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \mathbf{F} + \nu \nabla^2 \mathbf{u} \,, \tag{1b}$$

where $\mathbf{u}$ is the velocity field, p the pressure, ρ the density, $\mathbf{F}$ the body-force vector, and ν the kinematic viscosity. The first equation (1a) is known as the incompressibility condition, while the second set of equations (1b) is known as the momentum equation. When solving these equations, we can resort to grid-based methods or particle-based methods. In this work we use a PIC/FLIP solver for predicting single-phase free-surface flows, as it allows for real-time computation when implemented efficiently on a GPU. This numerical method is a combination of a grid-based method and a particle-based method, and is often used to simulate fluid motions in movies or computer games due to its easy parallel implementation. The particles represent the fluid, and the positions of these particles are evolved over time by using velocity values that are computed on a staggered grid. In this section we discuss the individual steps in the solver in more detail, as they determine the architecture of our neural network, which is discussed in section 3.

The PIC/FLIP framework comprises two stages; 1) initialisation of grid and particles, 2) time-stepping loop.

2.1 Initialisation

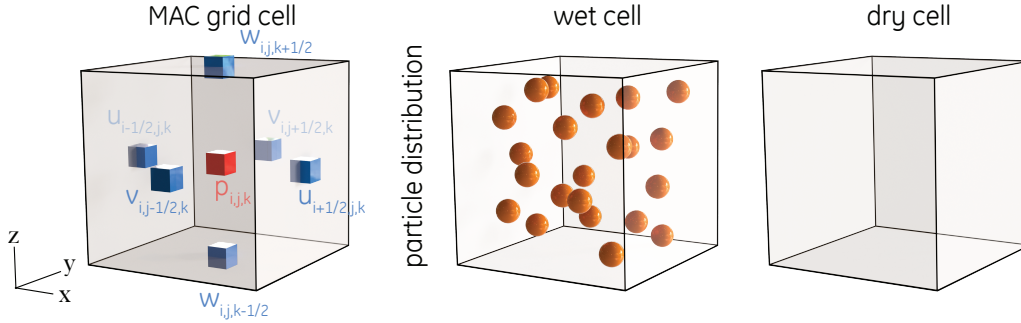The initialisation stage is shown schematically in figure 1.

Figure 1: The staggered grid quantities for the PIC/FLIP solver.

MAC staggered grid

A Marker And Cell (MAC) staggered grid is used, which defines the velocity vector $\mathbf{u} = (u, v, w)$ at the corresponding cell faces (blue cubes), while the pressure is defined at the centre of the cell (red cube). The main reason for using a staggered grid is to remove the odd-even coupling of velocity and pressure, which may occur when using a collocated grid where velocity and pressure are defined at the same locations. The staggered grid is characterised by the domain dimensions (L_x, L_y, L_z) and the number of cells in each of the three coordinate directions (N_x, N_y, N_z), which results in cells with dimensions $(\Delta x, \Delta y, \Delta z) = (\frac{L_x}{N_x}, \frac{L_y}{N_y}, \frac{L_z}{N_z})$. In the remainder of this paper, we assume that the number of grid cells is chosen such that $\Delta x = \Delta y = \Delta z$, and this uniform cell width is denoted as Δs. The indexing of the pressure and face velocities is shown in figure 1. The indices i, j, k correspond to the cell number in x, y, z-direction, respectively, where the first cell in each direction has an index 0. To clarify, $u_{-\frac{1}{2}, j, k}$ and $u_{N_x - \frac{1}{2}, j, k}$ denote the x-component of the velocity at the left and right boundaries of the domain, respectively.

Particle initialisation

After the grid has been initialised, we specify which of the cells need to be filled with particles, i.e., which cells are wet, corresponding to the initial state of the fluid. Following the implementation in [12] we place 8 randomly placed particles within each wet cell and set the initial velocity of these particles to zero. We denote the total number of particles by N_p, and by $\mathbf{p}^{(i)} = (p_x^{(i)}, p_y^{(i)}, p_z^{(i)})$ and $\mathbf{p_u}^{(i)} = (p_u^{(i)}, p_v^{(i)}, p_w^{(i)})$ we denote the position and velocity of the i-th particle, respectively. An example of an initialisation of grid cells and particles is shown in figure 2.
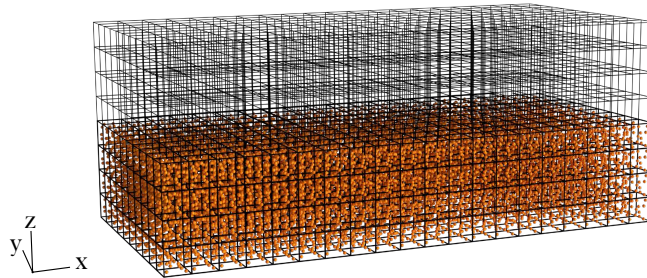

Figure 2: Grid and particle initialisation with $(N_x, N_y, N_z) = (20, 8, 8)$. The top half of the grid comprises dry cells while the cells in the bottom half are filled with particles.

2.2 Time-stepping loop

The time-stepping loop comprises 8 steps, which are performed in a sequential loop to advance the current solution at time t to the new time-level $t + \Delta t$, where Δt denotes the time step. These 8 steps are detailed below.

2.2.1 Particle to grid transfer

A weighted average of particle velocities is used to compute the velocities at the grid-cell faces. In order to compute the velocity at a given face centre, particles that lie within a sphere with a $2\Delta s$ radius, centred around this face centre, are used for computing the new velocity value. Consequently, the new face centre velocities are given by:

$$u_{i+\frac{1}{2},j,k} = \frac{\sum_{i=1}^{N_p} p_u^{(i)} h(\mathbf{p}^{(i)} - \mathbf{x}_{i+\frac{1}{2},j,k})}{\sum_{i=1}^{N_p} h(\mathbf{p}^{(i)} - \mathbf{x}_{i+\frac{1}{2},j,k})} , \tag{2a}$$

$$v_{i,j+\frac{1}{2},k} = \frac{\sum_{i=1}^{N_p} p_v^{(i)} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j+\frac{1}{2},k})}{\sum_{i=1}^{N_p} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j+\frac{1}{2},k})} , \tag{2b}$$

$$w_{i,j,k+\frac{1}{2}} = \frac{\sum_{i=1}^{N_p} p_w^{(i)} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j,k+\frac{1}{2}})}{\sum_{i=1}^{N_p} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j,k+\frac{1}{2}})} , \tag{2c}$$

where the vectors $\mathbf{x}$ correspond to the locations of the face centres, and where h is the so-called kernel. The choice of kernel h depends on the application. In general the commonly used cubic kernel is considered to be robust and is therefore employed in the remainder of this paper[12]:

$$k(\mathbf{r}) \begin{cases} (4\Delta s^2 - \|\mathbf{r}\|_2^2)^3 & , \|\mathbf{r}\|_2 \leq 2\Delta s , \\ 0 & , \text{otherwise} . \end{cases} \tag{3}$$

The particles that are used for the grid transfer for a single face velocity are schematically shown in figure 3.
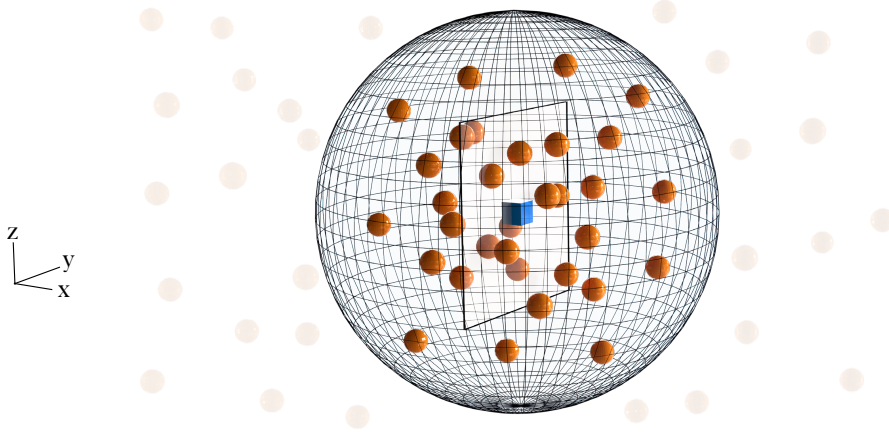

Figure 3: Particles that are used for computing one face velocity.

2.2.2 Store velocities

After the particle velocities have been transferred to the grid, the resulting face velocities are copied for later use, i.e., we define:

$$u^*_{i+\frac{1}{2},j,k} = u_{i+\frac{1}{2},j,k}, \quad v^*_{i,j+\frac{1}{2},k} = v_{i,j+\frac{1}{2},k}, \quad w^*_{i,j,k+\frac{1}{2}} = w_{i,j,k+\frac{1}{2}} . \tag{4}$$

The copied velocities remain unaltered for the rest of the time step.

2.2.3 Add body forces

The external forces, such as gravity, are added to the velocity field by a simple forward Euler time-integration:

$$u_{i+\frac{1}{2},j,k} = u_{i+\frac{1}{2},j,k} + \Delta t F_x(\mathbf{x}_{i+\frac{1}{2},j,k}) , \tag{5a}$$

$$v_{i,j+\frac{1}{2},k} = v_{i,j+\frac{1}{2},k} + \Delta t F_y(\mathbf{x}_{i,j+\frac{1}{2},k}) , \tag{5b}$$

$$w_{i,j,k+\frac{1}{2}} = w_{i,j,k+\frac{1}{2}} + \Delta t F_z(\mathbf{x}_{i,j,k+\frac{1}{2}}) , \tag{5c}$$

where F_x, F_y and F_z are the components of the body force acting on the fluid in each coordinate direction, calculated at the corresponding face centres. A more accurate time-integration scheme may be used, but forward Euler is very easy to implement, and is therefore considered as the standard in PIC/FLIP [11, 12]. When Δs is set, we need to choose Δt accordingly such that the forward Euler time integration remains stable, which is often not an issue when using PIC/FLIP due to the often coarse computational grid.

2.2.4 Enforce boundary conditions

We assume that the domain boundaries and obstacles are solid, and therefore solid-wall boundary conditions are imposed at these locations, which state that fluid is not allowed to flow out of the domain. As a result, the boundary condition is given by:

$$\mathbf{u} \cdot \mathbf{n} = \mathbf{u}_{\text{boundary}} \cdot \mathbf{n} , \tag{6}$$

where $\mathbf{n}$ is the normal vector pointing outside the domain. Furthermore, this boundary condition can be simplified when using a staggered grid, where the faces intersect with the domain boundaries:

$$u_{-\frac{1}{2},j,k} = u_{\text{boundary},0}, \quad u_{N_x-\frac{1}{2},j,k} = u_{\text{boundary},L_x} , \tag{7a}$$

$$v_{i,-\frac{1}{2},k} = v_{\text{boundary},0}, \quad v_{i,N_y-\frac{1}{2},k} = v_{\text{boundary},L_y} , \tag{7b}$$

$$w_{i,j,-\frac{1}{2}} = w_{\text{boundary},0}, \quad w_{i,j,N_z-\frac{1}{2}} = w_{\text{boundary},L_z} , \tag{7c}$$

where we assumed that the domain does not deform and the boundaries have a uniform velocity, i.e., $u_{\text{boundary},0} = u_{\text{boundary},L_x}$, $v_{\text{boundary},0} = v_{\text{boundary},L_y}$, $w_{\text{boundary},0} = w_{\text{boundary},L_z}$.

2.2.5 Determine which cells contain particles

The cells that contain at least one particle are referred to as wet cells. We define a new quantity $m_{i,j,k}$:

$$m_{i,j,k} \begin{cases} 1 & \text{, if cell } (i,j,k) \text{ contains at least one particle ,} \\ 0 & \text{, otherwise .} \end{cases} \tag{8}$$

2.2.6 Pressure correction

The PIC/FLIP method simulates incompressible fluid flow, which means that the velocities at the face centres of all i,j,k have to satisfy the following discrete version of incompressibility constraint (1a):

$$(\nabla \cdot \mathbf{u})_{i,j,k} \to \frac{u_{i+\frac{1}{2},j,k} - u_{i-\frac{1}{2},j,k}}{\Delta s} + \frac{v_{i,j+\frac{1}{2},k} - v_{i,j-\frac{1}{2},k}}{\Delta s} + \frac{w_{i,j,k+\frac{1}{2}} - w_{i,j,k-\frac{1}{2}}}{\Delta s} = 0 , \tag{9}$$

which states that the central second-order accurate divergence, computed at each cell centre, should be zero. In general, the face velocities from (5a)-(5c) do not satisfy this constraint. Therefore, the velocities are corrected by subtracting a pressure gradient. In order to calculate this pressure gradient, we solve the pressure equation in the cells that are wet ($m_{i,j,k} = 1$):

$$(\nabla^2 p)_{i,j,k} = \frac{1}{\Delta t} (\nabla \cdot \mathbf{u})_{i,j,k} , \tag{10}$$

where

$$(\nabla^2 p)_{i,j,k} \to \frac{p_{i+1,j,k} + p_{i-1,j,k}}{\Delta s^2} + \frac{p_{i,j+1,k} + p_{i,j-1,k}}{\Delta s^2} + \frac{p_{i,j,k+1} + p_{i,j,k-1}}{\Delta s^2} - \frac{6p_{i,j,k}}{\Delta s^2} , \tag{11}$$

5

and set $p_{i,j,k} = 0$ in dry cells. Notice that near the domain boundaries or obstacles we require the pressures located outside the domain, which are not defined, and a different formulation is necessary. To enforce that there is no flow through solid boundaries, we employ the commonly used homogeneous Neumann boundary condition for the pressure at these boundaries. Even though the homogeneous Neumann boundary condition may translate to a non-zero tangential velocity component along the boundaries, it is the easiest to implement while still enforcing that no fluid flows outside the domain. Other boundary conditions, e.g., strict no-slip boundary conditions, are not straightforward to implement on a staggered grid and are not considered in this paper. The homogeneous Neumann boundary conditions set the pressure coefficient of the cells outside the domain or inside an obstacle to zero and the central pressure coefficient 6 in (11) is decreased by the number of cells outside the domain or inside an obstacle, which translated to a pressure gradient that is zero on the boundary [12]. All this results in a sparse linear system of the form:

$$
A \begin{pmatrix} p_{0,0,0} \\ p_{1,0,0} \\ \vdots \\ p_{N_x-1,N_y-1,N_z-1} \end{pmatrix} = \frac{1}{\Delta t} \begin{pmatrix} (\nabla \cdot \mathbf{u})_{0,0,0} \\ (\nabla \cdot \mathbf{u})_{1,0,0} \\ \vdots \\ (\nabla \cdot \mathbf{u})_{N_x-1,N_y-1,N_z-1} \end{pmatrix} , \tag{12}
$$

where A is the pressure Poisson matrix. This system is solved using the preconditioned conjugate gradient method, as it does not require the explicit storage of the coefficient matrix A, and still exhibits relatively fast convergence for symmetric matrices A when compared to Jacobi iteration.

After the pressures have been computed, we correct the face velocities, that do not coincide with a domain boundary, as follows:

$$
u_{i+\frac{1}{2},j,k} = u_{i+\frac{1}{2},j,k} - \frac{\Delta t}{\Delta s}(p_{i+1,j,k} - p_{i,j,k}) , \tag{13a}
$$

$$
v_{i,j+\frac{1}{2},k} = v_{i,j+\frac{1}{2},k} - \frac{\Delta t}{\Delta s}(p_{i,j+1,k} - p_{i,j,k}) , \tag{13b}
$$

$$
w_{i,j,k+\frac{1}{2}} = w_{i,j,k+\frac{1}{2}} - \frac{\Delta t}{\Delta s}(p_{i,j,k+1} - p_{i,j,k}) , \tag{13c}
$$

which now satisfy the constraint (9).

2.2.7 Grid to particle transfer

Once the grid velocities are known, they have to be transferred back to the particles in order to advect the particles. To calculate the new particle velocities, we use *trilinear interpolation*. We will discuss how to compute the $u-$component of the particle velocity $p_u^{(i)}$, and the remaining two components can be computed in a similar fashion.

The face centre velocities that are used for the trilinear interpolation procedure for computing the $u-$component of the velocity $p_u^{(i)}$ for a single particle are shown in figure 4.
The $u-$component of the velocity is now calculated as:

$$
x^* = \frac{x - x_0}{x_1 - x_0}, \quad y^* = \frac{y - y_0}{y_1 - y_0}, \quad z^* = \frac{z - z_0}{z_1 - z_0} , \tag{14a}
$$

$$
\begin{aligned}
u_{00} &= u_{000}(1 - x^*) + u_{100}x^*, \quad u_{10} = u_{001}(1 - x^*) + u_{101}x^* , \\
u_{01} &= u_{010}(1 - x^*) + u_{110}x^*, \quad u_{11} = u_{011}(1 - x^*) + u_{111}x^* ,
\end{aligned} \tag{14b}
$$

$$
\begin{aligned}
u_0 &= u_{00}(1 - y^*) + u_{01}y^* , \\
u_1 &= u_{10}(1 - y^*) + u_{11}y^* ,
\end{aligned} \tag{14c}
$$

$$
p_u^{(i)} = u_0(1 - z^*) + u_1 z^* . \tag{14d}
$$

This trilinear interpolation is performed two times to perform the PIC/FLIP velocity update; with the new grid velocities $u_{i+\frac{1}{2},j,k}$ from (13b), and the old grid velocities $u_{i+\frac{1}{2},j,k}^*$ from (4). As a result, two
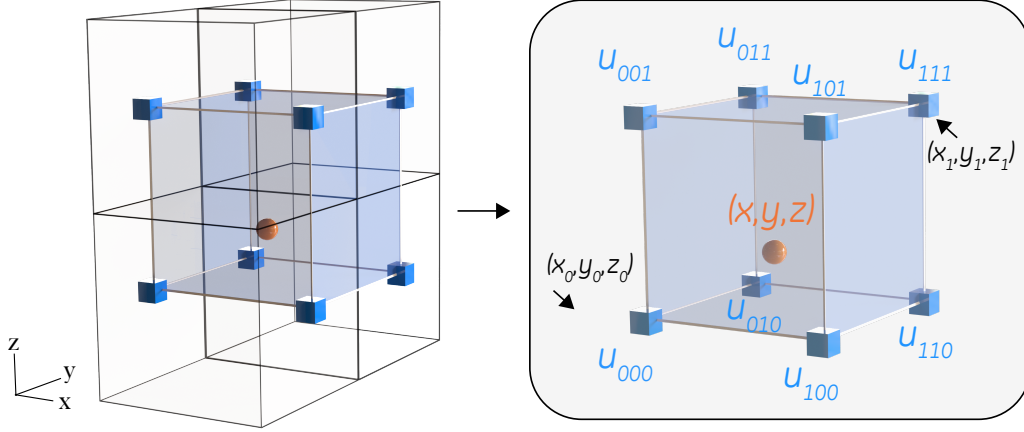
Figure 4: The face velocities that are used for the trilinear interpolation procedure to calculate the $u-$component of the particle velocity.

particle velocities are obtained $p_{\text{new},u}^{(i)}$ and $p_{\text{old},u}^{(i)}$. The new particle velocity is calculated as follows:

$$p_u^{(i)} \leftarrow (1-f)(p_u^{(i)} - p_{\text{old},u}^{(i)}) + p_{\text{new},u}^{(i)} \,, \tag{15}$$

where $p_u^{(i)}$ is the particle velocity from the previous time step, and $f \in [0,1]$ the so-called PICness parameter. If $f = 1$, then the velocity update is known as the PIC update, while setting $f = 0$ corresponds to the FLIP update. PIC is known for its good stability properties, but suffers from severe numerical diffusion. On the other hand, FLIP does not suffer from numerical diffusion, but can become unstable. Therefore, a weighted average between PIC and FLIP is taken, $f \in (0,1)$, to obtain a stable, but less diffusive solution. Notice that we did not include physical diffusion induced by the viscosity term in the method so far. In PIC/FLIP methods, the physical diffusion term in the Navier-Stokes equations is approximated by tweaking the numerical viscosity by choosing a specific value for the PICness parameter f, namely, f can be related to the kinematic viscosity as:

$$f = \max\left\{1, \frac{6\Delta t \nu}{\Delta s^2}\right\} \,, \tag{16}$$

and the value for ν should be set to the desired kinematic viscosity accordingly. However, it is common practice to use $f = 0.99$, which yields good results for water flow [12] and is used in the remainder of this paper. Physical diffusion by viscosity can be implemented in a PIC/FLIP solver but also requires no-slip boundary conditions and is not considered in this paper for ease of implementation.

2.2.8 Advect particles

A suitable time-integrator is used to advect the particles. A good trade-off between computational expense and accuracy is the second-order accurate Runge-Kutta method (RK2) which mimics the midpoint rule. The procedure for advecting the particles is given by:

1. Advect the particles for the duration of a half time step: $\mathbf{p}_{\text{halfway}}^{(i)} = \mathbf{p}^{(i)} + \frac{\Delta t}{2}\mathbf{p}_{\mathbf{u}}^{(i)}$,

2. Compute new particle velocities using trilinear interpolation, using the grid velocities $u_{i+\frac{1}{2},j,k}$ to obtain $\mathbf{p}_{\text{halfway},\mathbf{u}}^{(i)}$,

3. Compute new particle positions as $\mathbf{p}^{(i)} = \mathbf{p}^{(i)} + \Delta t \mathbf{p}_{\text{halfway},\mathbf{u}}^{(i)}$.

After computing new particle positions $\mathbf{p}_{\text{halfway}}^{(i)}$ and $\mathbf{p}^{(i)}$, it may appear that some particles have crossed the boundary of the domain. Particles that crossed the domain boundary are reflected back into the domain to prevent particles leaving the domain [12].

2.3 PIC/FLIP for fluid sloshing

The main focus of this paper is to enhance low-fidelity PIC/FLIP sloshing simulations. These sloshing simulations can be performed in a number of ways and we choose the one that is easiest to implement. In order to induce fluid sloshing, the gravity vector is rotated along the x-axis and y-axis independently. To clarify, the rotation of the gravity vector enters the fluid solver in the body force as $\mathbf{F} = \mathbf{g} = 9.81(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$, where ψ and ϕ are the rotation angles along the x and y axis, respectively. Rotating the gravity vector mimics rotation of the tank without the need to complicate the boundary conditions of the tank boundary to incorporate rotational accelerations.

2.4 Accuracy of PIC/FLIP

The accuracy of a PIC/FLIP simulation is determined by the number of particles N_p and the grid resolution Δs. As the particles do not react with each other, the steps that involve particles are easily implemented on a GPU. The number of particles can be chosen to be large (millions) while still being able to run in real-time. Most computational work occurs in the computation on the grid, especially in the solution of equation (12) to compute the pressures at the cell centres. It is not straightforward to efficiently implement these computations on a GPU. Solving the pressure equation is done in $O(N_c \log(N_c))$ time, where N_c is the total number of grid cells. Decreasing the grid resolution by a factor 2 in each coordinate direction, i.e., $\Delta s \to 2\Delta s$, results in a pressure solve that is approximately 10 times faster. As a result, when choosing a coarse grid, we can still easily simulate up to millions of particles in real-time, but computing solutions on a coarse grid results in a significant drop in accuracy of the particle positions when compared to simulating flows on a fine grid with the same number of particles. This drop in accuracy is mainly caused by the inaccurate incompressible velocity field that is computed on a the coarse grid, and the trilinear interpolation that is used to transfer these grid velocities to the particles. For instance, the error in the trilinear interpolation is composed of three linear interpolation errors R_{int}, which are bounded by [14]

$$R_{\text{int}} \leq C\Delta s^2 \, , \tag{17}$$

where C is a constant depending on the smoothness of the velocity field that is transferred to the particles. Equation (17) expresses that the error increases quadratically when coarsening the grid. Insufficiently accurate solutions may quickly arise when coarsening the computational grid. This quadratic scaling of errors holds for other grid computations as well (divergence computation (9), pressure computation (12)).

3 Deep learning for enhancing low-fidelity fluid flow predictions

Our approach is to incorporate a deep neural network in the PIC/FLIP solver that effectively increases the resolution of the computational grid by mapping a coarse-grid velocity field to its corresponding fine-grid counterpart, reducing the dominant errors that are caused by computing the incompressible velocity field and grid to particle transfer on a coarse grid. The resulting PIC/FLIP solver will be subsequently called the multi-fidelity PIC/FLIP, as it utilises both low and high-fidelity data to train the deep neural network. A schematic overview of the steps in a low-fidelity and multi-fidelity PIC/FLIP solver is shown in figure 5.
In order to train a neural network, we need to specify [15, 16]:

- Why this approach?
- Define low-fidelity and high-fidelity,
- Input/output quantities for the neural network,
- Training data,
- Neural network architecture,
- Training procedure.

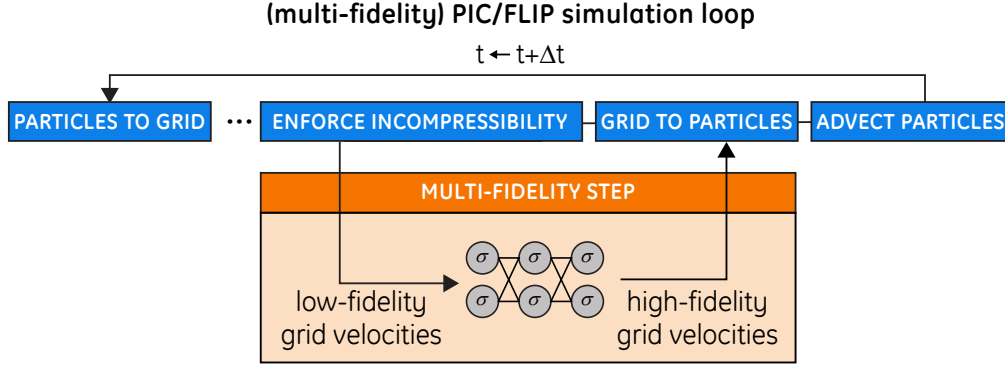The steps are discussed individually in the next subsections.

Figure 5: Steps in the simulation loop for a single time step in the (multi-fidelity) PIC/FLIP solver.

3.1 Why this approach?

The core idea is to learn low-level features of the low-fidelity simulations and map these to the corresponding fine-grid counterpart at every time step. Recent work shows that neural networks are perfectly suited for predicting fluid flows [17, 18, 19]. However, they are often used non-intrusively as post-processing tools or as standalone solvers. Strictly enforcing physical laws inside a neural network is challenging and often not possible. Using neural networks intrusively in a solver though allows the neural network to optimally utilise physical laws. Enhancing coarse-grid solutions by using low-level features is not new. It is done in for instance turbulence simulations using Large-Eddy Simulation (LES) [20], where subgrid features on a grid are modelled using quantities on that very only grid.

3.2 Defining low and high-fidelity PIC/FLIP

In order to apply our method for enhancing a low-fidelity PIC/FLIP solver, we need to define low and high-fidelity. As the computational bottleneck occurs in the computations on the grid, both the low and high-fidelity use the same number of particles, but differ in grid resolution. To clarify, a high-fidelity simulation corresponds to simulations on a fine-resolution computational grid with resolution Δs_{HF} where we place $n_{p,\text{init}} = 8$ particles in a wet cell during the initialisation, while a low-fidelity simulation corresponds to computations on a coarse-resolution computational grid with resolution Δs_{LF} and where we place $8 \left(\frac{\Delta s_{LF}}{\Delta s_{HF}} \right)^3$ particles in a wet cell during initialisation. Hence, the low-fidelity simulations have the same number of particles as the high-fidelity simulations and the combination of upscaling the resolution of the low-fidelity velocity grid and a large number of particles may result in an increase in accuracy. The grid resolutions Δs_{HF} and Δs_{LF} are specified in section 4. Lastly, the time step of the enhanced low-fidelity solver needs be such that the simulation remains stable when upscaling the velocity field. Therefore, we take the low-fidelity time-step the same as the high-fidelity time-step, which is chosen such that the high-fidelity simulation is stable. Even though a stable high-fidelity time-step is often orders of magnitude smaller than its low-fidelity counterpart, an upscaled low-fidelity solution is still beneficial as most of the computations are performed on a coarse grid which significantly reduces the time it takes to perform a single time-step when compared to high-fidelity computations.

3.3 Input/output quantities for the neural network

Our approach uses a deep neural network to enhance low-fidelity simulations at each time step. The inputs for the neural network have to be quantities that can be computed directly from the low-fidelity simulations at the current and/or previous time levels.

The choice of input and output quantities is motivated by the fact that the main error is caused by performing velocity calculations on a coarse grid. The input for the neural network is only the current state of the simulation as seen by the computational grid:

- Scaled number of particles $P_{i,j,k}^s$ in each grid cell:

$$P_{i,j,k}^s := \min \left(1, \frac{P_{i,j,k}}{n_{p,\text{init}} \left(\frac{\Delta s_{LF}}{\Delta s_{HF}} \right)^3} \right),$$ (18)

where $P_{i,j,k}$ is the number of particles in grid cell i, j, k.
- The face velocities defined on the low-fidelity grid $\mathbf{u}_{LF}$.

The scaled number of particles indicates if the cell is dry ($P_{i,j,k}^s = 0$), fully wet ($P_{i,j,k}^s = 1$), or partially wet ($0 < P_{i,j,k}^s < 1$). The $n_{p,\text{init}}$ in the definition of the scaled number of particles comes from the number of particles that are placed in a wet cell in the initialisation of a high-fidelity simulation (see section 2.1), and the ratio $\frac{\Delta s_{LF}}{\Delta s_{HF}}$ accounts for the difference in the number of particles placed in a wet cell in the initialisation. The scaled number of particles is also bounded to be within the interval $[0, 1]$. One can use the positions of all particles directly as input for the neural network, but we opt for using the scaled number of particles per cell as input, which significantly reduces the dimensionality of the input when many particles are simulated, at a slight reduction in the information content.

The outputs of the neural network are the new face velocities $\mathbf{u}_{HF}$, which are defined on the high-fidelity grid, hence, effectively increasing the resolution of the grid. This new and improved velocity field is then fed back into the solver and used to calculate the new particle velocities in the *grid to particles* transfer at the current time level with a smaller interpolation error (17).

3.4 Training data

The neural network is trained on a set of data that comprises both low- and high-fidelity data. The low-fidelity data is used as input for the neural network, while the high-fidelity data acts as a reference in the cost function, which is discussed in section 3.6. The amount of data required in the training set may vary depending on the application and is often found heuristically [15]. Furthermore, a validation set is constructed, which is used to tune the neural network architecture and hyperparameters, and a test set is used to test how well the network generalises.

We focus on performing sloshing simulations [21]. Sloshing is the movement of a liquid contained inside a (possibly moving) object. These types of fluid flow simulations are relevant in many industry and CGI applications. For our application we construct the data set that is used for training, validation and testing, as is shown in figure 6.
The procedure in figure 6 performs a specified number of time steps and the motion of the rectangular tank is imposed by rotating the gravity vector in the simulation, i.e., $\psi(t=0) = \phi(t=0) = 0$ and $\mathbf{g}(t) = 10(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$, where the angles change randomly over time. After the data set has been constructed, we split the entire set in 70% training, 20% validation and 10% testing samples [15].

The number of time steps, angle increments $\Delta\psi$, $\Delta\phi$, and the filling height determine how well the neural network performs and will be studied in detail in section 4.

3.5 Neural network architecture

Various options for the neural network type are available, e.g., fully-connected multilayer perceptron (MLP), convolutional neural network (CNN) or recurrent neural network (RNN), and determining which type to choose is often difficult. In general, the optimal neural network architecture depends on the application and a proper network type and architecture is often found by using expert knowledge.

Fully-connected, convolutional, recurrent, or ...?
In this study, there is a clear spatial component in both the input and the output of the neural network while there is no temporal component in the randomly picked training data. As a result, we opt for using a CNN architecture due to its inherent spatial nature. Furthermore, it reduces the total number of degrees of freedom significantly as compared to using an MLP architecture. The spatial nature of CNNs is caused by using local filters that take a weighted sum of neighbouring velocity values to compute a new velocity value. If we increase the number of layers, then the amount of neighbouring
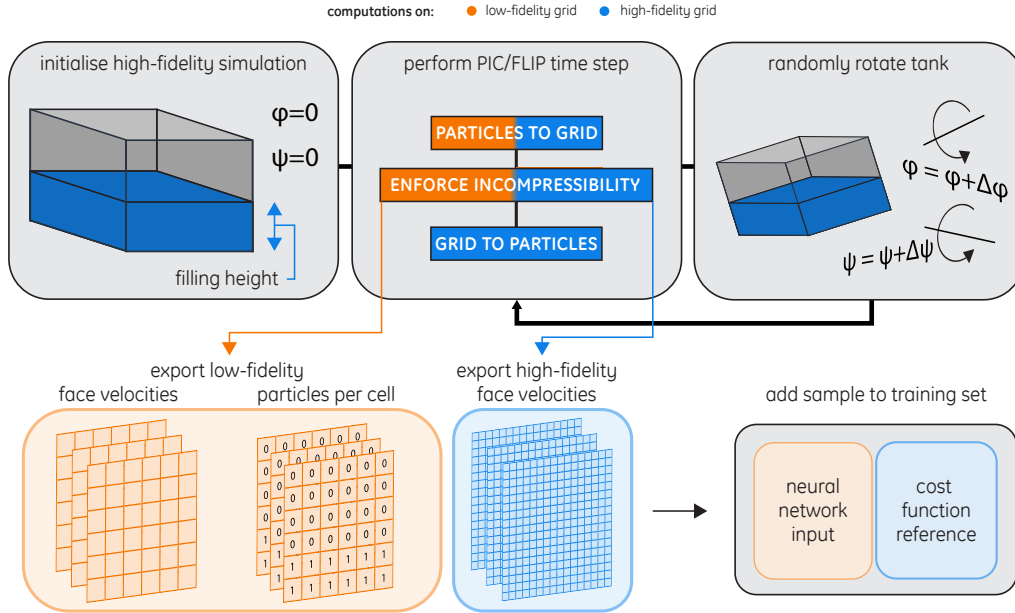
Figure 6: Schematic of how the data set is constructed that is used for training, validation, and testing. The magnitudes of the changes in angles $\Delta\psi$ and $\Delta\phi$ depend on the dimensions of the tank.

velocity values, that are used to calculate a new velocity value, increases. We do not know beforehand how many neighbouring velocity values should be used for accurately approximating the new velocity values and the number of CNN layers is therefore considered a hyperparameter in our approach.

How to increase the dimensions of the low-fidelity input to match the dimensions of the high-fidelity output?
When only using convolutional layers, the dimensionality of the input cannot be increased. This is problematic, because the goal of the neural network is to increase the effective grid resolution of the low-fidelity grid. To clarify, the low-fidelity input is of lower dimension than the high-fidelity output. This can be fixed by adding fully-connected layers after the convolutional part of the neural network. However these fully-connected layers significantly increase the number of degrees of freedom in the neural network and this leads to an increase of required training data. Constructing a larger training set requires more or longer high-fidelity simulations and is unwanted due to computational expense. The transposed convolutional layer, also known as deconvolutional layer [22, 23], solves this issue by performing a transposed matrix multiplication of the convolution matrix, which effectively increases the dimensions of the input and removes the need for adding fully-connected layers at the end of the network architecture. A schematic of the difference between a standard 2D convolutional layer and a 2D transposed convolutional layer is shown in figure 7.
The transpose convolutional layer is characterised by the number of filters, the filter size and the stride[15].

How to build the complete network architecture?
The complete neural network consists of a combination of both convolutional and transpose convolutional layers. The deconvolutional layers are used to increase the dimensionality of the neural network output, while the convolutional layers are used to effectively map the low-fidelity velocities to the high-fidelity velocities. A schematic overview of the complete network architecture is shown in figure 8.
The architecture is defined as follows:

- The network starts with a number of 3D standard convolutional layers with a specific number of filters (discussed later).
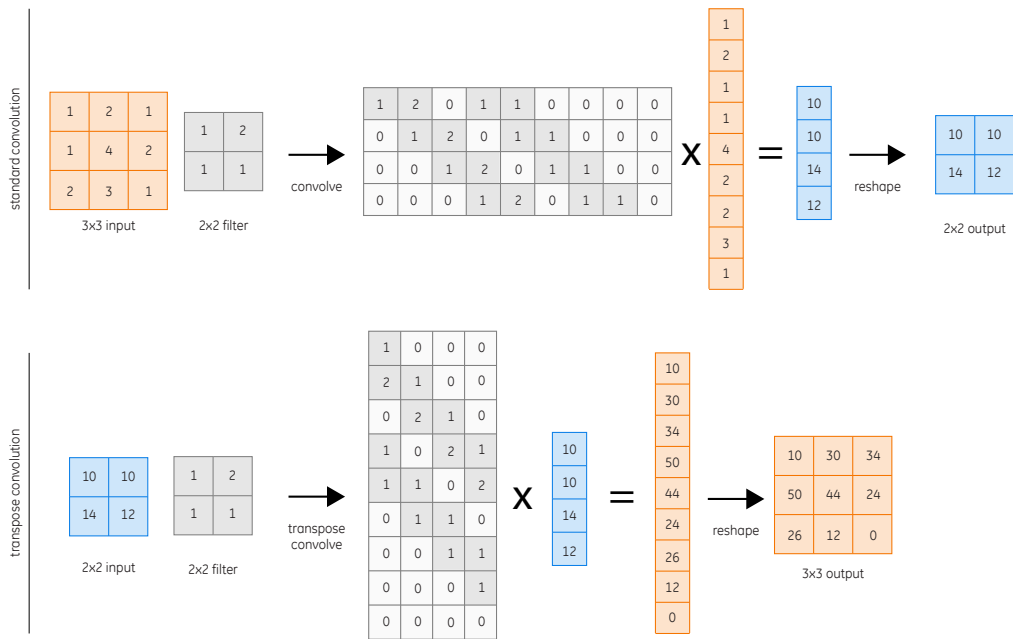
11

Figure 7: A standard 2D convolutional layer and a 2D transpose convolutional layer of a single kernel for a 1-channel input.
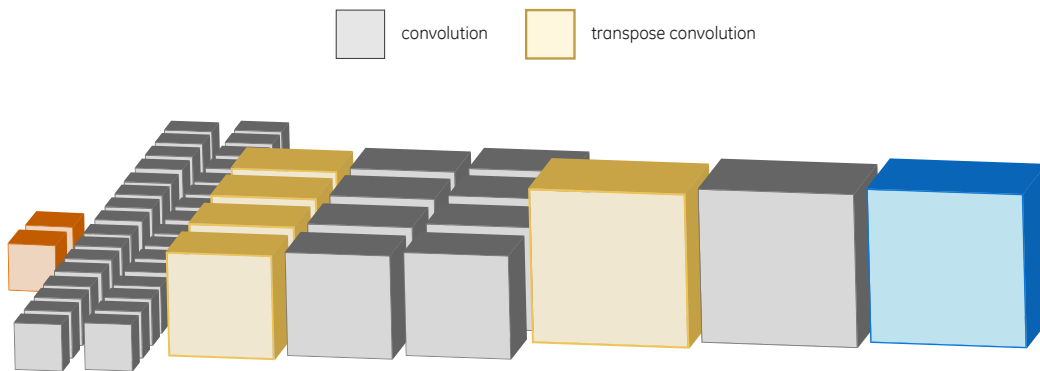

Figure 8: The complete neural network architecture with hyperparameter $N_{CNN} = 2$. The architecture consists of a combination of convolutional and transpose convolutional layers.

- After the first layer the network consists of groups of a single 3D transpose convolution, and N_{CNN} 3D standard convolutional layers, where N_{CNN} is a hyperparameter.
  - The transpose convolutional layers increase the dimensions of their input with a factor 2 in every dimension, by using a 2×2-filter and a stride 2.
  - The standard convolutional layers use a 3×3-filter, which is a commonly used filter size in deep convolutional neural networks [15].
  - Zero padding is used at the standard convolutional layers to keep the input and the output dimensions of these layers unaltered.

The number of pairs is determined by the input (low-fidelity grid) and the output (high-fidelity grid) and the number of filters for each pair of deconvolution and convolutional layers decreases with a factor 4, ending with a single filter for the output layer representing the high-fidelity velocity field. For example, if the input corresponds to a low-fidelity grid of $16 \times 16 \times 16$ cells and the output corresponds to a high-fidelity grid of $256 \times 256 \times 256$ cells, 4 pairs of transpose and standard convolutional layers are needed to match the dimensions of the output, where the first convolutional layer uses 64 filters. To assure that this makes sense, we require that the number of cells in each dimension on the high-fidelity grid is a multiple of 2 of the number of cells in the same dimension on the low-fidelity grid.

The network architecture is further characterised by the activation function used for each layer. In this work, we use a linear output activation to allow for unbounded values, and an exponential linear unit (ELU) [24] activation function for the remaining layers:

$$
\sigma(z) = \left\{ \begin{array}{ll} z & , z > 0 \, , \\ \gamma(e^z - 1) & , z \leq 0 \, , \end{array} \right. \tag{19}
$$

where γ is a hyperparameter. We prefer the ELU activation function over conventional activation functions, such as ReLU and hyperbolic tangent, as it is easy to evaluate, suffers less from the dying neuron problem as compared to conventional ReLU, suffers less from the vanishing gradient problem as compared to hyperbolic tangent activation [25], and because there is evidence that it speeds up training when compared to ReLU [24].

3.6 Training the neural network

The cost function c is a function of the weights and biases and indirectly depends on the hyperparameters. To prevent overfitting, regularisation will be used [15].

The randomly generated training data in section 3.4 is used to train neural networks with different choices for the hyperparameters discussed in section 3.5. The cost function that is minimised is given by:

$$
c_\lambda(W) = \sum_{i=1}^{N_t} \|(NN(\mathbf{u}_i^{LF}, \mathbf{p}_{\text{cell},i}) - \mathbf{u}_i^{HF}\|_2^2 + \lambda \left(\|W\|_2^2 \right) \, , \tag{20}
$$

where W are the trainable parameters (filter coefficients, biases) of the neural network NN, and where λ is the regularisation parameter, an additional hyperparameter for which a proper value needs to be determined during training. The cost function is minimised using the Adam optimizer with default parameter values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a step size α that is treated as a hyperparameter. Additionally, batch minimisation with batch size 256 is used to resolve out-of-memory errors, which may occur when passing a large dataset entirely for minimising the cost function.

We find proper values for the hyperparameters $(N_{CNN}, \gamma, \lambda, \alpha)$ by performing a grid search on the tensor grid H with 4 values for each hyperparameter. The 4 values for each hyperparameter are:

- *Number of CNN layers after a deconvolution layer*: $N_{CNN} \in \{1, 2, 3, 4\}$,
- *ELU shape parameter*: $\gamma \in \{0.0001, 0.001, 0.01, 0.1\}$,
- *Regularisation constant*: $\lambda \in \{10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}\}$,
- *Optimiser step size*: $\alpha \in \{0.0001, 0.001, 0.01, 0.1\}$.

As a result, we need to train $4^4 = 256$ neural networks, one for each different combination of the hyperparameters. From this set of trained neural networks, the optimal network is defined as the one

that has the smallest validation error

$$\arg \min_{(N_{CNN}, \gamma, \lambda, \alpha) \in H} \left[\sum_{i=1}^{N_v} \|(NN(\mathbf{u}_i^{LF}, \mathbf{p}_{\text{cell},i}) - \mathbf{u}_i^{HF}\|_2^2 \right] . \tag{21}$$

The test set can be used to determine if the found optimal neural network generalises well to unseen data. The resulting neural network is used to enhance the low-fidelity PIC/FLIP simulations.

4 Results

This section discusses the results of training and applying a multi-fidelity PIC/FLIP solver for fluid sloshing. The Tensorflow library for Python3 [26] is used for constructing and training the neural networks, and it runs on 4 NVIDIA GTX 1080Ti GPUs. The PIC/FLIP solver is also implemented to run on 4 NVIDIA GTX 1080Ti GPUs. The section is divided into four parts:

- Enhancing low-fidelity PIC/FLIP sloshing simulations.
- Generalisation capabilities for solver parameters.
- Generalisation to a 3D dambreak problem.
- A discussion of weaknesses of the multi-fidelity PIC/FLIP solver.

For all simulations we take a computational domain $(x, y, z) \in [0, 10] \times [0, 5] \times [0, 5]$, unless stated otherwise. The high-fidelity PIC/FLIP solutions are computed with $\Delta s = 1/50$ ($500 \times 250 \times 250$ cells) and simulate up to 250 million particles. The low-fidelity PIC/FLIP solutions are computed with $\Delta s = 1/10$ ($100 \times 50 \times 50$ cells) and simulate up to 2 million particles. The grid resolution of the low-fidelity simulations is chosen such that it is able to run in real-time, while the high-fidelity simulation grid contains 125 times more grid cells and can not be performed in real-time. The time step needs to be the same for both fidelities and is set to $\Delta t = 1/350$ which results in stable low- and high-fidelity simulations. Lastly, the PICness f is set to 0.99 for both fidelities, which is a commonly used value in literature for simulating free surface water flow [11].

4.1 Enhancing low-fidelity PIC/FLIP sloshing simulations

We show that the proposed multi-fidelity approach can indeed enhance low-fidelity PIC/FLIP sloshing simulations.

The type of solutions that are enhanced
In order to study the effectiveness of a multi-fidelity PIC/FLIP solver, we prescribe a tank motion with $\mathbf{g} = 9.81(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$, where ψ and ϕ are given by:

$$\psi(t) = \begin{cases} \sin(2t) & , t < 2\pi , \\ 0 & , t \geq 2\pi , \end{cases} \quad \phi(t) = \begin{cases} \sin(t) & , t < 2\pi . \\ 0 & , t \geq 2\pi . \end{cases} \tag{22}$$

We choose a deterministic motion as we have full control over the sloshing phenomenon encountered in the simulation, which is not possible when picking a random tank motion. This particular motion causes heavy liquid sloshing in the tank and is perfectly suited for testing our method. The filling height is varied to study generalisation capabilities of the method, and ranges between $(0, 100)\%$.

Construction of the training set
The procedure for constructing the training data is shown in figure 6. A summary of all training parameters is given by:

- Data set: 10^6 generated training samples (see figure 6) with a filling height of 40%, $\Delta\psi, \Delta\phi$ are picked randomly every time step in the ranges $[-2\Delta t, 2\Delta t]$, $[-\Delta t, \Delta t]$, respectively.
  - 70% training,
  - 20% validation,
  - 10% testing,
- Training parameters: batch-size=256 , number of epochs=1000.

14

The training set only comprises samples with a single filling height of 40%, which corresponds to a filling height that allows for heavy sloshing motions and therefore a wide range of fluid configurations. This allows us to study if the neural network generalises well to cases with a different filling height. The range for $(\Delta\psi, \Delta\phi)$ corresponds to the same oscillation frequency as the one in (22). However, the probability that the tank motion (22) is comprised in the training data is negligible.

Training the neural networks
A total of 256 neural networks need to be trained (1 for each set of hyperparameters), and the resulting training and validation errors are shown in figure 9.
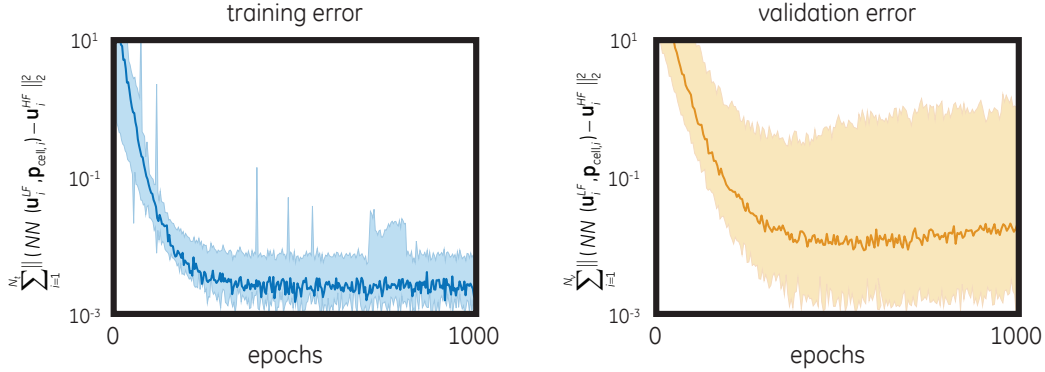

Figure 9: The mean and envelope of both the training and validation error. The envelope shows the minimum and maximum values over all 256 trained neural networks.

We clearly see that all 256 trained neural networks converge to a similar error in the training set, while the variation in the validation error is more pronounced. This indicates under/overfitting and a lack of generalisation capabilities for specific hyperparameter values. Remarkable is that the 103 networks with the largest validation errors are the networks where $N_{CNN} = 1, 2$ which indicates that these relatively local multi-fidelity velocity approximations are not capable of generalisation outside the training set. This indicates that to approximate a local multi-fidelity velocity, we may need a large area of surrounding low-fidelity velocities. This global character of the required approximation may be caused by the incompressibility of the fluid, which leads to a coupling of velocities by means of the pressure Poisson equation (12) which is also felt globally. The hyperparameters $(N_{CNN}, \gamma, \lambda, \alpha) = (3, 0.01, 10^{-4}, 0.001)$ satisfy (21), and the trained neural network with these hyperparameter values is used in the remainder of this paper.

Enhancing solution with the same filling height as used during training
First, we start by using the trained neural network to enhance a sloshing simulation with the motion given by (22) given a filling height of 40%. Notice that this is the same filling height that is used for constructing the training set and is therefore considered a consistency test for the trained neural network. The low-fidelity, multi-fidelity, and high-fidelity PIC/FLIP simulations are compared at $t = \pi, 3\pi$. The rendered comparison is shown in figure 10.
When we compare both low- and multi-fidelity solution to the reference high-fidelity solution, we see a clear qualitative improvement when using the multi-fidelity solver. Notice that the multi-fidelity solution at $t = \pi$ shows smaller scale droplets when compared to the low-fidelity solution, which is caused by effectively increasing the resolution of the grid. Furthermore, the multi-fidelity and the high-fidelity solutions at $t = 3\pi$ show a similar fluid surface, while the low-fidelity fluid surface has a completely different shape.

To study the difference between the three fidelities quantitatively, we perform a simulation until $t = 4\pi$ for each of the three fidelities, and compare the so-called fluid-match. The fluid-match is defined as the percentage of the total volume that coincides with the high-fidelity solution, resulting in a fluid-match that is close to 100% if a solution is close to the high-fidelity reference and almost 0% when the simulation is completely off. This is used as a measure to quantify how close a solution is to the reference high-fidelity solution, i.e., if the fluid-match is 100%, then the fluid surface in the performed simulation is the same as the high-fidelity fluid surface. Notice that this does not necessarily mean that the neural network velocities are the same as the high-fidelity velocities. A quantitative comparison between the different fidelities is shown in figure 11.
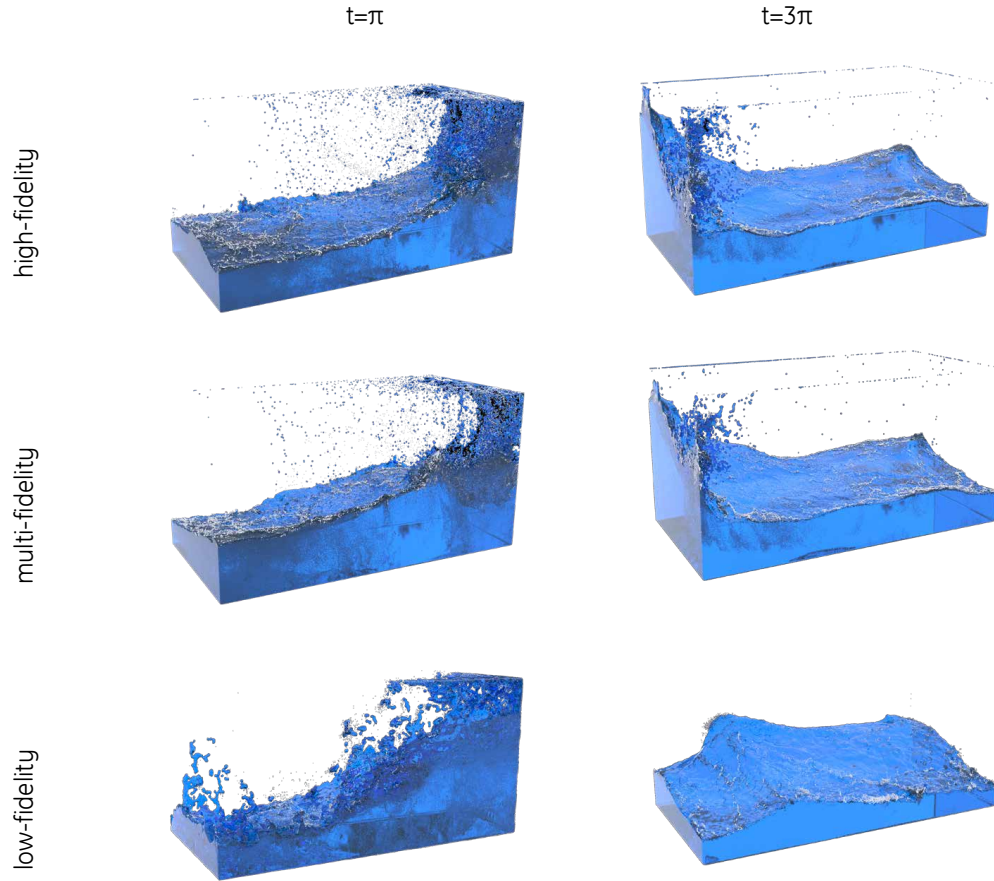
Figure 10: Rendered fluid surface for the three different fidelities at $t = \pi, 3\pi$ with a filling height of 40%. The white particles are passive tracers that represent foam and spray [27] and which are added in a post-processing stage by detecting where air entrapment is occurring during the simulation, i.e., parts with high velocity gradients and areas where the fluid surface is concave.

A smooth initial part of the fluid-match resembles the period before the fluid impacts the boundaries of the domain. After the first impact, the fluid-match oscillates. When simulating for a longer period, the fluid-match converges back to almost 100%, due to the fluid coming to rest again. Notice that the fluid-match is not exactly 100% after the fluid being almost at rest, which is due to the fact that the PIC/FLIP solver does not preserve fluid volume in the presence of large particle velocities. These large velocities may result in particles being more densely clustered which causes the fluid volume to shrink. We clearly see a significant improvement in fluid-match when using the multi-fidelity PIC/FLIP solver, while not significantly increasing computational time. The neural network produces slightly compressible face-velocities, which may also cause the shrinking and growing of the total fluid volume. Lastly, the multi-fidelity and high-fidelity velocity fields show a gradual increase of mismatch in velocity values. The oscillating behaviour with large outliers of the velocity mismatch is caused by the difference in droplet distributions, as shown in figure 10 at $t = \pi$. These droplets often have high velocities and a slight difference in droplet positions may cause a large mismatch in the face velocities. However, because these large outliers in velocity mismatch are caused by droplets, this large mismatch is not seen in the fluid-match, as they only affect the flow locally in a small part of the computational domain. ***To summarise, the multi-fidelity solver significantly increases accuracy in terms of fluid-match for a sloshing simulation with the same filling height that was used during training, but generates slightly compressible velocity fields.***

Enhancing solutions with different filling heights
Enhancing solutions where the filling height differs from the filling height that was used during
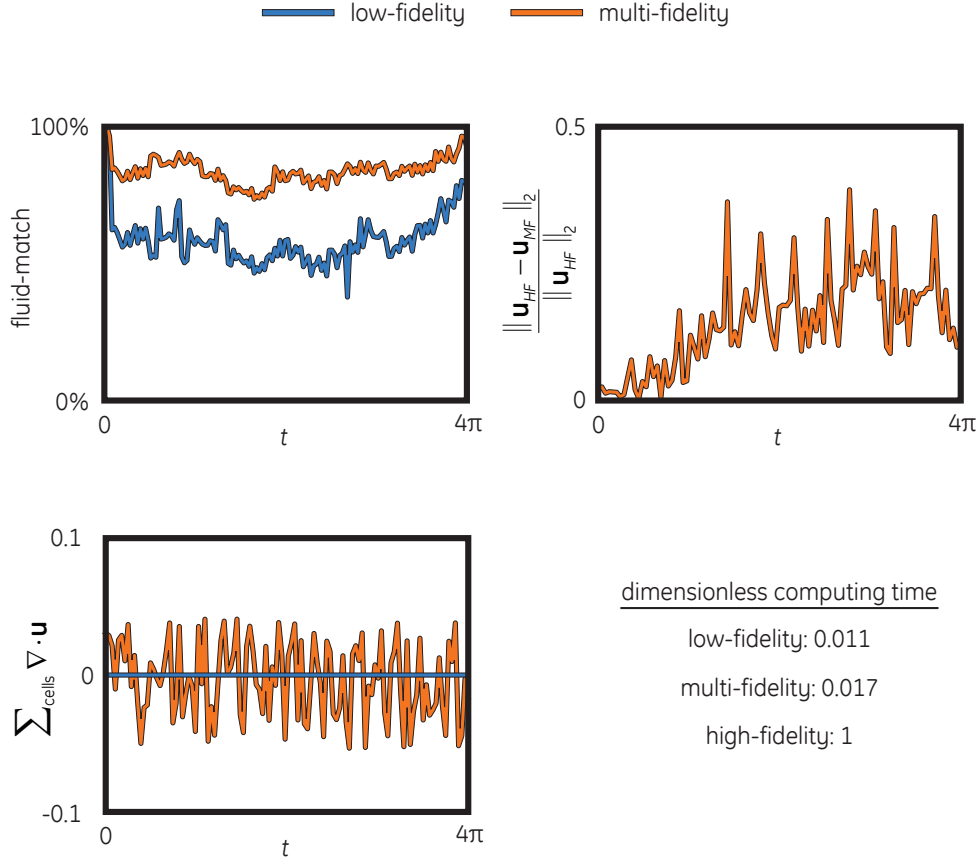
Figure 11: Fluid-match, multi-fidelity incompressibility computed with (9), and multi-fidelity (MF) relative velocity error.

training is more difficult, as the trained neural network has never seen inputs that are associated with different filling heights. Figure 12 shows how well the neural network generalises for filling heights outside the training set.

The multi-fidelity solver outperforms the low-fidelity solver for a significant portion of filling heights (indicated by the green area in figure 12). The low-fidelity fluid-match shows improvement in fluid-match with increasing filling height, which eventually leads to out-performance of the multi-fidelity solver. This increasing trend is caused by the fact that fluid sloshing affects the fluid-match less and less when the filling height goes to 100% (completely filled tank without fluid sloshing for which low-fidelity is already close to high-fidelity simulation).

To summarise, the multi-fidelity solver is able to the generalise to other filling heights, but leads to inaccurate results when deviating too much from the training set. Furthermore, without a high-fidelity reference, which is not available after training, it is difficult to predict the limits of the generalisation capabilities of the trained neural network. This is a common problem in machine learning and is an active field of research.

4.2 Generalisation capabilities for solver parameter changes

The filling height is only one out of several parameters that were used in the sloshing test case. Studying how the multi-fidelity solver generalises when changing solver parameters provides information on how sensitive the neural network is to the training data. In this section we show how the multi-fidelity solver generalises when changing the following parameters:

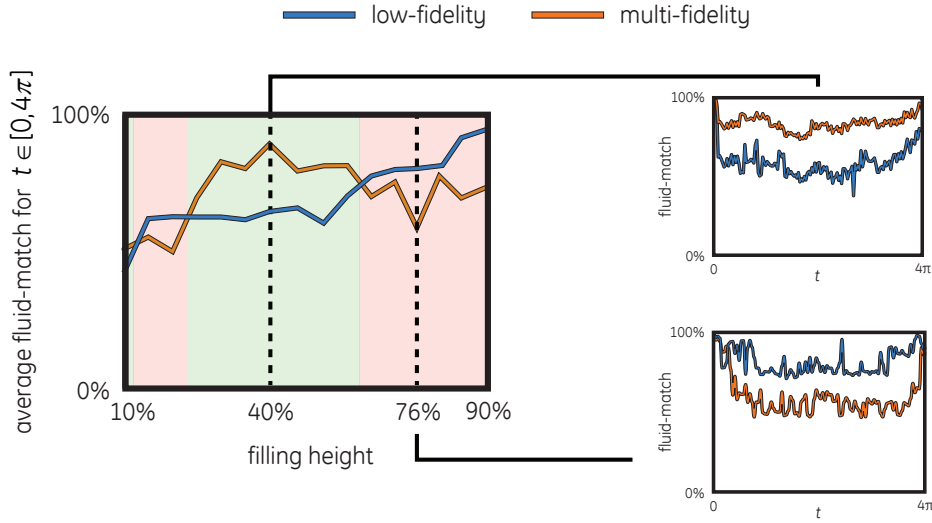- ***Time step***: $\Delta t \in \left[\frac{1}{275}, \frac{1}{500}\right]$.

17

Figure 12: Error in enhanced sloshing simulations for filling heights that were not used during training.

  - The time step is picked such that the high-fidelity simulations are stable, which makes it easy to compare all three fidelities.

- **Computational domain size scaling**: $c_{\Delta s} \in [\frac{1}{2}, \frac{3}{2}]$.

  - While keeping the number of cells in the low- and high-fidelity simulations the same, we can change $\Delta s \to c_{\Delta s}\Delta s$ to effectively increase the size of the computational domain.

- **Number of particles in wet cell during initialisation**: $n_{p,\text{init}} \in [4, 12]$.

  - The wet cells in the multi-fidelity simulation are initialised with $n_{p,\text{init}} \left(\frac{\Delta s_{LF}}{\Delta s_{HF}} \right)^3$ particles ($n_{p,\text{init}} = 8$ was used for training).

- **PICness parameter**: $f \in [0, 1]$.

  - Determines how diffusive the fluid motions are. Changing this value significantly changes the behaviour of the fluid ($f = 0.99$ was used for training).

- **Gravity vector scaling**: $c_{\mathbf{g}} \in [5, 15]$.

  - The gravity vector will be defined as $\mathbf{g} = c_{\mathbf{g}}(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$ ($c_{\mathbf{g}} = 9.81$ was used for training). The value for $c_{\mathbf{g}}$ determines the strength of the gravitational force and changing this value significantly changes fluid behaviour.

The fluid-matches for the low- and multi-fidelity simulations are compared for the sloshing simulation characterised by the sloshing motion (22) for $t \in [0, 4\pi]$. The results are shown in figure 13.

We clearly see that the multi-fidelity solver generalises well for all solver parameters apart from the PICness parameter f and gravity scaling constant $c_{\mathbf{g}}$. The parameters Δt, Δs and $n_{p,\text{init}}$ do not significantly change the fluid motions and the neural network will receive inputs that are similar to the training data. As a result, the multi-fidelity solver attains a significant increase in accuracy when compared to the low-fidelity solver. The parameters f and $c_{\mathbf{g}}$ do significantly change the fluid behaviour though, which results in neural network inputs that significantly deviate from the training data.

To summarise, for a large portion of the parameter value ranges the multi-fidelity solutions show an increase in accuracy when compared to the low-fidelity solutions. However, it is not possible to generalise this to cases where c_g and f deviate significantly from the training data.
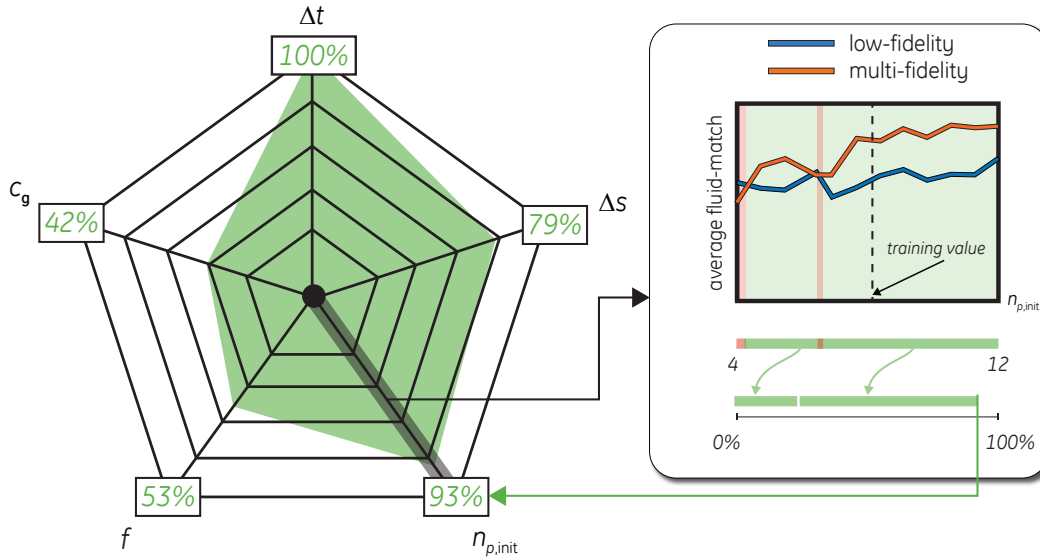
18

Figure 13: Generalisation capabilities for different solver parameters. The green regions indicate the fraction of parameter values for which the multi-fidelity fluid-match is higher than the low-fidelity fluid-match for the sloshing simulation for $t \in [0, 4\pi]$.

4.3 Generalisation to a 3D dambreak problem

It was shown in sections 4.1 that the trained neural network, that was used in the multi-fidelity solver, was able to generalise to different filling heights and solver parameters values. However, the simulations were still similar in the sense that the prescribed motion (22) was the same and the initial fluid configuration was similar. In this section we show that the trained neural network is able to generalise to a different test case as well.

We consider a wet dambreak problem of which the initial fluid configuration is shown in figure 14.
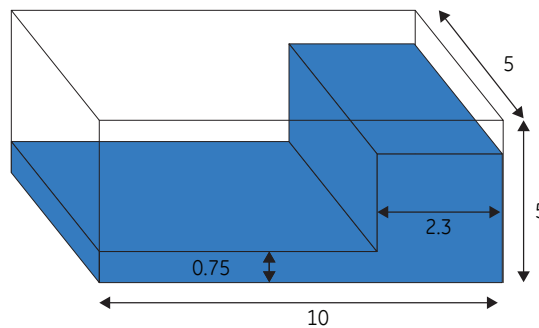

Figure 14: The initial fluid configuration of the wet dambreak problem.

This particular configuration is chosen such that the total volume corresponds to a 30% filling height. This allows us to study if the neural network generalises to a case with a filling height different from the one that was used for training, and was also initialised in a different configuration. The dambreak problem leads to a breaking wave that impacts the left domain boundary. The low and multi-fidelity solutions are compared for $t \in [0, 15]$, i.e., the period before, during, and after the wave impact. A qualitative and quantitative comparison of the three fidelities is shown in figure 15.

Both the low- and multi-fidelity simulation show good agreement with the high-fidelity solution. The effectiveness of the neural network approach is again emphasised by the fluid-match, which shows an increase in accuracy for the multi-fidelity solution, even for this dambreak problem for which
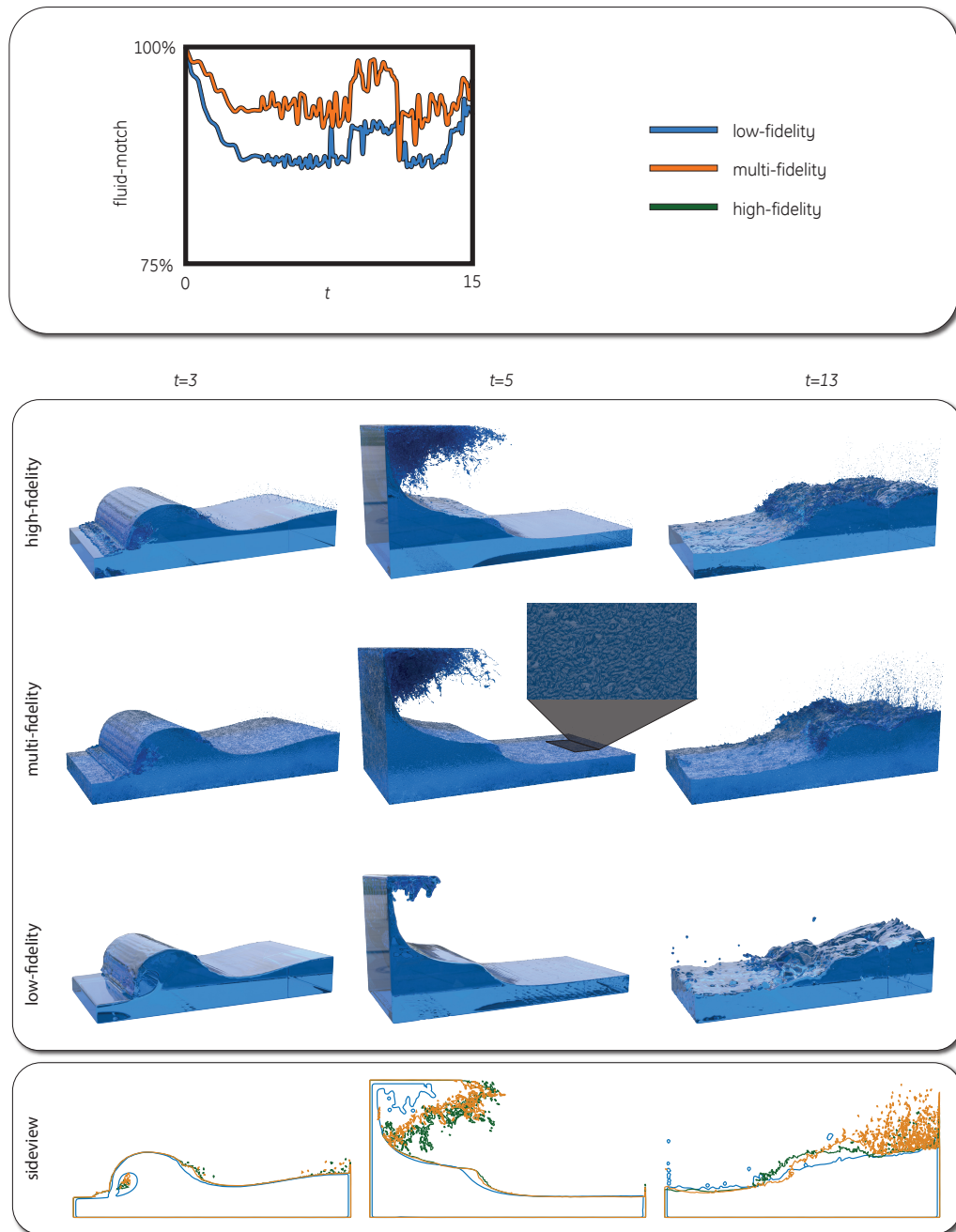
Figure 15: A comparison of the low-, multi-, and high-fidelity solutions for the dambreak problem. The multi-fidelity solution at $t = 5$ shows a close-up of the tiny oscillations on the fluid surface.

the training set was not tailored. However, the multi-fidelity solution shows tiny oscillations on the fluid surface, which is caused by oscillatory velocity predictions. The frequency of the oscillation corresponds to the largest frequency that can be resolved on the high-fidelity computational grid . They immediately appear in the first few time steps of the multi-fidelity simulation and grow slightly until the wave impacts the domain boundary. This phenomenon is most likely caused by the configuration of the initial condition which is not comprised in the training set and therefore not seen by the neural network during training. Even though these oscillations are unwanted, they do not

affect the accuracy of the multi-fidelity solution severely. In order to possibly remove the unwanted oscillations, one may enlarge the training set to encapsulate more test cases.

To summarise, these results indicate that the generalisation to other test cases is possible, but a carefully constructed training set may be needed to remove unwanted artefacts in the enhanced velocity field.

4.4 Weaknesses of the multi-fidelity PIC/FLIP solver

It was shown that our approach is able to generalise to solver parameters and test cases slightly outside the training set. However, the problem of generalisation to cases further away from the training data still exists. In this section we give an overview of the problems we encountered during training of the neural network and of the limitations of the proposed approach. In short the limitations of our multi-fidelity approach concern:

- Sensitivity to solver parameters.
- Does not generalise to other test cases.
- Difficult to obtain physical interpretation of the multi-fidelity approach.

Sensitivity to solver parameters
As is shown in figure 13, the neural network is able to simulate cases with slightly different solver parameters. It is possible to construct a training set that comprises simulations with multiple solver parameters, which increases the range of solver parameters for which the multi-fidelity solver produces satisfactory results. However, as the training set is constructed with low- and high-fidelity data, we need to make sure that the high-fidelity simulations are stable, which results in a small time step. We chose to use this small time step for the low-fidelity simulations, which results in a multi-fidelity solver that also uses a similar small time step. An alternative is to construct the training set with low-fidelity simulations, computed with a large time step, and a high-fidelity counterpart, computed with a small time step. The resulting training set then comprises all low-fidelity data and the sub sampled high-fidelity simulation data at the same time levels as the low-fidelity simulation. This approach leads to unstable multi-fidelity solutions which indicates that the multi-fidelity solver needs to satisfy similar stability conditions as the high-fidelity solver.

Generalisation to other test cases
We showed that the neural network, which was trained on sloshing simulation data, is able to generalise to a wet dambreak problem. This is a promising result, but using the multi-fidelity solver for still more differing cases leads to unsatisfactory results. In figure 16 we show three example high-fidelity simulations where the multi-fidelity solver became unstable after only a few time steps. These cases all have features that the neural network has never encountered before, i.e., a dry bed, static obstacles in the flow, and obstacles moving through the fluid. This is consistent with the results in [9, 10] where it is noted that generalisation is often an issue when using neural networks for fluid flow predictions. A carefully constructed training set may be needed to be able to generalise to a wide range of test cases.

Physics interpretation of the multi-fidelity approach
The idea of our approach is to learn low-level features of the low-fidelity simulations and map these to the corresponding fine-grid counterpart. An in-depth analysis of why this approach works when using neural networks is challenging, due to the inherent non-linearities of the neural networks and the underlying physics.

5 Conclusion

We investigated the use of deep deconvolutional neural networks to enhance a low-fidelity PIC/FLIP fluid solver. An important novelty of our approach is that it is used intrusively in the low-fidelity solver and is able to enhance the solution at every time step. This is different from the common non-intrusive approaches that use neural networks either as a stand-alone solver or as a post-processing tool. The neural network in this paper was trained on randomly generated fluid sloshing data. After training, the neural network provided a significant increase in accuracy when compared to the low-fidelity solution, for the case of a sloshing simulation that was not comprised in the training data. Although the neural network introduced small oscillations on the fluid surface when considering a dambreak problem, it
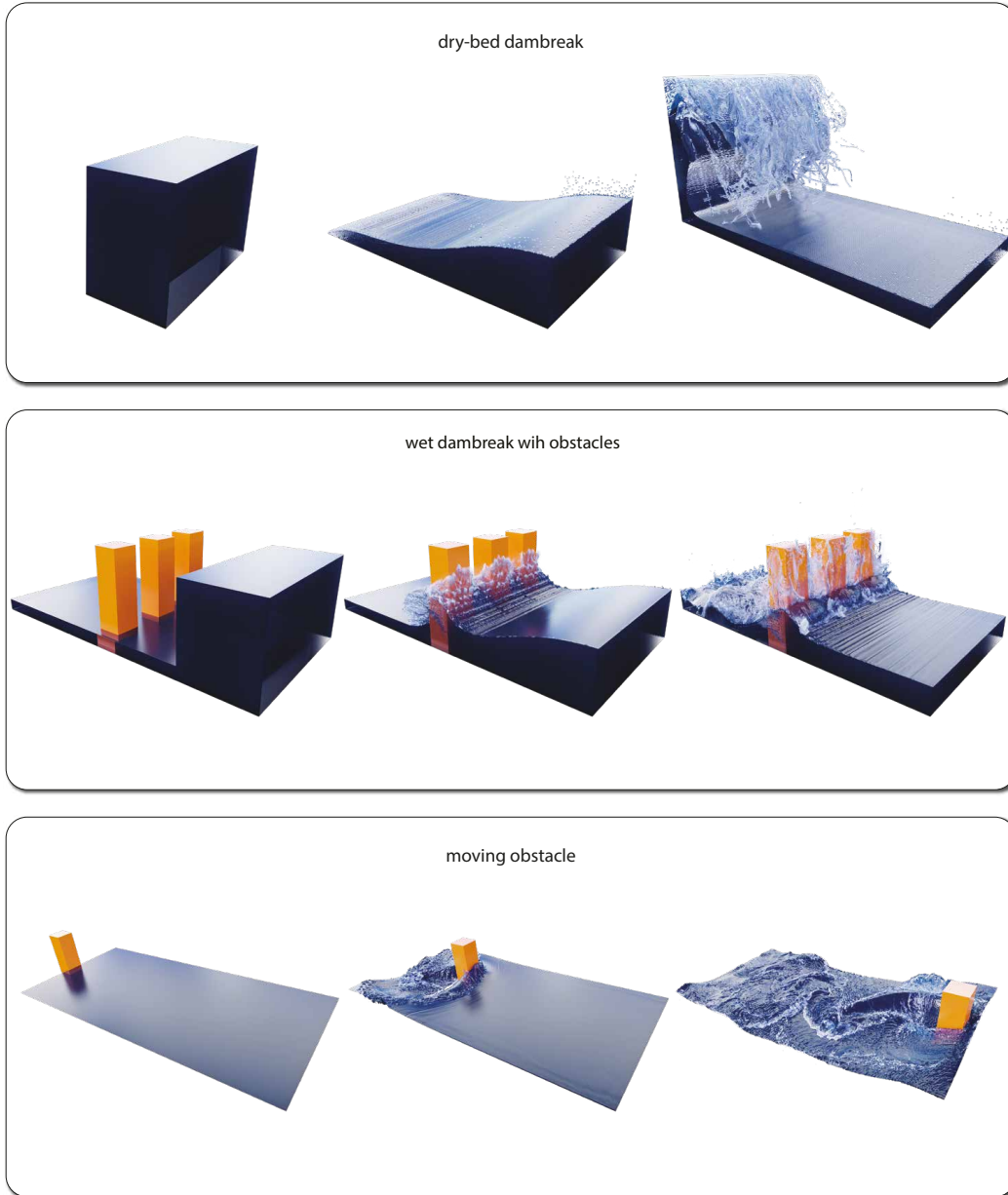
Figure 16: Three example high-fidelity simulations where the multi-fidelity solver fails. The dry-bed dambreak consists of a column of fluid that is released at $t = 0$, resulting in a wave impact on the domain boundary. The dambreak with obstacle uses the same initial fluid configuration as shown in figure 14 but has solid obstacles (orange boxes) in the middle of the domain. The moving obstacle case consists of an obstacle (orange box) that is moving in a sinusoidal motion through a tank that is 40% filled with fluid.

was still able to significantly enhance accuracy for this problem, which hints at the generality of our approach.

Our approach still has some issues. Firstly, the choice for the time step is rather restrictive, the time step needs to be kept small to keep the multi-fidelity solution stable. Secondly, our approach fails to enhance solutions that introduce new phenomena that were not encountered in the training set, e.g., obstacles in the flow. We expect that a carefully constructed training set may alleviate this issue and developing a procedure for this is scheduled for future work.

References

[1] A. Barreiro, A. J. C. Crespo, J. M. Domínguez, and M. Gómez-Gesteira, "Smoothed particle hydrodynamics for coastal engineering problems," *Computers & Structures*, vol. 120, pp. 96–106, Apr. 2013.

[2] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, USA), pp. 481–490, ACM, 2016.

[3] A. McAdams, A. Selle, K. Ward, E. Sifakis, and J. Teran, "Detail preserving continuum simulation of straight hair," in *ACM SIGGRAPH 2009*, SIGGRAPH '09, (New York, USA), pp. 62:1–62:6, ACM, 2009.

[4] B. Frost, A. Stomakhin, and H. Narita, "Moana: performing water," in *ACM SIGGRAPH 2017*, SIGGRAPH '17, (New York, USA), pp. 30:1–30:2, ACM, 2017.

[5] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, "A material point method for snow simulation," *ACM Trans. Graph.*, vol. 32, pp. 102:1–102:10, July 2013.

[6] A. Angelidis, F. Neyret, K. Singh, and D. Nowrouzezahrai, "A controllable, fast and stable basis for vortex based smoke simulation," in *ACM SIGGRAPH 2006*, SCA '06, (Aire-la-Ville, Switzerland), pp. 25–32, Eurographics Association, 2006.

[7] S. S. Ravindran, "Reduced-order adaptive controllers for fluid flows using POD," *Journal of Scientific Computing*, vol. 15, no. 4, pp. 457–478, 2000.

[8] C. Audouze, F. D. Vuyst, and P. B. Nair, "Reduced-order modeling of parameterized PDEs using time–space-parameter principal component analysis," *International Journal for Numerical Methods in Engineering*, vol. 80, no. 8, pp. 1025–1057, 2009.

[9] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven fluid simulations using regression forests," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 199:1–199:9, 2015.

[10] L. Ladický, S. Jeong, N. Bartolovic, M. Pollefeys, and M. Gross, "Physicsforests: real-time fluid simulation using machine learning," in *ACM SIGGRAPH 2017*, pp. 22–22, 2017.

[11] Y. Zhu and R. Bridson, "Animating sand as a fluid," in *ACM SIGGRAPH 2005*, SIGGRAPH '05, (New York, USA), pp. 965–972, ACM, 2005.

[12] R. Bridson, *Fluid Simulation for Computer Graphics*. CRC Press, 2015.

[13] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin, "The affine particle-in-cell method," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 51:1–51:10, 2015.

[14] G. M. Phillips, *Interpolation and Approximation by Polynomials*. Springer-Verlag, 2003.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Springer, 2018.

[17] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, 2020.

[18] Y. Shin, J. Darbon, and G. E. Karniadakis, "On the convergence and generalization of physics informed neural networks," 2020.

[19] X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "NSfnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations," 2020.

[20] C. Meneveau and P. Sagaut, *Large Eddy Simulation for Incompressible Flows: An Introduction*. Scientific Computation, Springer Berlin Heidelberg, 2006.

[21] R. A. Ibrahim, *Liquid Sloshing Dynamics: Theory and Applications*. Cambridge University Press, 2005.

[22] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016.

[23] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015.

[24] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by Exponential Linear Units (ELUs)," *arXiv:1511.07289 [cs]*, Nov. 2015. arXiv: 1511.07289.

[25] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," in *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[26] M. A. et al., "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[27] M. Ihmsen, N. Akinci, G. Akinci, and M. Teschner, "Unified spray, foam and air bubbles for particle-based fluids," *The Visual Computer*, vol. 28, pp. 669–677, 06 2012.